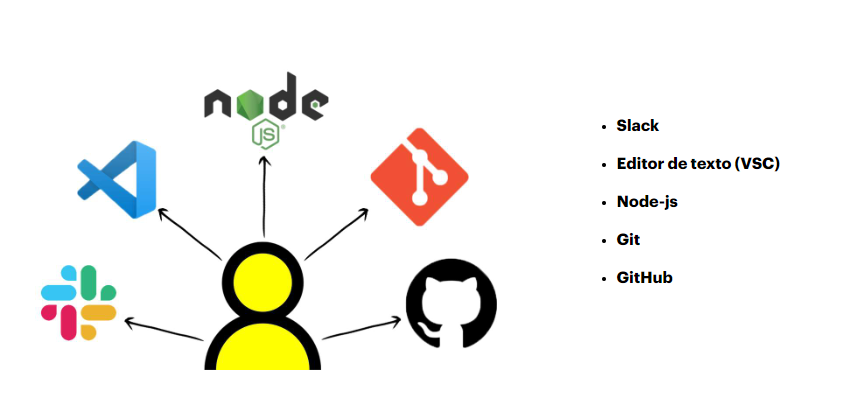
<https://rise.articulate.com/share/O0SjmUFcjx_3AXSA2lizESbHitgJIOpq#/lessons/LodEYjBDWszK3077zvPcuJrhbxQLx16G>

https://rise.articulate.com/share/O0SjmUFcjx\_3AXSA2lizESbHitgJIOpq#/lessons/KSire0WgNTOeIoTA2xlNr2A9136cqQvP

Recuerden unirse a cada charla de Zoom con el mismo correo con el que se inscribieron en Henry.  
Link de Zoom (siempre es el mismo)  : <https://us02web.zoom.us/j/86923049990>





 -Nos dirigimos al sitio oficial de [slack.com(opens in a new tab)](https://slack.com/)

-  Al ingresar, el sitio detectará nuestro Sistema Operativo y nos sugerirá que descarguemos el instalador adecuado. Para ello selecciona el menú **Producto,** luego **Descargar Slack.**

-Una vez finalizada la descarga, procedemos a ejecutar el instalador.  
- Abre la app de [Admissions(opens in a new tab)](https://www.admissions.soyhenry.com/" \t "_blank) y presiona en el link de invitacion a slack.

- Presiona sobre la opción de **seguir con google** y **elije la cuenta con la cual te registraste en el Henry Prep Course**. Si te pide un codigo, debes revisar tu correo para ver el mensaje que te mando Slack con el mismo.

-Te aparecerá una ventana emergente para permitir abrir la aplicacion de Slack. - Debes darle a la **opcion de abrir** y ya podrás estar comunicado con todos tus compañeros y miembros del staff.

**¿Qué es Node JS?**

Fue ideado como un **entorno de ejecución de javascript**, Node JS nos va a permitir ejecutar nuestro código, lo leerá y si algo anda mal, nos retornará lo que debemos corregir o en caso contrario, nos brindara el resultado de la ejecución de nuestro código JS.

**Pasos para instalar Node Js**

En el siguiente tutorial te explicamos paso a paso la instalación de Node Js.

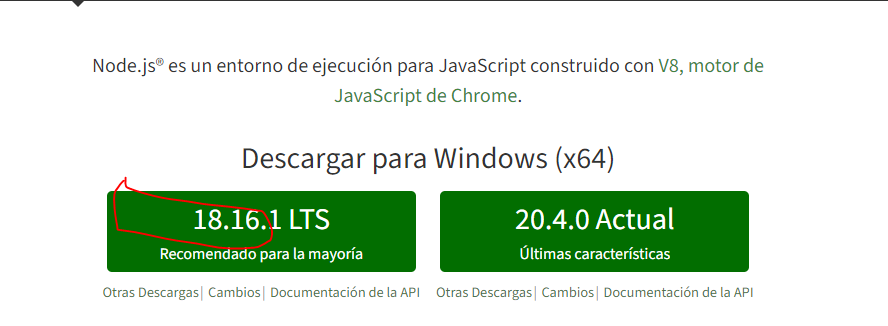
**Enlaces para para descargar Node JS...**

* 1

Nos dirigimos al sitio oficial: [https://nodejs.org/es/(opens in a new tab)](https://nodejs.org/es/)

* 2

Al ingresar, el sitio detectará nuestro Sistema Operativo y nos sugerirá que descarguemos el instalador adecuado.Para proceder a la descarga, seleccionamos la versión LTS, que es la versión estable.



* 3 Una vez finalizada la descarga, procedemos a ejecutar el instalador.
* 4 Para corroborar que Node js se instaló correctamente, procedemos a ejecutar el siguiente comando por la terminal de nuestro sistema operativo:

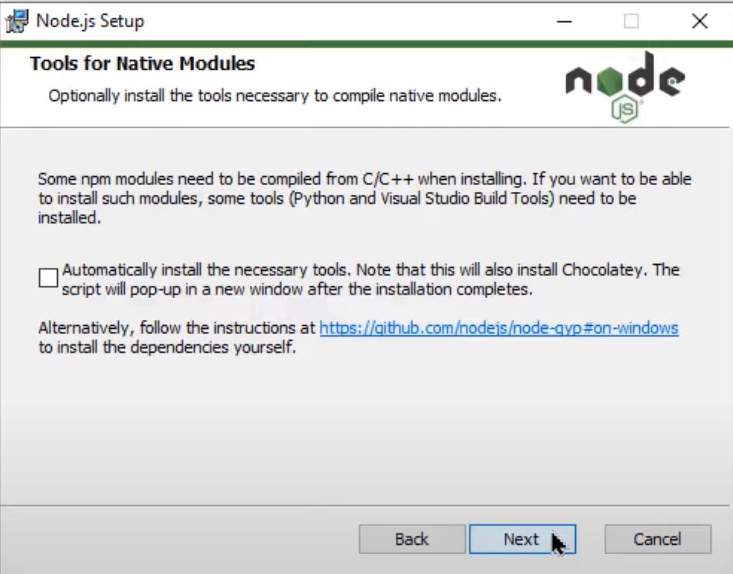
**node -v**

* 5 Luego de seguir los pasos anteriores, la terminal nos devolverá la versión de Node JS que tenemos instalada

**IMPORTANTE**

En caso de que, al instalar **Node.js**, encuentres una pestaña como las siguiente en la que pregunte si quieres instalar herramientas complementarias, simplemente has click en **next**.

¿No es necesario que instales ninguna dependencia o herramienta extra para Node!



Validar que se instalo bien:

C:\Users\mcampos>node -v

v18.16.1

**Pasos para instalar Visual Studio Code**

* Nos dirigimos al sitio oficial, en la sección Dowload y descargamos el instalador según nuestro Sistema Operativo: [https://code.visualstudio.com/download(opens in a new tab)](https://code.visualstudio.com/download)

Una vez finalizada la descarga, procedemos a ejecutar el instalador.

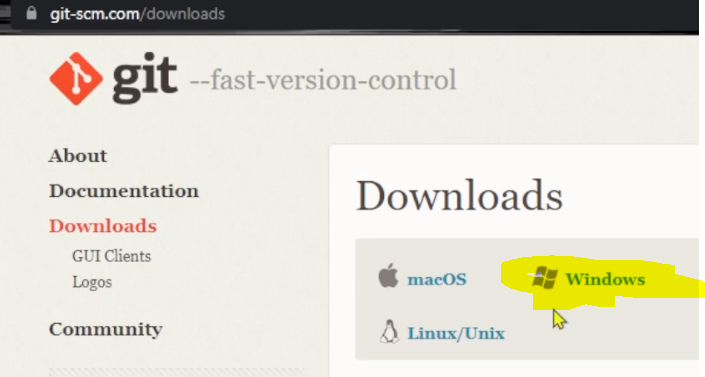
**¿Qué es Git?**

**Git**es un sistema de**control de versiones de código**, es decir; un sistema que registra los cambios realizados en un archivo o conjunto de archivos a lo largo del tiempo, de modo que puedas recuperar versiones específicas más adelante.

