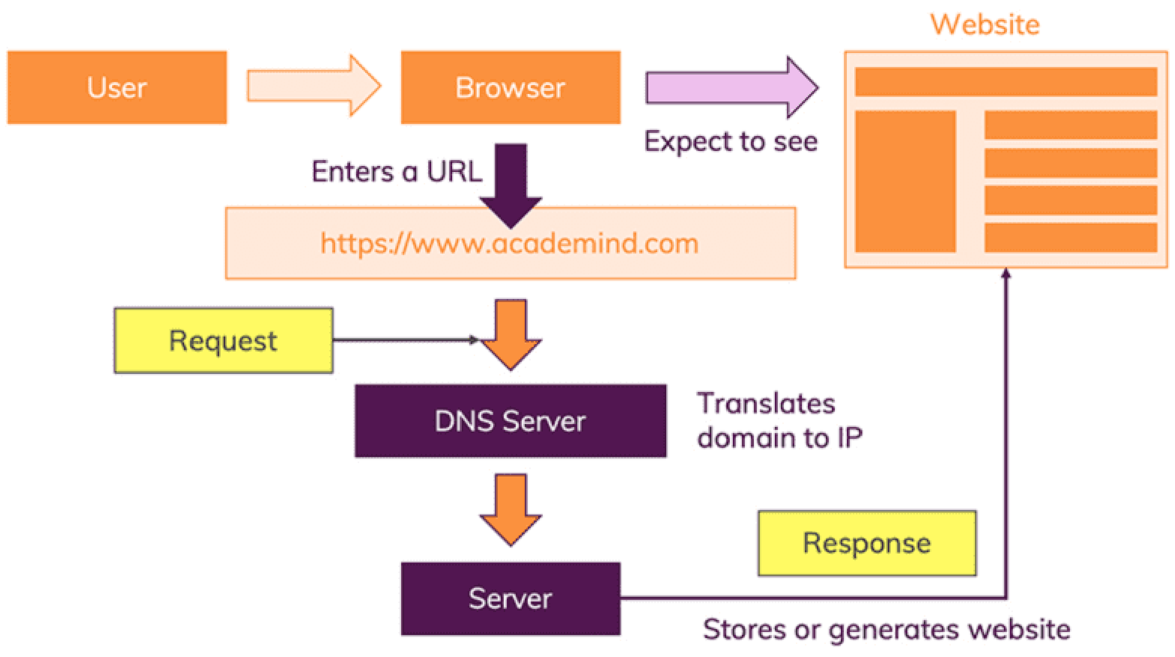
How does a website work :

The moment you enter this address in your browser and you hit ENTER, a lot of different things happen:

