

¿Qué es la programación?

La **programación** es el proceso de crear instrucciones que una computadora o dispositivo digital puede seguir para realizar tareas específicas. Estas instrucciones se escriben en un lenguaje de programación, el cual puede ser entendido tanto por humanos como por máquinas. La programación implica el uso de lógica, control de flujo, estructuras de datos, y algoritmos para resolver problemas.

Los pasos principales de la programación son:

1. **Entender el problema:** Analizar el objetivo o tarea que debe cumplir el programa.
2. **Diseñar la solución:** Crear un plan, a menudo usando diagramas de flujo o pseudocódigo.
3. **Codificar:** Escribir el código en un lenguaje de programación.
4. **Probar y depurar:** Ejecutar el programa para asegurarse de que funcione correctamente y corregir errores.
5. **Mantener:** Hacer mejoras o correcciones en el programa a lo largo del tiempo.

Algoritmos

¿Qué es un algoritmo?

Un algoritmo se puede definir como una serie de pasos lógicos, ordenados y finitos para llegar a la solución de un problema. Un algoritmo consta de:

- Entrada de datos: son los datos que requiere el algoritmo para ser ejecutado.
- Proceso: es la secuencia de pasos lógicos que se llevan a cabo durante la ejecución del algoritmo.
- Salida: son los datos que se obtienen después de la ejecución del algoritmo.

Ejemplo: un algoritmo para preparar limonada

Ingredientes(Entrada):

- Limones
- Azúcar
- Agua
- Hielo

Preparación(Proceso):

1. Exprima los limones
2. Agregue agua
3. Agregue azúcar y revuelva
4. Agregue hielo

Resultado(Salida):



Ejercicios: construya un algoritmo para:

- Preparar arroz
- Enviar un correo electrónico
- Cepillarse los dientes

Conceptos básicos de lógica

En tu proceso de aprendizaje, ten muy presente los siguientes conceptos para aplicarlos en los ejercicios de lógica de programación o desarrollo.

CONCEPTO	EJEMPLO
Variables y Tipos: Los datos que almacenamos en las variables tienen diferentes tipos. El tipo de dato determina las operaciones que podemos realizar con ese valor.	<pre>//Declaración de una variable let nombre; //Inicializamos la variable nombre nombre = "Ana"; // String // Declaramos variables de diferentes tipos y las inicializamos let edad = 25; // Number let esEstudiante = true; // Boolean let coloresFavoritos = ["rojo", "verde", "azul"]; // Array let persona = { nombre: "Pedro", edad: 30 }; // Object</pre>
Operadores Aritméticos: Los operadores aritméticos	<pre>let x = 10;</pre>

son símbolos especiales que nos permiten realizar cálculos matemáticos sobre números en JavaScript. Estos operadores nos permiten sumar, restar, multiplicar, dividir, obtener el módulo (resto de una división)

```
let y = 5;

let suma = x + y; // suma será 15
let resta = x - y; // resta será 5
let multiplicacion = x * y; //
multiplicación será 50
let division = x / y; // división será
2
let modulo = x % y; // módulo será 0
suma++; //aumento de la variable suma
en 1
resta--; //decremento de la variable
resta en 1
```

Operadores Lógicos: Los operadores lógicos nos permiten combinar expresiones booleanas (verdaderas o falsas) para crear condiciones más complejas. Son fundamentales para tomar decisiones en nuestros programas.

```
let tieneHambre = true;
let tieneDinero = false;
let esFinDeSemana = true;
Let tieneHambreyDinero = tieneHambre &&
tieneDinero; // false
Let tieneHambreyDinero = esFinDeSemana
|| tieneDinero; // true
let contrarioTieneHambre =
!tieneHambre; // false
```

Operadores Relacionales: Los operadores relacionales nos permiten comparar dos valores y obtener un resultado booleano (verdadero o falso). Estos resultados son cruciales para las estructuras condicionales como if y else if.

```
let edad = 25;
let nombre = "Juan";
// Igualdad
console.log(edad == 25); // true
(igualdad no estricta)
console.log(edad === "25"); // false
(igualdad estricta, tipos diferentes)
// Desigualdad
console.log(edad != 30); // true
console.log(nombre !== "Pedro"); //
true
```

