Week 2

```
// synchronous function
function sum(a,b){
    return a+b;
}
console.log(sum(1,2));
console.log(sum(2,3));
console.log(sum(3,6))
console.log(sum(43,1));
```

```
node /tmp/ItbJ3JUXHv.js
3
5
9
44
```

This is the synchronous function .

As we know that the javascript is a single threaded language. So in these type of function the thread is always busy.executing each line of code line by line .

Q What is Asynchronous function ?

Your javascript thread doesn't have access to everything immediately. There are some tasks it needs to wait for.

Example -

Reading a file , sending a network request , a deliberate timeout .

Now suppose you want to access the file from you file system .so now I tell the javascript go read this file .

Remember that your code doesn't have access to all the file on your computer system . It always need to ask your OS system . Then the OS will decide whether to give the file or not .

This is not the case that your javascript asked for a file immediately . The os takes it time to give the file access to your code it can be 1 sec,2 sec or whatever.but the OS will not immediately give the content of this file .

So for this kind of operation we need async function . So that our thread doesn't get block waiting for an OS to complete its operation.

PROMISES

Underthehood promises uses callbacks . It just a syntactical sugar.

It just a way to write a more readable code to write an async calls .

Q. Now the question come can we create our own async function ?

You can, but it is not needed . Most of the time you just wrap around another async function.
For example -

Read and write operation -

```js
const fs=require("fs");

function myownReadandWriteFile(){
  fs.readFile("a.txt","utf-8",function(err,data)
{
    console.log("current data",data)
    data=data+"i am a cse student"
    fs.writeFile("a.txt",data,function(err){
      console.log("file successfully updated")
    })


  })
}
myownReadandWriteFile()
```

```
current data hi my name is y
ash rawat.
file successfully updated
```

Another example -

```js
// my own asynchronous function
function myOwnSetTimeout(fn,duration){
    setTimeout(fn,duration);
}
myOwnSetTimeout(function(){
    console.log("hello there")
},1000)
```

```
node /tmp/ItbJ3JUXHv.js
hello there
```

Q why do we need PROMISES if it simply uses the callbacks underthehood ?

We need Promises to write a more readable code and to avoid callbacks .
For example

```js
function callBacksHells(){
    console.log("inside the function")
    setTimeout(function(){
        console.log("hello this is 1 ")
        setTimeout(function(){
            console.log("hello this is 2")
            setTimeout(function(){
                console.log("hello this is 3")
                setTimeout(function(){
                    console.log("hello this is 4")
                    setTimeout(function(){
                        console.log("hello this is 5")

                    },6000)
                },5000)
            },4000)
        },3000)
    },2000)
}
callBacksHells()
```

```
node /tmp/ItbJ3JUXHv.js
inside the function
hello this is 1
hello this is 2
hello this is 3
hello this is 4
hello this is 5
```

This approach uses a callback .as you can the unnecessary indentation. This is what we call as callback hells.

And this is the promisified version of it -

```
function myOwnSetFunction(duration){
    var p=new Promise(function(resolve){
        setTimeout(resolve,duration)
    })
    return p;
}
myOwnSetFunction(2000).then(function(){
    console.log("hello this is 1")
})
myOwnSetFunction(3000).then(function(){
    console.log("hello this is 2")
})
myOwnSetFunction(4000).then(function(){
    console.log("hello this is 3")
})
myOwnSetFunction(5000).then(function(){
    console.log("hello this is 4")
})
myOwnSetFunction(6000).then(function(){
    console.log("hello this is 5")
})
```

```
node /tmp/mXP1aQyjYA.js
hello this is 1
hello this is 2
hello this is 3
hello this is 4
hello this is 5
```

Look how readable is the code from the previous one.

NOTE - .then method tells the function that once the output is ready you have to run the code written in the then method .

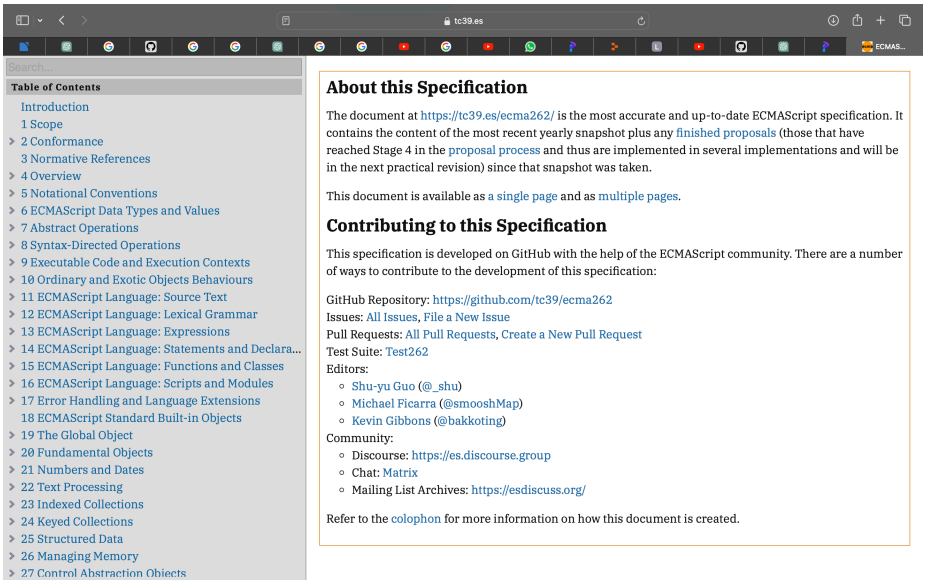
Mainly you need the async code -

1.do a network call
2.sleep/wait for some time
3.read a file
4.database call

CLASS 2.2 - NODE.JS RUNTIME || HTTP SERVER

Q. WHAT IS ECMASCRIPT ?

ECMAScript (often abbreviated as ES) is a set of rules that defines how JavaScript should work. It's like a guidebook that developers follow to ensure that JavaScript behaves consistently across different platforms and environments. So, when people talk about ECMAScript, they're usually referring to the standards that shape the JavaScript language.



Whoever wants to create a complier for javascript needs to make sure that meets all the requirement mentioned in ECMASCRIPT.

JAVASCRIPT - It's a programming language that implements the ECMAScript standard.

BEYOND ECMASCRIPT

JavaScript, as a language, often includes additional features or APIs beyond what is specified in the ECMAScript standard. These extra features are usually specific to the environment where JavaScript is running, such as web browsers or server-side environments like Node.js. Here are a few examples:

- DOM (Document Object Model): JavaScript in web browsers has additional features for interacting with the Document Object Model, allowing manipulation of HTML and XML documents.
- Web APIs: Browsers provide additional APIs like fetch for making network requests, localStorage for client-side storage, and Canvas for drawing graphics.
- Node.js APIs: When using JavaScript on the server side with Node.js, you have access to various APIs for file system operations, networking, and other server-related tasks.
- Browser-specific features: JavaScript in browsers may have features specific to certain browsers, like event handling or CSS styling through JavaScript.

These additional features go beyond the core ECMAScript standard and provide capabilities that are specific to the runtime environment in which JavaScript is executed.

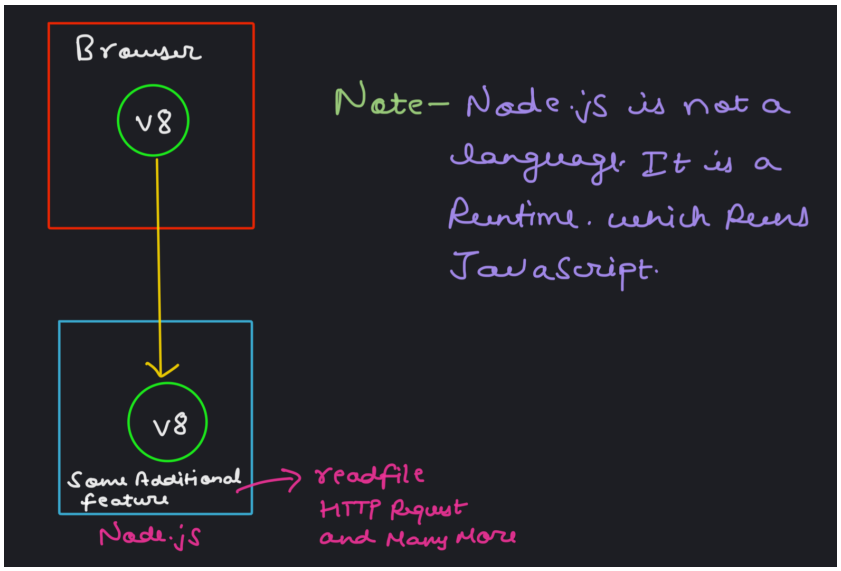
COMMON JS BROWSER ENGINES

1.V8 - used by google chrome to convert the javascript code into zeroes and ones and whoever written this code had to made sure that they are updated by new ECMASCRIPT standard.

2.SPIDERMONKEY - used by firefox - written in c and rust . They also need to make sure that they update there code as per ECMASCRIPT.

Q. What is Node.JS ?

Node.js is an open-source, cross-platform JavaScript runtime environment that executes JavaScript code outside of a web browser. It allows developers to use JavaScript for server-side programming, enabling them to build scalable and efficient network applications. Node.js is built on the V8 JavaScript runtime engine, the same engine that powers the Google Chrome browser.



Some smart people took a v8 engine.added some backend things(filesystem reads) on top to create a new runtime to compete with backend language like java.

Js was never meant to be run in the backend eventually become popular choice of runtime on the backend.

Q. What is BUN ?

Other than the fact that javascript is a single threaded. Node.js is slow (multiple reason for it) some smart people said they wanted to re- write the JS runtime for the backend and introduced bun.it is significantly provides the faster runtime.it is written in zig.

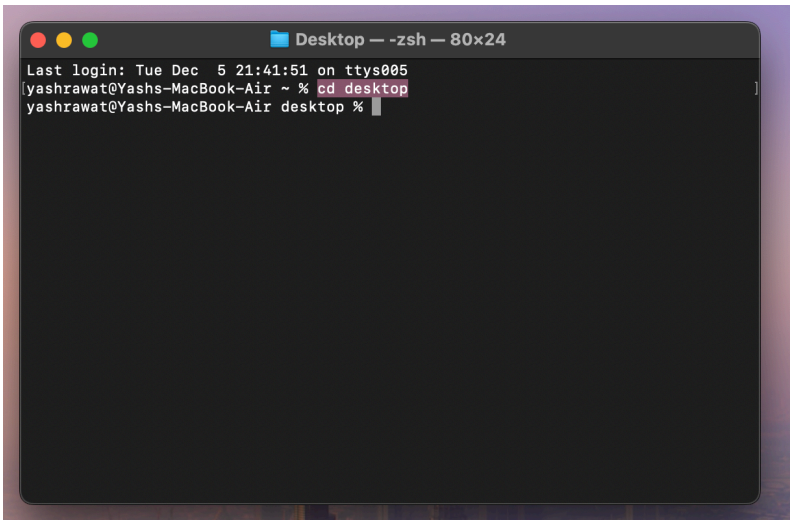
Q. What can you do with Node.JS?

1. Create CLIS

CLI - command line interface

A CLI allows users to interact with a software application or system by entering text commands in a terminal or command prompt.

For example - cd (change directory)



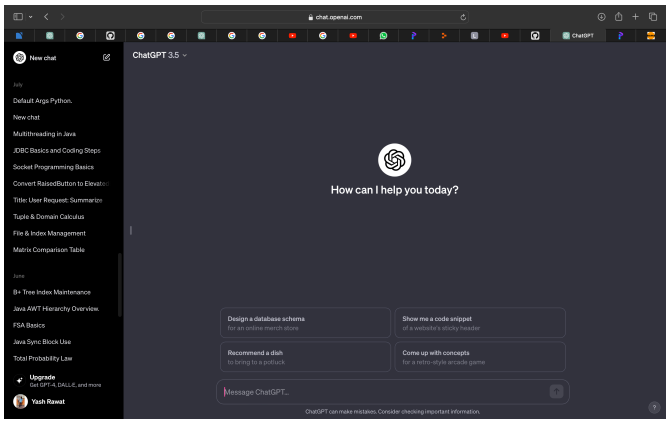
2. Create a video player
3.create a game
4.create an HTTP server.

Mostly the NODE.JS is used for creating the HTTP server.

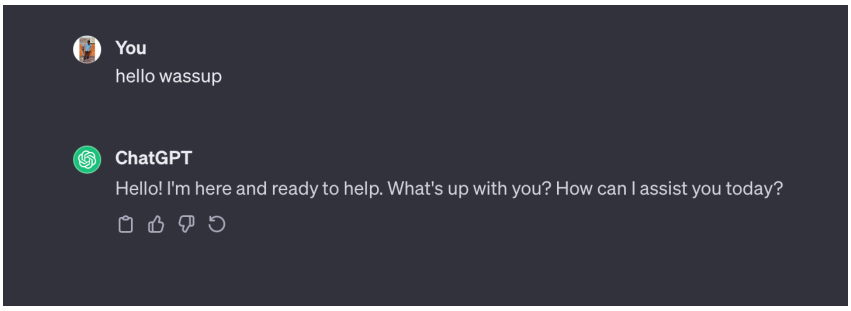
Q. What is HTTP server?

HTTP - hypertext transfer protocol

1. **A protocol that is defined for machines to communicate .**
2. **Specifically for websites, it is the most common way for your websites frontend to talk to its backend .**



FRONTEND



When I write some input to the chat gpt. It generates response. What do you think this response is coming from ?

You might have heard that openai has close source model.which means they never reveal how the response is being generated .this machine learning model will never run on my browser. So the conversion from input to output is done on the openai server.that is what we called as a BACKEND.

So to write backend we need NODE.JS.
And to write frontend we need HTML,CSS,JS.

Q. What is HTTP Server?

Some code that follows the HTTP protocol and is able to communicate with clients(browser, mobile app)

Think of it as similar to the call app in your phone which lets you communicate with your friends .

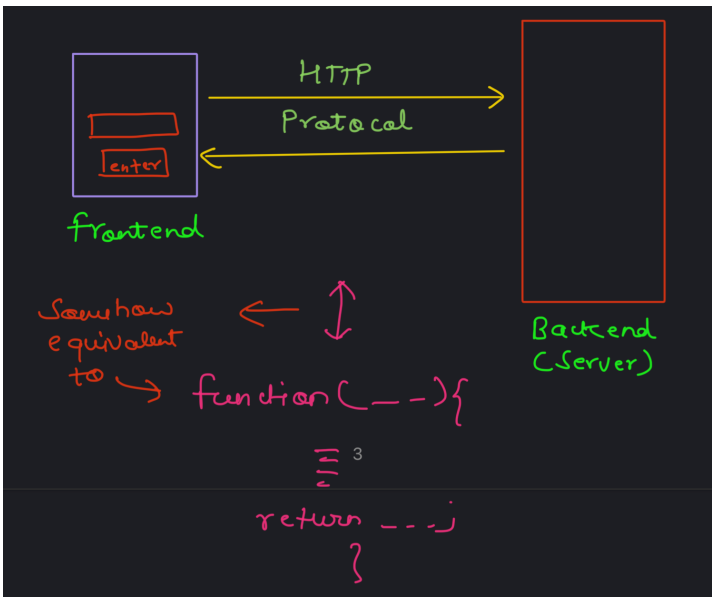
Q. Difference between HTTP protocol and HTTP server ?

The HTTP protocol defines the rules for communication on the web, specifying how messages are formatted and transmitted. An HTTP server is software that adheres(means that the server follows and complies with the rules and specifications defined by the HTTP protocol. It suggests that the server implements the protocol correctly, ensuring that it processes requests and delivers responses in accordance with the standards set by the HTTP protocol.) to these rules, implementing the protocol to handle requests and provide the requested content or perform other actions. The protocol and the server work together to facilitate communication between clients (e.g., web browsers) and servers on the internet.

HTTP Protocol

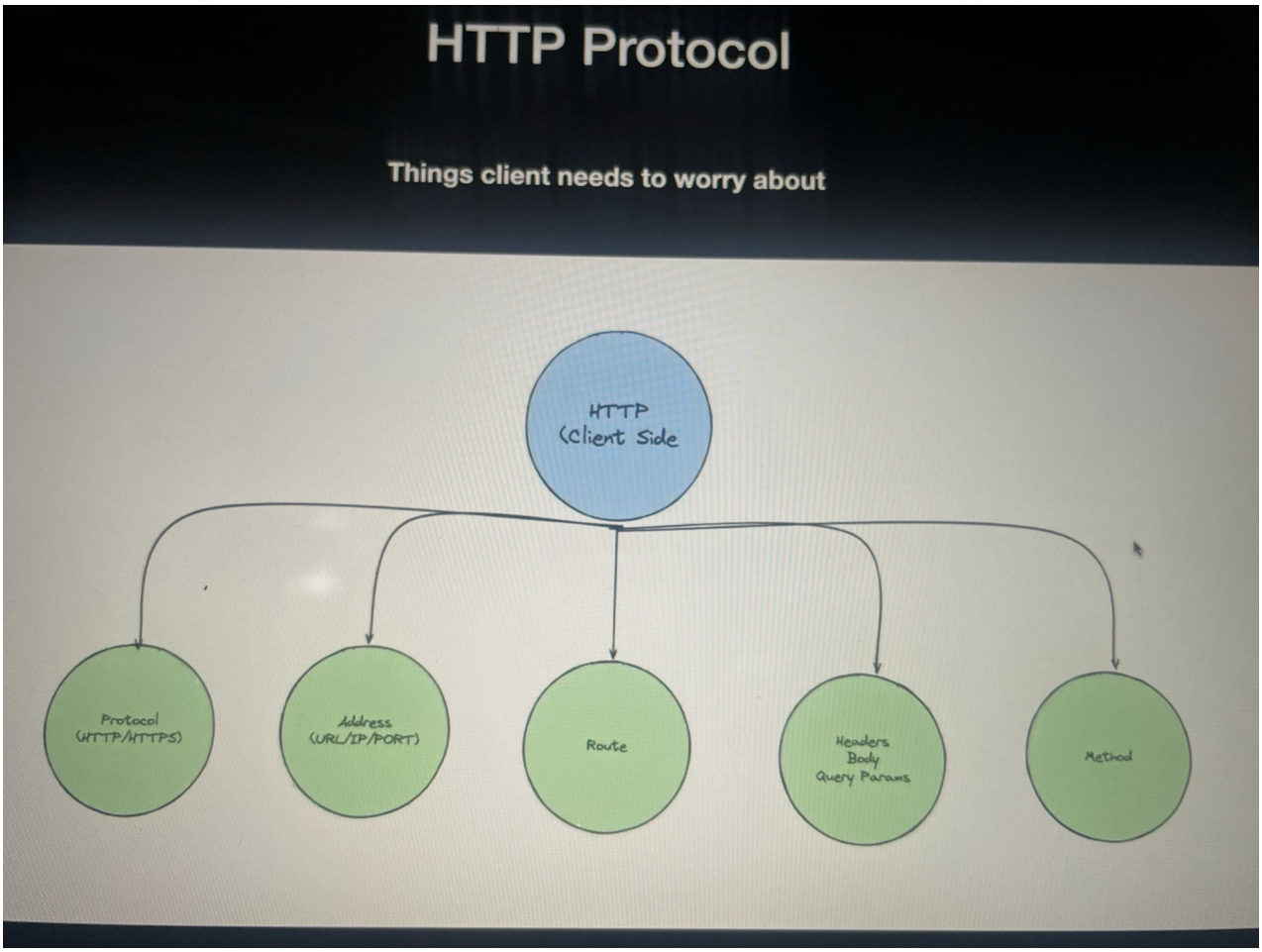
In the end it is the client throwing some information at a server .server doing something with that information . Server responding back with the final result .

Think of it as a functions where arguments are something the client sends rather than calling a function using its name , the client uses a url . Rather than the function body the server does something with the request . Rather than the function returning a value ,the server responds with some data .



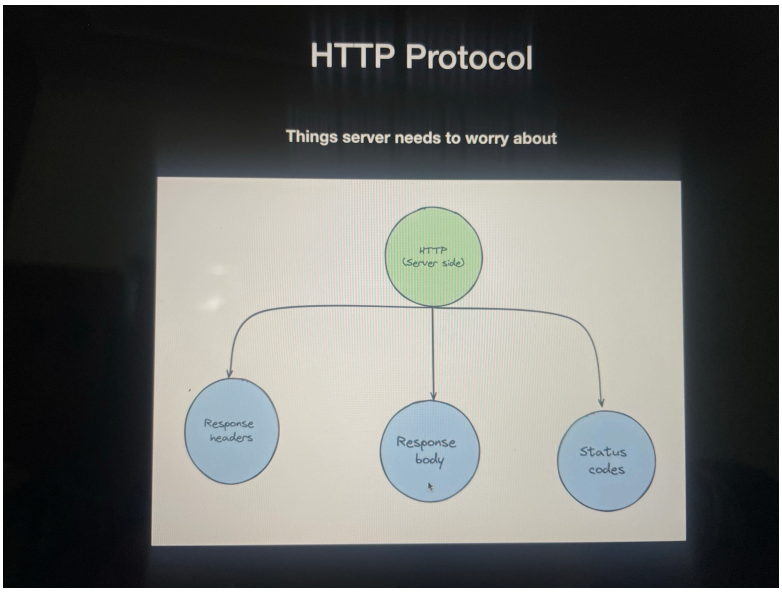
Now lets deep dive into http protocol .so lets find out what are things you need to know when you are sending a request and what are the things you need to know when you are receiving a request .and what are things that the server needs to know when it is responding back .

HTTP PROTOCOL (client side)

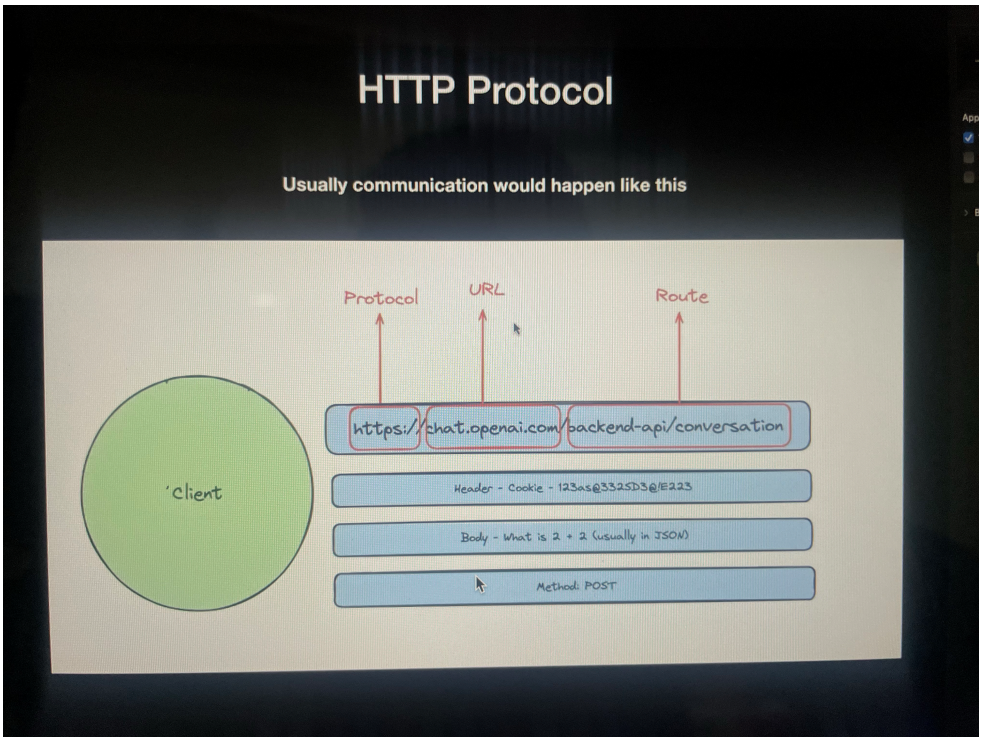


It needs to specify this specification in order to send http request to the backend or server .just like the function needs an argument when you call the function .