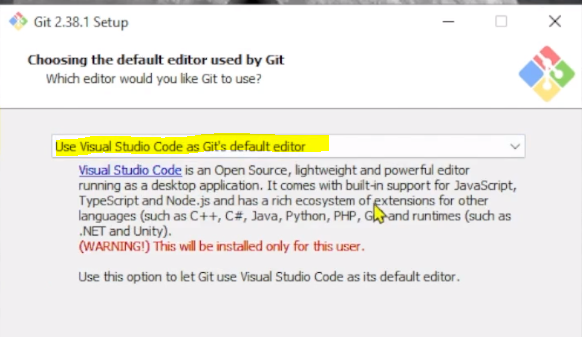
**Pasos para la instalación de Git**

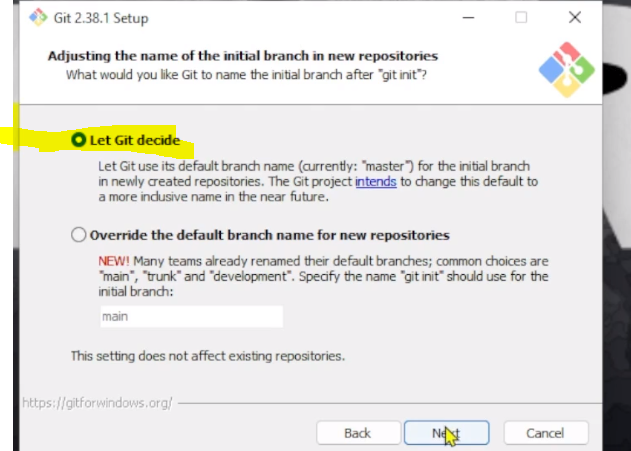
Para instalar **Git**es necesario **tener instalado Visual Studio Code previamente**(u otro editor de texto). En el siguiente tutorial te explicaremos cómo instalar y configurar correctamente **Git**en tu equipo.

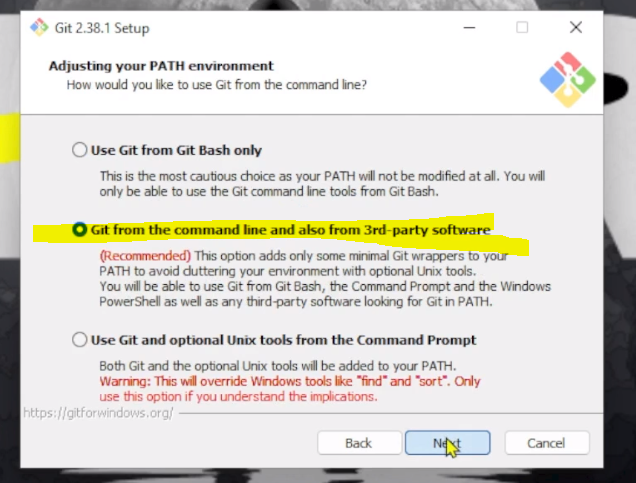
* 1 Desde Windows, ingresamos a [**https://git-scm.com**(opens in a new tab)](https://git-scm.com/) y descargamos la última versión.
* 2 Una vez descargado, ejecutar el archivo .exe y seguimos los pasos hasta la ventana donde elegimos el editor de texto que vamos a utilizar: "**Use Visual studio code as git default editor**".
* 3 Continuamos precionando “Next” y luego “Install”.
* 4 Por último, debes presionar finalizar. Si seleccionas la opción "**Launch Git Bash**", una vez que finalizan la instalación se va a abrir la terminal.

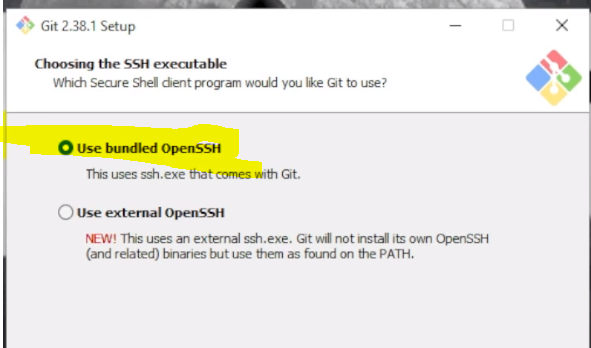


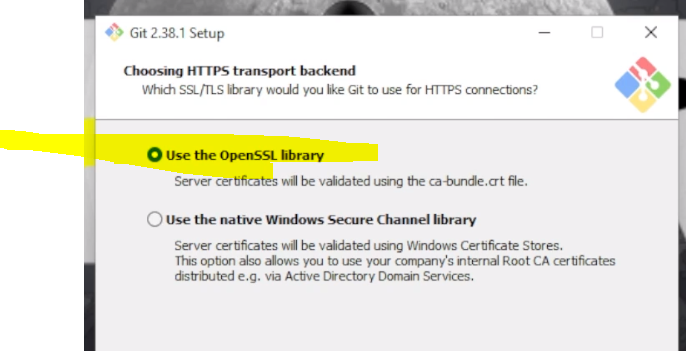


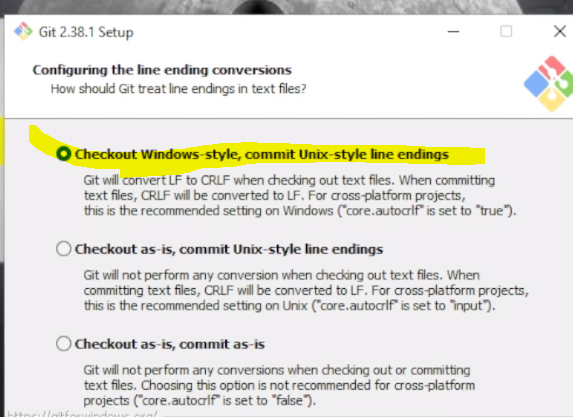


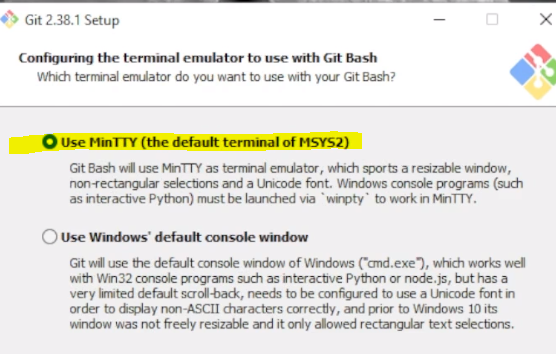


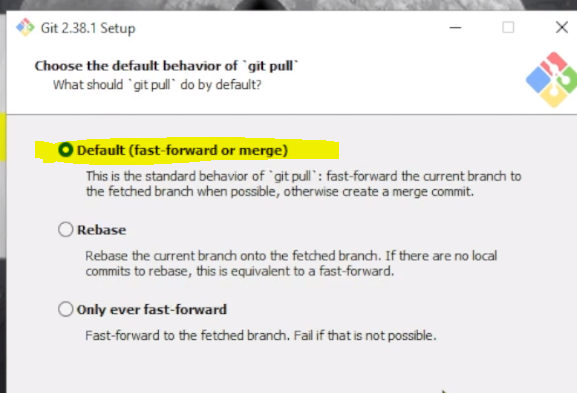


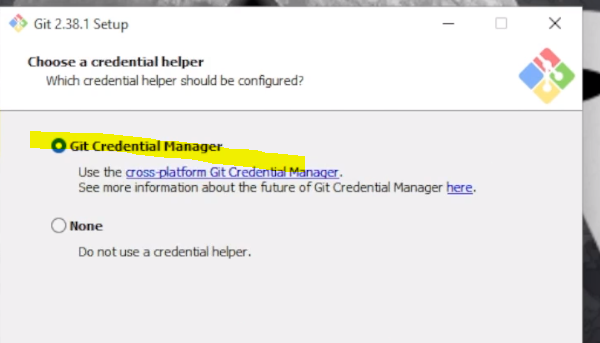


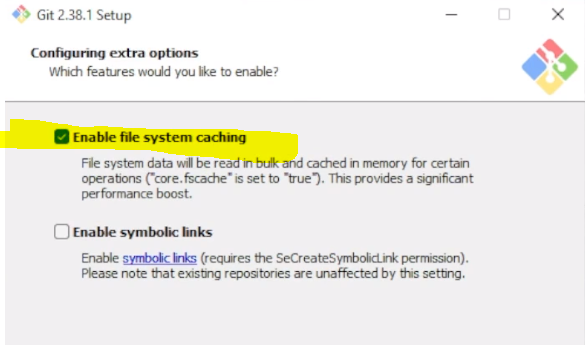


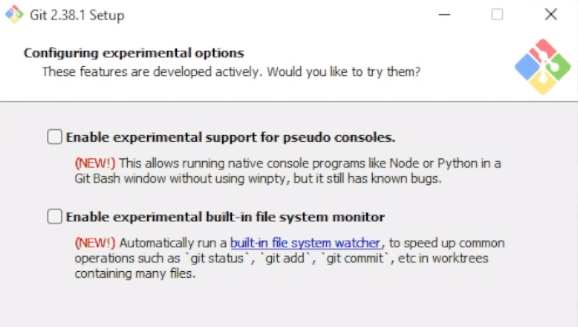












Para saber si esta instalado:

C:\Users\mcampos>git --version

git version 2.35.1.windows.2

**Github**

**¿Qué es?**

Con **Git** ya estamos controlando las versiones de nuestro código, pero también deseamos guardar este código en **internet** (en la nube) para poder compartirlo, y tener un respaldo de él.