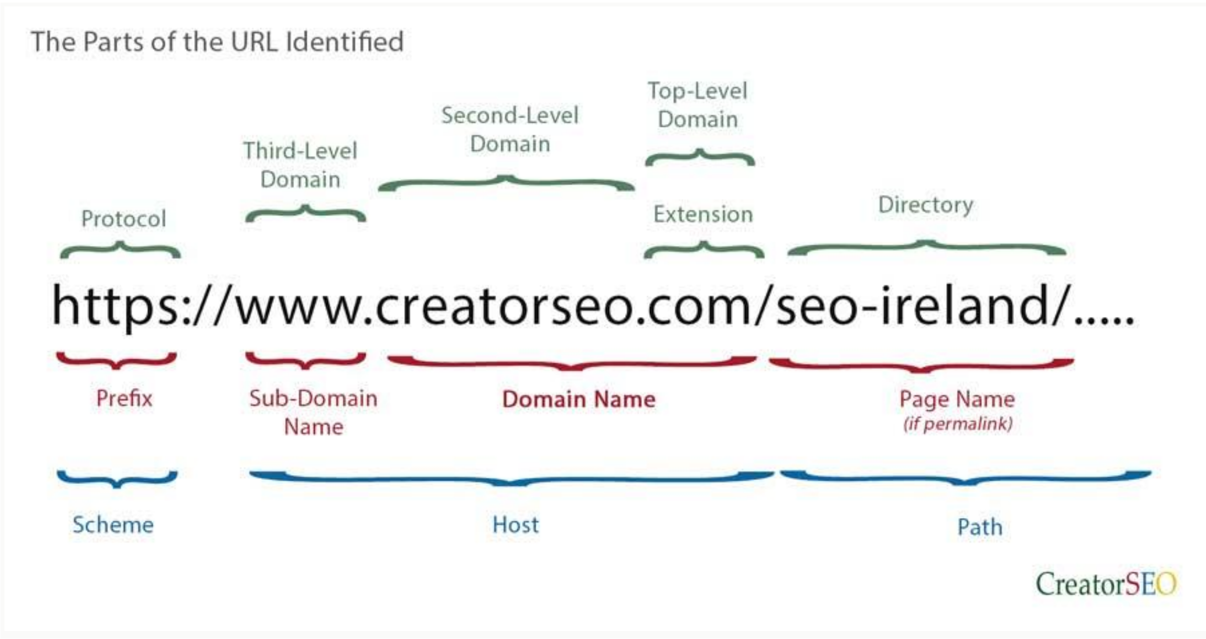


1. The URL gets resolved - Uniform Resource Locator.
   The website code is obviously not stored on your machine and hence needs to be fetched from another computer where it is stored. This "other computer" is called a "server". Because it serves some purpose, in our case, it serves the website.

   Lets say, You enter "google.com" (that is called "a domain") but actually, the server which hosts the source code of a website, is identified via IP (= Internet Protocol) addresses. The browser sends a "request" to the server with the IP address you entered (indirectly - you of course entered "google.com").

   Now, I would like to discuss two terminologies :

   - Domain : A **domain** name is a string of text that maps to a numeric IP address, used to access a website from client software.



- IP Address : An **IP address** is a unique address that identifies a device on the internet or a local network. IP stands for "Internet Protocol,"
  IP Addresses are assigned to a computer network and not a device. MAC address is unique to a device.

Now, The question arises : How is the domain "google.com" translated to its IP address?

There's a special type of server out there on the internet - not just one but many servers of that type. A so-called "name server" or "DNS server" (where DNS = "Domain Name System").

The job of these DNS servers is to translate domains to IP addresses. You can imagine those servers as huge dictionaries that store translation tables: **Domain => IP address**.

When you enter "google.com", the browser therefore first fetches the IP address from such a DNS server.

DNS Servers are located in ISPs(Internet service provider) data centres. If a mapping is not found it calls other DNS on the internet.

Once, Ip address is found we move to the next step.

2. A Request is sent to the server of the website

With the IP address resolved, the browser goes ahead and makes **a request** to the server with that IP address.

"A request" is not just a term. It really is a technical process that happens behind the scenes. We will discuss it later in the course.

The browser bundles up a bunch of information (What's the exact URL? Which kind of request should be made? Should metadata be attached) and send that data package to the IP address.

The data is sent via the "HyperText Transfer Protocol" (known as "HTTP") - a standardised protocol which defines what a request (and response) has to look like, which data may be included (and in which form) and how the request will be submitted

The server then handles the request appropriately and returns a so-called "response". Again, a "response" is a technical thing and kind of similar to a "request". You could say it's basically a "request" in the opposite direction.

Like a request, a response can contain data, metadata etc. When requesting a page like academind.com, the response will contain the code that is required to render the page onto the screen.

3. The response of the server is parsed

The browser receives the response sent by the server. This alone doesn't display anything on the screen though.

Instead, the next step is that the browser parses the response. Just as the server did it with the request. Again, the standardisation enforced by HTTP helps of course.

The browser checks the data and metadata that's enclosed in the response. And based on that, it decides what to do.