HTTP PROTOCOL (server side)



HTTP PROTOCOL (client side)



Routes - routes in a chatbot like ChatGPT refer to different paths or functionalities based on user inputs. These routes include greetings, informational responses, task-specific actions, small talk, handling errors, and saying goodbye. They help structure the bot's behavior for diverse user interactions. For example in above image we are having a conversation route.

Headers(cookies) - Headers play a crucial role in controlling the behavior of the HTTP communication and conveying metadata between the client and server. They help in negotiating content types, handling authentication, caching, and more. The structure of headers is key to the functioning of the HTTP protocol.

Body - Think of the body in HTTP like the payload of a message. When you send a request to a website, the body might contain data like the information you entered in a form. When the website responds, the body could hold the actual content you see on the page. It's where the real data lives in the communication between your browser and the web server.

Methods -

GET:

- Purpose: Used to request data from a specified resource.
- Example: Retrieving the content of a web page.

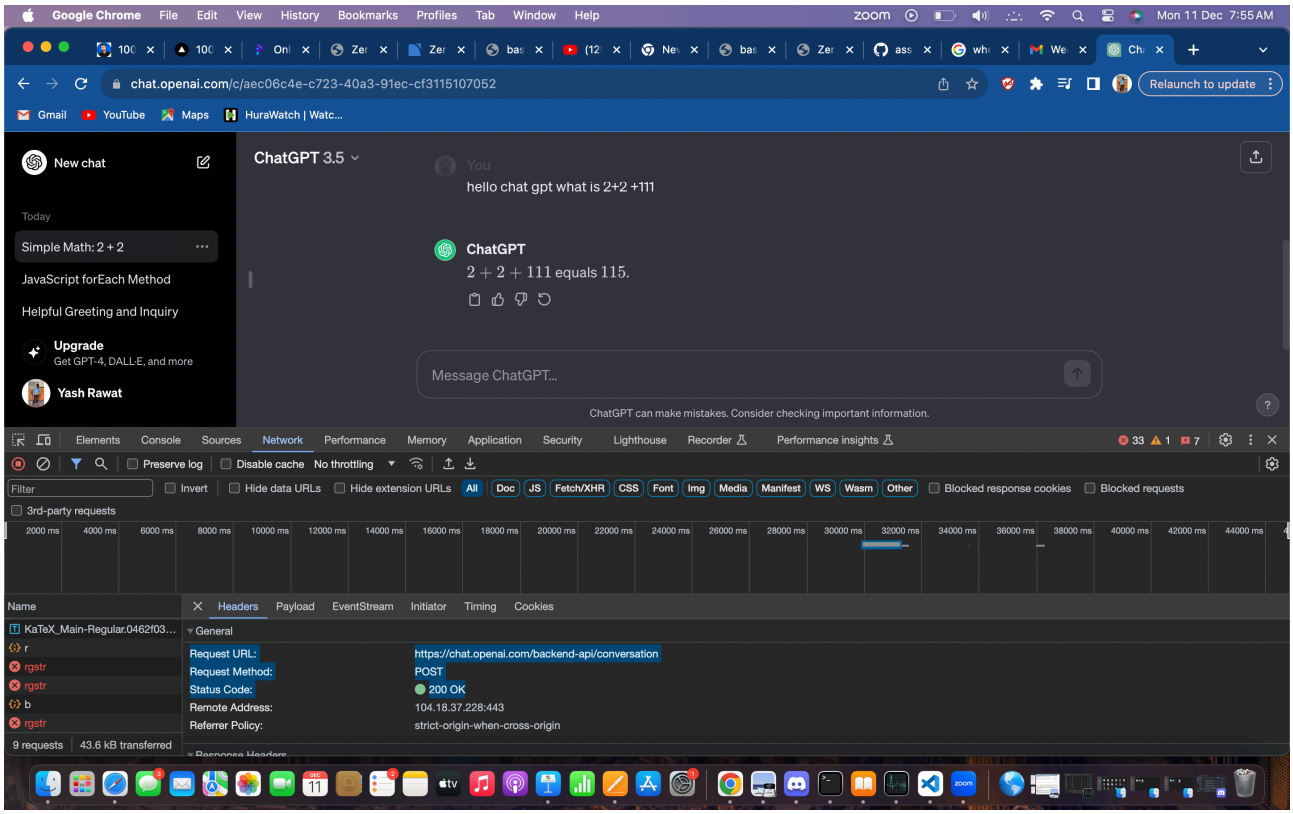
- **POST:**
- Purpose: Used to submit data to be processed to a specified resource.
- Example: Submitting a form with user input.

- **PUT:**
  - Purpose: Used to update a resource or create a new resource if it doesn't exist.
  - Example: Updating the content of a blog post.

- **DELETE:**
  - Purpose: Used to request the removal of a resource.

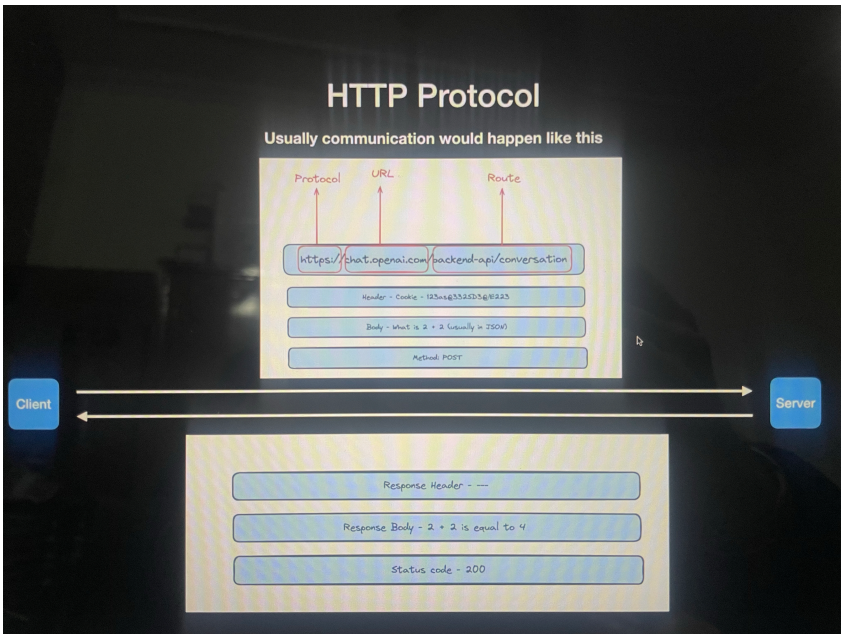
- Example: Deleting a user account.

For example -



And if you go to Network — request Headers — you will see the cookies(authorisation) it needs to be confidential .so don't share it anyone else .

Client - server communication



Q. What will happen in the browser when you fire a request ?

When you fire a request to Google (e.g., by entering a search query in your browser's address bar), several actions take place behind the scenes. Here's a simplified overview of the process:

1.It parses the URL(uniform resource locator). for example - Domain name inside the URL - google.com,chat.openai.com

2. DNS RESOLUTION - our browser initiates a Domain Name System (DNS) resolution to convert the human-readable domain name "google.com" into an IP address. This IP address is needed to locate the server on the internet.

3. Once the IP address is obtained, your browser establishes a Transmission Control Protocol (TCP) connection to Google's server.

4. if you're accessing Google over HTTPS (secure HTTP), a TLS (Transport Layer Security) handshake takes place to establish a secure encrypted connection.

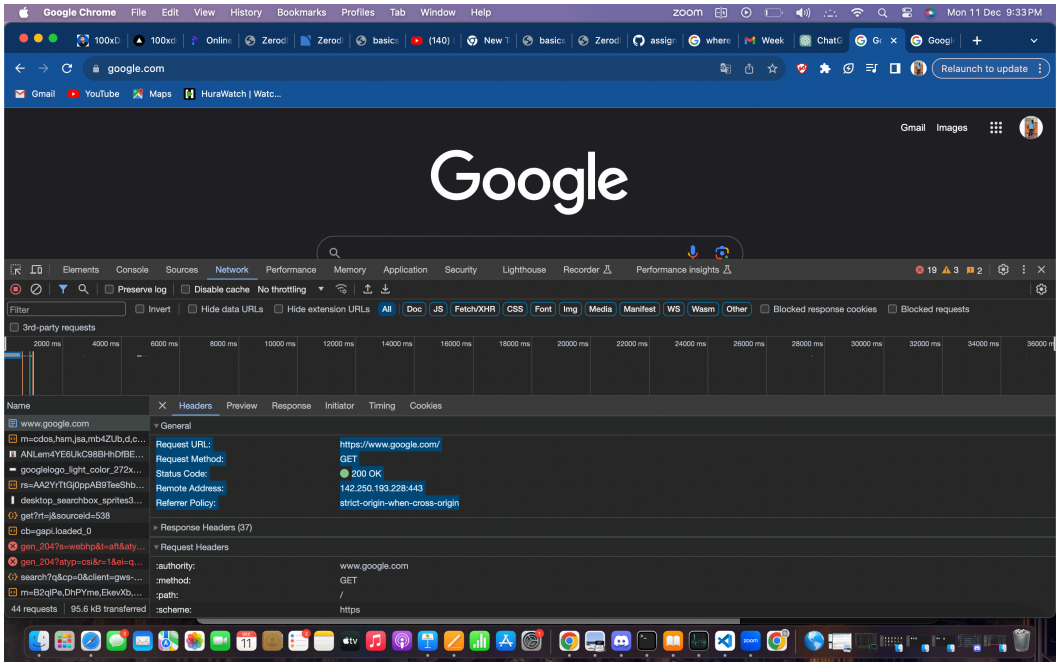
5. HTTP request - Your browser asks Google for information about the thing you entered in the search bar and entered it.

6. Google's server processes the request, which may involve querying databases, executing algorithms to determine relevant search results, and more.

7. Google's server sends back an HTTP response to your browser. This response includes headers (e.g., Content-Type specifying the type of data) and the actual content (e.g., HTML for displaying the search results).

8. Your browser receives the response and begins rendering the page. It interprets the HTML, executes JavaScript, and fetches additional resources (e.g., images, stylesheets) as needed.

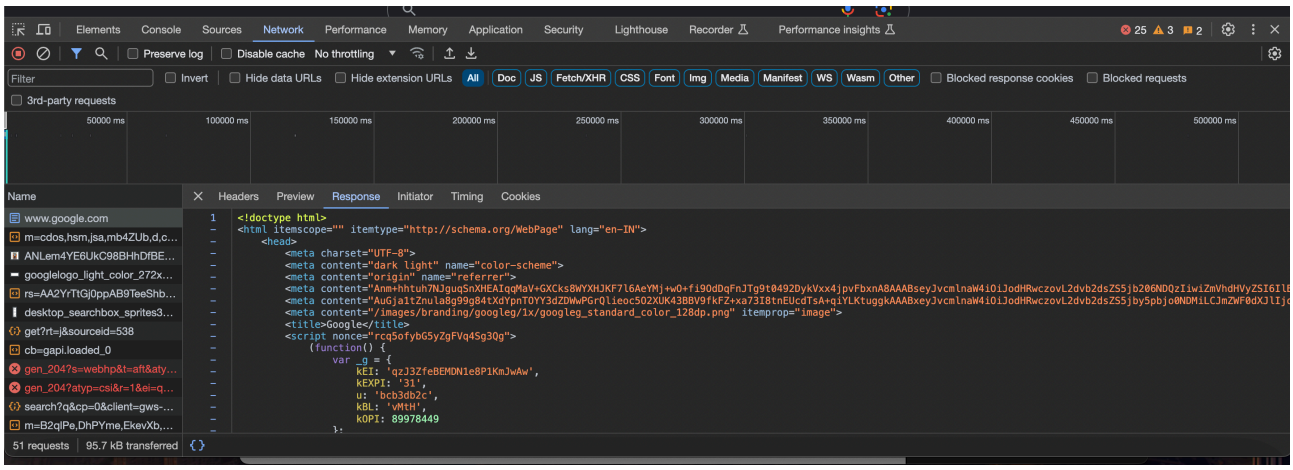
9. finally, your browser displays the Google search results page on your screen.



Now if you go to google.com and click on inspect and then go to network . And then to www.google.com you will see -

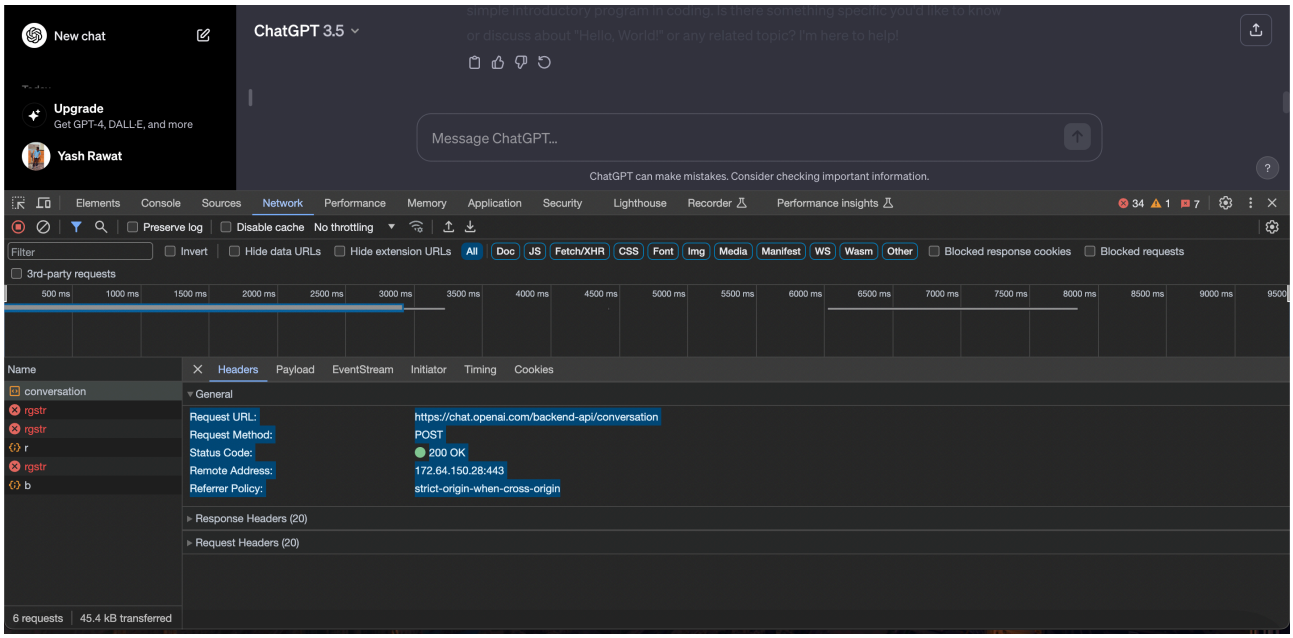
That the GET request was made to https://www.google.com/ , the server successfully processed the request (status code - 200 means OK) ,and the servers IP address is 142.250.193.228 with port 443. The referrer policy is set to ‘strict-origin-when-cross-origin’,indicating a security measure for handling the “referer” header.

And if you click on response then -
You will see the server is responded with html,jss,css file.



HTTP Protocol (Server side)

1.You get the inputs(route, body, headers)
2.You do some logic on the input ,calculate the output.
3.You return the output body, headers and status code.



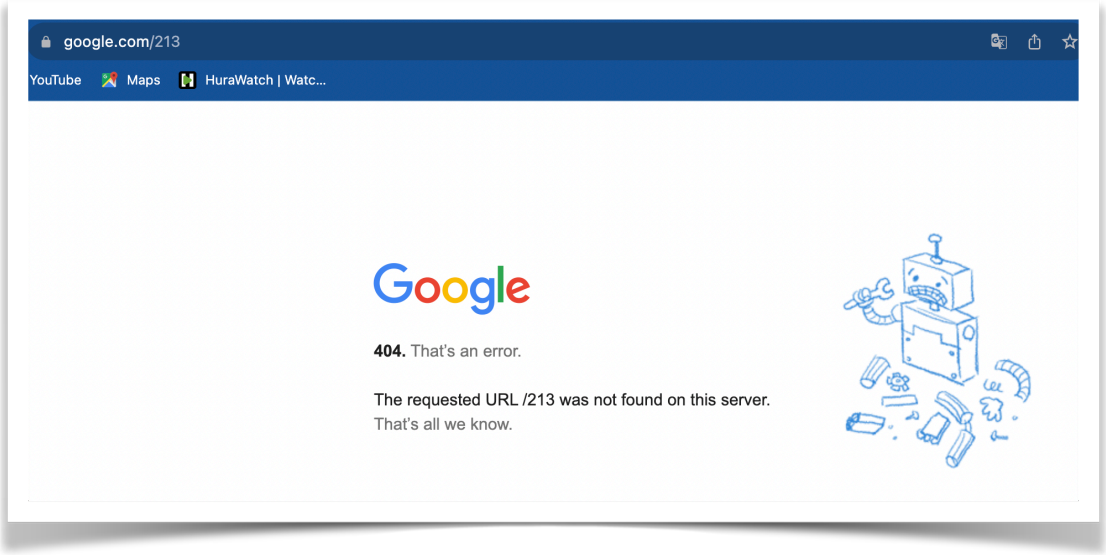
This log indicates that a POST request was made to https://chat.openai.com/backend-api/conversation, the server successfully processed the request (status code 200 OK), and the server's IP address is 172.64.150.28 with port 443. The referrer policy is set to strict-origin-when-cross-origin, indicating a security measure for handling the Referer header during cross-origin requests.

Different request of the HTTP

- **GET Method:**
  - Purpose: Retrieve data from a specified resource.
  - Example: Asking ChatGPT to provide information about programming languages.for example previous conversation you asked to chat GPT.
  - HTTP Request: GET /info?topic=programming_languages
- **POST Method:**
  - Purpose: Submit data to be processed to a specified resource.
  - Example: Asking ChatGPT to generate a new response based on user input.
  - HTTP Request: POST /generate-response with a payload containing the user input.
- **PUT Method:**
  - Purpose: Update a resource or create a new resource if it doesn't exist.
  - Example: Updating ChatGPT's training data with new information.
  - HTTP Request: PUT /training-data with the updated data.
- **DELETE Method:**
  - Purpose: Request the removal of a resource.
  - Example: Deleting a specific conversation thread in ChatGPT.
  - HTTP Request: DELETE /conversation/123 to delete the conversation with ID 123.

Status code Section -

200 - Everything is OK
404 -Page/route is not found. This happens when the url is not correct .



403 - Authentication issues. It happens when for example you are trying to access chat gpt 4 but you don't have access to it .

500 - internal server error

Now we will create the HTTP server

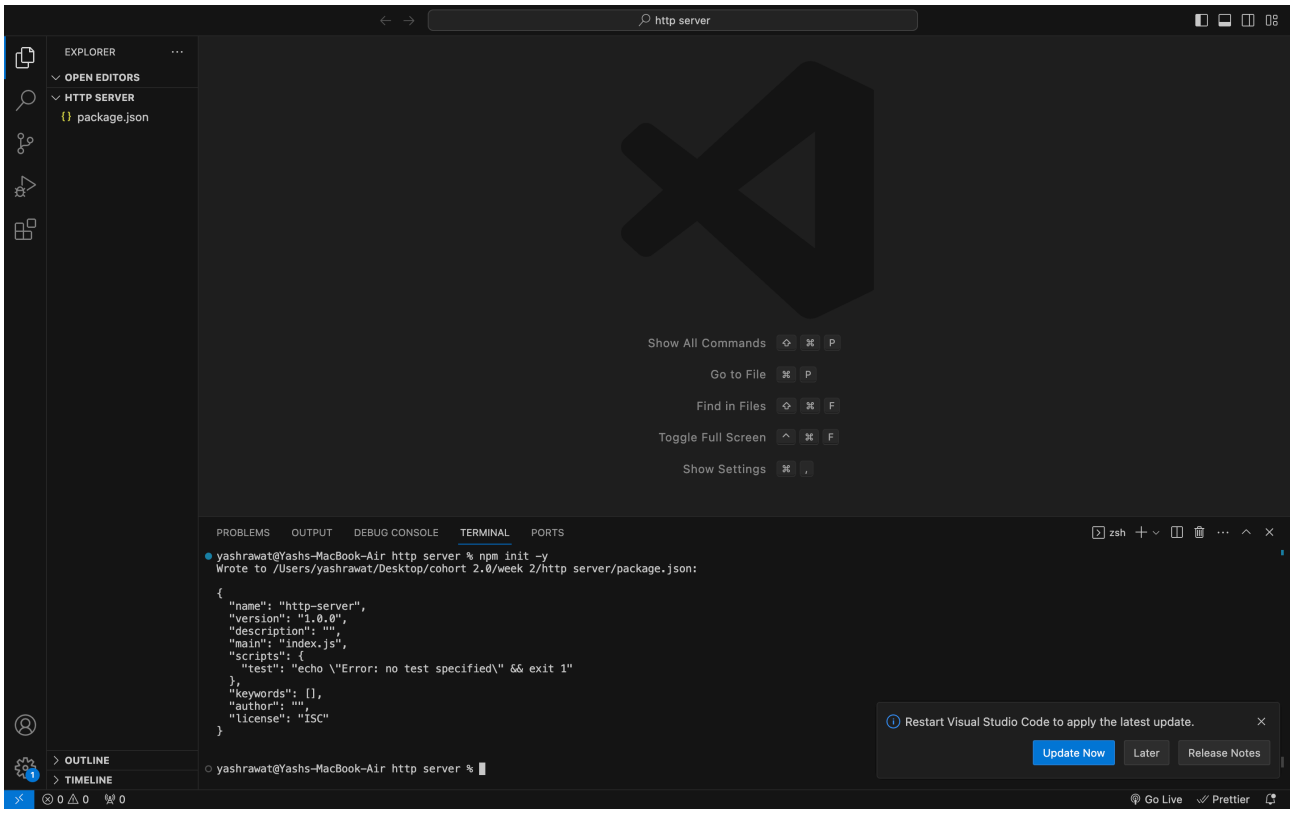
Q. How do you write an HTTP server?

We can create in any language (C++,node.js,springboot) the HTTP server.

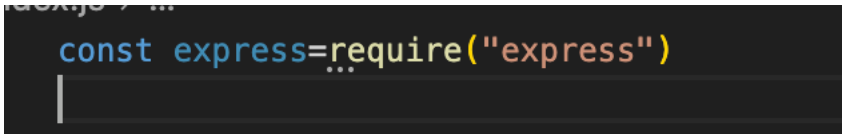
We will create the HTTP server using node.js and by using the external library know as express.js

1. Go to VS code .create a new folder and then open it .

2. Now open the terminal and in the directory of the folder you created write a command npm init -y . And then you will see the package.json file will be created in the folder.

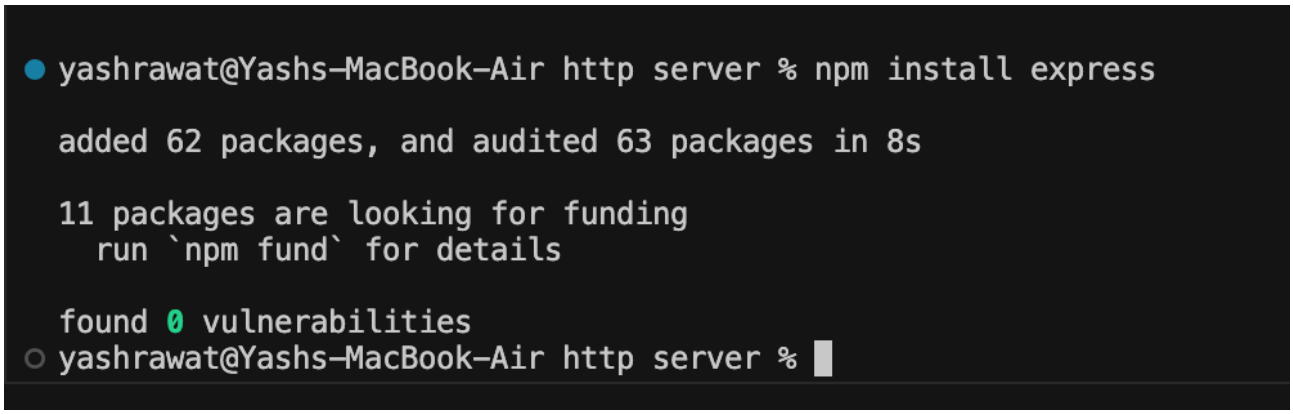


3. Then create another file in the folder name index.js
4. And then import the express lib in it.



5. Unlike fs is available locally in your node.js package . The express you have to import from the browser because it is not present locally in the package got node js .