Para cumplir esta función  existen diferentes herramientas como: gitLab, bitbucket o **Github**, nosotros recomendamos utilizar esta última. **Github**.com es una red para almacenar tus repositorios, esencialmente es un repositorio de repositorios, y actualmente es el más popular entre la comunidad de desarrolladores. Es importante aclarar que Git no es igual que Github, aunque funcionan muy bien juntos.

**¿Cuál es la diferencia entre Git y Github?**

La diferencia principal entre Git y GitHub es que Git es una herramienta de código abierto que los desarrolladores instalan localmente para gestionar el código fuente, mientras que GitHub es un**servicio en línea al que los desarrolladores que utilizan Git pueden conectarse y cargar o descargar recursos.**  
En resumen, Git es el sistema de control de versiones y GitHub es un servicio de alojamiento para los repositorios de Git.

**Para configurar Github...**

Como **Github**es un servicio en línea no necesitamos instalar ningún software localmente, únicamente necesitamos **crear nuestra cuenta en su web y configurarla,**para ello sigue los siguientes pasos:

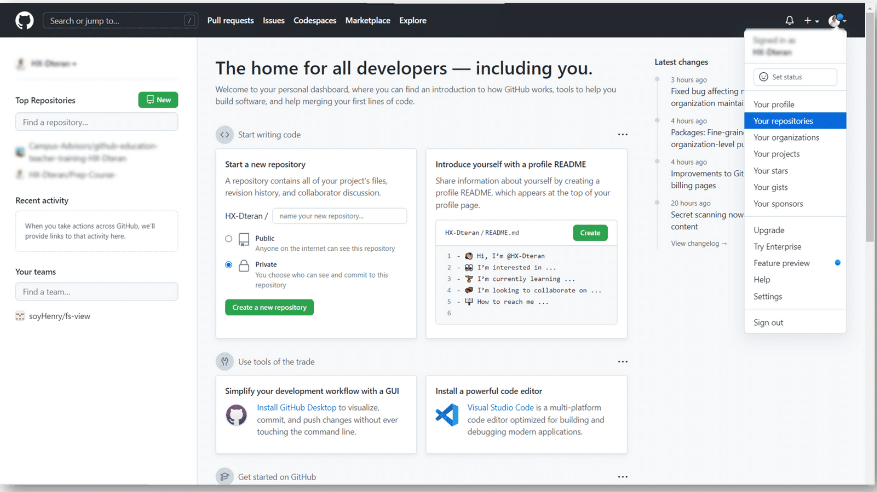
* 1 Ingresa a https://github.com
* 2 Crea una cuenta siguiendo los pasos e [**https://github.com/signup**(opens in a new tab)](https://github.com/signup), te recomendamos crear un nombre de usuario compuesto con tus iniciales junto con tu nombre y apellido real, por ejemplo *DMorenoTrejo*.

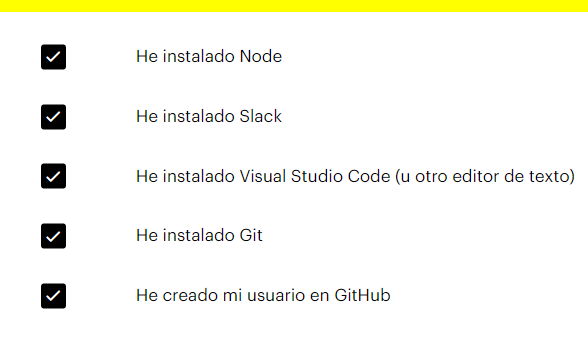
**Este usuario lo verán los reclutadores al mostrar tu portafolio. Te recomendamos elegir un nombre de usuario profesional y fácil de recordar para que sea  identificable en futuros proyectos y colaboraciones.**

3 Una vez registrado, ingresa con tu usuario y contraseña.

**Es importante mantener el nombre de usuario de GitHub una vez que lo creas. Cambiarlo puede afectar la visibilidad y reconocimiento de tu trabajo dentro de HENRY.**

* 4hora verás una página como la siguiente:





**Comandos para otros sistemas operativos (Linux y Mac)**

**Linux:**

1. Presiona Ctrl + Alt + T para abrir una terminal.
2. Presiona en el menú de aplicaciones y busca “Terminal” o “Consola”.
3. Usa el atajo de teclado específico de tu entorno de escritorio (por ejemplo, Ctrl + Alt + T en GNOME y Unity, Super + T en XFCE).

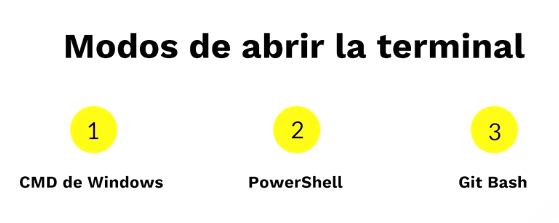
**Mac:**

1. Abre la carpeta “Aplicaciones” y luego abre la carpeta “Utilidades”. Haz doble clic en “Terminal”.
2. Usa Spotlight para buscar “Terminal” y presiona Enter.
3. Presiona Cmd + Espacio para abrir Spotlight y escribe “Terminal”, luego presiona Enter.

**¿Cómo cambiar la configuración de mi terminal?**

Como ya sabes, estaremos utilizando la terminal de **Git Bash** a lo largo del Prep Course. Uno de los beneficios que tiene el editor de texto Visual Studio Code es que nos permite integrar esta terminal allí mismo. Por defecto este editor de texto utiliza la terminal de **powershell**. ¡A continuación te dejamos un pequeño GIF de cómo hacer el cambio!





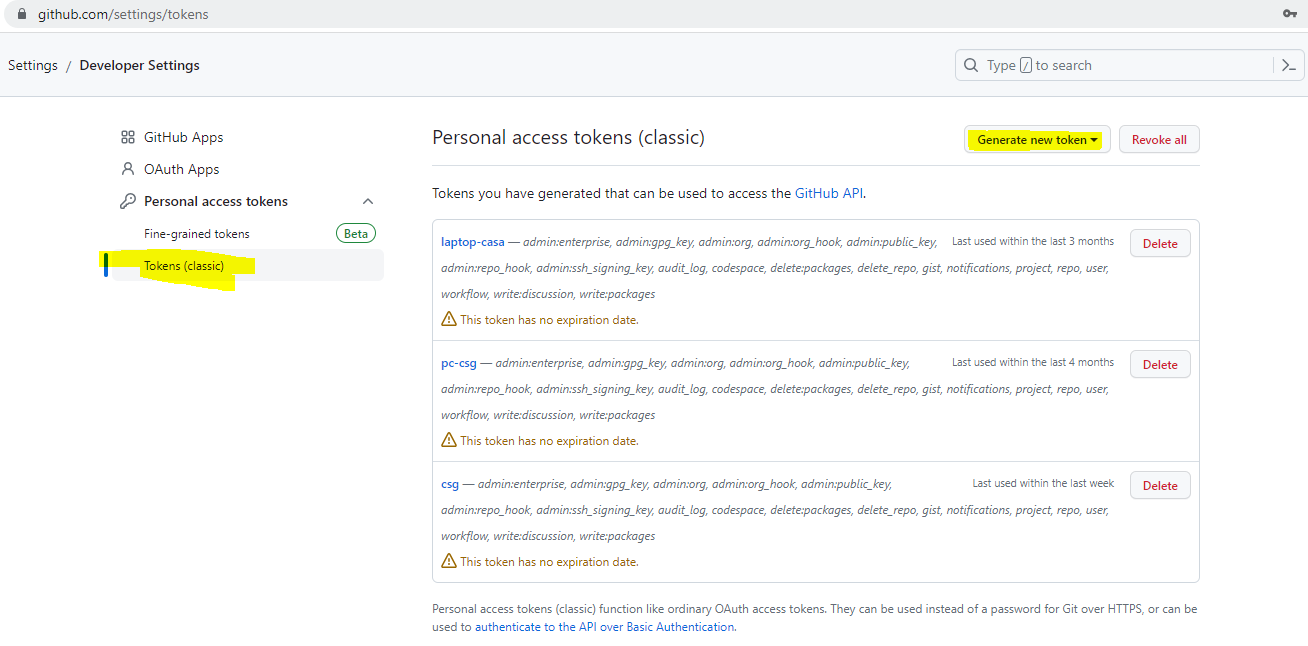
**Comandos más importantes para el manejo de la terminal:**