In that case, the response would contain a specific piece of metadata, that tells the browser that the response data is of type `tex`

This allows the browser to then parse the actual data that's attached to the response as HTML code.

HTML is the core "programming language" (technically, it's not a programming language - you can't write any logic with it) of the web. HTML stands for "Hyper Text Markup Language" and it describes the structure of a webpage.

The browser knows how to parse HTML and now simply goes through the entire response data (also called "the response body") to render the website.

4. The page is rendered and displayed

As mentioned, the browser goes through the HTML data returned by the server and builds a website based on that.

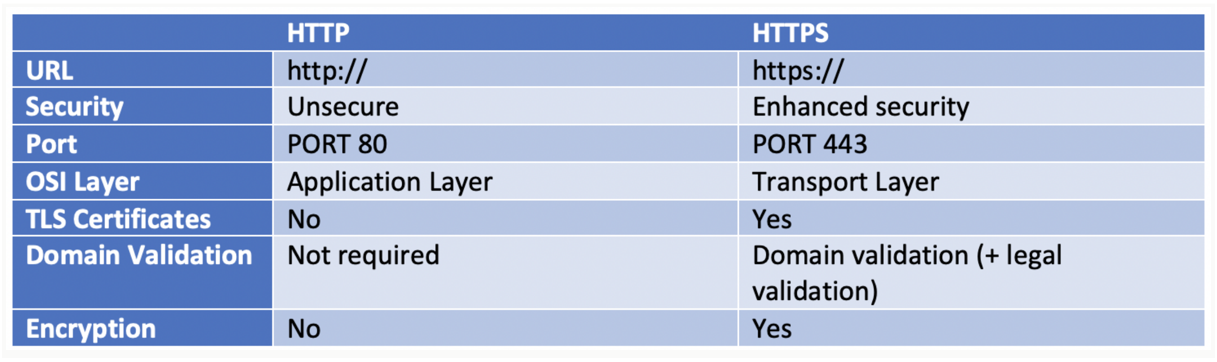
So, a website is displayed, But there are two other things which are part of this website.
It's css, and javascript. We will learn about them later in the course.

First let's talk about HTTP and HTTPS ;

HTTP: HyperText Transfer Protocol. These are a set of rules which allows us to communicate b/w different systems. Most commonly, between servers and browsers in order for us to view webpages.

HTTPS: (secure) - An HTTP request over a SSL(secure socket layer). It provides security and encryption data. Why ? PAyment, Data privacy etc.

SSL - IS standard security technology for establishing an encrypted link b/w server and client.



	HTTP	HTTPS
URL	http://	https://
Security	Unsecure	Enhanced security
Port	PORT 80	PORT 443
OSI Layer	Application Layer	Transport Layer
TLS Certificates	No	Yes
Domain Validation	Not required	Domain validation (+ legal validation)
Encryption	No	Yes

A port is **a virtual point where network connections start and end.**

Caching - In common language, caching means placing something in storage on the chance that it may come in useful later. A browser or Web cache does exactly that, except with program and website assets. When you visit a website, your browser takes pieces of the page and stores them on your computer's hard drive. Some of the assets your browser will store are:

- Images - logos, pictures, backgrounds, etc.
- HTML
- CSS
- JavaScript

In short, browsers typically cache what are known as "static assets" - parts of a website that do not change from visit to visit.

What to cache and for how long is determined by the website. Some assets are removed from your machine in a few days while others may remain in your cache for up to a year.