	<pre>// Mayor que, menor que, etc. console.log(edad > 18); // true console.log(edad < 18); // false console.log(edad >= 18); // true console.log(edad <= 20); // false</pre>
Imprimir: Mostrar información en la consola	<pre>let nombre = "Juan"; let edad = 30; console.log("Hola, mi nombre es", nombre, "y tengo", edad, "años.");</pre>
Entrada de Datos: La función <code>prompt()</code> crea un cuadro de diálogo donde se le solicita al usuario que ingrese un valor.	<pre>let edad = prompt("¿Cuántos años tienes?");</pre>
Conversiones: Si tenemos un número almacenado como una cadena y queremos realizar operaciones matemáticas con él, debemos convertir esa cadena a un número. Si tenemos un número y queremos concatenarlo con una cadena, podemos convertir el número a cadena.	<pre>//Convierte un valor a un número. let numero = Number("42"); // número será 42 //Convierte una cadena a un número entero. let entero = parseInt("10.5"); // entero será 10 //Convierte una cadena a un número de punto flotante let flotante = parseFloat("3.14"); // flotante será 3.14 //Convierte un valor a una cadena. let cadena = String(42); // cadena será "42"</pre>
Constantes: Una constante es una variable cuyo valor no puede cambiar una vez asignado	<pre>const PI = 3.14159; // Una constante se declara usando la palabra clave "const" PI = 3.14159; // Otorgarle otro valor no es posible y arrojaría error</pre>
Comentarios: Los comentarios son líneas de texto que el motor de JavaScript ignora al ejecutar el código. Su principal función es documentar el código, haciendo que sea más fácil de entender tanto para ti como para otros desarrolladores.	<pre>// Este es un comentario de una línea let x = 10; // Inicializamos la variable x /*</pre>

	<pre>Este es un comentario de múltiples líneas. Podemos escribir tanto como queramos. */</pre>
--	------------------------------------------------------------------------------------------------

ANÁLISIS DE UN PROBLEMA EN PROGRAMACIÓN

A continuación, se sugerirá una manera ordenada de resolver problemas a través de la programación.

Para resolver problemas en programación seguiremos los siguientes pasos:

1. Analizar el problema determinando qué respuesta debe dar nuestro algoritmo a construir y cuál es el tipo de ésta.
2. (ENTRADA DE DATOS) Leer el problema que debemos resolver y extraer lo siguiente:
 - Datos que requiere el algoritmo para trabajar.
 - De los datos que requiere nuestro algoritmo, ¿cuáles nos los proporciona el enunciado del problema ?
 - De los datos que requiere nuestro algoritmo, ¿cuáles no los proporciona el enunciado del problema y debemos obtenerlos de otras fuentes?
3. (PROCESO) ¿Cuáles son las acciones que debe realizar nuestro algoritmo una vez le hemos proporcionado los datos que necesita para trabajar? Estas acciones se deben describir y enumerar.
4. (SALIDA DE DATOS) Construir la respuesta de nuestro algoritmo

Para expresar los pasos anteriores escribiremos pseudocódigo, ya sea siguiendo un estándar o no, algo muy personal.

Veamos un ejemplo:

Problema: **Realizar un algoritmo para determinar la distancia de dos puntos en el espacio bidimensional.**

Analizando el problema, podemos determinar que la respuesta de nuestro algoritmo debe ser numérica ya que se nos pide “**determinar la distancia**”:

tipo salida: numérica

(ENTRADA DE DATOS)

Los datos que requiere nuestro algoritmo para trabajar son:

Como se habla de distancia entre dos puntos en el espacio bidimensional, o sea, el plano cartesiano, necesitaremos las coordenadas de dos puntos en el espacio, las cuales no las proporciona el problema, por lo cual, el usuario deberá ingresarlas por teclado.

coordenada_x1: lo ingresará el usuario

coordenada_y1: lo ingresará el usuario

coordenada_x2: lo ingresará el usuario

coordenada_y2: lo ingresará el usuario

(PROCESO)

¿QUÉ SE DEBE HACER?