* 1**pwd**: imprimir el directorio en donde nos encontramos.
* 2**ls**: imprime archivos o carpetas dentro del cual estamos trabajando.
* 3**mkdir**carpetaPrueba: crea una carpeta llamada 'carpetaPrueba' en el directorio que estamos situados. Si queremos crear una carpeta que lleve espacios en su nombre se debe utilizar comillas. Por ej "PrepCourse Henry"
* 4**cd**: para ir hacia el directorio que contiene la carpeta o directorio en donde nos encontramos. También podemos usarla para movernos entre carpetas, por ejemplo al usar **cd carpetaPrueba** cambiamos de la carpeta actual hacia "carpetaPrueba"
* 5**touch**archivo.txt : crea un archivo con nombre "archivo.txt"
* 6**rm**archivo.txt : elimina el archivo "archivo.txt"
* 7**rm -r** carpetaPrueba: elimina la carpeta de nombre "carpetaPrueba"
* 8 ls –a : ver archivos ocultos

# Repositorio, configuración de credenciales de git y token de GitHub

# 

# 



Ojo:el token generado copiarlo y colocarlo en un archivo en el pc. Ya que no se volverá a ver

Entendiendo que un repositorio no es más que un almacén o lugar donde se guardan ciertas cosas y que hoy en día esto gracias a la tecnología se da mayoritariamente en la nube, se creo GIT. Este sistema nos permite...

* 1**Guardar, mantener y manipular archivos a lo largo del tiempo**
* 2**Posibilidad de trabajo colaborativo**
* 3**Proyectos guardados en un almacén “repositorio”**

Configurar las credenciales de GIT:

$ git config --global user.email "marbellys.campos@gmail.com"

$ git config --global user.name "marbellys"

 Crear un **repositorio** y hacer nuestro primer **commit**

**Ojo: $code hola.txt : abre el archivo hola.txt en VSCode**

Desde VSCode abrir terminal y colocar:

Crear repositorio Local:

$ git init

Initialized empty Git repository in C:/Marbellys/HENRY/PrepCourse Henry/.git/

MCampos@COM-119 MINGW64 /c/Marbellys/HENRY/PrepCourse Henry (master)

$ dir

Mi primer repositorio.txt

$ ls -a

./ ../ .git

'Mi primer repositorio.txt'

Un **commit** no es más que una captura instantánea de los cambios preparados en ese momento del proyecto. Las instantáneas confirmadas pueden considerarse como versiones "seguras" de un proyecto.

A continuación de compartimos los comandos más usados con **git**:

* 1 **git init**: comando para inicializar un repositorio local.
* 2 **git add:** comando para que nuestro repositorio sepa de la existencia de un archivo o de sus últimos cambios, no almacena las actualizaciones de forma definitiva, únicamente las guarda en algo que conocemos como “Staging Area” (área de montaje o ensayo). Se puede armar como git add ArchivoEjemplo.js (solo agrega ese archivo) o git add (agrega todos los archivos modificados de la carpeta donde estas con el punto.
* 3 **git commit -m** **"**mensaje**"**: comando para almacenar definitivamente todos los cambios que por ahora viven en el staging área. En el mensaje ponemos el mensaje que explica los cambios commiteados.
* 4 **git push**: comando para mandar nuestros commits a un servidor remoto.
* 5 **git status**: ofrece una descripción del estado de los archivos.
* 6 **git pull**: sirve para recibir cambios de repositorio remoto a local.

Recuerda que Git nunca cambiará las versiones seguras de tu proyecto, a no ser que se lo pidas expresamente. De esta forma, estamos guardando el estado actual de nuestro proyecto, para volver a visitarlo de ser necesario y por supuesto para mantenerlo a lo largo del tiempo.

Guardar el estado actual de un proyecto: COMMIT

$git add . o git add “nombre archivo”

MCampos@COM-119 MINGW64 /c/Marbellys/HENRY/PrepCourse Henry (master)

$ git add .

$ git commit -m "Mi primer trabajo con Henry"

[master (root-commit) 9950bf7] Mi primer trabajo con Henry

1 file changed, 2 insertions(+)

create mode 100644 Mi primer repositorio.txt

# Creando repositorio con GITHUB

**repositorio en la nube** utilizando **Github**.

**Posibles errores**

A continuación veremos algunos de los errores más típicos que pueden encontrarte cuando trabajas con Git y GitHub.

**Agregar un repositorio dentro de otro**

–

✅ Antes de clonar un repositorio, fíjate que no estés parado dentro de un repositorio. Utiliza el comando ***"cd .."*** para salir de esa carpeta contenedora.

**git init | Carpeta del sistema**

–

✅Si alguna vez inicializas un repositorio de git dentro de una carpeta del sistema de tu computadora, ¡Descuida! Solo tienes que entrar a la terminal, pararte dentro de esta carpeta y utilizar el comando: "***rm -rf .git"***.

**No funciona el comando: "code ."**

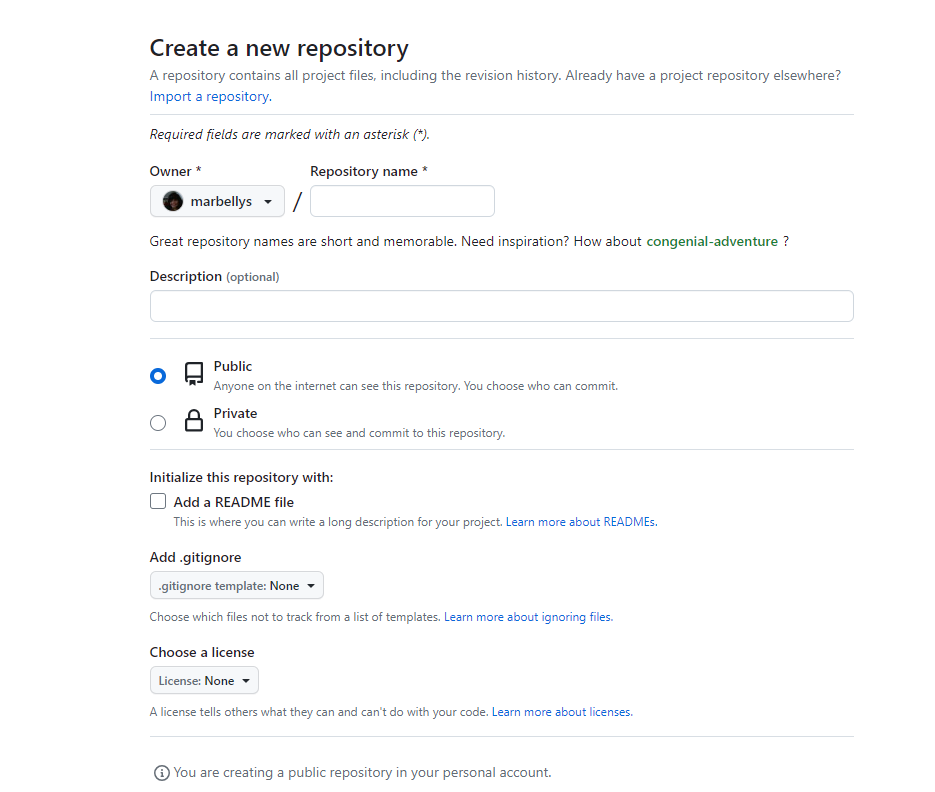
–

Hay dos formas en las que puedes abrir el repositorio en tu Visual Studio Code:

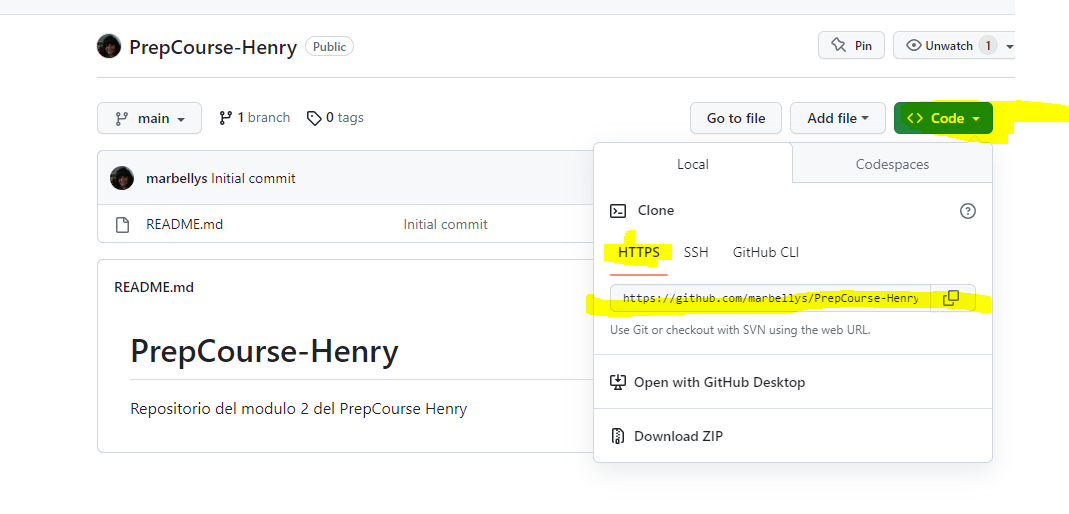
**1.** Dirígete a la carpeta del repositorio. Has click derecho encima de la carpeta, y preciona: "***Abrir con Code***".

**2.**Abre el Visual Studio Code. Busca la opción "archivo"/"file" en la barra de herramientas. Has click sobre el botón que dice "abrir carpeta". Localiza tu repositorio y simplemente ábrelo.

1.- creamos el repositorio en github:



2.- lo clonamos en el pc:



Desde el git bash en el pc:

$ git clone https://github.com/marbellys/PrepCourse-Henry.git

Vamos a trabajar con VSCode.

$ code .

Editamos el archimos.

**Se guarda en el repositorio local:**

MCampos@COM-119 MINGW64 /c/Marbellys/HENRY/PrepCourse-Henry (main)

$ git add "README.md"

MCampos@COM-119 MINGW64 /c/Marbellys/HENRY/PrepCourse-Henry (main)

$ git commit -m "Modificacion de readme"

[main 5bb1339] Modificacion de readme

1 file changed, 2 insertions(+)

**Se sube al repositorio en la nube:**

MCampos@COM-119 MINGW64 /c/Marbellys/HENRY/PrepCourse-Henry (main)

$ git push origin main

Enumerating objects: 5, done.

Counting objects: 100% (5/5), done.

Delta compression using up to 8 threads

Compressing objects: 100% (2/2), done.

Writing objects: 100% (3/3), 360 bytes | 360.00 KiB/s, done.

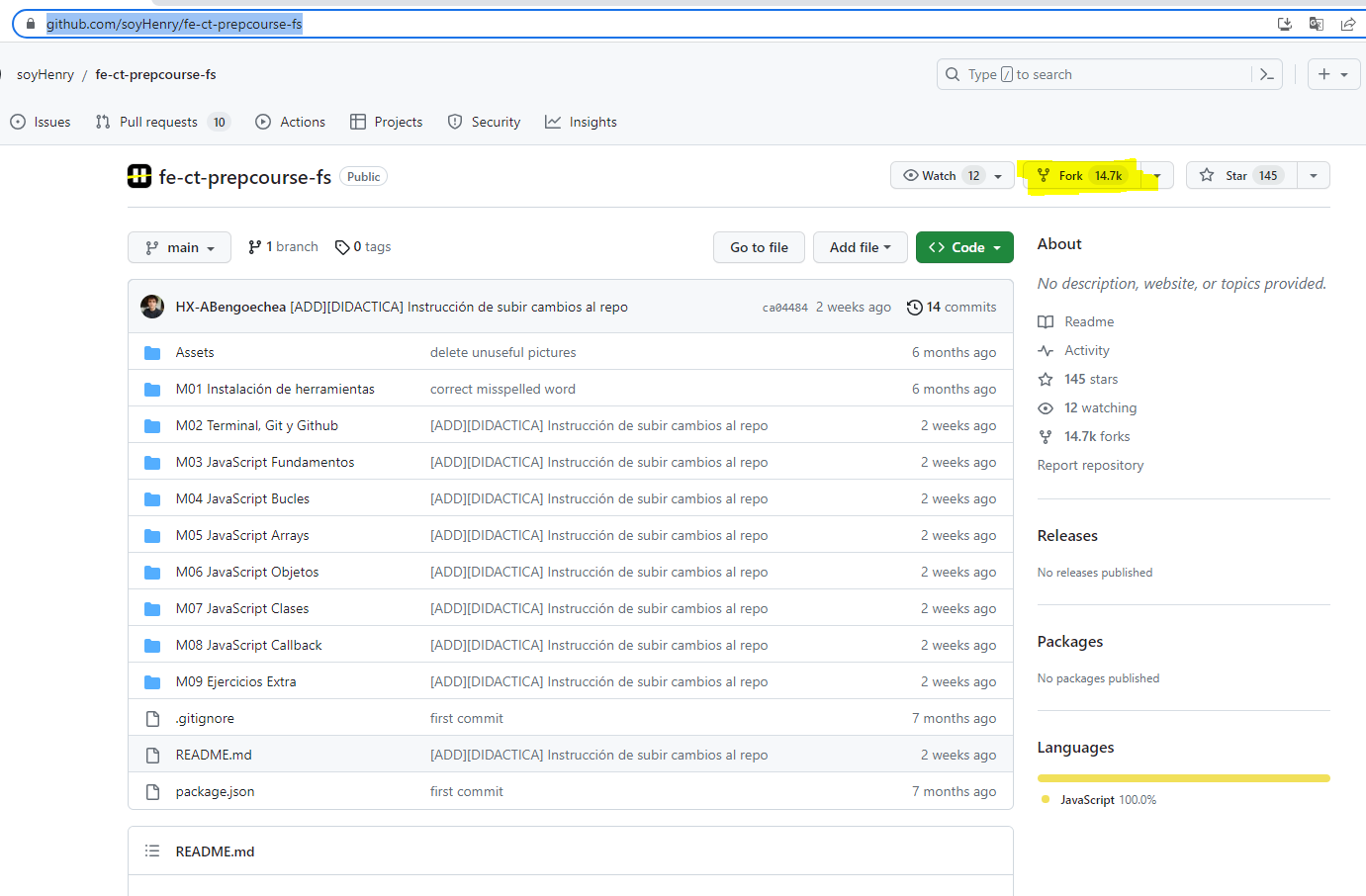
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0