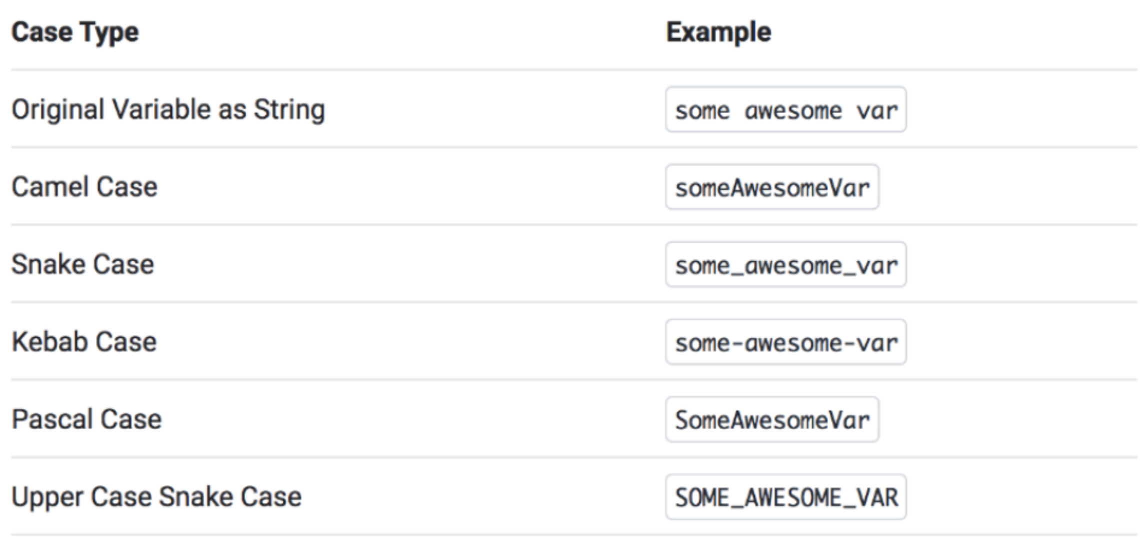
The benefits of caching :

As your browser reads the HTML code, it sends out more requests to the server to send more pieces of the page, mostly the static assets mentioned above. This process takes up bandwidth. Some Web pages will take a great deal of time to fully download and become functional because they have a lot of pieces or their assets are large.

- **Caching improves Speed at which the webpage loads. Once you've downloaded an asset, it lives (for a time) on your machine. Retrieving files from your hard drive will always be faster than retrieving them from a remote server, no matter how fast your Internet connection.**

- **Along with large images, complex sites use large JavaScript files - necessary for applications such as shopping carts, interactive images and wish lists.**

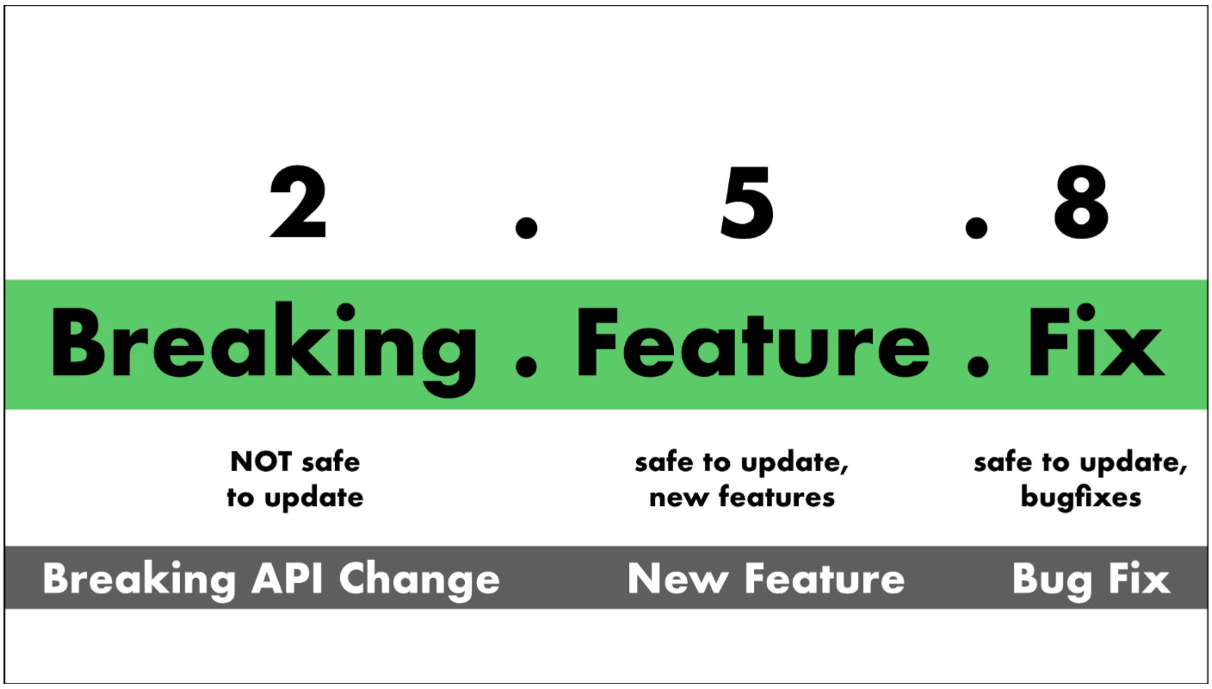
Some pitfalls : Old versions of cached files cause all sorts of issues for users if their devices don't have the latest version of the file - mismatched formatting, broken JavaScript and incorrect images are just a few.How does a website work :