The command you have to use is - npm install express



Q. What is Port?

In networking, a port is a communication endpoint that is used to identify a specific process to which data can be sent . Ports allow multiple applications or services to run on the same device and use network resources without interfering with each other.

Server Side Port number -

- Identification of Services:
    - Imagine a server as a big building with different rooms. Each room (service or application) has a unique number, which is the port number.
- Incoming Requests:
    - When a request (data) arrives at the server, it's like someone knocking on the building's door.
- Checking the Room Number (Port):
    - The server checks the port number mentioned in the incoming request to figure out which room (service or application) should handle the request.
- Routing to the Right Service:
    - The server routes the incoming data to the specific service or application associated with the mentioned port.
- Example:
    - If the request is for a website, it might be directed to the room (service) handling web traffic, which is often associated with port 80 (HTTP) or 443 (HTTPS).

In summary, port numbers on the server side act as identifiers for different services or applications. They ensure that incoming data reaches the correct destination within the server, allowing multiple services to operate independently on the same server.

Client side Port -

On the client side, the concept of ports is still relevant, but it's often handled automatically by the operating system and applications. Here's a simplified explanation:

- Dynamic Port Assignment:
    - When your computer initiates a connection to a server, the operating system assigns a dynamic or "ephemeral" port to the client-side of the connection.
- Random Port Number:
    - This dynamically assigned port is usually a random number chosen by the operating system from a range of available ports.
- Temporary Door on the Client:
    - Think of this client-side port as a temporary door that your computer uses for communication with the server. It ensures that the response from the server comes back to the right place.
- Managing Multiple Connections:
    - Because a computer can have multiple applications connecting to various servers simultaneously, using dynamic ports helps manage these connections without conflicts.
- Example:
    - Let's say your web browser (client) is accessing a website (server) on the internet. Your computer might use a dynamic port number (e.g., 50000) for this specific connection.

In summary, on the client side, dynamic or ephemeral ports are used to facilitate communication between your computer and servers. These temporary ports ensure that multiple connections can be managed simultaneously without interference. The exact port numbers are typically managed by the operating system and the networking stack of the client machine.

Q. Are client side port number are same or differ in number with server side port number?

Yes they differ in the port number , the server-side port is a well-known port associated with a specific service, while the client-side port is dynamically assigned by the client's operating system to manage multiple connections. They are generally different port numbers and play distinct roles in facilitating communication between the client and server.

Example:

- When a client, such as a web browser, connects to a web server, the client-side port might be something like 50000, and it is dynamically chosen by the client's operating system. Meanwhile, the server-side port would be the well-known port associated with the HTTP service, which is typically port 80.

Basic HTTP server using express on port 3000

```
// import the express library
const express=require("express")
//define the port number
const port=3000;
// create an instance of the express application
const app=express()
//define the route for handling GET requests to the root path ("/")
app.get('/',function(req,res){
    // send the response hello world when this route is accessed
    res.send("hello world ")
})
// Start the express applicaiton ,and make it listen on the specified port (3000)
app.listen(port)
```

Q. Difference between public and private IP addresses .

- Public Address:
  - Definition: A public IP address is assigned to a device (such as a router or server) that is directly connected to the internet. It is globally unique and routable across the internet.
  - Usage: Public IP addresses are used to identify devices on the public internet. Websites, servers, and other services accessible from the internet typically have public IP addresses.
- Private Address:
  - Definition: A private IP address is assigned to devices within a private network (e.g., home or office network) and is not directly accessible from the internet. It is designed for use within a specific, confined network environment.
  - Usage: Devices like computers, printers, or smartphones within a local network often have private IP addresses. These addresses are not reachable directly from the internet; instead, they are accessed through a router that performs Network Address Translation (NAT).

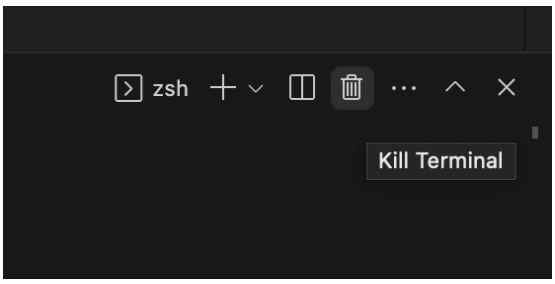
- Example:
  - A home network might have a router with a public IP address facing the internet and private IP addresses assigned to devices within the home (e.g., 192.168.1.x).

- Network Address Translation (NAT):
  - Role: NAT is a crucial mechanism that allows devices with private addresses to share a single public IP address when accessing the internet.

  - How it works: When a device with a private address sends a request to the internet, the router translates its private address to the router's public address. The response from the internet is then sent back to the router, which uses NAT to route it to the correct private device.

In essence, public addresses are used for communication over the global internet, while private addresses are for internal communication within a local network. NAT acts as a bridge between private and public addresses, allowing devices in a private network to access resources on the internet using a single public address.

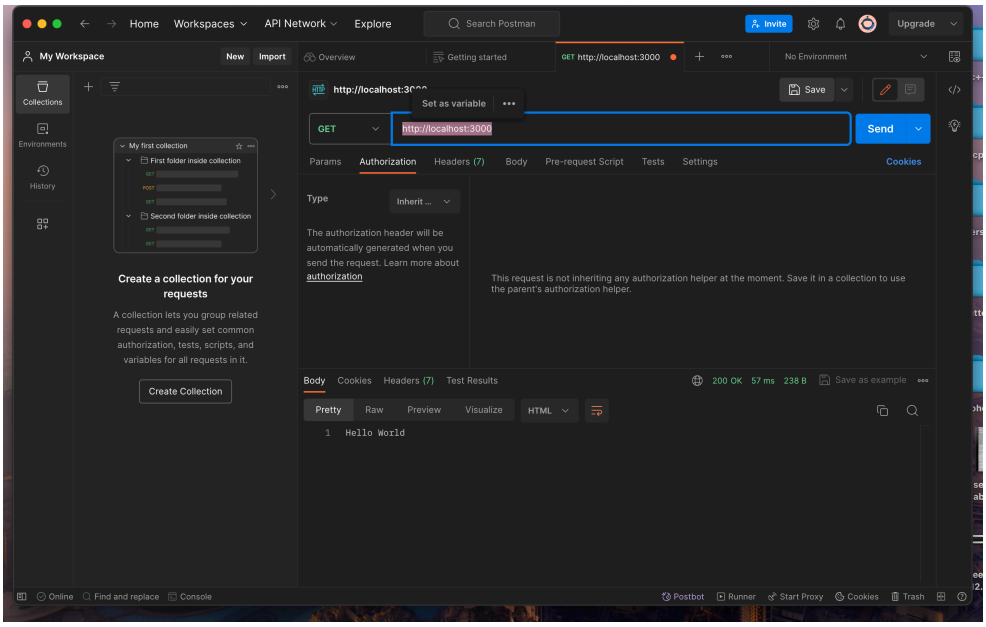
NOTE - when you change the code of express. You first need to exit the terminal



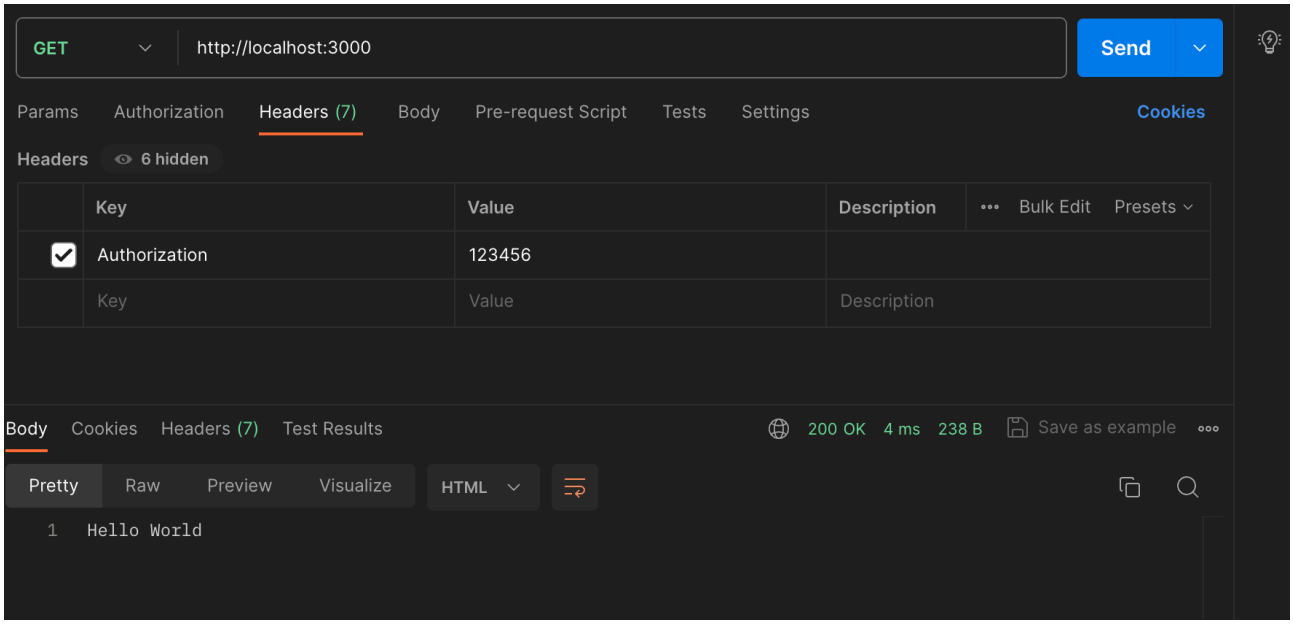
And then save the code and then open new terminal and write the same command as node index.js .

Postman

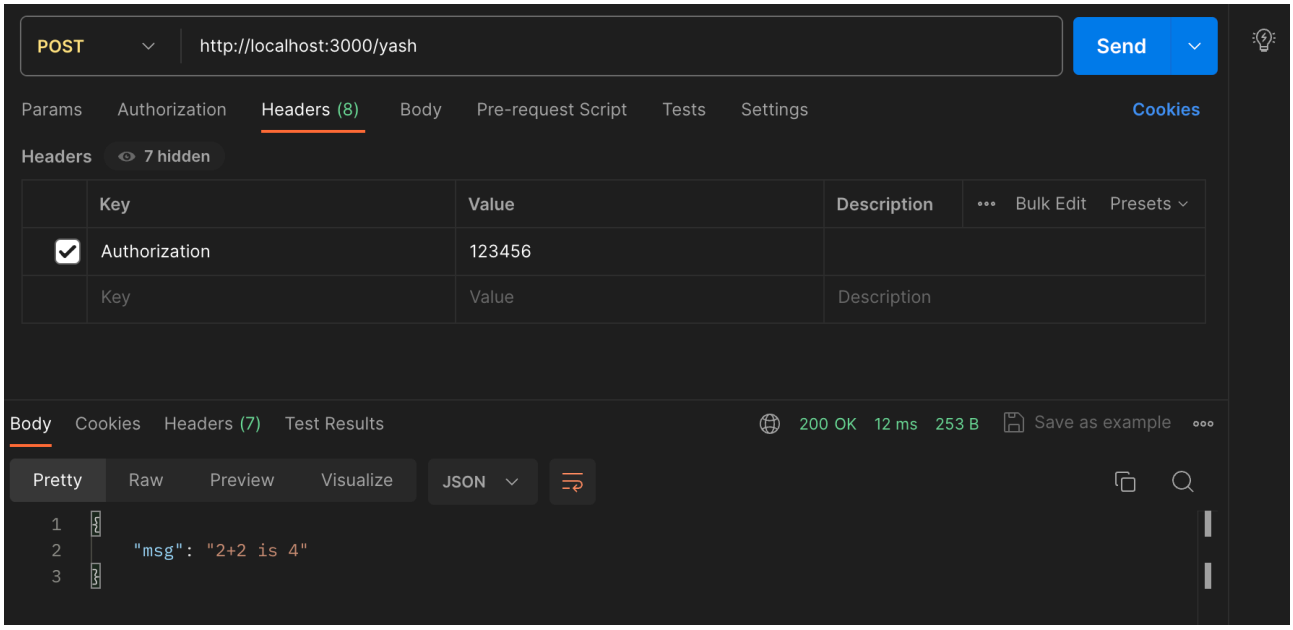
Postman acts as a client application that allows you to send various HTTP requests (such as GET, POST, PUT, DELETE, etc.) to test and interact with a backend API that you've created.



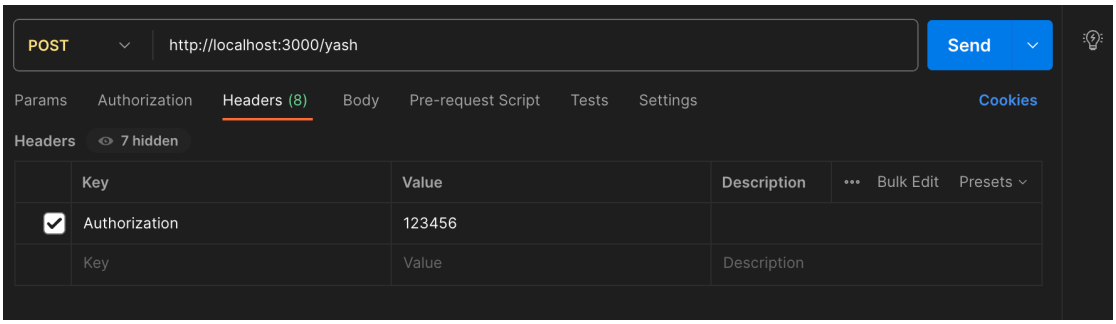
GET request in Postman



Post request in Postman



Now I will add some authentication inn my header section in my postman



And then you click on Send in the Postman .(your postman will send headers to the backend)

And then in your code you can log it .

```
const express = require("express");
const port = 3000;
const app = express();

// Define a route for handling GET requests to "/route-handler"
app.post("/yash",function(req,res){
    console.log(req.headers)
    res.send({
        msg:"2+2 is 4"
    })
})

// Define a route for handling GET requests to the root path "/"
app.get('/', function (req, res) {
    res.send("Hello World");
});

// Start the Express application, making it listen on the specified port (3000)
app.listen(port, () => {
    console.log(`Server is running on http://localhost:${port}`);
});
```

```
yashrawat@Yashs-MacBook-Air http server % node index.js
Server is running on http://localhost:3000
{
  authorization: '123456',
  'user-agent': 'PostmanRuntime/7.35.0',
  accept: '*/*',
  'postman-token': '86a17fdd-f6aa-4458-a6cf-b5a5ef03e0c1',
  host: 'localhost:3000',
  'accept-encoding': 'gzip, deflate, br',
  connection: 'keep-alive',
  'content-length': '0'
}
```

Note - There is multiple domain name that can point to one server(same IP address).and then its server duty to figure out from which domain the request is coming and establish connection to it .

NOTE - express ignores the body. So to read body you have to install another library name - body parser

```
yashrawat@Yashs-MacBook-Air http server % npm install body-parser

added 2 packages, changed 2 packages, and audited 65 packages in 582ms

11 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
yashrawat@Yashs-MacBook-Air http server %
```

```
const bodyParser=require("body-parser")
```

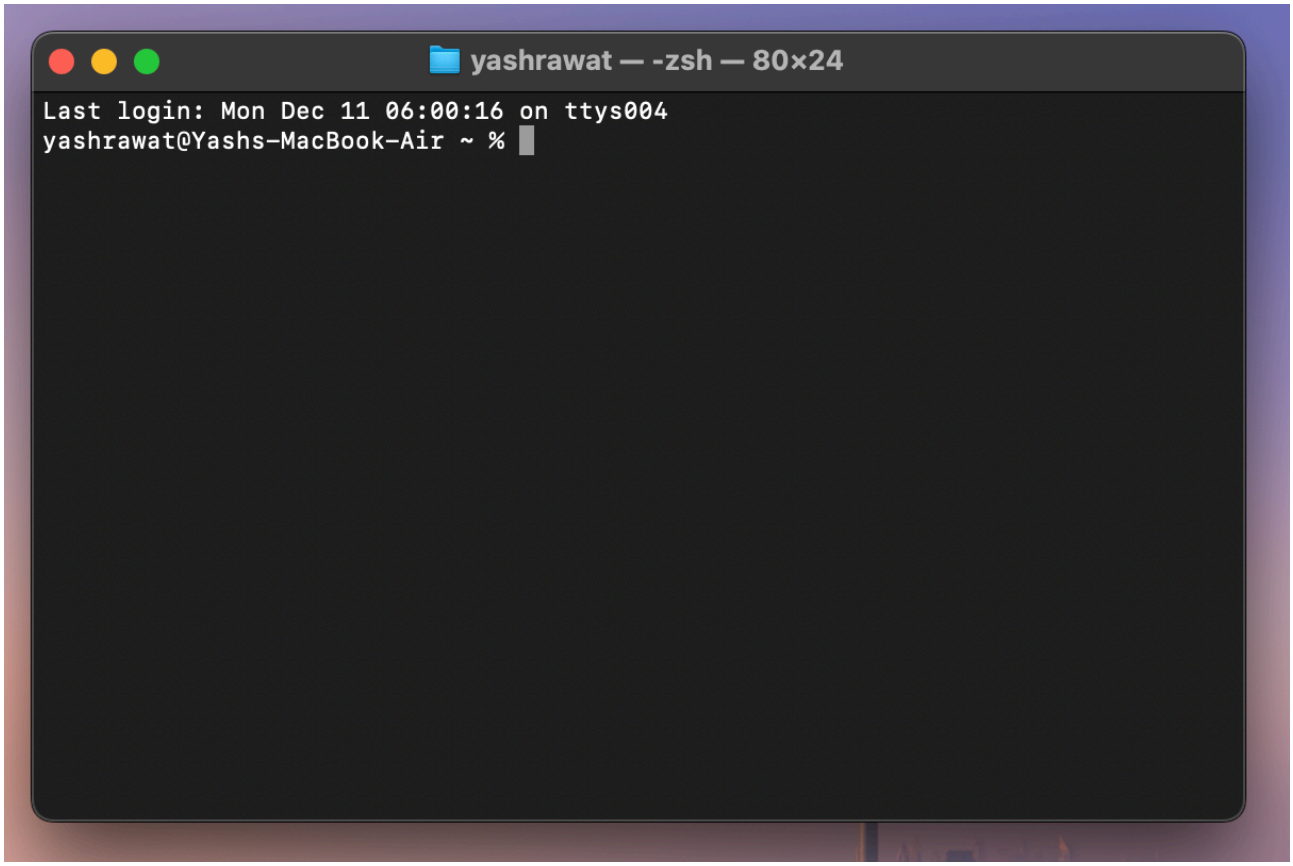
```
app.use(bodyParser.json())
```

NOTE - we have to restart the terminal again and again its an hectic task.so to avoid it we can write a command - rpm install nodemon

```
yashrawat@Yashs-MacBook-Air http server % npx nodemon index.js
Need to install the following packages:
  nodemon@3.0.2
Ok to proceed? (y) y
[nodemon] 3.0.2
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting `node index.js`
Server is running on http://localhost:3000
rs
[nodemon] starting `node index.js`
Server is running on http://localhost:3000
```

BASH AND TERMINAL

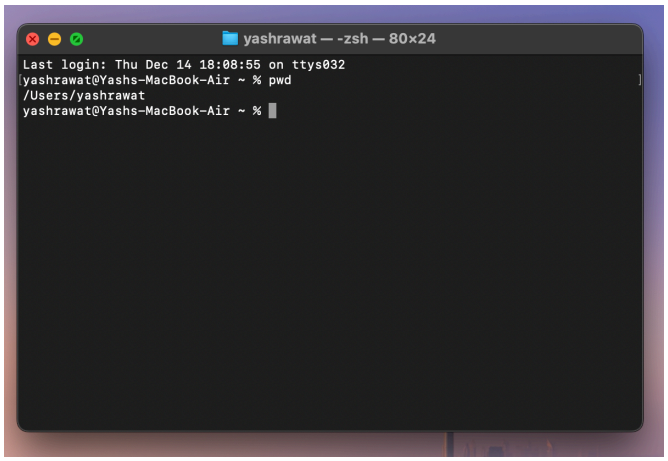
Bash - Its let you interact with your operating system .
Open the terminal.



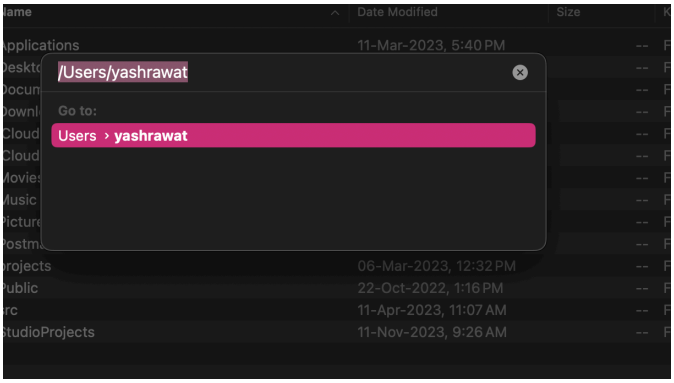
Terminal is nothing but another interface to do things on your machine (Just like the GUI windows,macOS) .

Pwd command

- What it Does: pwd stands for "print working directory."
- What it Shows: When you run pwd in a terminal, it shows you the full path of the directory you are currently in.
- Why it's Useful: It helps you know where you are in your computer's folders. Imagine it like a map telling you the exact location (directory) you're exploring.
- Example: If you see /home/user/documents after running pwd, it means you are currently inside the "documents" folder, which is inside the "user" folder, which, in turn, is inside the "home" folder.



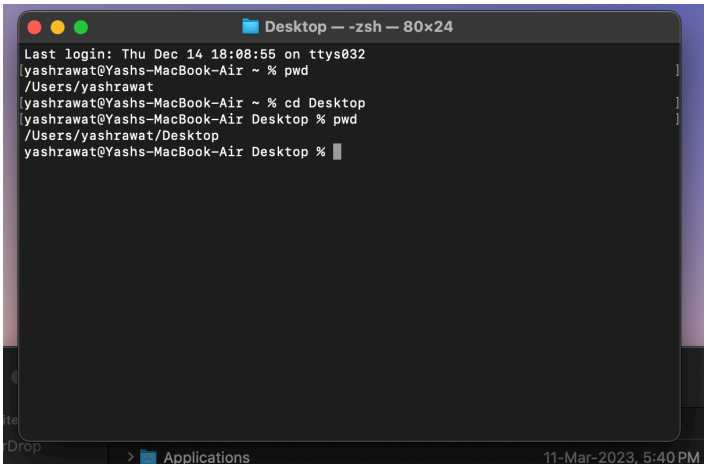
NOTE - to enter a file path into finder and search for it you can use .
Command + shift + G

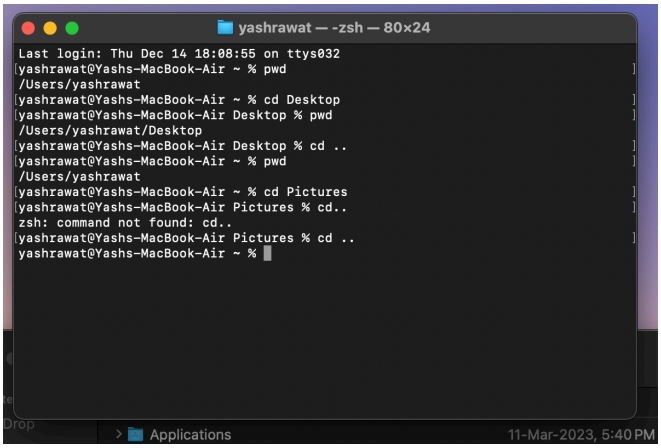


Cd command

cd is a fundamental command for navigating the file system in a command-line environment. It allows you to move around, explore directories, and specify the location for your subsequent commands.