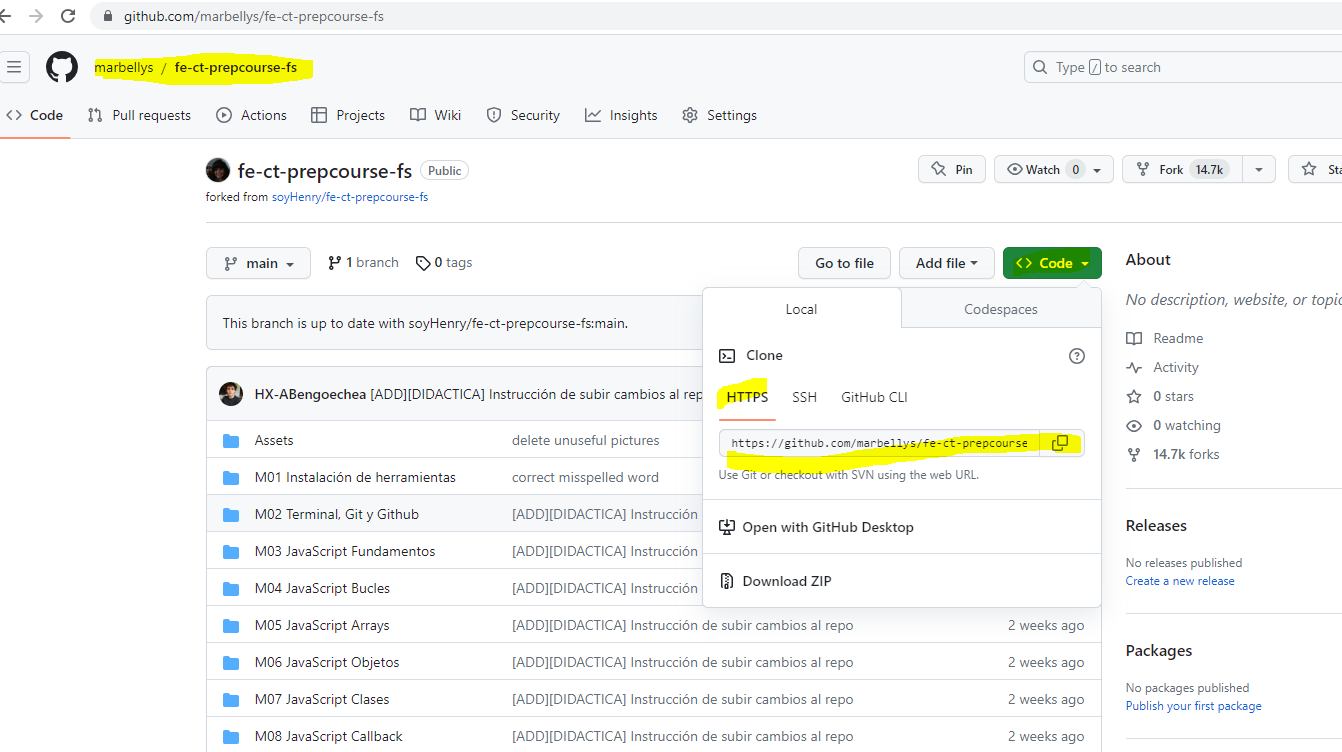
To https://github.com/marbellys/PrepCourse-Henry.git

aaa5742..5bb1339 main -> main

**Como utilizar el repositorio del PrepCourse:**

Ir al repositorio: <https://github.com/soyHenry/fe-ct-prepcourse-fs> y presionar el boton fork para que cree una copia en tu repositorio



Luego vamos a copiar 

Abrimos el VSCode

Nuevo Terminal

MCampos@COM-119 MINGW64 /c/Marbellys/HENRY

$ git clone https://github.com/marbellys/fe-ct-prepcourse-fs.git

Cloning into 'fe-ct-prepcourse-fs'...

remote: Enumerating objects: 129, done.

remote: Counting objects: 100% (19/19), done.

remote: Compressing objects: 100% (12/12), done.

remote: Total 129 (delta 6), reused 15 (delta 6), pack-reused 110

Receiving objects: 100% (129/129), 142.71 KiB | 283.00 KiB/s, done.

Resolving deltas: 100% (41/41), done.

Si no puedes asistir al espacio en vivo, podrás ver la grabación. Puedes encontrar el calendario con las clases, el link de acceso y el link de la grabación, aquí: [**\*\*CALENDARIO\*\***](https://prep.soyhenry.com/calendario/)

MODULO 3

conocerás los fundamentos de programación de Javascript, iniciaremos por comprender lo que es un lenguaje de programación en general, las cualidades de Javascript y luego abordaremos una serie de conceptos básicos como variables, tipos de datos, operadores y funciones para ejecutar nuestras primeras líneas de código.

Iniciemos por definir... ¿qué es un lenguaje de programación?

Podemos denominarlo como un lenguaje formal (o artificial, es decir, un lenguaje con reglas gramaticales bien definidas) que le proporciona a una persona, en este caso el programador, la capacidad de escribir (o programar) una serie de instrucciones o secuencias de órdenes en forma de algoritmos con el fin de controlar el comportamiento físico o lógico de un sistema informático, de manera que se puedan obtener diversas clases de datos o ejecutar determinadas tareas.

El popular portal devjobsscanner(opens in a new tab) recientemente ha publicado los lenguajes de programación más demandados en el último año y los listamos a continuación:

**El pilar del Front y el Back**

JavaScript es un **lenguaje de programación** dinámico que tiene la capacidad de ser utilizado en variados dispositivos, desde ordenadores hasta teléfonos inteligentes.

Basado en objetos, dinámico y amigable para principiantes, JavaScript es el pilar de múltiples herramientas que sus fanáticos fueron ampliando con nuevas funcionalidades a lo largo del tiempo.

Si bien era conocido como un lenguaje de programación solamente de **front-end**, en los últimos años tomó una posición muy fuerte también en el lado del servidor, a través de Node.js. Esto significa que alguien puede saber programar todo el stack, esto es: tanto el front como el **back-end**, ¡aprendiendo un único lenguaje!

JavaScript es un lenguaje de programación que se utiliza **client-side**. Permite crear la interfaz de usuario de sitios web, y realizar tareas del lado del servidor.

Además, es un lenguaje de **tipado dinámico**, lo que significa que no necesitamos declarar y establecer un tipo –como número o cadena de caracteres– específico e inflexible para las variables, lo cual da mucha libertad y sencillez, en especial al aprender el lenguaje. Técnicamente, JavaScript es un lenguaje de programación **interpretado**, por lo que no es necesario compilar los programas para ejecutarlos.

Junto con Python y Java, **JavaScript** es uno de los lenguajes de programación más utilizados por las empresas en el desarrollo de sus proyectos en el **mundo del software**.

Sin ir más lejos, las compañías que están construyendo el futuro destacan, entre las herramientas de su**tech stack**, a JavaScript como uno de los lenguajes de programación preferidos para desarrollar sus productos.

Lo cierto es que la mayoría de las aplicaciones que vemos hoy en día usan JavaScript. Los desarrolladores web de todo el mundo lo eligen por su facilidad de uso y porque su curva de aprendizaje es una de las más sencillas.

Aprenderlo te permitirá **crear cualquier aplicación para cualquier dispositivo o plataforma**. Además de ser el lenguaje de programación más popular, es el más utilizado en el mundo según la última encuesta de Stack Overflow (puedes ver más [aquí, en este reporte(opens in a new tab)](https://insights.stackoverflow.com/survey/2021#technology-most-popular-technologies))

**¿Cuáles son las principales cualidades de JavaScript?**

**By HENRY ㅤ**HENRY ㅤ

**Velocidad 🏃**

En esencia, JavaScript tiende a ser muy rápido. A menudo se ejecuta inmediatamente en el navegador; en tanto no requiera recursos externos, JavaScript no tiene permitido retrasarse por llamados del **servidor backend**.

**Popularidad 🏆**

JavaScript está en todos lados y se espera que su popularidad siga en alza. Si navegas por recursos y plataformas tales como **StackOverflow** y **GitHub** verás la cantidad de proyectos desarrollados en base a este lenguaje. Es más, con la llegada de Node.js, se ha incrementado su uso en el back-end y por lo tanto incorporado cada vez con más frecuencia en los tech stack de todo tipo de startups y compañías.

**Server Load 💻**

JavaScript trabaja del lado del cliente lo que reduce la demanda de servidores en general.

**Compatibilidad 🔁**

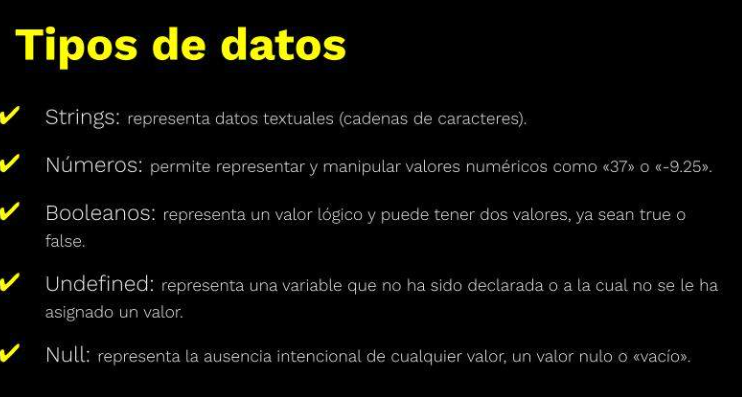
A diferencia de otros lenguajes scripting, JavaScript puede ser usado en cualquier página web y en diferentes tipos de aplicaciones gracias al soporte que ofrecen otros lenguajes como **Pearl** y **PHP**.