Case Type	Example
Original Variable as String	`some awesome var`
Camel Case	`someAwesomeVar`
Snake Case	`some_awesome_var`
Kebab Case	`some-awesome-var`
Pascal Case	`SomeAwesomeVar`
Upper Case Snake Case	`SOME_AWESOME_VAR`

VERSION CONTROL :

Major, Minor, Patch. In general.



Version Control System (VCS) is a software that helps software developers to work together and maintain a complete history of their work.

Listed below are the functions of a VCS −

- Allows developers to work simultaneously.
- Does not allow overwriting each other’s changes.

- Maintains a history of every version.

Following are the types of VCS −

- Centralised version control system (CVCS).
- Distributed/Decentralised version control system (DVCS).

In a centralised version control system (CVCS), a server acts as the main repository which stores every version of code. Using centralised source control, every user commits directly to the main branch, so this type of version control often works well for small teams, because team members have the ability to communicate quickly so that no two developers want to work on the same piece of code simultaneously. Strong communication and collaboration are important to ensure a centralised workflow is successful.

Disadvantages :

A single point of failure risks data

Slow speed delays development

Users often have a difficult time branching quickly, because users must communicate with the remote server for every command, which slows down code.

Distributed Version Control System **Mercurial, Git and Bazaar.**

A distributed version control system (DVCS) is **a type of version control where the complete codebase — including its full version history** — is mirrored on every developer's computer. It's abbreviated DVCS. Changes to files are tracked between computers.

Git is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency.

Advantages of Git :