The cd command is used to change the current working directory in a command-line interface (CLI) or terminal.



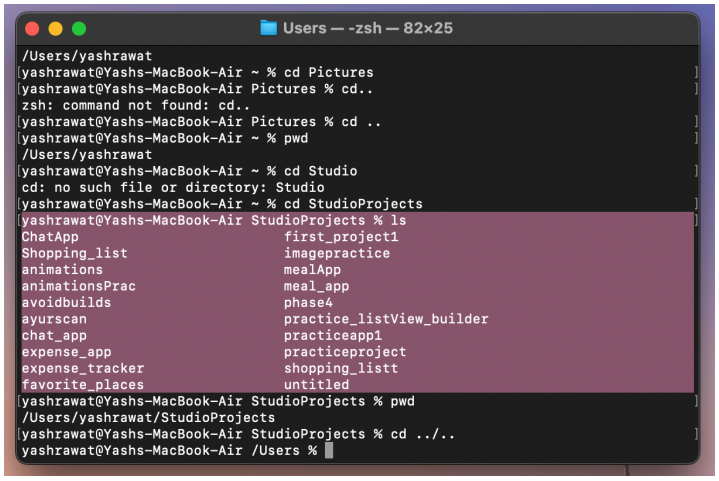


You can use **cd ..** to move back one folder .

Ls command

The ls command is used to list the contents of a directory in a command-line interface (CLI) or terminal.

NOTE - to auto complete the directory name or file name — on a command line type one and more characters ,then press tab.



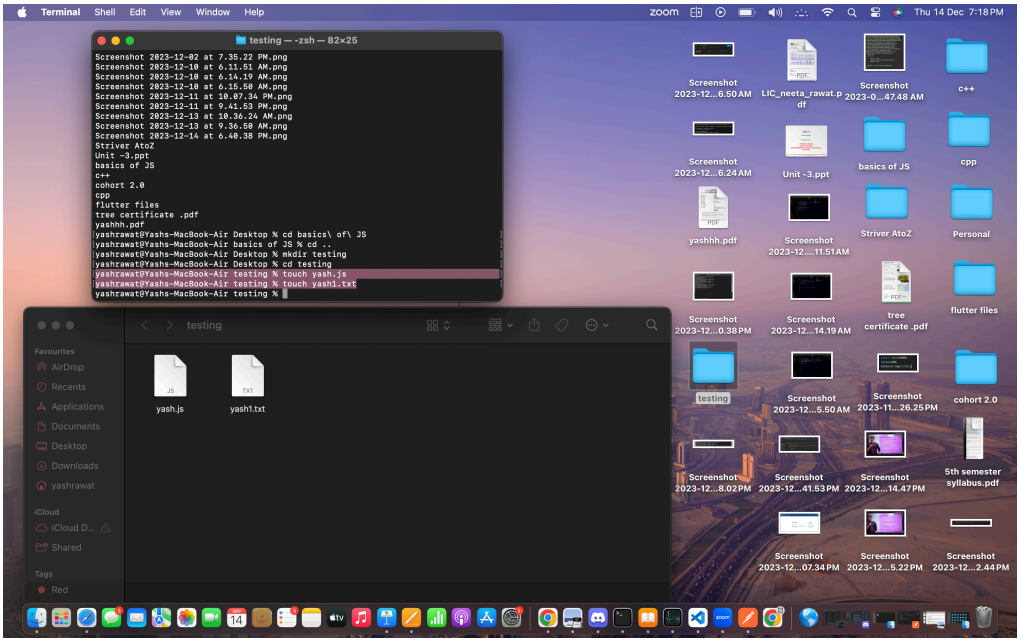
Mkdir command

The mkdir command is used to create a new directory (folder) in a command-line interface or terminal.

```
yashhh.pdf
yashrawat@Yashs-MacBook-Air Desktop % cd basics\ of\ JS
yashrawat@Yashs-MacBook-Air basics of JS % cd ..
yashrawat@Yashs-MacBook-Air Desktop % mkdir testing
yashrawat@Yashs-MacBook-Air Desktop %
```

Touch command

The touch command is used to create empty files or update the access and modification timestamps of existing files.



Cat Command

The cat command, short for "concatenate," is used in Unix-like operating systems to display the contents of files or concatenate files and display the result.

```
yashrawat@Yashs-MacBook-Air testing % cat yash1.txt
Hello Yash baby
yashrawat@Yashs-MacBook-Air testing %
```

Mv command

The mv command, short for "move," is used in Unix-like operating systems to move or rename files and directories

```
yashrawat@Yashs-MacBook-Air testing % cd testing2
yashrawat@Yashs-MacBook-Air testing2 % ls
yash.js
yashrawat@Yashs-MacBook-Air testing2 % mv yash.js /Users/yashrawat/Desktop
yashrawat@Yashs-MacBook-Air testing2 %
```

Cp command

The cp command, short for "copy," is used in Unix-like operating systems to copy files and directories from one location to another.

```
yashrawat@Yashs-MacBook-Air testing2 % pwd
/Users/yashrawat/Desktop/testing/testing2
yashrawat@Yashs-MacBook-Air testing2 % cp yashh.txt /Users/yashrawat/Desktop/testi
ng
yashrawat@Yashs-MacBook-Air testing2 %
```

Node Command

The node command is used to execute JavaScript code outside of a web browser. It is part of the Node.js runtime environment, allowing developers to run JavaScript on servers, build scripts, and perform various tasks on the command line.

It also helps you to run the javascript command in the terminal.

```
yashrawat@Yashs-MacBook-Air ~ % node
Welcome to Node.js v19.8.1.
Type ".help" for more information.
> let a=2;
undefined
> console.log(a)
2
undefined
>
(To exit, press Ctrl+C again or Ctrl+D or type .exit)
>
yashrawat@Yashs-MacBook-Air ~ %
```

```
yashrawat@Yashs-MacBook-Air basics of JS % node index.js
hello
{ name: 'rawat', age: 30 }
kanji
mosh
mary
4
hello yash
hello yash rawat
4
yashrawat@Yashs-MacBook-Air basics of JS %
```

Npm command

The npm command is the Node Package Manager command-line tool, and it is used to manage packages and dependencies in Node.js projects.

Website - npm.JS package .

```
yashrawat@Yashs-MacBook-Air Desktop % npm install express

added 62 packages in 1s

11 packages are looking for funding
  run `npm fund` for details
yashrawat@Yashs-MacBook-Air Desktop %
```

DEEP DIVING INTO BASH COMMAND

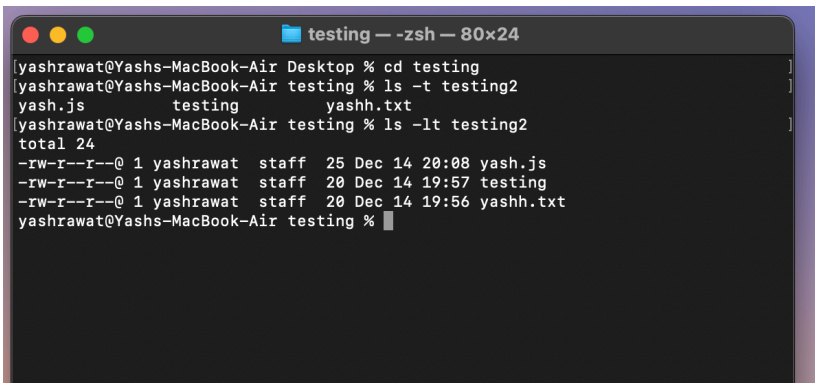
Ls -l The ls -l command is used in Unix-like operating systems, including Linux and macOS, to list detailed information about files and directories in a directory.

1.D - stands for it is a directory .
2.Rwxr-xr-x - file permission(read,write,and execute) to yashrawat(owner)
3.staff - it has only read and excuse permission .
4.160 - total size in bytes.
5.dec-14-19:57 - modification time .
6.testing2 - it is a directory name .

7- - this means it is a file.

```
testing — -zsh — 80×24
Last login: Thu Dec 14 18:25:23 on ttys032
yashrawat@Yashs-MacBook-Air ~ % pwd
/Users/yashrawat
yashrawat@Yashs-MacBook-Air ~ % cd Desktop
yashrawat@Yashs-MacBook-Air Desktop % cd testing
yashrawat@Yashs-MacBook-Air testing % ls
testing2        yash1.txt       yashh.txt
yashrawat@Yashs-MacBook-Air testing % ls -l
total 16
drwxr-xr-x  5 yashrawat  staff  160 Dec 14 19:57 testing2
-rw-r--r--@ 1 yashrawat  staff   17 Dec 14 19:22 yash1.txt
-rw-r--r--@ 1 yashrawat  staff   20 Dec 14 19:57 yashh.txt
yashrawat@Yashs-MacBook-Air testing %
```

Ls -t The **ls -t** command is used to list files and directories in a directory, sorted by modification time, with the newest files first.



Ls anyNameYouWannaSearch* -

For example - ls yash*

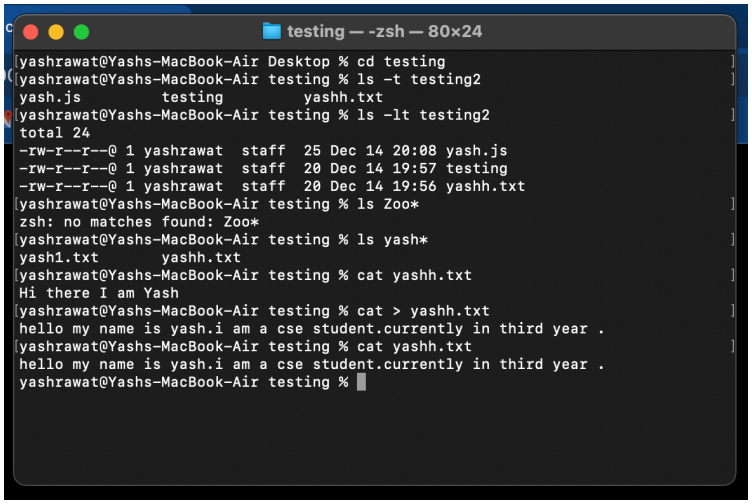
The **ls Yash*** command is used to list files and directories in the current directory that match the specified pattern "Yash*".

- In this example, the command lists files and directories that start with the prefix "yash" in the current directory. In this case, the files yash1.txt and yashh.txt match the pattern.
- Wildcard (*):
  - The * (asterisk) is a wildcard character that matches any sequence of characters. In this context, it matches any file or directory name that starts with “yash".
- Case-Sensitive:
  - The matching is case-sensitive, so it will only match filenames that exactly start with "yash".

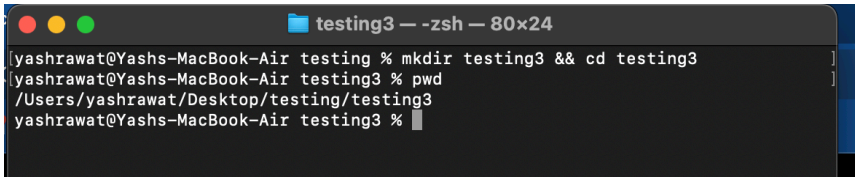
Editing the file

Now if you want to write a file you can use the command **cat > yashh.txt**
And then you can edit it (write what you want) and then you want to exit you can press **control + D**

Another way to append in the file is **cat >> yashh.txt**



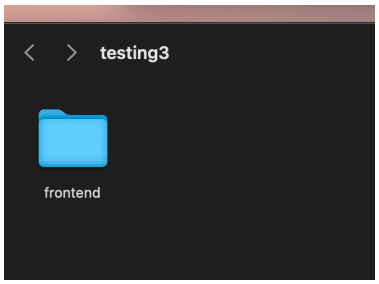
This command lets you create a new directory and then cd to move to the new directory you created -

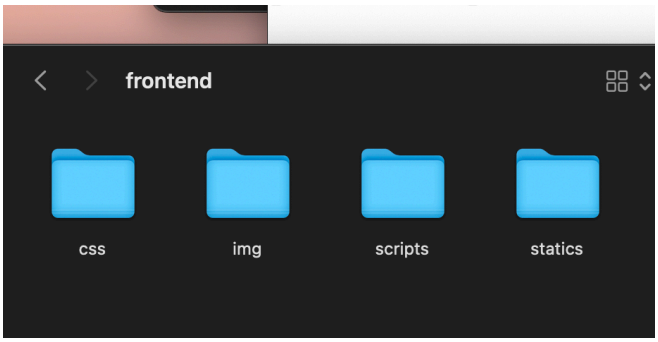


Suppose you are working for a project and you want to create a directory recursively .you can use command **mkdir -p frontend/css**

What it will do?
It create a Frontend directory.and then inside it .it will create directories of name Scripts , css,img,statics.

```
frontend — -zsh — 80×24
yashrawat@Yashs-MacBook-Air testing % mkdir testing3 && cd testing3
yashrawat@Yashs-MacBook-Air testing3 % pwd
/Users/yashrawat/Desktop/testing/testing3
yashrawat@Yashs-MacBook-Air testing3 % cd ../..
yashrawat@Yashs-MacBook-Air Desktop % cd testing
yashrawat@Yashs-MacBook-Air testing % ls
testing2        testing3        yash1.txt       yashh.txt
yashrawat@Yashs-MacBook-Air testing % cd testing3
yashrawat@Yashs-MacBook-Air testing3 % mkdir -p frontend/scripts
yashrawat@Yashs-MacBook-Air testing3 % mkdir -p frontend/css
yashrawat@Yashs-MacBook-Air testing3 % mkdir -p frontend/img
yashrawat@Yashs-MacBook-Air testing3 % mkdir -p frontend/statics
yashrawat@Yashs-MacBook-Air testing3 % ls
frontend
yashrawat@Yashs-MacBook-Air testing3 % cd frontend
yashrawat@Yashs-MacBook-Air frontend % ls
css     img     scripts statics
yashrawat@Yashs-MacBook-Air frontend %
```





Mv command

It also help us to rename a file .

```
yashrawat@Yashs-MacBook-Air testing % mv yash1.txt maggi.txt
yashrawat@Yashs-MacBook-Air testing % ls
maggi.txt       testing2        testing3        yashh.txt
yashrawat@Yashs-MacBook-Air testing % mv testing2 maggi2
yashrawat@Yashs-MacBook-Air testing % ls
maggi.txt       maggi2          testing3        yashh.txt
yashrawat@Yashs-MacBook-Air testing %
```

It will move the file + rename it too.

```
maggi2 — -zsh — 80×24
yashrawat@Yashs-MacBook-Air testing % cd maggi2
yashrawat@Yashs-MacBook-Air maggi2 % ls
testing         yash.js         yashh.txt
yashrawat@Yashs-MacBook-Air maggi2 % pwd
/Users/yashrawat/Desktop/testing/maggi2
yashrawat@Yashs-MacBook-Air maggi2 % mv yash.js /Users/yashrawat/Desktop/testing
/maggi3.js
yashrawat@Yashs-MacBook-Air maggi2 %
```

Now if you want to delete the particular file or folder

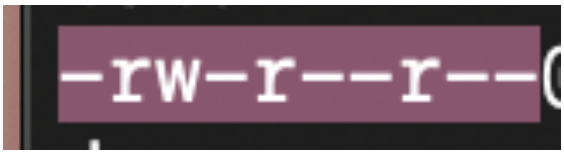
rm command

the rm command is used in Unix-like operating systems to remove (delete) files and directories

```
rm: frontend: is a directory
yashrawat@Yashs-MacBook-Air testing3 % touch test23.txt
yashrawat@Yashs-MacBook-Air testing3 % rm test23.txt
yashrawat@Yashs-MacBook-Air testing3 % cd ..
yashrawat@Yashs-MacBook-Air testing % rm -r testing3
yashrawat@Yashs-MacBook-Air testing %
```

Permission

There are three types of permission - read(r),write(w) and execute(x)



And you can see there is three sets of permission first one is for

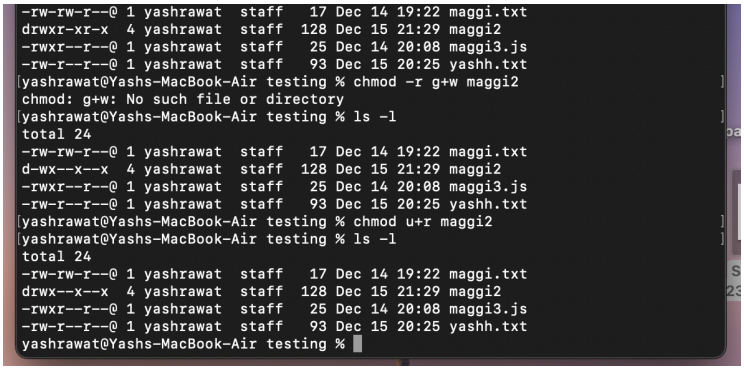
```
-rw-r--r--@ 1 yashrawat  staff   17 Dec 14 19:22 maggi.txt
```

This blue one is for first set .then the second one is only for staff. And third one you can create .

And to change this permission there is command know as chmod.

Chmod command

The chmod command in Unix-like operating systems is used to change the permissions (read, write, and execute) of files and directories.



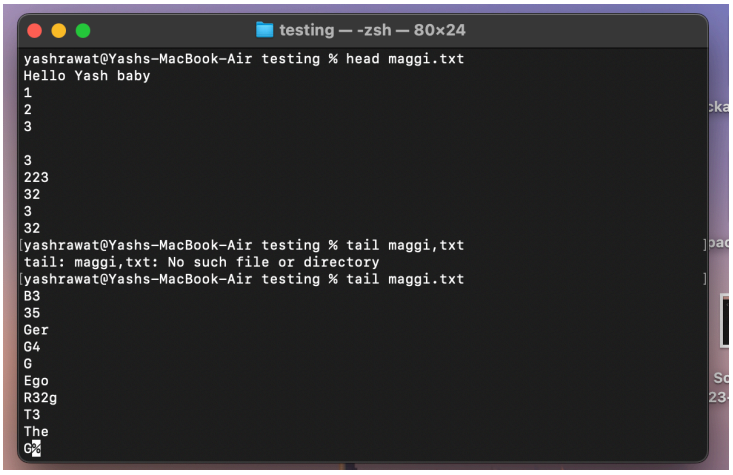
Now symbolic method of how you can do it -

- u (user/owner), g (group), o (others), a (all).
- + (add permissions), - (remove permissions), = (set exact permissions).
- r (read), w (write), x (execute).

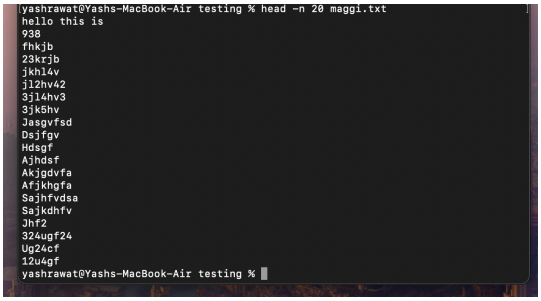
Head command and tail command

Head command(default) is used to print the top content (top ten line of the content).

Tail command(default) is used to print the last content (last ten line of the content)



If you want to print top 20 you can use this command



Pipe command (|)

This is a pipe symbol, which takes the output from the command on its left and uses it as input for the command on its right.

Now we can see the command use of tail and head -

```
yashrawat@Yashs-MacBook-Air testing % tail -n +17 maggi.txt | head -n 9
Jhf2
324ugf24
Ug24cf
12u4gf
21guf4l1
12u4yf12
1uyf4
1uf4
4f12i
yashrawat@Yashs-MacBook-Air testing %
```

It will Start printing from the line 17(starting from top of the file) to 9 line below 17 line .

Wc command

The wc command in Unix-like operating systems is used to display information about the number of lines, words, and bytes in a file or standard input.

```
yashrawat@Yashs-MacBook-Air testing % wc maggi.txt
      34      36     286 maggi.txt
yashrawat@Yashs-MacBook-Air testing %
```

Number of lines - 34, number of words - 36, number of characters(bytes) - 286

Common Options:

- -l: Display only the number of lines.
- -w: Display only the number of words.
- -c: Display only the number of bytes.
- -m: Display only the number of characters.
- -L: Display the length of the longest line.

```
yashrawat@Yashs-MacBook-Air testing % wc -l maggi.txt
      34 maggi.txt
yashrawat@Yashs-MacBook-Air testing % wc -w maggi.txt
      36 maggi.txt
yashrawat@Yashs-MacBook-Air testing % wc -c maggi.txt
     286 maggi.txt
yashrawat@Yashs-MacBook-Air testing % we -m maggi.txt
zsh: command not found: we
yashrawat@Yashs-MacBook-Air testing % wc -m maggi.txt
     286 maggi.txt
yashrawat@Yashs-MacBook-Air testing % wc -L maggi.txt
      14 maggi.txt
yashrawat@Yashs-MacBook-Air testing %
```

Grep command

The grep command in Unix-like operating systems is a powerful tool for searching and filtering text patterns within files or the output of other commands.

```
yashrawat@Yashs-MacBook-Air testing % grep "f" maggi.txt
fhkjb
Jasgvfsd
Dsjfgv
Hdsgf
Ajhdsf
Akjgdvfa
Afjkhgfa
Sajhfvdsa
Sajkdhfv
Jhf2
324ugf24
Ug24cf
12u4gf
21guf4l1
12u4yf12
1uyf4
1uf4
4f12i
21h4jf5g
23iohf252p
23uf432i
```

In combination with wc command

```
yashrawat@Yashs-MacBook-Air testing % grep "f" maggi.txt | wc
      24      24     194
yashrawat@Yashs-MacBook-Air testing % grep "f" maggi.txt | wc -l
      24
yashrawat@Yashs-MacBook-Air testing %
```

If you use it with command -n it will also show us the line of the occurrence .

```
yashrawat@Yashs-MacBook-Air testing % grep -n  "f" maggi.txt
3:fhkjb
9:Jasgvfsd
10:Dsjfgv
11:Hdsgf
12:Ajhdsf
13:Akjgdvfa
14:Afjkhgfa
15:Sajhfvdsa
16:Sajkdhfv
17:Jhf2
18:324ugf24
19:Ug24cf
20:12u4gf
21:21guf4l1
22:12u4yf12
23:1uyf4
24:1uf4
25:4f12i
26:21h4jf5g
27:23iohf252p
28:23uf432i
29:2pify21i
```

If you want to count the number of occurrences -

```
yashrawat@Yashs-MacBook-Air testing % grep -c "f" maggi.txt
24
yashrawat@Yashs-MacBook-Air testing %
```

If you want the search for particular word -

```
yashrawat@Yashs-MacBook-Air testing % grep -w "f" maggi.txt
f
f
yashrawat@Yashs-MacBook-Air testing %
```

It will print all the occurrence which don't involve "f" in it .

```
yashrawat@Yashs-MacBook-Air testing % grep -v "f" maggi.txt
hello this is
938
23krjb
jkhl4v
jl2hv42
3jl4hv3
3jk5hv
12ougy4i1
F
21i4ygip1g1
12ipg4i12pg12
F
yashrawat@Yashs-MacBook-Air testing %
```

History command

History command is used to view all the previous executed commands.

```
yashrawat@Yashs-MacBook-Air testing % history 0
    1  cd "/Users/yashrawat/Desktop/cpp/milestone 3/pointer/dyanmicallocation/" && g++ dynamicAllocation.cpp -o dynamicAllocation && "/Users/yashrawat/Desktop/cpp/milestone 3
namicAllocation
    2  cd "/Users/yashrawat/Desktop/cpp/milestone 3/pointer/dyanmicallocation/" && g++ dynamicAllocation.cpp -o dynamicAllocation && "/Users/yashrawat/Desktop/cpp/milestone 3
namicAllocation
    3  cd "/Users/yashrawat/Desktop/cpp/milestone 3/pointer/dyanmicallocation/" && g++ dynamicAllocation.cpp -o dynamicAllocation && "/Users/yashrawat/Desktop/cpp/milestone 3
namicAllocation
    4  cd "/Users/yashrawat/Desktop/cpp/milestone 3/pointer/dyanmicallocation/" && g++ dynamicAllocation.cpp -o dynamicAllocation && "/Users/yashrawat/Desktop/cpp/milestone 3
namicAllocation
    5  cd "/Users/yashrawat/Desktop/cpp/milestone 3/pointer/dyanmicallocation/" && g++ macrosAndGlobalVariables.cpp -o macrosAndGlobalVariables && "/Users/yashrawat/Desktop/c
allocation/"macrosAndGlobalVariables
    6  cd "/Users/yashrawat/Desktop/cpp/milestone 3/pointer/dyanmicallocation/" && g++ macrosAndGlobalVariables.cpp -o macrosAndGlobalVariables && "/Users/yashrawat/Desktop/c
allocation/"macrosAndGlobalVariables
    7  cd "/Users/yashrawat/Desktop/cpp/milestone 3/pointer/dyanmicallocation/" && g++ macrosAndGlobalVariables.cpp -o macrosAndGlobalVariables && "/Users/yashrawat/Desktop/c
allocation/"macrosAndGlobalVariables
```

Sed command

The sed command, short for stream editor, is a powerful tool for processing and transforming text. It operates by reading input line by line and applying specified operations to each line.