- 1. Pedir por teclado las coordenadas de los puntos**
- 2. Calcular la distancia entre los puntos obtenidos**

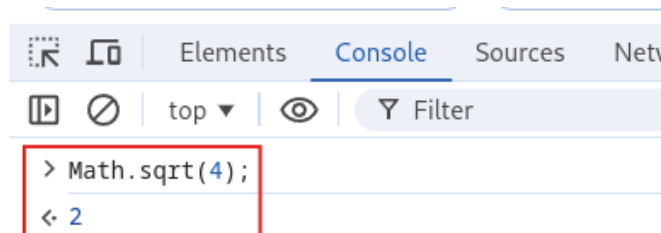
¿CÓMO SE HACE LO QUE SE DEBE HACER?

Pedir por teclado las coordenadas de los puntos - se hace con el método *prompt()*. Los datos ingresados deben ser convertidos con *parseFloat()*, ya que operaremos con ellos.

Determinar la distancia entre los puntos obtenidos - se hace con la fórmula

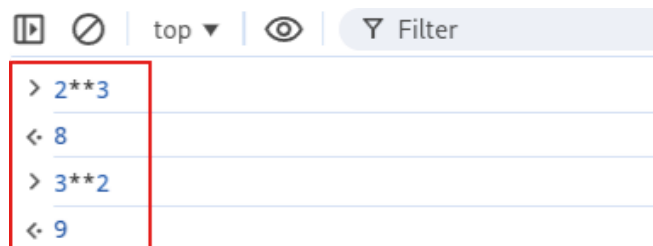
de distancia entre dos puntos $\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$ Acá, notamos que la fórmula hace uso de raíz cuadrada y elevación al cuadrado, por lo tanto, necesitamos averiguar cómo se hacen estas dos operaciones en Javascript. Haciendo la investigación sabemos que:

Para usar raíz cuadrada en Javascript podemos recurrir a la librería *Math* y usar su función *sqrt*, la cual recibe un número y devuelve su raíz cuadrada:



```
> Math.sqrt(4);  
↩ 2
```

Para elevar un número a una potencia en Javascript usar el operador de potenciación ******, el cual toma un número y lo eleva a la potencia indicada:



```
> 2**3  
↩ 8  
> 3**2  
↩ 9
```

(SALIDA DE DATOS)

¿QUÉ SE DEBE HACER?

1. Imprimir un mensaje al usuario mostrando la distancia entre los puntos tecleados por éste


¿CÓMO SE HACE LO QUE SE DEBE HACER?

Imprimir un mensaje al usuario mostrando la distancia entre los puntos tecleados por éste - se hace con *console.log()*.

PASANDO NUESTRO PSEUDOCÓDIGO O CÓDIGO JAVASCRIPT

Ahora, debemos transcribir y adaptar las acciones expresadas en el análisis del problema a código Javascript:

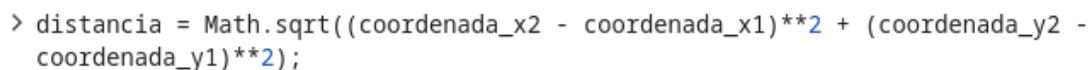
(ENTRADA DE DATOS) Según lo planeado, las coordenadas de los puntos deben ser ingresadas por teclado por el usuario usando *prompt()*. No podemos olvidar convertir estos datos a flotantes para poder operar con ellos.



```

> coordenada_x1 = parseFloat(prompt("Ingrese coordenada en x del primer punto"))
< 3
> coordenada_y1 = parseFloat(prompt("Ingrese coordenada en y del primer punto"))
< 6
> coordenada_x2 = parseFloat(prompt("Ingrese coordenada en x del segundo punto"))
< 8
> coordenada_y2 = parseFloat(prompt("Ingrese coordenada en y del segundo punto"))
< 7
```