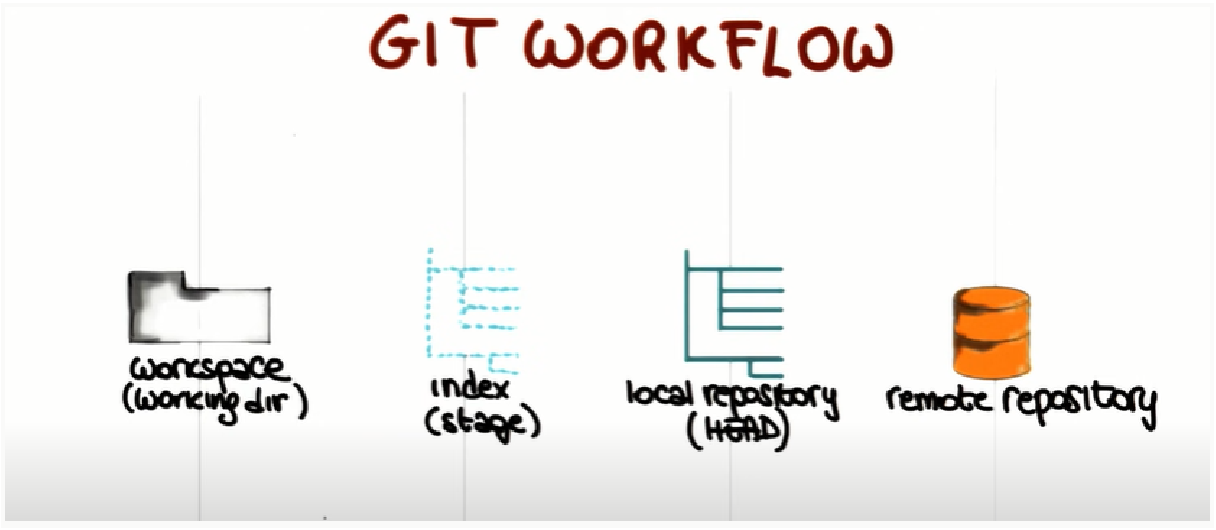
- Free and open source - **open source software** (OSS). Open source software is code that is designed to be publicly accessible
- **Fast & small - Fast as in it mirrors take care of changes not the entire thing. Small - Good quality Data compression is used**
- Implicit backup - Every developer has a compy

- Security - Uses SHA1 - secure hash function
- Scalable

Now, What is Github?

GitHub is a **code hosting platform for version control and collaboration**. It lets you and others work together on projects from anywhere.

In github we have a repository(remote),



Now, Let's talk about some terminologies :

1. Commits - Commit holds the current state of the repository. A commit is also named by SHA1 hash.
2. Branches - Branches are used to create another line of development. By default, Git has a master/main branch Usually, a branch is created to work on a new change. Once the change is completed, it is merged back with the master branch and we delete the branch.
3. Clone - Clone operation creates the instance of the repository. Clone operation not only checks out the working copy, but it also mirrors the complete repository.
4. Head - HEAD is a pointer, which always points to the latest commit in the branch. Whenever you make a commit, HEAD is updated with the latest commit.
5. Pull Request PR - A pull request occurs when a developer asks for changes committed to the remote repository.
6. Merge - The git merge command lets you take the independent lines of development created by the git branch and integrate them into a single branch.

7. Fork - A fork is **a rough copy of a repository**. Forking a repository allows you to freely test and debug with changes without affecting the original project.

First time -

git config --global user.name "Your Name Here"
git config --global user.email your@email.com

Some Git commands :

- **Git init -** This command turns a directory into an empty Git repository. This is the first step in creating a repository. After running git init, adding and committing files/directories is possible**.**
- **Git clone** - To create a local working copy of an existing remote repository,
- **git add file_name/ git add . -** Adds files to the staging area for Git. Before a file is available to commit to a repository, the file needs to be added to the Git index (staging area)
- **Git commit** - Record the changes made to the files to a local repository. For easy reference, each commit has a unique ID. It's best practice to include a message with each commit explaining the changes made in a commit. Adding a commit message helps to find a particular change or understanding the changes.
- **Git status** - This command returns the current state of the repository.
- **Git remote -v** - Returns the remote url.
- **Git branch** - To determine what branch the local repository is on, add a new branch, or delete a branch.
- **Git checkout branch_name** - To start working in a different branch, use *git checkout* to switch branches.
- **Git checkout -b new_branch_name** - create a new branch and switch to that branch.
- **Git merge** - merge one branch to another.
- **Git pull** - This pulls the changes from the remote repository to the local computer.
- **Git push** - Sends local commits to the remote repository.

- **git push -u** option sets upstream, it basically says to git that link the local branch with the remote branch automatically.
- **Git stash/ Git stash pop** - To save changes made when they're not in a state to commit them to a repository
- **Git log** - To show the chronological commit history for a repository.
- **Git diff -** Show changes between commits