Command - **sed "s/f/p/" maggi.txt**

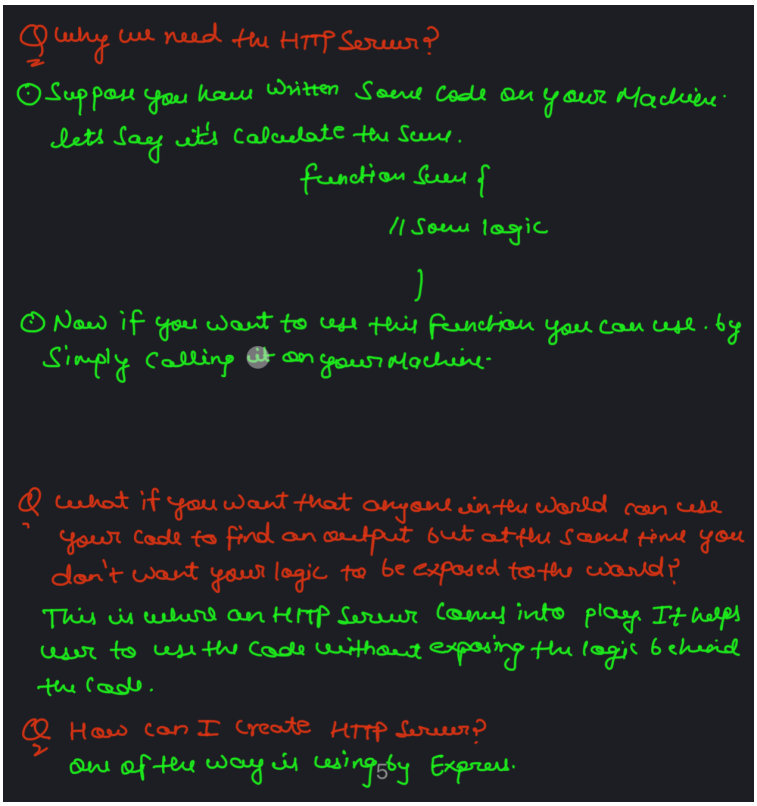
It will replace f with p .

```
yashrawat@Yashs-MacBook-Air testing % sed "s/f/p/" maggi.txt

hello this is
938
phkjb
23krjb
jkhl4v
jl2hv42
3jl4hv3
3jk5hv
Jasgvpsd
Dsjpgv
Hdsgp
Ajhdsp
Akjgdvpa
Apjkhgfa
Sajhpvdsa
Sajkdhpv
Jhp2
324ugp24
Ug24cp
12u4gp
21gup4l1
12u4yp12
1uyp4
1up4
4p12i
21h4jp5g
23iohp252p
23up432i
2pipy21i
12ougy4i1
F
21i4ygip1g1
1ig4lipg12
12ipg4i12pg12
U214ip1i4
p
p
F
yashrawat@Yashs-MacBook-Air testing %
```

It will create a backup of the file .

```
yashrawat@Yashs-MacBook-Air testing % sed -ibackup "s/f/p/" maggi.txt
yashrawat@Yashs-MacBook-Air testing % ls
maggi.txt       maggi.txtbackup maggi2          maggi3.js       yashh.txt
yashrawat@Yashs-MacBook-Air testing %
```

Week 2.5
Cohort 2.0
Express with example



The idea is to allow users to utilize your code and obtain its output without revealing the underlying logic. An HTTP server, such as one created with Express, acts as an intermediary, facilitating interaction with your code while keeping the internal details protected. This approach is beneficial for maintaining code privacy while offering a service accessible through an API or web interface.

Q. What is const express = require("express") ?

It is the library written by very smart people to create an HTTP server .it sort of encapsulate the complexity of creating an http server.

For example you want to Run a hospital .before that you want to build it too.so for that you don't need an architect degree first.you will be like I will be delegate this task to someone .

Just like that express is a library where really smart people have written HTTP server that are optimized .you just have to call a function on it and it will expose your functionality over the http protocol.

Q. What is **const app=express();** ?

It will help, you create an instance of the express library object .

For example its the room in the hospital where doctor is sitting .

Q. What is **app.get("/",function (req,res){**
Const n=req.query.n;
Const ans =calculateSum(n)
res.send(ans);

)
}. ?

It is basically one of the functionality.
For example exposing doctors one of the functionality . Doctor can have multiple room inside the hospital it is one of them .
For example doctor be like I am brain surgeon for that query come to me , and another one be like I am backpain specialist so anyone with backpain query come here .

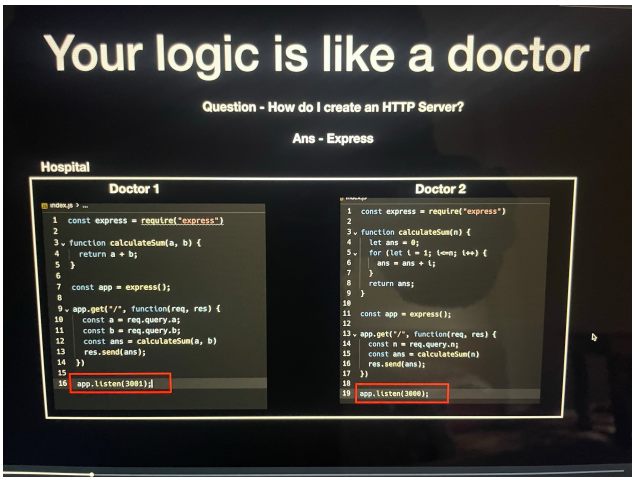
Q. What is **app.listen(3000)** ?

Deciding the address of the clinic .

Here the doctor 1 listens on port 3001 and there is another doctor 2 which is in the same hospital sits in the room which has a port number 3000.

So basically here we have to http server.

Yes, it's possible to run multiple HTTP servers on a single machine. Each HTTP server would typically listen on a different port, allowing them to operate independently.



Express

This express library is not included into the node so we have to install it explicitly .using the command npm install express.

```
const express=require("express")

const app=express()

app.listen(3000)
```

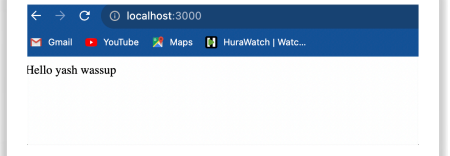
This is the basic http server.

Now lets create a route -

In these we have created a route “/”.

So any query with these specific route will be answered by these route .

```
const express=require("express")

const app=express()

app.get("/",function(req,res){
    res.send("Hello yash wassup")
})

app.listen(3000)
```



Now people can access your logic .now it does means I will give the url to my friend .and he can access it ?

No not now ,for that you have to deploy it on the Internet or on the aws machine .right now you have deployed on your local network .so you can access it from your phone .

Ifconfig command

The ifconfig command on Unix-based systems displays information about network interfaces. This will provide details about each network interface, including their flags, IP addresses, and other relevant information.

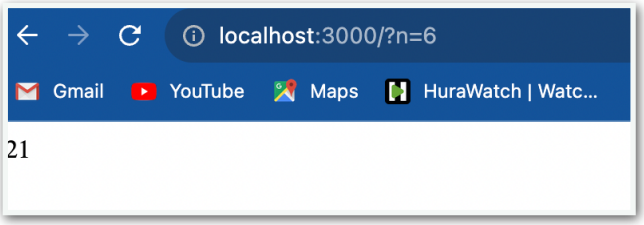
You can also search for private ip address.

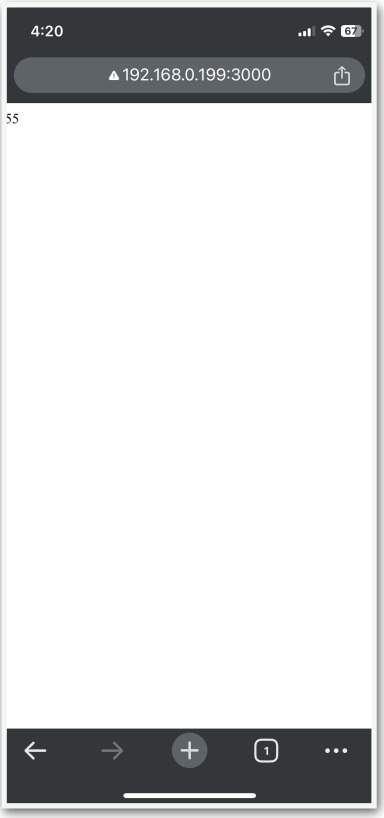
Query parameter

Query parameter is the way by which you can send data to your backend as a query in url it is popular among get request .

Whatever written after “?” Is a query parameters.

```
const express=require("express")

const app=express()

function sum(n){
    let ans=0
    for(let i=0;i<=n;i++){
        ans=ans+i;
    }
    return ans;
}

app.get("/",function(req,res){
    const n=req.query.n;
    const ans=sum(n)
    res.send(ans.toString())
})

app.listen(3000)
```





Accessing the code written on my Mac by my phone through router .

What’s happening is my phone is sending a request to the http server written on my Mac where the logic is running and the server is sending me the output.

Request method

Get - going for a consultation to get a check up .(for example in linked in you refresh your post)

Post - going to get new kidney inserted . (For example posting on linked in)

Put - going to get kidney replaced (for example changing your linked in password of editing your lined in post)

Delete - going to get kidney removed (for example deleting the post on linked in)

Q. What does **in memory** means ?
It means that all the data will be saved in the memory (variable) instead of database .

Another way to send data to the backend is through the **body** . It is popular among the POST method .

Postman

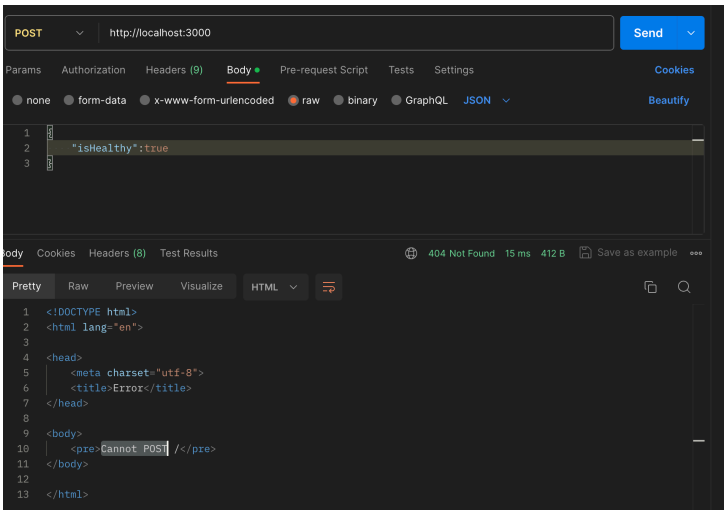
Postman is a popular API development and testing tool that provides a user-friendly interface for creating, sending, and testing HTTP requests. It is widely used by developers and teams to streamline the process of working with APIs.

Post Method

Code which is written.

```
app.post("/",function(req,res){
    const isHealthy=req.body.isHealthy
    users[0].kidneys.push({
        healthy:isHealthy
    })
    res.json({
        msg:"done!"
    })
})
```

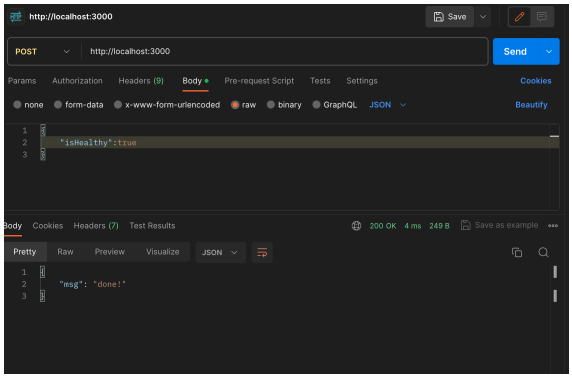
But when we send a post request it shows us an error



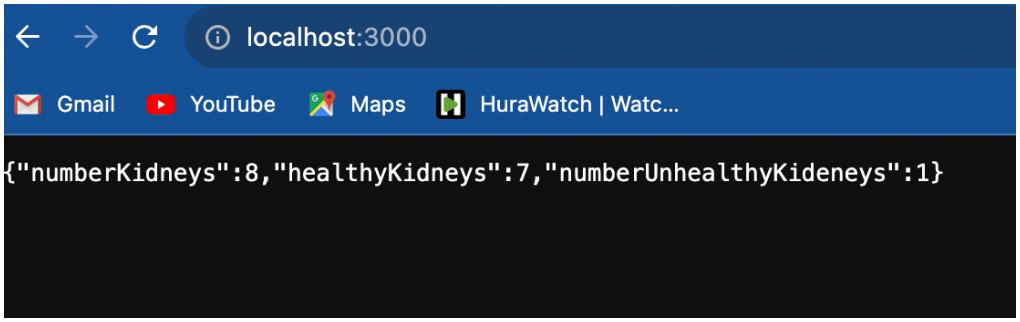
NOTE - As we know that our express by default ignore the body part .so we have to explicitly need to add one more line globally .

app.use(express.json())

After using it we can successfully send the post request to the backend —->



So how I can confirm that .as we know earlier I have 2 kidneys . And then I have sent 6 post request . So ideal if I send one more get request the kidneys will be updated to 8 kidneys .



As I see my kidney gets updated to 8 .

It is same as example when we upload a post on linked in from mobile and then we refresh the site of linked on browser then we see the uploaded post on the browser . It means that there is common backend where the data is uploaded(means we have common place where multiple client can update the data) .

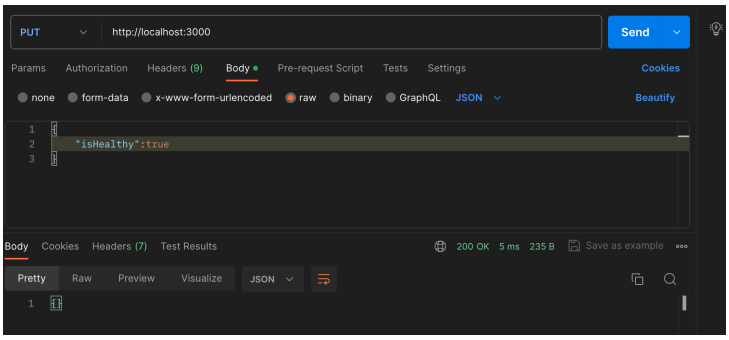
NOTE - as I restart the server my data gets restarted (means lost).this is what happens when we save our data in memory instead of databases.

PUT REQUEST

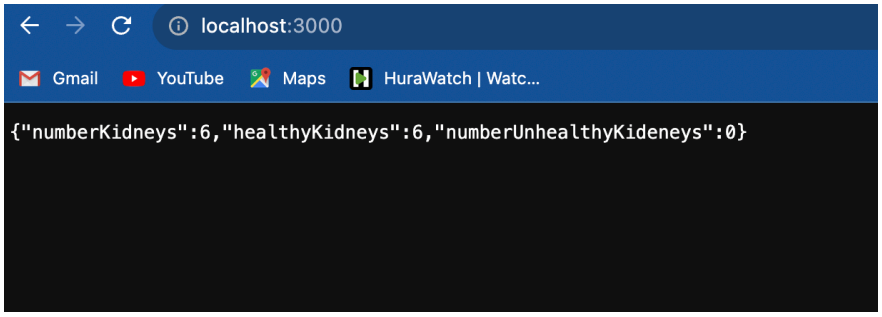
Code written

```
app.put("/",function(req,res){
    for(let i=0;i<users[0].kidneys.length;i++){
        users[0].kidneys[i].healthy=true;
    }
    res.send({})
})
```

Put request



Output



DELETE METHOD

Code written

```
app.delete("/",function(req,res){
    let updatedKidneys=[]
    for(let i=0;i<users[0].kidneys.length;i++){
        if(users[0].kidneys[i].healthy){
            updatedKidneys.push(
                {healthy:true}
            )
        }
    }
    users[0].kidneys=updatedKidneys;
    res.send({})
})
```

output

Gmail YouTube Maps HuraWatch | Watc...

```
{"numberKidneys":6,"healthyKidneys":6,"numberUnhealthyKideneys":0}
```

DELETE http://localhost:3000 Send

Params Authorization Headers (9) Body Pre-request Script Tests Settings Cookies

none form-data x-www-form-urlencoded raw binary GraphQL JSON Beautify

```
1 {
2     "isHealthy":true
3 }
```

Body Cookies Headers (7) Test Results 200 OK 4 ms 235 B Save as example

Pretty Raw Preview Visualize JSON

```
1 {}
```

NOTE - always remember to send the res no matter if its empty or not .

MAP,FILTER

Q. Given an array ,give me back a new array in which every value is multiplied by 2.
Ans
[1,2,3,4,5] —-> [2,4,6,8,10]

Dumb way of doing it

```
const input=[1,2,3,4,5]
let newArray=[]
for(let i=0;i<input.length;i++){
    newArray.push(input[i]*2)
}
console.log(newArray)
```

smart way of doing it

Smart way => using By Map.

Map wants two things :-

1. Input array.
2. Transformation function

Map(arr, function) => [2, 4, 6, 8, 10]

```
const input=[1,2,3,4,5]
const newArray=input.map((i)=>{return i * 2})
console.log(newArray)
```

```
node /tmp/eJjmrONDCL.js
[ 2, 4, 6, 8, 10 ]
```

Q. Given an array filter out only an even element from it and then return back the new array?

Dumb way of doing it

```
const arr=[1,2,3,4,5]
let newArray=[]
for(let i=0;i<arr.length;i++){
    if(arr[i]%2==0){
        newArray.push(arr[i])
    }
}
console.log(newArray)
```

```
node /tmp/eJjmrONDCL.js
[ 2, 4 ]
```

Smart way of doing it => filter

filter wants two things :-

1. Input array.
2. filtering out logic

filter(arr, filteringOutlogic) => [2,4]

```
const arr=[1,2,3,4,5]
let newArray=arr.filter((i)=>{
    if(i%2==0){
        return true
    }else{
        return false
    }
})
console.log(newArray)
```

```
node /tmp/eJjmrONDCL.js
[ 2, 4 ]
```

Cohort 2.7
GIT AND GITHUB

Q. What is GIT ?
Free and open source version control system .

Q. What is version control system ?
1. A system that keeps track of our files or projects .

2. It allows you to revert selected files to a previous state , revert the entire project to a previous state ,compare changes over time ,see who last modified something so that we can know what might be causing a problem ,or what is the issue ,who made it ,and when with the details .

Q. Why we need the version control system?

A version control system (VCS) is crucial in software development for several reasons:

- **Collaboration:**
  - VCS enables multiple developers to work on the same project simultaneously. It allows them to make changes independently, and the system merges those changes seamlessly.
- **History and Audit Trail:**
  - VCS maintains a history of changes made to the codebase. This history includes who made the changes, when they were made, and what exactly was changed. This audit trail is valuable for debugging, accountability, and understanding the evolution of the project.
- **Code Revert and Rollback:**
  - Developers can revert to previous versions of the code easily. If a new feature introduces bugs or issues, it's possible to roll back to a stable version.
- **Branching and Merging:**
  - VCS allows developers to create branches, which are separate lines of development. This is useful for working on new features or bug fixes without affecting the main codebase. Merging allows the integration of these changes back into the main code.
- **Concurrency:**
  - Version control handles concurrent access to the codebase. Multiple developers can make changes simultaneously, and the system manages the merging of those changes.
- B**ackup and Recovery:**
  - The codebase is stored in a centralized repository, providing a backup mechanism. If local copies are lost or corrupted, developers can retrieve the code from the repository.
- **Collaborative Development:**

    - VCS facilitates collaborative development, enabling teams distributed across locations to work together seamlessly. Each developer can contribute without interfering with others.

- **Traceability:**
    - VCS helps in tracing when and why a change was made. This traceability is essential for understanding the rationale behind code modifications.

- **Experimentation and Feature Development:**
    - Developers can experiment with new ideas or features in separate branches without affecting the stability of the main codebase. This promotes a culture of innovation.

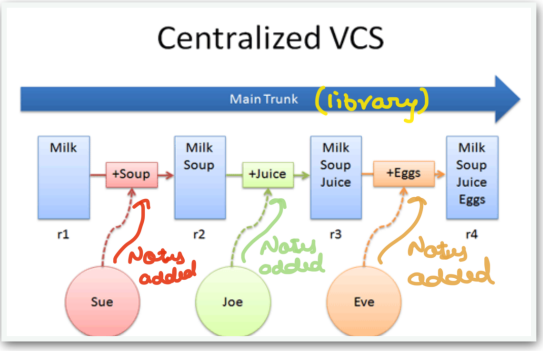
- **Release Management:**
    - VCS plays a crucial role in release management. It allows for the tagging of specific versions, making it easy to identify and deploy releases.

There are two types of VCS(version control system)

1. Centralized version control system(CVCS)

Imagine a Library:

- CVCS is like a library where everyone in a team checks out books to read and makes notes in them.
- There's a central librarian (repository) who keeps the main copy of all the books (code).
- When someone wants to make changes to a book, they take it out, make their notes, and then put it back in the library.
- Others can see those changes, but there's one main place where all the important stuff is stored.



Key Points:

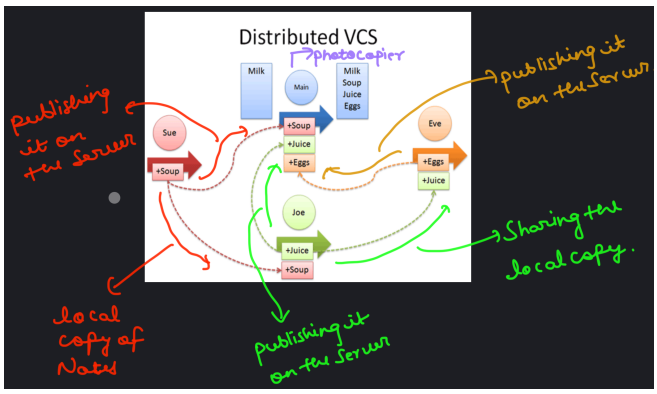
- Central Hub: There's a central server/repository that holds the "official" version of the project.
- Check-Out/Check-In: Developers "check out" files to work on them, and "check in" changes to the central repository.
- Dependency on Central Server: Everyone needs to connect to the central server to get the latest code or to save their changes.