**Simplicidad 💆‍♀️**

La **sintaxis** de JavaScript está inspirada en Java, y en relación a otros lenguajes de programación igualmente populares, es relativamente sencillo de aprender.

**Versatilidad 🚀**

Hay muchos métodos para usar JavaScript mediante servidores Node.js.  Si tú estás en bootstrap Node.js con Express, puedes usar un documento database como **MongoDB**, y usar JavaScript en el frontend para clientes.  Una gran ventaja de JavaScript es que es el único lenguaje que te permite trabajar en modo **Full Stack**



<https://developer.mozilla.org/es/docs/Web/JavaScript/Data_structures#estructuras_y_tipos_de_datos>

* Seis **tipos de datos** [primitivos](https://developer.mozilla.org/es/docs/Glossary/Primitive), controlados por el [operador typeof](https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Operators/typeof)
  + [Undefined](https://developer.mozilla.org/es/docs/Glossary/Undefined): typeof instance === "undefined"
  + [Boolean](https://developer.mozilla.org/es/docs/Glossary/Boolean): typeof instance === "boolean"
  + [Number](https://developer.mozilla.org/es/docs/Glossary/Number): typeof instance === "number"
  + [String](https://developer.mozilla.org/es/docs/Glossary/String): typeof instance === "string"
  + [BigInt](https://developer.mozilla.org/es/docs/Glossary/BigInt): typeof instance === "bigint"
  + [Symbol](https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Global_Objects/Symbol): typeof instance === "symbol"
* [Null](https://developer.mozilla.org/es/docs/Glossary/Null): typeof instance === "object".  Tipo [primitivo](https://developer.mozilla.org/es/docs/Glossary/Primitive) especial que tiene un uso adicional para su valor: si el objeto no se hereda, se muestra null;
* [Object](https://developer.mozilla.org/es/docs/Glossary/Object): typeof instance === "object". Tipo estructural especial que no es de datos pero para cualquier instancia de objeto [construido](https://developer.mozilla.org/es/docs/Learn/JavaScript/Objects#the_constructor) que también se utiliza como estructuras de datos: new [Object](https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Global_Objects/Object), new [Array](https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Global_Objects/Array), new [Map (en-US)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Map" \o "Currently only available in English (US)), new [Set](https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Global_Objects/Set), new [WeakMap](https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Global_Objects/WeakMap), new [WeakSet](https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Global_Objects/WeakSet), new [Date](https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Global_Objects/Date) y casi todo lo hecho con la [palabra clave new](https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Operators/new);
* [Function](https://developer.mozilla.org/es/docs/Glossary/Function): una estructura sin datos, aunque también responde al operador typeof: typeof instance === "function". Esta simplemente es una forma abreviada para funciones, aunque cada constructor de funciones se deriva del constructor Object.

Ten en cuenta que el único propósito valioso del uso del operador typeof es verificar el tipo de dato. Si deseamos verificar cualquier Tipo Estructural derivado de Object, no tiene sentido usar typeof para eso, ya que siempre recibiremos "object". La forma correcta de comprobar qué tipo de Objeto estamos usando es la palabra clave [instanceof](https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Operators/instanceof). Pero incluso en ese caso, puede haber conceptos erróneos.

//variables

//String

var nombre ="Marbellys";

//Numericas

var nronega = -4;

var nrodec = 3.2;

//booleanas

var entiendom = true;

var noentiendo = false;

//Undefined

var apellido;

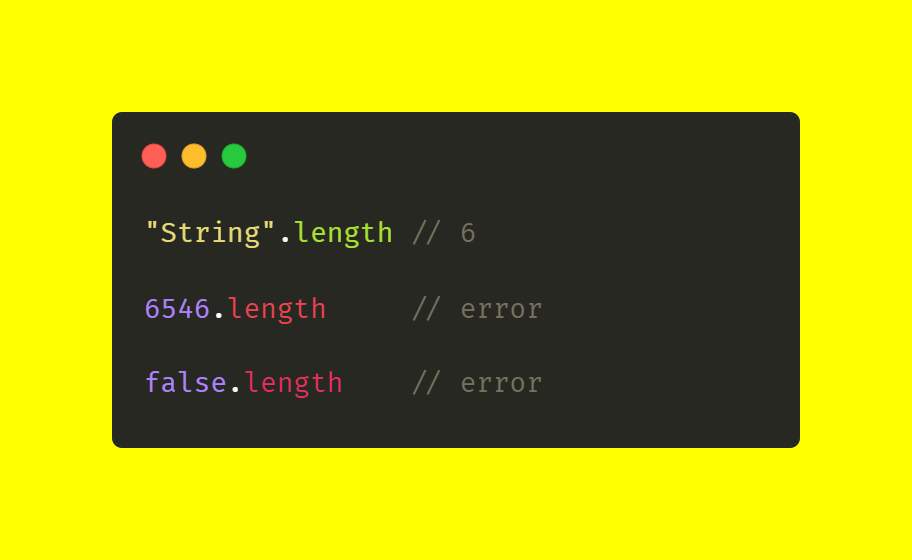
//null

var sinNada = null;

**MÉTODO LENGTH**

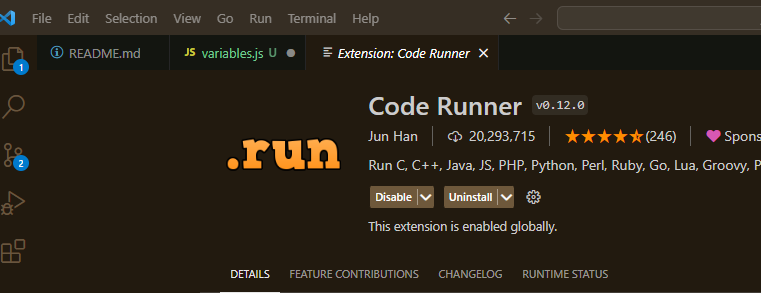
Este es un método que solo se puede aplicar al tipo de dato string (más adelante veremos que también lo podemos utilizar en arreglos). En este ejemplo podemos ver que si lo intentamos aplicar en un número o en un booleano, nos devuelve error.

Apellido.length



# Operadores y precedencia

Intalar la etension code runner



console.log(3 + 4 \* 5); // 3 + 20

// Expected output: 23

console.log(4 \* 3 \*\* 2); // 4 \* 9

// Expected output: 36

let a;

let b;

console.log(a = b = 5);

// Expected output: 5

Tabla de precedencia:

|  |  |  |  |
| --- | --- | --- | --- |
| **Precedencia** | **Tipo de operador** | **Asociatividad** | **Operadores individuales** |
| 19 | [Agrupamiento](https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Operators/Grouping) | n/a | ( … ) |
| 18 | [Acceso a propiedades (notación por punto)](https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Operators/Property_accessors#notaci.c3.b3n_por_punto) | a la izquierda | … . … |
| [Acceso a propiedades (notación por corchetes)](https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Operators/Property_accessors#notaci.c3.b3n_por_corchetes) | a la izquierda | … [ … ] |
| [new](https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Operators/new) (con lista de argumentos) | n/a | new … ( … ) |
| [Llamada a función](https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Functions) | a la izquierda | … ( … ) |
| [Encadenamiento opcional](https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Operators/Optional_chaining) | a la izquierda | ?. |
| 17 | [new](https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Operators/new) (sin lista de argumentos) | a la derecha | new … |
| 16 | [Incremento sufijo](https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Operators#incremento_y_decremento) | n/a | … ++ |
| [Decremento sufijo](https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Operators#incremento_y_decremento) | … -- |
| 15 | [NOT lógico (!) (en-US)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Logical_NOT) | a la derecha | ! … |
| [NOT a nivel de bits (~) (en-US)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Bitwise_NOT) | ~ … |
| [Suma unaria (+) (en-US)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Unary_plus) | + … |
| [Negación unaria (-) (en-US)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Unary_plus) | - … |
| [Incremento prefijo](https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Operators#incremento_y_decremento) | ++ … |
| [Decremento prefijo](https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Operators#incremento_y_decremento) | -- … |
| [typeof](https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Operators/typeof) | typeof … |
| [void](https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Operators/void) | void … |
| [delete](https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Operators/delete) | delete … |
| [await](https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Operators/await) | await … |
| 14 | [Potenciación (\*\*) (en-US)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Exponentiation) | a la derecha | … \*\* … |
| 13 | [Multiplicación (\*) (en-US)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Multiplication) | a la izquierda | … \* … |
| [División (/)](https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Operators/Division) | … / … |
| [Resto (%)](https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Operators/Remainder) | … % … |
| 12 | [Adición (+)](https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Operators/Addition) | a la izquierda | … + … |
| [Sustracción (-)](https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Operators/Subtraction) | … - … |
| 11 | [Desplazamiento de bits a la izquierda (<<) (en-US)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Left_shift) | a la izquierda | … << … |
| [Desplazamiento de bits a la derecha (>>) (en-US)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Right_shift) | … >> … |
| [Desplazamiento de bits a la derecha sin signo (>>>) (en-US)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Unsigned_right_shift) | … >>> … |
| 10 | [Menor a (<) (en-US)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Less_than) | a la izquierda | … < … |
| [Menor o igual a (<=) (en-US)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Less_than_or_equal) | … <= … |
| [Mayor a (>) (en-US)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Greater_than) | … > … |
| [Mayor o igual a (>=) (en-US)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Greater_than_or_equal) | … >= … |
| [in](https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Operators/in) | … in … |
| [instanceof](https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Operators/instanceof) | … instanceof … |
| 9 | [Igualdad (==)](https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Operators/Equality) | a la izquierda | … == … |
| [Desigualdad (!=) (en-US)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Inequality) | … != … |
| [Igualdad estricta (===)](https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Operators/Strict_equality) | … === … |
| [Desigualdad estricta (!==) (en-US)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Strict_inequality) | … !== … |
| 8 | [AND a nivel de bits (&) (en-US)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Bitwise_AND) | a la izquierda | … & … |
| 7 | [XOR a nivel de bits (^) (en-US)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Bitwise_XOR) | a la izquierda | … ^ … |
| 6 | [OR a nivel de bits (|) (en-US)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Bitwise_OR) | a la izquierda | … | … |
| 5 | [AND lógico (&&) (en-US)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Logical_AND) | a la izquierda | … && … |
| 4 | [OR lógico (||) (en-US)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Logical_OR) | a la izquierda | … || … |
| [Operador de coalescencia nula (??) (en-US)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Nullish_coalescing) | a la izquierda | … ?? … |
| 3 | [Operador condicional (ternario)](https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Operators/Conditional_operator) | a la derecha | … ? … : … |
| 2 | [Asignación](https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Operators#operadores_de_asignaci%C3%B3n) | a la derecha | … = … |
| … += … |
| … -= … |
| … \*\*= … |
| … \*= … |
| … /= … |
| … %= … |
| … <<= … |
| … >>= … |
| … >>>= … |
| … &= … |
| … ^= … |
| … |= … |
| … &&= … |
| … ||= … |
| … ??= … |
| [yield](https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Operators/yield) | a la derecha | yield … |
| [yield\*](https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Operators/yield*) | yield\* … |
| 1 | [Operador coma](https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Operators/Comma_operator) | a la izquierda | … , … |

# Operadores de comparación

Anteriormente conocimos los operadores básicos y su precedencia, ahora nos enfocaremos en los operadores de comparación: mayor que (>), menor que (<), mayor o igual (>=), menor o igual (<=), igual == (se representa con doble igual)  y distinto de (!=).

console.log (7 == "7"); true

console.log (7 === "7"); false

== verifica solo en valor

=== verifica tanto el tipo de datos como el valor

//Asignación y asociatividad

var a = 1;

var b = 2;

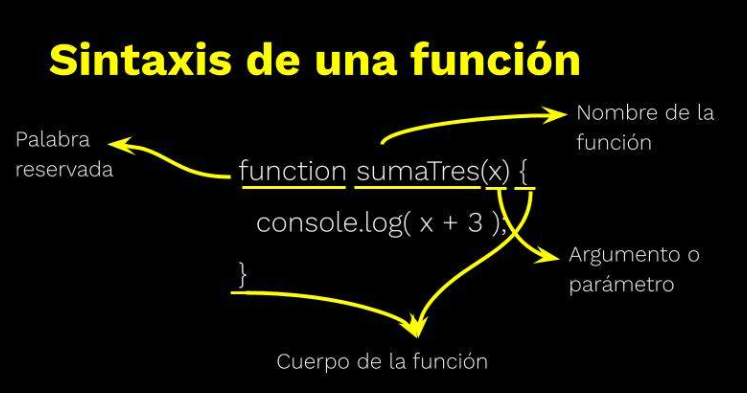
var c = a = b;

console.log (c);

2

console.log (8/2/2); 2

Funciones:



Es similar, declarae estas funciones:

//forma1:

function sumaTres1(x){

    return x +3;

};

sumaTres1(2);

//forma2:

var sumaTres2 = function (x){

    return x +3;

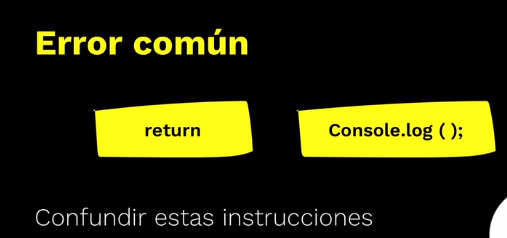
};

//forma3: funcion de flecha

var sumaTres3 = (x)=>{

    return x +3;

};



function CuidadoConElConsoleLog(nombre){

    console.log(nombre);

    return nombre;

}

Para ejecutar esto, abrimos un terminal y utilizamos el motor de node.

$ node

Welcome to Node.js v18.16.1.

Type ".help" for more information.

> function CuidadoConElConsoleLog(nombre){

... console.log(nombre);

... return nombre;

... }

Undefined

Retorna undefine.

Pero si llamamos la función:

> CuidadoConElConsoleLog('Marbellys')

Marbellys

'Marbellys'

Muestra lo que tiene la variable y también Retorna el string ‘Marbellys’

Realmente son diferentes

Ojo: Nunca olvidar colocar el return a la función

Ojo: después de un return no se ejecutará nada:

function CuidadoConElConsoleLog(nombre){

    return nombre;

console.log(nombre);

}

**Nombrando funciones...**

Habrás visto que en la anterior video clase comenzamos a ponerle nombres a nuestras funciones, ¿verdad? Bueno, existen diferentes maneras de escribir nombres dentro del mundo del código. Antes de continuar a la siguiente lección, queremos contartelas.

**¿Qué es una nomenclatura?**

Las nomenclaturas son formas de llamar a las cosas. En programación existen ciertas reglas sobre las cuales llevamos uniformidad en el código y nos proporcionan un estándar de trabajo especialmente dentro de equipos de desarrollo de software.

Existen 3 tipos de nomenclatura principal: **camelCase, PascalCase, snake\_case.**

**camelCase** Este modo de nombrar implica la unión de dos o más palabras sin espacios entre ellas, pero diferenciadas por una letra mayúscula inicial a partir de la segunda palabra, por ejemplo: holaMundo / funcionEdadAños.

**PascalCase** Es similar a camelCase pero la primera letra de la frase también se inicia con mayúscula, al igual que cada palabra. Se mantiene la práctica de no tener espacios entre palabras y cada palabra siempre empieza con su primera letra mayúscula. Por ejemplo: HolaMundo / FuncionEdadAños

**snake\_case** La nomenclatura snake\_case  es definida de esta forma porque siempre la usamos sobre el piso. Las letras siempre  serán minúsculas y las  palabras están separadas por un guion bajo de esta forma: hola\_mundo / funcion\_edad\_años

**Control de flujo (if/else)**

En esta lección abordaremos la forma de direccionar nuestro código a través de los **condicionales (If/else)**  son estructuras fundamentales para la resolución de problemas. En la siguiente video clase detallaremos su importante con ejemplos prácticos.