(PROCESO) Según lo planeado, debemos aplicar la fórmula de distancia usando el método *sqrt* para trabajar con raíces cuadradas y usar el operador **** para elevar al cuadrado en este caso. El resultado que obtenemos al aplicar la fórmula de distancia lo guardamos en la variable *distancia*.



```

> distancia = Math.sqrt((coordenada_x2 - coordenada_x1)**2 + (coordenada_y2 -
  coordenada_y1)**2);
```

(SALIDA DE DATOS) Según lo planeado, se debe usar *console.log()* para mostrar al usuario un mensaje indicando la distancia buscada. Recordar que el mensaje al usuario debe tener información del dato que se le muestra, de esta manera el usuario no se confundirá preguntándose qué significa el dato que nuestro programa le está mostrando.


```
> console.log("La distancia entre los puntos es: ", distancia)
La distancia entre los puntos es: 5.0990195135927845
```

Por último, nuestro programa completo quedaría así:

```
> coordenada_x1 = parseFloat(prompt("Ingrese coordenada en x del primer punto"))
< 3
> coordenada_y1 = parseFloat(prompt("Ingrese coordenada en y del primer punto"))
< 6
> coordenada_x2 = parseFloat(prompt("Ingrese coordenada en x del segundo punto"))
< 8
> coordenada_y2 = parseFloat(prompt("Ingrese coordenada en y del segundo punto"))
< 7
> distancia = Math.sqrt((coordenada_x2 - coordenada_x1)**2 + (coordenada_y2 -
  coordenada_y1)**2);
< 5.0990195135927845
> console.log("La distancia entre los puntos es: ", distancia)
La distancia entre los puntos es: 5.0990195135927845 VM
```

Ahora, aplicando los mismos pasos, resuelva en Javascript:

1. Realice un programa que calcule el índice de masa corporal de una persona.
2. Realice un programa que calcule el área de un trapecio isósceles.
3. Realice un programa que calcule la distancia entre dos puntos en el espacio tridimensional.

Introducción a JavaScript y su entorno

JavaScript es un lenguaje de programación ampliamente utilizado para el desarrollo web, tanto en el frontend como en el backend. Fue creado para agregar interactividad a las páginas web, pero ahora se puede utilizar para crear aplicaciones completas en múltiples entornos gracias a tecnologías como Node.js.

Características principales de JavaScript:

1. Lenguaje interpretado(necesita un programa intérprete para funcionar): JavaScript se ejecuta en tiempo real dentro del navegador o en un entorno como Node.js, sin necesidad de compilación.
2. Dinámico: Los tipos de datos se asignan automáticamente en tiempo de ejecución.
3. Orientado a eventos: La mayoría del desarrollo frontend en JavaScript gira en torno a la interacción del usuario y eventos como clics de botón o movimientos del mouse.
4. Multiplataforma: JavaScript funciona en casi cualquier navegador y dispositivo.
5. Versatilidad: Se usa tanto en el lado del cliente (navegador) como en el lado del servidor (con Node.js).

Entorno de desarrollo:

- Navegador: El entorno más común para ejecutar JavaScript es el navegador web, que viene con una consola de desarrollo para probar y depurar código.
- Node.js: Un entorno de ejecución que permite usar JavaScript en el lado del servidor, permitiendo desarrollar aplicaciones completas fuera del navegador.
- Editores de texto/IDEs: Para escribir código JavaScript, se puede utilizar editores como Visual Studio Code, Sublime Text, o Atom.

Repositorios: Git/Github

Git y GitHub son herramientas fundamentales para el control de versiones y la colaboración en proyectos de desarrollo de software.