2. Distributed version control system (don't always you need to available online to make changes. You can make changes in your local repo and at the time of publish to main repo you just need a connection)

Imagine a Photocopier and Shared Notes:

- DVCS is like having a personal notebook and a photocopier.

- Each person in the team has their own notebook (local repository) with all the project notes (code).
- When someone wants to share their notes or make changes, they use the photocopier (pull and push changes) to share with others.
- If the photocopier (central server) is not available, they can still work and share notes with others using their own copies.



Key Points:

- Local Copies: Everyone has a complete copy of the project with its entire history on their computer.
- Independent Work: Developers can work independently and make changes without needing a constant connection to a central server.
- Flexibility: If the central server goes down, work can still continue using local copies.

Comparison:

- **CVCS:** Like everyone going to the same library and checking out books from a central place.
- **DVCS:** Like each person having their own set of books and a photocopier to share changes.

In simple terms, CVCS has one central place where everything is stored, while DVCS gives everyone their own complete set of project notes, making it more flexible and allowing independent work.

Q. Why GIT ?
1. Free
2.open source
3.scalable
4.super fast
5. Cheap branching and merging

Q. How is GitHub is different from git ?

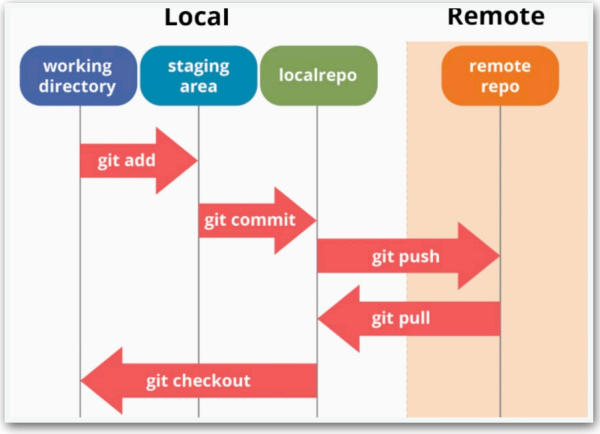
- Git: It's like a personal time machine for your code. You use Git to manage and track the history of your code changes on your local machine.
- GitHub: It's like a platform on the internet where you can store your time machine (Git repository). GitHub makes it easy to share your code with others, collaborate, and have a backup of your work in the cloud. It is like a web interface to view your file .

So, in simple terms:

- Git is for local history management.
- GitHub is for sharing and collaborating with others on the internet.

Basic workflow of Git

- Clone an Existing Repository:
  - Use git clone <repository> to copy an existing repository to your local machine.
- Make Changes:
  - Work on your code, modify files, add new features, etc.
- Stage Changes:
  - Use git add . to stage all changes or git add <filename> to stage specific changes. Staging is like preparing changes for a commit.



- Commit Changes:
  - Use git commit -m "Your commit message" to save the staged changes as a commit. This creates a snapshot of your project with those changes.
- Push Changes (Optional):
  - If you cloned a repository from a remote source (like GitHub), you might want to push your changes back to that source using git push origin <branch-name>.

So, you stage changes first, and then you commit them. The staging area is like a middle step where you decide which changes you want to include in your next commit. Once you're happy with the staged changes, you commit them to your local repository. If you want to share those changes with others, you can then push them to a remote repository (like GitHub).

BLOBS AND TREE

BLOBS

Each time you make a change to the content of a file, Git creates a new Blob to represent that specific version of the file. This new Blob is identified by a unique SHA-1 hash, capturing the content at that moment.
So, every change results in a new snapshot (Blob) of the file, allowing Git to keep track of the entire history of changes to each file in your project.

TREE

- Tree Doesn't Contain File Content:
  - Correct! A Tree doesn't store the actual content of files; instead, it contains references to Blobs (file content) and other Trees (subdirectories).
- Paths to File Content:
  - Exactly! A Tree holds information about the filenames, permissions, and SHA-1 hashes (like an address or path) pointing to Blobs or other Trees that store the actual content.

So, a Tree acts as a kind of directory snapshot, organizing the paths and references to where Git can find the actual file content (Blobs) or nested directories (other Trees). It's like a guide indicating where to locate the different parts of your project.

Q. If I change the content of the file the new blob is created this means the new tree will also be created ?

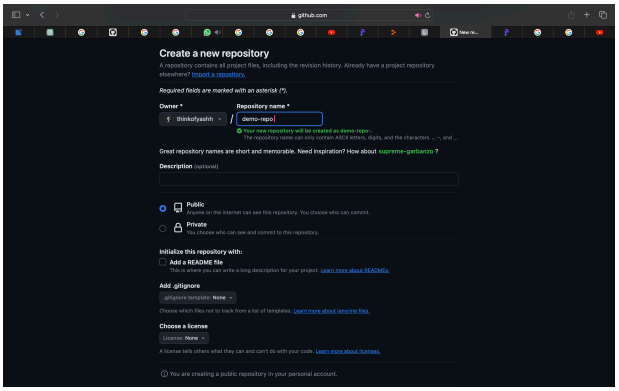
When you change the content of a file, a new Blob is created to represent that specific version of the file. Since a Tree in Git contains references to Blobs (file content), changing the content of a file results in a change to the SHA-1 hash of the Blob.
When you create a new commit, the Tree associated with that commit reflects the updated SHA-1 hash of the changed file, creating a new Tree. This new Tree captures the updated references to the latest Blobs, forming a snapshot of the updated project state.
So, changing file content leads to a new Blob, and if this file is part of your project's directory structure, it contributes to the creation of a new Tree in the subsequent commit.

GIT COMMANDS

Clone - bring a repository hosted somewhere like GitHub into a folder or your local machine .
Add- track your files and changes in git
Commit - saves your files in git (staging area)
Push - uploads your commits to a git repo like GitHub.
Pull- Download changes from a remote repository to your local repository

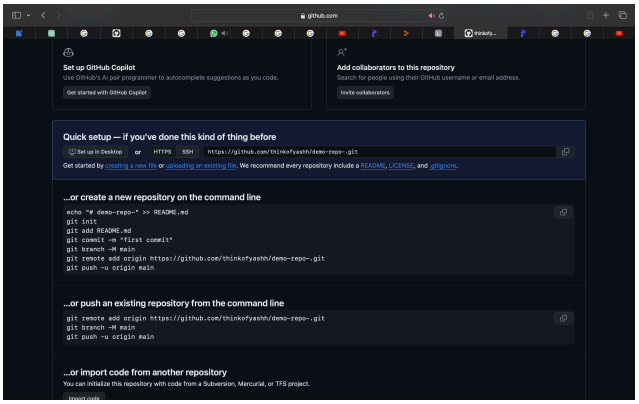
GITHUB

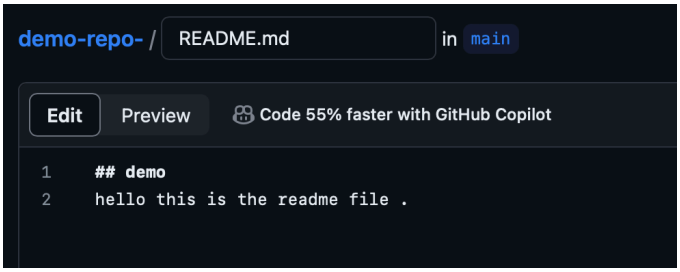
Step -1 create a repo



Now lets commit the file .using the GitHub ui .

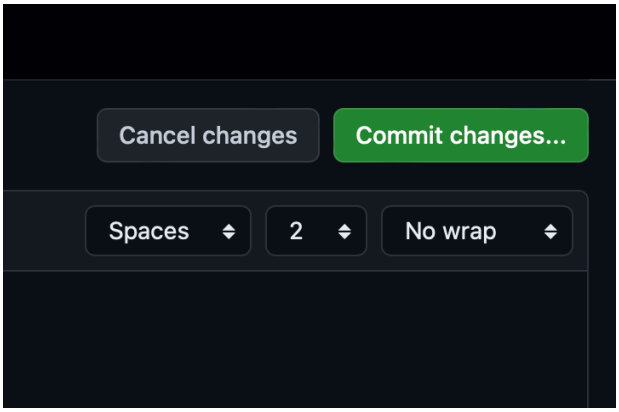
Click on [creating a new file](#) .

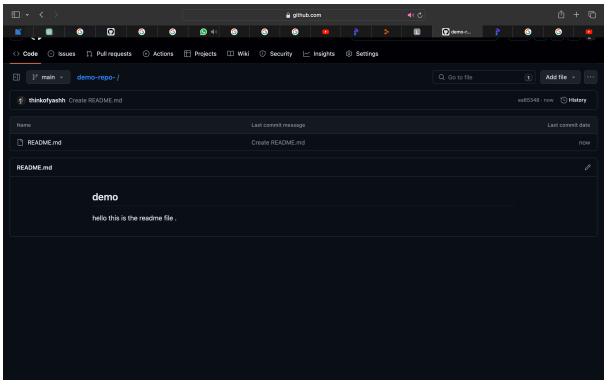




Then give a name to the file with the extension of .md . And then write the content of the file .

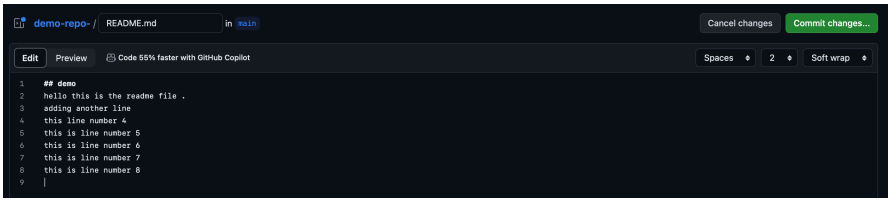
And then click on commit changes .button to save the file .

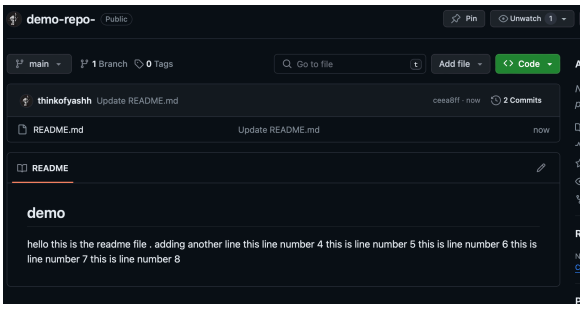




New file is created in your repo .and now lets edit it by clicking on the pencil icon .

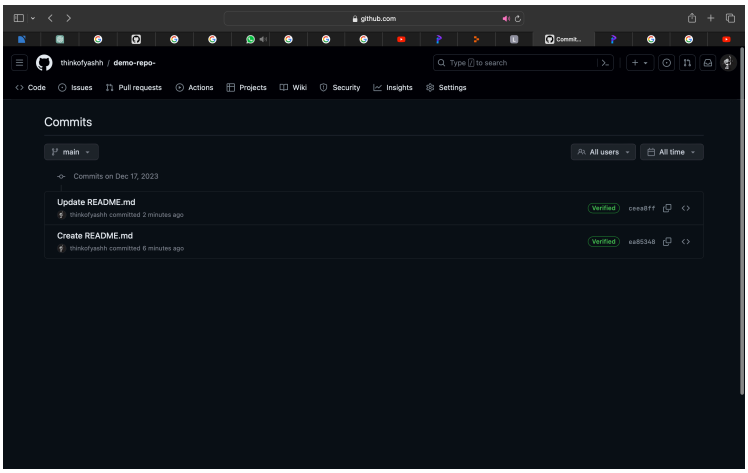
After editing the file click on commit changes .



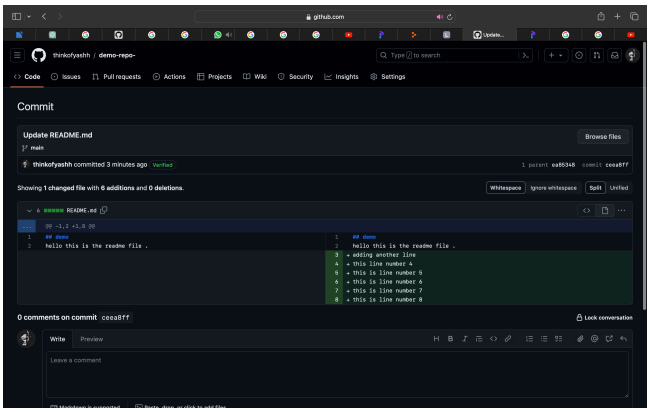


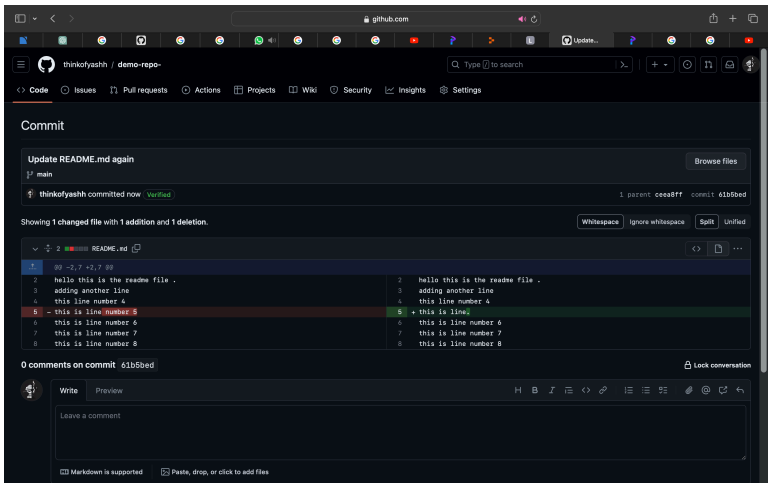
File is updated. And then we click on commits option or history option we can see the history of the file version .

This the version of the file . And if click on the file you will the commit details .



Here in the green line is the updated file .or the new changes introduced in the file .

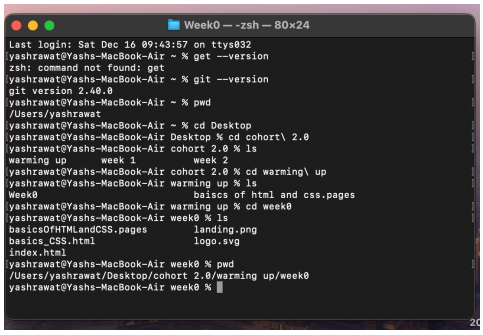




And this red line says that in these commit this line is changed from “this is line number 5” to “this is line ”

Now lets understand that how can we create a repo using command line .

Step 1 we will navigate to the folder we trying to push .



Step 2 run command git init in the folder which you want to push to the GitHub.
This git init command will initialise the empty git repo in the folder(week 0) .
And to see the git repo you need to run another command name **ls -la.**

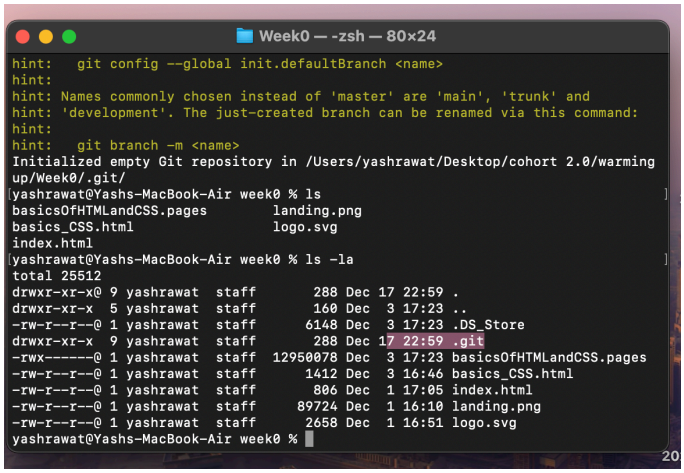
Ls -la command

The ls -la command is used in Unix-based systems (like Linux) to list detailed information about files and directories in a directory. Here's a breakdown of what each part of the command does:

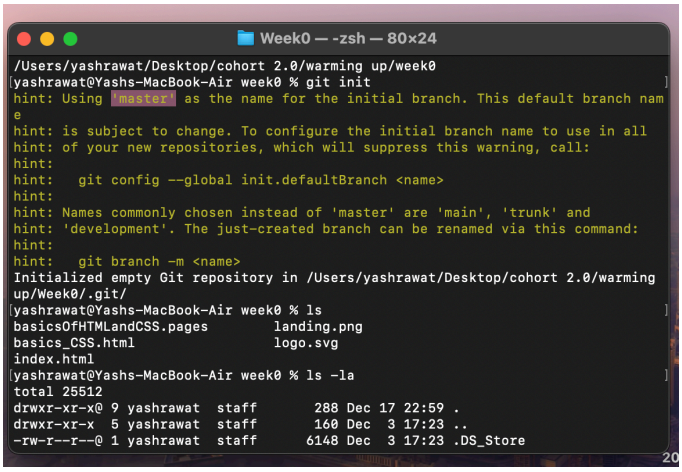
- ls:
  - Stands for "list." It's a command to display the contents of a directory.
- -la:
  - The options passed to ls modify its behavior.

- -l: Provides a long listing format, giving detailed information about each file or directory.
- -a: Includes hidden files and directories in the listing. Hidden files start with a dot (.).

So, when you run ls -la, you get a detailed list of all files and directories in the current directory, including hidden ones, with information such as permissions, owner, group, size, modification date, and the name of each entry.



This git will contain all of the commit ,config and all of the history and all the operation we are gonna perform on this particular local repo.

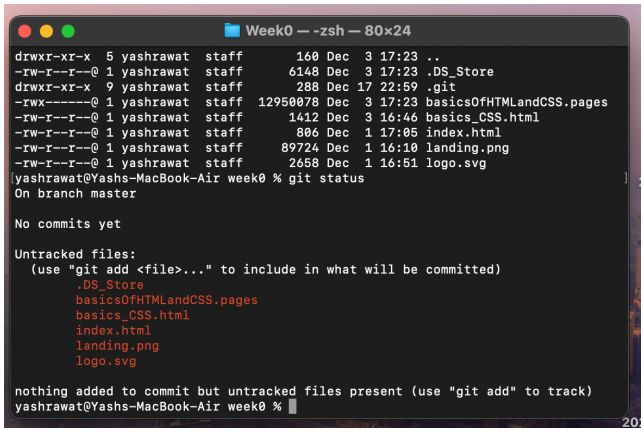


This master is the branch we are working on .

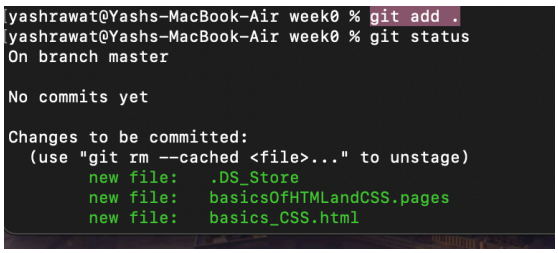
Now as we know we first need to stage this changes before we commit this changes or save this changes in our repo.

Inorder to look at the current status we can use the command **git status**

Right now we haven't added any of this file .



Now to add this files we can use the command know as **git add .** And then check the status by **git status** command



And now it is saying you can un stage particular file that you don't wanna commit . And now we can add those file in our repo by using command **git commit -m"particular message"-m"another message "**

But this files are not present in our GitHub repo ,currently they are located in my local git repo.

```
Week0 — -zsh — 80×24
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   .DS_Store
        new file:   basicsOfHTMLandCSS.pages
        new file:   basics_CSS.html
        new file:   index.html
        new file:   landing.png
        new file:   logo.svg

yashrawat@Yashs-MacBook-Air week0 % git commit -m"adding project files"-m"along
with readme"
[master (root-commit) 92a375f] adding project files-malong with readme
 6 files changed, 66 insertions(+)
 create mode 100644 .DS_Store
 create mode 100755 basicsOfHTMLandCSS.pages
 create mode 100644 basics_CSS.html
 create mode 100644 index.html
 create mode 100644 landing.png
 create mode 100644 logo.svg
yashrawat@Yashs-MacBook-Air week0 %
```

And to push this code to the GitHub repo we need to go to the GitHub repo and copy the link .

And before doing that we need check if there is any remote repo available on or not . We can check it by running the command **git remote -v**

```
yashrawat@Yashs-MacBook-Air week0 % git remote -v
yashrawat@Yashs-MacBook-Air week0 %
```

If it doesn't show something it means there is no repo that have been configured to push our changes from our local to a remote repo.

```
yashrawat@Yashs-MacBook-Air week0 % git remote -v
yashrawat@Yashs-MacBook-Air week0 % git remote add origin https://github.com/thinkofyashh/demo-repo-.git
yashrawat@Yashs-MacBook-Air week0 %
```

Then use command **git remote add origin [link of the GitHub repo].git**

```
yashrawat@Yashs-MacBook-Air week0 % git remote -v
origin  https://github.com/thinkofyashh/demo-repo-.git (fetch)
origin  https://github.com/thinkofyashh/demo-repo-.git (push)
yashrawat@Yashs-MacBook-Air week0 %
```

Now if you use **git remote -v** command you will see the repo that have been configured to push your changes from local repo to a remote repo .

```
yashrawat@Yashs-MacBook-Air week0 % git status
On branch master
nothing to commit, working tree clean
yashrawat@Yashs-MacBook-Air week0 %
```

Now if use the **git status** command . You will see it will say I am on the branch master and I have nothing to commit because there are no further changes that need to be added .

Now you need to push the changes from your local repo to remote repo. So you use the command git push but it shows error saying on which remote repo and branch it needs to push .

```
yashrawat@Yashs-MacBook-Air week0 % git push
fatal: The current branch master has no upstream branch.
To push the current branch and set the remote as upstream, use

    git push --set-upstream origin master

To have this happen automatically for branches without a tracking
upstream, see 'push.autoSetupRemote' in 'git help config'.

yashrawat@Yashs-MacBook-Air week0 %
```

So for that instead of **git push** we need to use **git push origin master**

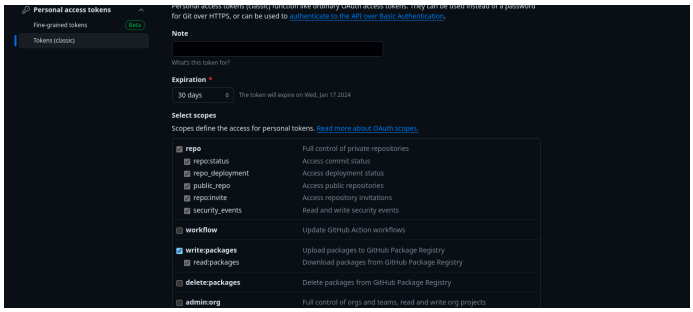
```
yashrawat@Yashs-MacBook-Air week0 % git push origin master
Username for 'https://github.com': thinkofyashh
Password for 'https://thinkofyashh@github.com':
Enumerating objects: 8, done.
Counting objects: 100% (8/8), done.
Delta compression using up to 8 threads
Compressing objects: 100% (6/8), done.
Writing objects: 100% (6/8), 9.61 MiB | 3.56 MiB/s, done.
Total 8 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), done.
remote:
remote: Create a pull request for 'master' on GitHub by visiting:
remote:      https://github.com/thinkofyashh/demo-repo-/pull/new/master
remote:
To https://github.com/thinkofyashh/demo-repo-.git
 * [new branch]      master -> master
yashrawat@Yashs-MacBook-Air week0 %
```

Personal access token- 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

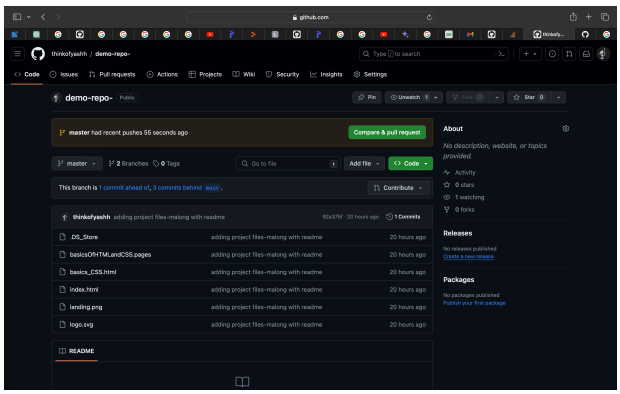
Error -

```
yashrawat@Yashs-MacBook-Air week0 % git push origin master
Username for 'https://github.com': thinkofyashh
Password for 'https://thinkofyashh@github.com':
remote: Permission to thinkofyashh/demo-repo-.git denied to thinkofyashh.
fatal: unable to access 'https://github.com/thinkofyashh/demo-repo-.git/': The requested URL returned error: 403
```

Solution - we need to generate token and while generating the token allow repo and write package permission



Now you will see that the code gets updated to the GitHub repo



CLONE THE REPOSITORY

Q. Difference between cloning the repository and initialising the repository ?

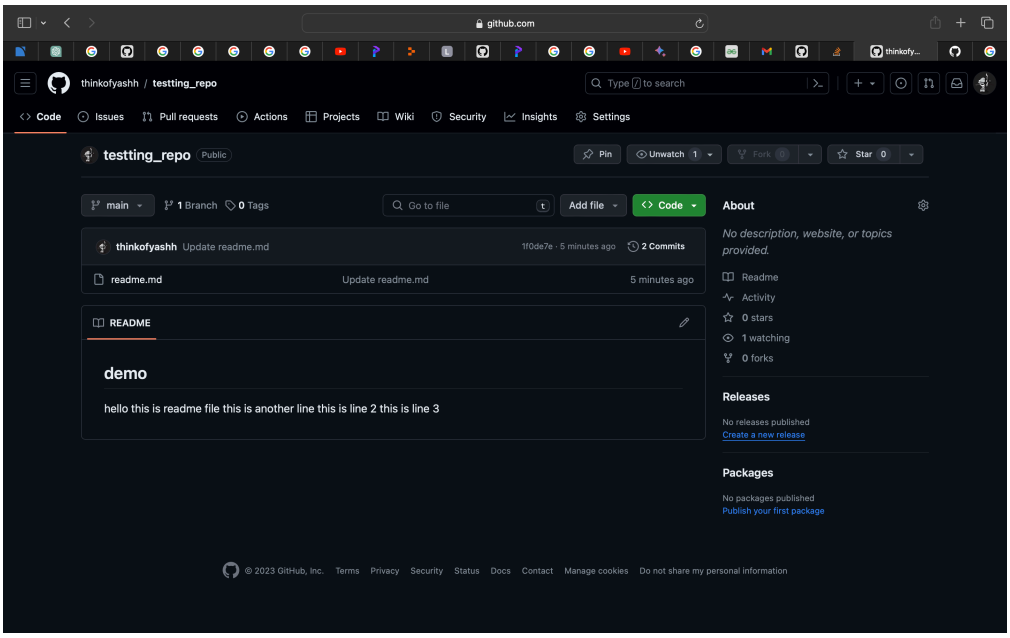
- **Cloning a Repository:**
  - Definition: Cloning refers to creating a copy of a remote repository onto your local machine.
  - Purpose: The primary purpose of cloning is to get a complete copy of an existing repository, including all its branches, commits, and files. This allows you to work on the project locally.
  - Command: The command to clone a repository is typically:
    bash

    Copy code clone
  -
  - Usage: Cloning is used when you want to collaborate on an existing project or contribute to an open-source project. It establishes a connection between your local copy and the remote repository.

- **Initializing a Repository:**
  - Definition: Initializing a repository means setting up a new Git repository in a directory that wasn't previously under version control.
  - Purpose: The main purpose of initialization is to start tracking an existing project or create a new project from scratch. This action creates a .git directory, which is the repository's metadata and where Git stores information about the project.
  - Command: The command to initialize a new repository is:
    bash

    Copy code
  - Usage: Initialization is used when you are starting a new project and want to begin tracking changes. After initialization, you add files to the repository, commit changes, and manage version control locally.

Step 1 Lets clone this repository. Copy this URL.



Step 2 now change directory to repos folder where you want to clone the repo. Using the command **git clone [link of repo from GitHub]**

```
yashrawat@Yashs-MacBook-Air repos % git clone https://github.com/thinkofyashh/testting_repo
Cloning into 'testting_repo'...
remote: Enumerating objects: 6, done.
remote: Counting objects: 100% (6/6), done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 6 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (6/6), done.
yashrawat@Yashs-MacBook-Air repos %
```

Here is your repo

```
yashrawat@Yashs-MacBook-Air repos % ls
testting_repo
yashrawat@Yashs-MacBook-Air repos % cd testting_repo
yashrawat@Yashs-MacBook-Air testting_repo % ls
readme.md
yashrawat@Yashs-MacBook-Air testting_repo %
```

STEP 2 - lets make changes to this repo . We have added new file and modified the existing file to.

Now if I do git status on it .we will show this changes .

```
yashrawat@Yashs-MacBook-Air testting_repo % 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)
        readme1.txt

no changes added to commit (use "git add" and/or "git commit -a")
yashrawat@Yashs-MacBook-Air testting_repo %
```

STEP 3 now if you are changing only the existing file you can use **git commit -ma** . But if you are also adding new file to repo you should have to use **git add .** Command

But if you see we have also added new file so we have to use **git add.**

Also if you want to add a particular file (the new ones) you have to use the command **git add particular file name**

```
yashrawat@Yashs-MacBook-Air testting_repo % git add .
```

STEP 4 now check the status of the git by **git status command**

```
yashrawat@Yashs-MacBook-Air testting_repo % 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:   readme1.txt
```

STEP 5 now we will commit this changes to our local repo

```
yashrawat@Yashs-MacBook-Air testting_repo % git commit -m "updated my readme and added new file "
[main 91e9a32] updated my readme and added new file
 2 files changed, 2 insertions(+)
 create mode 100644 readme1.txt
```

STEP 6 now we will again use git status

```
yashrawat@Yashs-MacBook-Air testting_repo % 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
yashrawat@Yashs-MacBook-Air testting_repo %
```

STEP 7 now we will **git remote -v** command and then **git push** command

```
yashrawat@Yashs-MacBook-Air testting_repo % git remote -v
origin  https://github.com/thinkofyashh/testting_repo (fetch)
origin  https://github.com/thinkofyashh/testting_repo (push)
yashrawat@Yashs-MacBook-Air testting_repo % git push
Enumerating objects: 6, done.
Counting objects: 100% (6/6), done.
Delta compression using up to 8 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (4/4), 420 bytes | 420.00 KiB/s, done.
Total 4 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/thinkofyashh/testting_repo
   1f0de7e..91e9a32  main -> main
yashrawat@Yashs-MacBook-Air testting_repo %
```

GIT BRANCHING

Now there can be multiple people working on the project for example some are working on the new feature within the project .

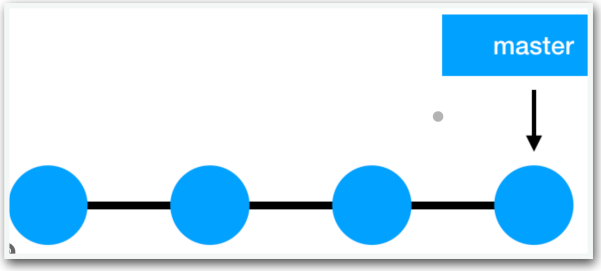
So that in that case you need way in which you can still keep the main code (which is stable) running while you can work on the copy of that code .

In Git, branching is a powerful feature that allows you to diverge from the main line of development and work on different features, fixes, or experiments without affecting the main codebase.

MASTER BRANCH

So for what we did is that we did a commit and then further did some more commit and so on. We are pushing all this thing in a one single branch . And master is the default branch we have .

The "master" branch in Git is the default branch that is created when you initialize a new Git repository. Historically, it has been the main branch where the source code for a project is stored and represents the latest stable release.



But let say there is the new feature you are working on or it could be your friends or other developers working on adding a new feature .

And we first need to test the feature before they can actually push that changes into the actual code which is running in the production system.

So what you wanted to do is you want to create a new feature branch

Q. Difference between main branch, feature branch ,hotfix branch

Main Branch (or Master Branch):

- The main branch (or master branch, depending on the naming convention) is considered the default and primary branch in a Git repository.
- It often represents the latest stable version of the project.
- Developers typically branch off from the main branch to work on new features or bug fixes.

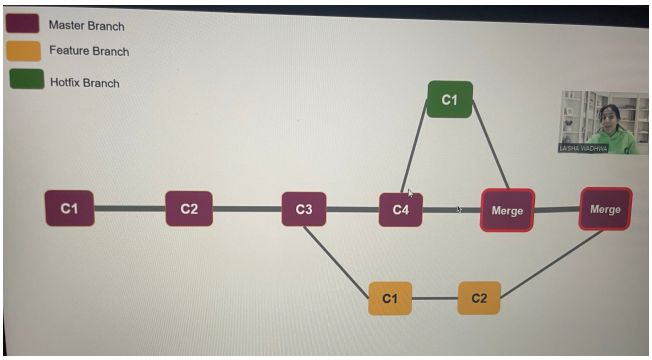
Feature Branch:

- Feature branches are created to implement new features or functionalities.

- Developers work on these branches independently of the main branch.
- Once the feature is complete and tested, it can be merged back into the main branch.

Hotfix Branch:

- Hotfix branches are used to quickly address critical issues or bugs in the production code.
- They are created based on the main branch, not on feature branches.
- Once the hotfix is ready, it is merged back into both the main branch and the active feature branches.



Lets deeper dive into the concept

Git branch command is the command to check which branch I am currently on -
Here we are currently on the main branch .

```
yashrawat@Yashs-MacBook-Air testting_repo % git branch
* main
yashrawat@Yashs-MacBook-Air testting_repo %
```

Now to move to the different branch we use **git checkout { name of the branch }**command .

But here since we don't have any other branch we will create it by using **git checkout -b [name you wanna kept of the branch]**

```
yashrawat@Yashs-MacBook-Air testting_repo % git checkout -b feature_updated_files
Switched to a new branch 'feature_updated_files'
yashrawat@Yashs-MacBook-Air testting_repo %
```

Now if you use **git branch** you will see you are on the new branch you have created

```
yashrawat@Yashs-MacBook-Air testting_repo % git branch
* feature_updated_files
  main
yashrawat@Yashs-MacBook-Air testting_repo %
```

Now if you use **git branch main** you will switched to the main branch

```
yashrawat@Yashs-MacBook-Air testting_repo % git checkout main
Switched to branch 'main'
Your branch is up to date with 'origin/main'.
yashrawat@Yashs-MacBook-Air testting_repo %
```

Now get back to the feature branch and do some changes in the file . And then use git status command to see those changes

```
yashrawat@Yashs-MacBook-Air testting_repo % git status
On branch feature_updated_files
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
        modified:   readme1.txt

no changes added to commit (use "git add" and/or "git commit -a")
yashrawat@Yashs-MacBook-Air testting_repo %
```

And then add those changes in the feature branch by using the command **git add .** And then checking the **git status**

```
yashrawat@Yashs-MacBook-Air testting_repo % git add .
yashrawat@Yashs-MacBook-Air testting_repo % git status
On branch feature_updated_files
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   readme.md
        modified:   readme1.txt

yashrawat@Yashs-MacBook-Air testting_repo %
```

We have added this changes in the staging area.

And then add to the feature repo we need to use **git commit -m [the message you wanna insert]**

```
yashrawat@Yashs-MacBook-Air testting_repo % git commit -m "updated  the readme file again"
[feature_updated_files f0edec5] updated  the readme file again
 2 files changed, 7 insertions(+), 1 deletion(-)
```

Now if you checkout to the main branch you will see that the changes you did in the file are not reflected in the main branch .

Because the changes are only exclusive to that branch only .

```
## demo
hello this is readme file
this is another line
this is line 2
this is line 3
hello this is the change i am making to this repo .
~
~
~
~
~
~
~
```

Now if you want that changes to be available in the main branch to . And then you have to merge the feature branch to the master branch .

Now if you want to check what changes we did in the feature branch we can use **git diff [name of the repo you did the change]** as compared to main branch

The red line indicates the change .that are present in the feature branch but not in the main branch .

```
yashrawat@Yashs-MacBook-Air testting_repo % git diff feature_updated_files
diff --git a/readme.md b/readme.md
index f44353d..a78f8fd 100644
--- a/readme.md
+++ b/readme.md
@@ -4,7 +4,3 @@ this is another line
 this is line 2
 this is line 3
 hello this is the change i am making to this repo .
-
-## developemt enviroment
-hello i am writing this new branch .
-hello i am wrting it again in the feature branch only
diff --git a/readme1.txt b/readme1.txt
index abc8f2c..cc17906 100644
--- a/readme1.txt
+++ b/readme1.txt
@@ -1,3 +1 @@
-Hello this is the text to test the readme file .
-
-Hello I am writing this new feature .
\ No newline at end of file
+Hello this is the text to test the readme file .
\ No newline at end of file
yashrawat@Yashs-MacBook-Air testting_repo %
```

And the way to merge this feature branch into the main branch we use **git merge {the branch you want to merge }**

But in almost all the practical case this is not the way you push these changes to the main branch .in general in whatever organisation you are working you will first review those changes and when everything looks good after that you will merge them .

So what we are going to do is

1. You will switch to feature branch using git checkout feature command
2. And then you checked the git status to check if there is anything you need to commit .
3. And then you used git push -u origin [name of the feature branch]

Using -u in git push **-u** origin feature_updated_files is like telling Git, "Remember this connection between my local branch and the remote branch on the

```
yashrawat@Yashs-MacBook-Air testting_repo % git checkout feature_updated_files
Switched to branch 'feature_updated_files'
yashrawat@Yashs-MacBook-Air testting_repo % git status
On branch feature_updated_files
nothing to commit, working tree clean
yashrawat@Yashs-MacBook-Air testting_repo % git push -u origin feature_updated_files
Enumerating objects: 7, done.
Counting objects: 100% (7/7), done.
Delta compression using up to 8 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 499 bytes | 499.00 KiB/s, done.
Total 4 (delta 0), reused 0 (delta 0), pack-reused 0
remote:
remote: Create a pull request for 'feature_updated_files' on GitHub by visiting:
remote:      https://github.com/thinkofyashh/testting_repo/pull/new/feature_updated_files
remote:
To https://github.com/thinkofyashh/testting_repo
 * [new branch]      feature_updated_files -> feature_updated_files
branch 'feature_updated_files' set up to track 'origin/feature_updated_files'.
yashrawat@Yashs-MacBook-Air testting_repo %
```

server." It makes future interactions more straightforward by setting up a shortcut.and now next time you wanna push to this branch you just have to use git push.

As you see in the upper image you will see that it is suggested us to pull our request (commonly know as PR)

```
Create a pull request for 'feature_updated_files' on GitHub by visiting:
     https://github.com/thinkofyashh/testting_repo/pull/new/feature_updated_files
```

PULL REQUEST

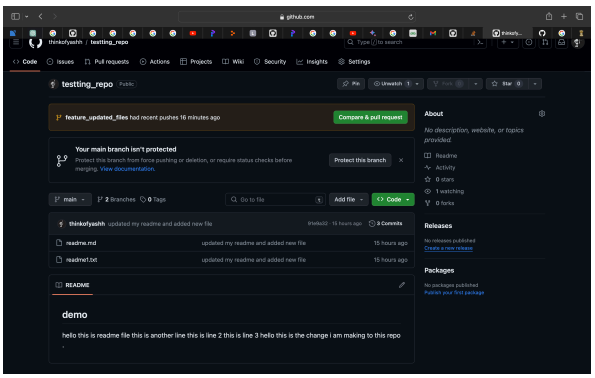
A pull request (PR) is a way to propose changes to a code repository in Git. Here's a simple breakdown

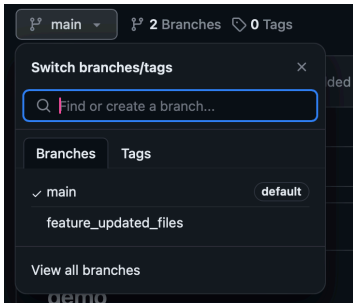
- Branch Creation:
  - You create a new branch in your repository to work on a new feature or fix a bug. This is often called a feature branch.
- Commits:
  - You make changes and commit them to your feature branch. Each commit represents a set of changes.
- Push:
  - You push your feature branch to the remote repository (like GitHub, GitLab, or Bitbucket).
- Pull Request:
  - Once your branch is pushed, you open a pull request. This is a request to the repository owner to pull in your changes and merge them into the main or another branch.
- Discussion and Review:
  - Others can review your code, leave comments, and discuss the proposed changes within the pull request. It's a collaborative space for feedback.
- Merge:
  - If everyone agrees that the changes look good, the repository owner (or someone with the right permissions) can merge your changes into the main branch.
- Closure:
  - After the merge, your feature branch can be deleted, and your changes are now part of the main branch.

In summary, a pull request is a mechanism for proposing, reviewing, and discussing changes before they are merged into the main codebase. It adds a layer of collaboration and ensures that code changes are well-vetted before becoming a permanent part of the project.

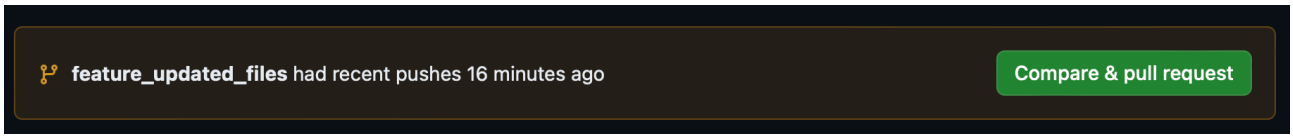
Now if I go to my GitHub repo -

And if you click on main you will see other branches .

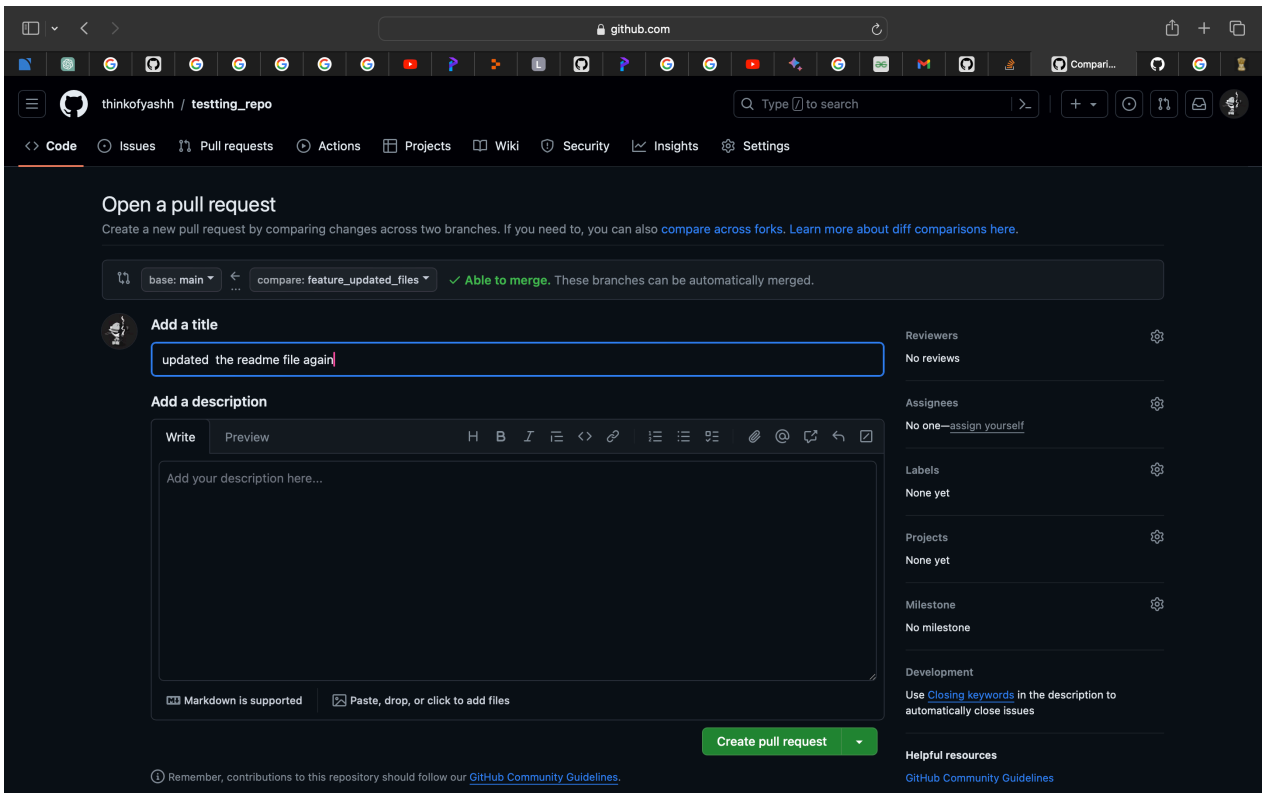




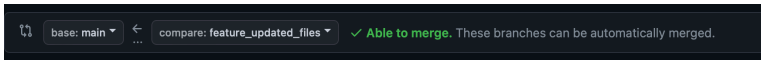
As you can see that GitHub as already identified that we have created a new branch .



Now if you click on it . You will see -



Indicates that feature_updated_files needs to be merged into the main branch



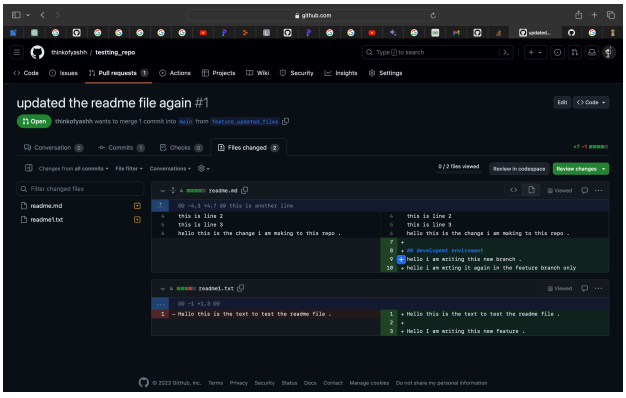
Add a title - it shows us the commit message

Add a description - while PR you should have to add good description which may include -
Q. What PR sands for ? Basically what are the things you added, what are the things you subtracted .

Once you have done that you can click on create pull request .

And since we are the owner of the repo we do have the access to merge it as well.

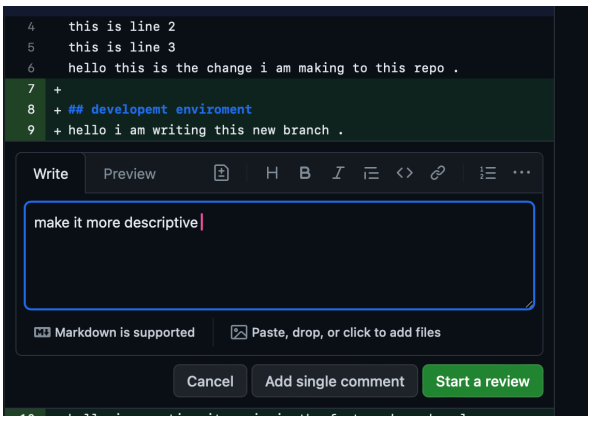
But generally when you create pull request . You have multiple people collaborating on that pull request . People can comeback with certain updates . They can ask you to make some changes as well .



People can look at your code . And review it like -

And then they click on start a review .