Git:

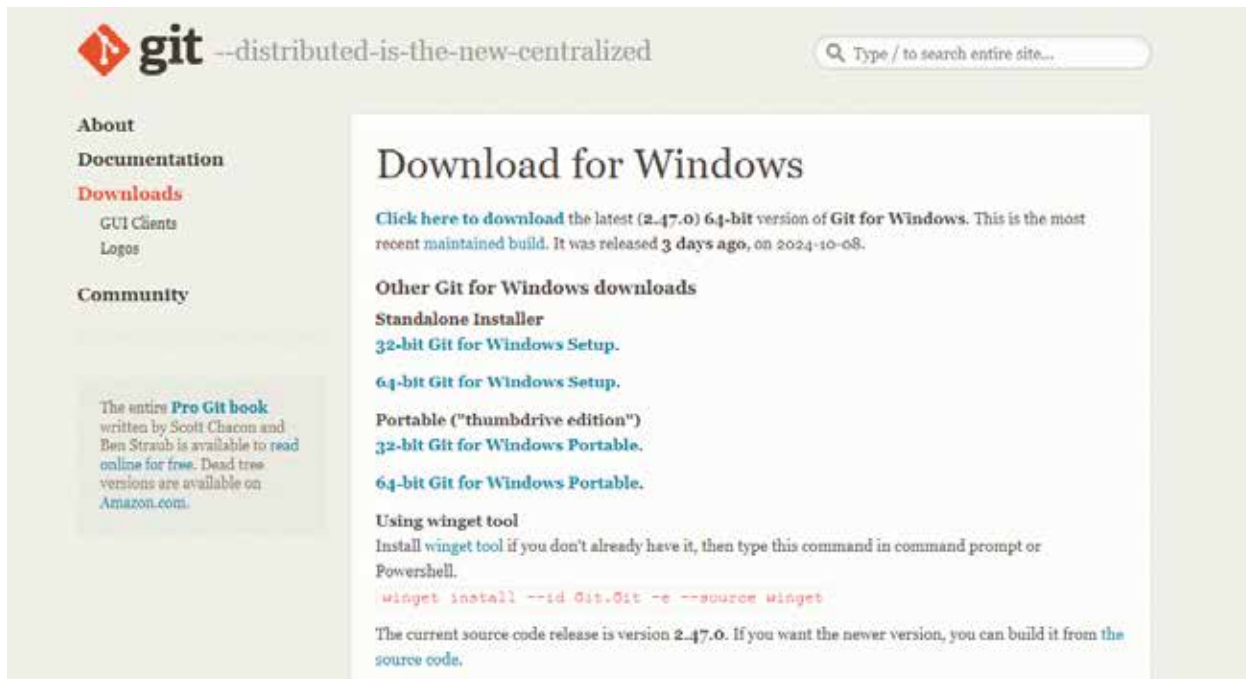
- Sistema de control de versiones distribuido
- Permite rastrear cambios en el código fuente
- Facilita la colaboración entre desarrolladores
- Funciona localmente en tu máquina

Ventajas de Git:

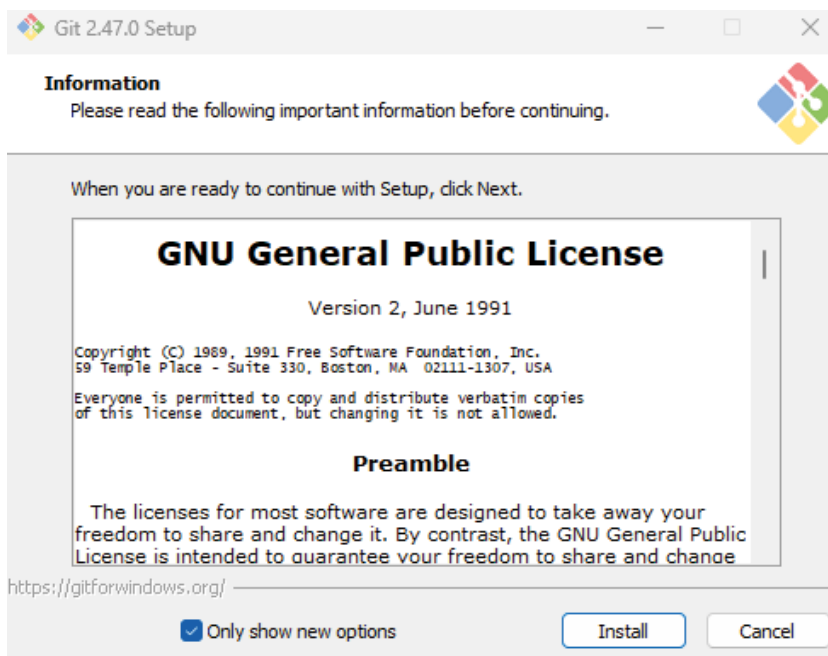
- Control detallado del historial de cambios
- Trabajo offline
- Branching y merging eficientes