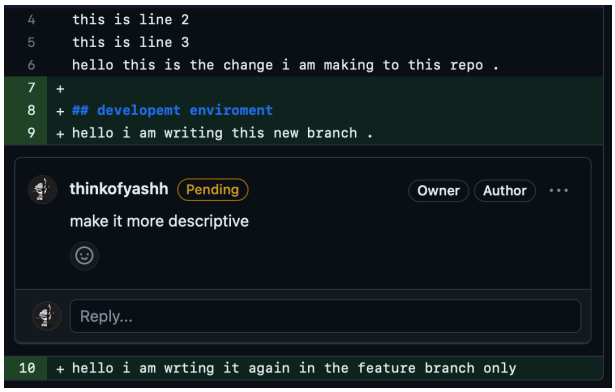
And then they can start the review at the specific line of code in your code .



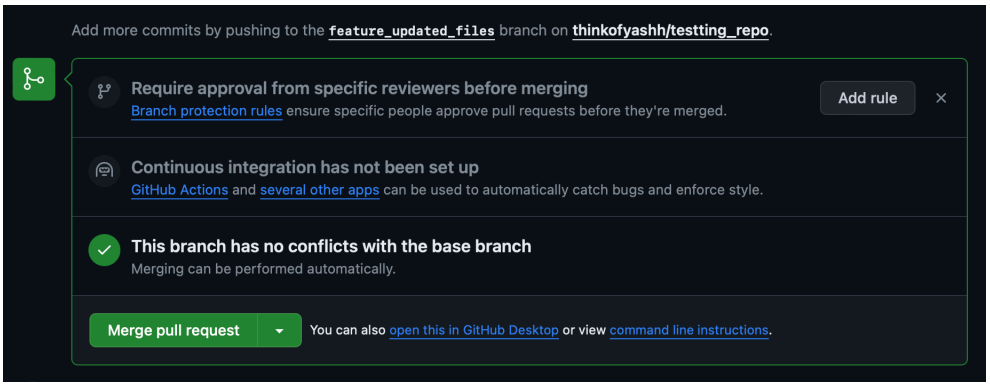
With this someone can review your PR and help you to get your PR more improved .

And that's how generally PR works .

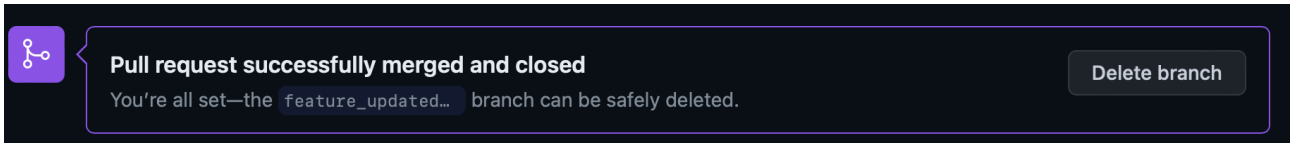
And after you done and reviewed what other are telling about your code .



Now lets see how we merge it . Since we are the owner of the repo we can also merge it by



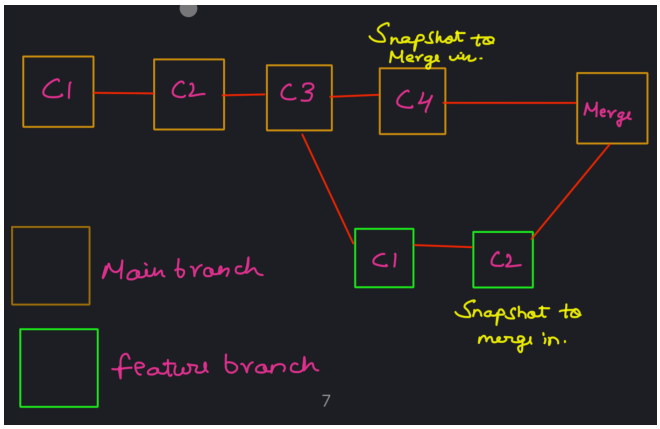
After merging it you will see -



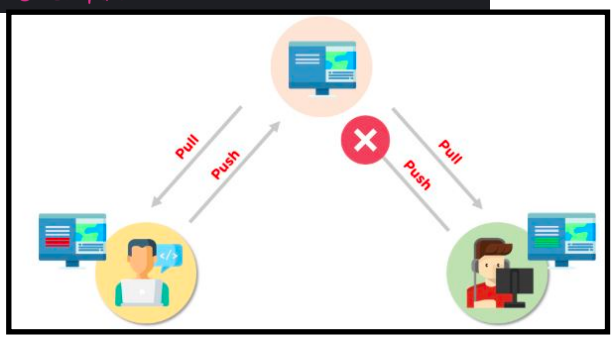
And now you refresh it you will see all the changes you have done in the main branch are also reflected in the main branch as well .

NOTE - in practical use case it is not going to be that simple . When you are working on the project . There are chances that master branch is constantly keeps on updating by multiple people .

GIT MERGE CONFLICT

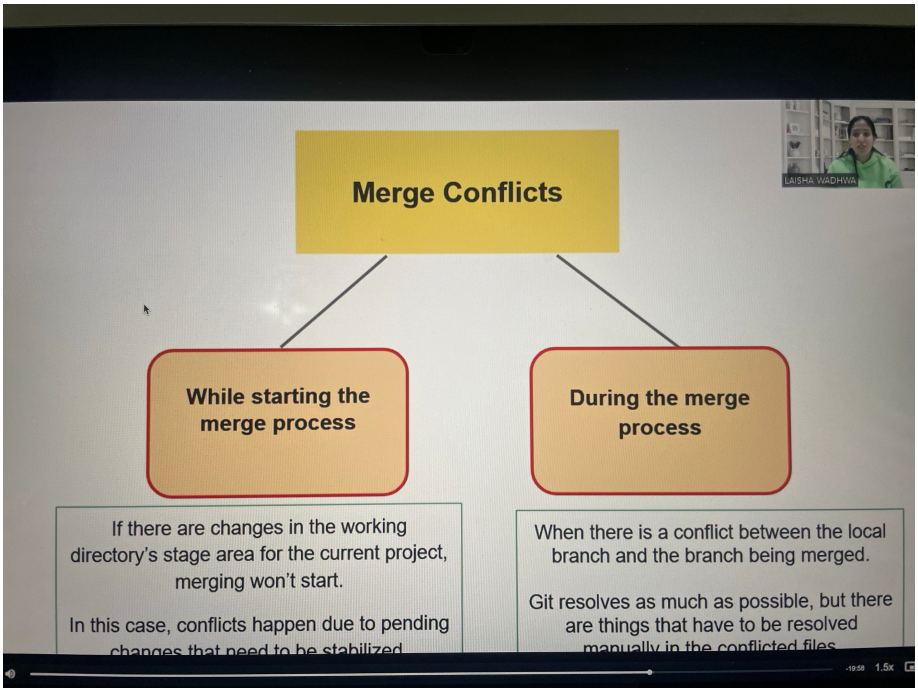


- Currently if we see Image you will see that the new Commit has been added to the Main branch before we initialised the new feature branch
- And what actually happens when we internally merge two branches.
  - ↳ It is going to look at the common ancestors (It is just the common commit).
  - ↳ Compare the two branch using git diff Command.
  - ↳ And if there is no file which is common across both of the branches where there is a Common change. Then its Simply going to Merge those two branches together. (By taking the change from feature branch and Merging them into Main branch).
  - ↳ It is Simple unless there is a file which is Common in both the branch that has been changed in both Master/Main branch and the feature branch. And that's create a Conflict.



A Git merge conflict happens when different branches change the same part of a file. Git can't automatically decide which changes to keep, so it marks the conflict. You need to manually choose and edit the conflicting parts, then commit the resolution.

Types of conflict-



MERGE CONFLICT

Occasionally this process doesn't go smoothly .

If you changed the same part of the same file differently in the two branches you're merging Git wouldn't be able to. Merge them cleanly . If your fix for feature branch modified the same part of a file as the hotfix branch you will get a merge conflict that looks something like this :

```
git merge iss53
Auto-merging index.html
CONFLICT (content): Merge conflict in index.html
Automatic merge failed; fix conflicts and then commit the
result.
```

Git hasn't automatically created a new merge commit. It has paused the process while you resolve the conflict. If you want to see which files are unmerged at any point after a merge conflict, you can run git status:

```
$ git status
On branch master
You have unmerged paths.
  (fix conflicts and run "git commit")

Unmerged paths:
  (use "git add <file>..." to mark resolution)

    both modified:      index.html

no changes added to commit (use "git add" and/or "git commit
-a")
```

Anything that has merge conflicts and hasn't been resolved is listed -

manually and resolve those conflicts. Your file contains
thing like this:

```
<<<<<<< HEAD:index.html
<div id="footer">contact : email.support@github.com</div>
=======
<div id="footer">
 please contact us at support@github.com
</div>
>>>>>>> feature_branch:index.html
```

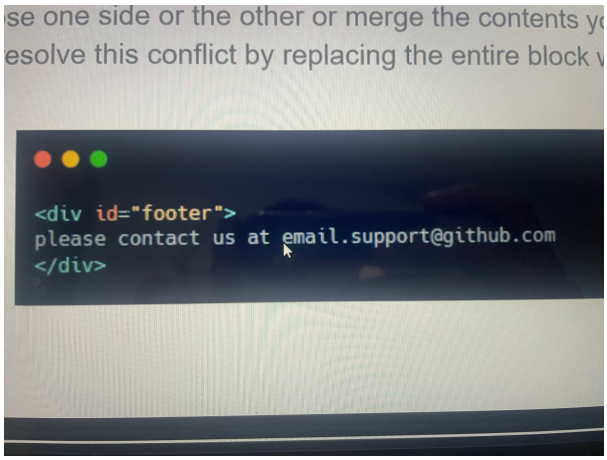
The conflict resolution markers are special annotations that Git places in a file to indicate areas where conflicts occurred during a merge. These markers help identify conflicting changes made in different branches. The standard conflict markers include:

- <<<<<<< HEAD: Marks the beginning of the conflicting changes from the current branch (HEAD).

- =======: Separates the conflicting changes from the changes in the branch being merged.
- >>>>>>> branch-name: Marks the end of the conflicting changes from the branch being merged.

When resolving conflicts, you need to manually edit the file, removing the markers and deciding which changes to keep. After resolving conflicts, you use git add to stage the changes and continue with the merge process.

In order to resolve the conflict you have to either choose one side or the other or merge the contents yourself . For instance you might resolve this conflict by replacing the entire block with this -



Some useful commands to resolve git conflict -

Git log - - merge :

The git log --merge command is used to display a commit history that includes information about merge commits. This command shows a concise log of commits, emphasizing the branches and merges in the commit history.

Here's a breakdown of what the git log --merge command does:

- git log: This command is used to display the commit history of a Git repository.
- --merge: This option filters the log to include only merge commits. Merge commits are commits that result from merging two branches.

By using git log --merge, you get a view of the commit history that highlights the points where branches were merged. It helps you understand the structure of your project's development and how different branches were integrated over time.

Git diff

The git diff command is used to show the differences between various states of a Git repository. It allows you to see changes that have been made to files, whether they are in your working directory, staging area, or committed in the repository. (Has multiple functions)

Git reset -
Used at the time of merge conflict to reset the conflicted files to their original state .

Git merge - - abort -

Helps in exiting the merge process and returning back to the state before the merging began .

Git reset - - mixed -

Used to undo changes to the working directory and staging area .

DELETING THE BRANCH

Deleting the branch which you no longer need is a good practice .for example our feature branch is already merged into the main branch .

Step 1 - For that go to the main branch first .

```
yashrawat@Yashs-MacBook-Air testting_repo % git checkout main
Switched to branch 'main'
Your branch is up to date with 'origin/main'.
yashrawat@Yashs-MacBook-Air testting_repo % git branch
  feature_updated_files
* main
yashrawat@Yashs-MacBook-Air testting_repo %
```

Step 2 - delete the branch using the command name - **git branch -d [name of the branch you wanna delete]**

```
yashrawat@Yashs-MacBook-Air testting_repo % git branch -d feature_updated_files
warning: deleting branch 'feature_updated_files' that has been merged to
         'refs/remotes/origin/feature_updated_files', but not yet merged to HEAD.
Deleted branch feature_updated_files (was f0edec5).
yashrawat@Yashs-MacBook-Air testting_repo % git branch
* main
yashrawat@Yashs-MacBook-Air testting_repo %
```

Now lets deep dive into conflicts -

Now lets add an example that you are working on the branch of your choice and there are some fixes that are done on the main branch and you are also working on the similar file .and now you want to push your changes from your current branch . You need to make sure that your branch and master branch are in sync but when you try to do that there are conflicts that come up . So lets create that scenario -

Lets create a new branch first -

```
yashrawat@Yashs-MacBook-Air testting_repo % git checkout -b hotfix
Switched to a new branch 'hotfix'
yashrawat@Yashs-MacBook-Air testting_repo %
```

We created a branch and then did some changes in readme.md file and then check the git status

```
yashrawat@Yashs-MacBook-Air testting_repo % vi readme.md
yashrawat@Yashs-MacBook-Air testting_repo % git status
On branch hotfix
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

no changes added to commit (use "git add" and/or "git commit -a")
yashrawat@Yashs-MacBook-Air testting_repo %
```

and then we commit those changes in the branch name hotfix with the message creating the conflict

```
yashrawat@Yashs-MacBook-Air testting_repo % git commit -am "creating the conflict "
[hotfix 368b7c3] creating the conflict
 1 file changed, 1 insertion(+)
yashrawat@Yashs-MacBook-Air testting_repo %
```

Now lets move to the main branch and some changes to the same file we did in the hot fix branch

```
yashrawat@Yashs-MacBook-Air testting_repo % git checkout main
Switched to branch 'main'
Your branch is up to date with 'origin/main'.
yashrawat@Yashs-MacBook-Air testting_repo % ls
readme.md       readme1.txt
yashrawat@Yashs-MacBook-Air testting_repo % vi readme.md
yashrawat@Yashs-MacBook-Air testting_repo %
```

Commit those changes in the main branch

```
yashrawat@Yashs-MacBook-Air testting_repo % git commit -am "updated the same file from the main branch ."
[main 1b1c3eb] updated the same file from the main branch .
 1 file changed, 1 insertion(+)
yashrawat@Yashs-MacBook-Air testting_repo %
```

Now lets move back to the branch we created for example hotfix

```
yashrawat@Yashs-MacBook-Air testting_repo % git checkout hotfix
Switched to branch 'hotfix'
yashrawat@Yashs-MacBook-Air testting_repo %
```

And then try merging this branch with the main branch using the command name **git merge main**

```
yashrawat@Yashs-MacBook-Air testting_repo % git merge main
Auto-merging readme.md
CONFLICT (content): Merge conflict in readme.md
Automatic merge failed; fix conflicts and then commit the result.
yashrawat@Yashs-MacBook-Air testting_repo %
```

Now here is the **CONFLICT t**hat is generated while you merge the file .

We know that auto merging is failed due to the readme.md file . So now we have to do some manually intervention and then you can fix those changes from the GitHub ui or you can fix it with any editor of your choice . The best way is go to the file and fix it .because we know that auto merging is failed due to readme.md file .

So lets open that file in the vi .

```
yashrawat@Yashs-MacBook-Air testting_repo % vi readme.md
```

Head is the branch you are currently in (hotfix) and the main is the brach from which the conflict is arising . And (=====) these are the separation between both the conflict .

```
## demo
hello this is readme file
this is another line
this is line 2
this is line 3
hello this is the change i am making to this repo .
<<<<<<< HEAD
hello checkin for a conflict .
=======
hello this changes i am making from the main branch to produce the conflict .
>>>>>>> main
~
~
~
```

So we can manually decide whether to keep one or both .

```
## demo
hello this is readme file
this is another line
this is line 2
this is line 3
hello this is the change i am making to this repo .
hello checkin for a conflict .
hello this changes i am making from the main branch to produce the conflict .
~
~
```

Using git diff to check the changes -

```
yashrawat@Yashs-MacBook-Air testting_repo % git diff
diff --cc readme.md
index f336a0a,12c4d9e..0000000
--- a/readme.md
+++ b/readme.md
@@@ -4,4 -4,4 +4,5 @@@ this is another lin
  this is line 2
  this is line 3
  hello this is the change i am making to this repo .
 +hello checkin for a conflict .
+ hello this changes i am making from the main branch to produce the conflict .
yashrawat@Yashs-MacBook-Air testting_repo % git diff main
diff --git a/readme.md b/readme.md
index 12c4d9e..1818170 100644
--- a/readme.md
+++ b/readme.md
@@ -4,4 +4,5 @@ this is another line
 this is line 2
 this is line 3
 hello this is the change i am making to this repo .
+hello checkin for a conflict .
 hello this changes i am making from the main branch to produce the conflict .
yashrawat@Yashs-MacBook-Air testting_repo % git diff hotfix
diff --git a/readme.md b/readme.md
index f336a0a..1818170 100644
--- a/readme.md
+++ b/readme.md
@@ -5,3 +5,4 @@ this is line 2
 this is line 3
 hello this is the change i am making to this repo .
 hello checkin for a conflict .
+hello this changes i am making from the main branch to produce the conflict .
yashrawat@Yashs-MacBook-Air testting_repo %
```

And then commit these changes from the hotfix branch with the message - resolved merge conflict

```
yashrawat@Yashs-MacBook-Air testting_repo % git commit -am "resolved merge conflict"
[hotfix 578da8a] resolved merge conflict
yashrawat@Yashs-MacBook-Air testting_repo %
```

And now if you are going to merge

```
yashrawat@Yashs-MacBook-Air testting_repo % git merge main
Already up to date.
yashrawat@Yashs-MacBook-Air testting_repo %
```

And that's how you resolve conflicts .

Q. What are issues in Git Hub ?

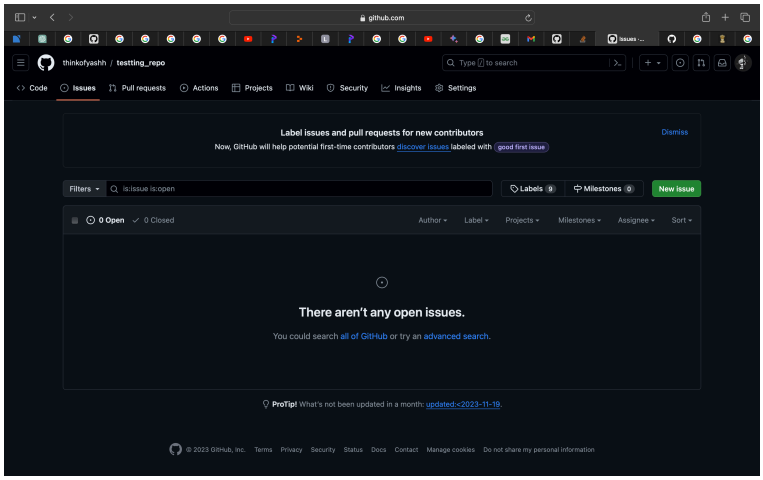
GitHub issues are a feature that allows you to track and manage tasks, enhancements, bugs, and other kinds of questions or discussions on your repositories. They serve as a communication tool

between project contributors, and users can open new issues, comment on existing ones, and close issues when the task is completed. Here are some common types of issues on GitHub:

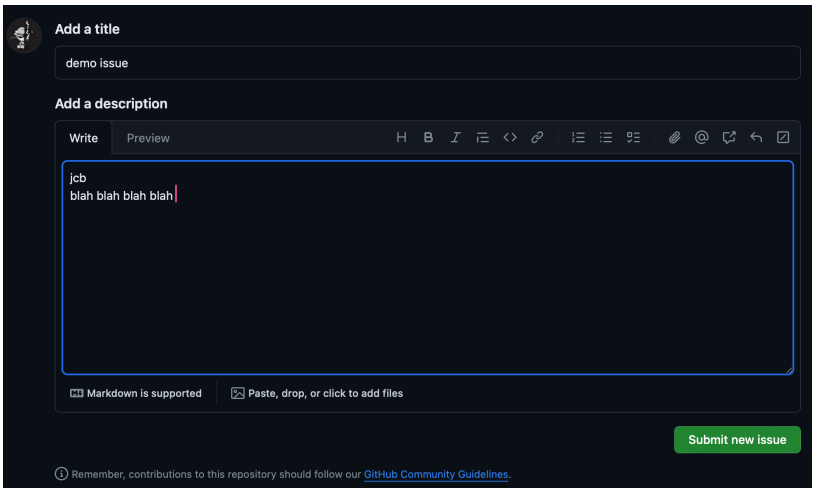
- Bug Report:
  - Users report problems or unexpected behavior they have encountered in the software. This helps developers identify and fix issues.
- Feature Request:
  - Users suggest new features or enhancements they'd like to see in the project. This can help project maintainers understand user needs.
- Documentation:
  - Issues related to documentation improvements or errors. Users may point out unclear instructions or propose changes to the documentation.
- Question:
  - Users may have questions about how to use a certain feature, clarification on documentation, or general inquiries about the project.
- Enhancement:
  - Similar to feature requests, enhancements are suggestions to improve existing features or make them more efficient.
- Discussion:
  - Open-ended discussions on a particular topic related to the project. This can be a place for brainstorming or planning new features.
- Help Wanted / Good First Issue:
  - Projects often tag issues as "help wanted" or "good first issue" to invite new contributors to the project. These are typically tasks that are considered suitable for beginners.

GitHub issues provide transparency and a structured way for teams to collaborate on software projects. They help manage the development process by keeping track of tasks, bugs, and improvements.

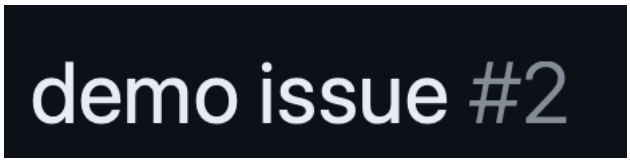
To create a new issues click on it .



Click on submit issue .



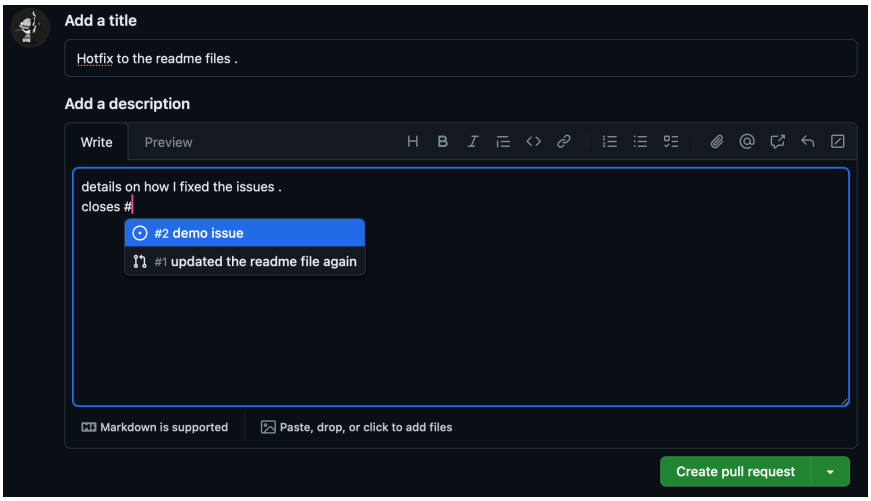
After you created the issue . It is assigned to a number .



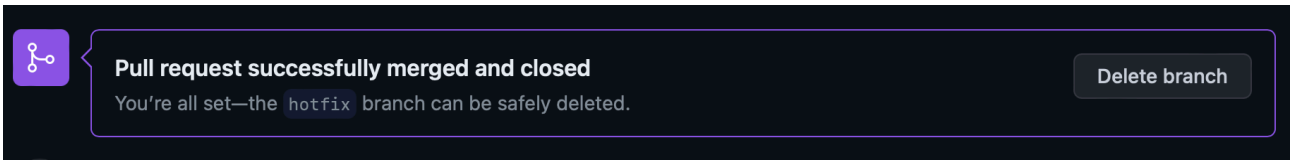
Suppose you try to solve the issue . So you need to pull the request closing the issue like -

First make a pull request then click on it -

And then -



And then click on create pull request . So finally you are able to solve the issues .



Now the issue will automatically be closed .