Pasos para su descarga

1. Ingresamos al sitio oficial: <https://git-scm.com/downloads>

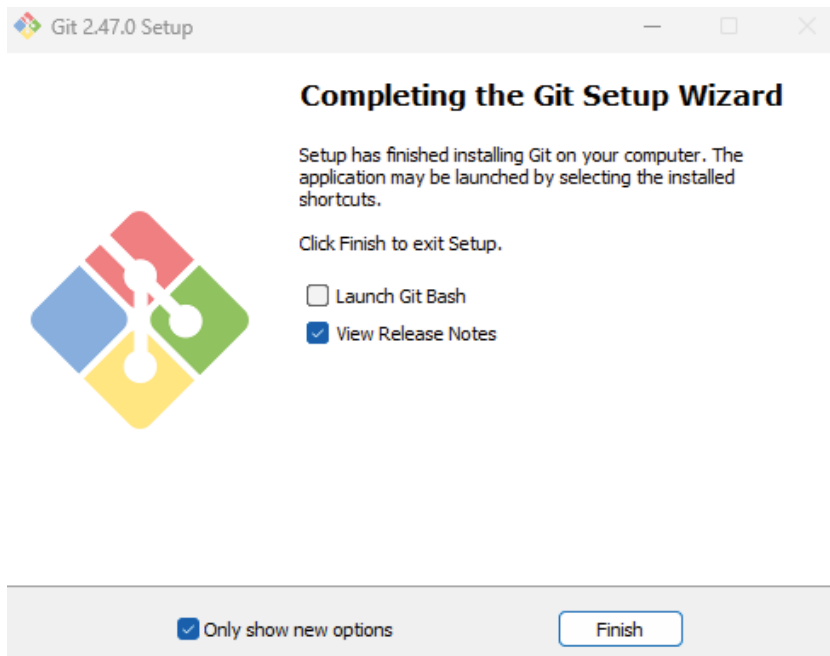


2. Presionamos en donde dice **“Click here to download”** que descargara la ultima versión estable para windows
3. Ejecutaremos el archivo descargado y presionamos donde dice instalar en la parte inferior derecha



4. Presionamos **“Next”** dejando la configuración por defecto tanto de la ruta de instalación y los componentes a instalar además del editor por defecto y todas sus demás opciones

5. Al completar el proceso de instalación presionamos **“Finish”**



6. Y con esto finalizamos la instalación de git para más información podemos consultar el siguiente videotutorial <https://www.youtube.com/watch?v=JWnZvHOYBDc>

Los comandos git a usar en nuestro trabajo son:

INICIA UN REPOSITORIO
`git init`

IDENTIFICARSE CON UN CORREO DE USUARIO
`git config --global user.email "usuarioejemplo@ejemplo.com"`

IDENTIFICARSE CON UN NOMBRE DE USUARIO
`git config --global user.name "nombreusuario"`

GUARDAR CREDENCIALES
`git config --global credential.helper store`

MOSTRAR LA CONFIGURACIÓN
`git config --list`

AGREGAR UNA NUEVA VERSIÓN AL REPOSITORIO(HACER UN SNAPSHOT DEL PROYECTO)

```
git commit -m "descripción del commit"
```

CLONAR UN REPOSITORIO

```
git clone url_repositorio_remoto
```

MOSTRAR EL ESTADO DEL PROYECTO

```
git status
```

AGREGA UN ARCHIVO AL STAGING AREA.

```
git add index.html
```

AGREGA TODOS LOS ARCHIVOS AL STAGING AREA

```
git add.
```

MOSTRAR EL HISTORIAL DE COMMITS EN USO CON SU AUTOR, FECHA-HORA Y CÓDIGO HASH

```
git log
```

MOSTRAR EL HISTORIAL DE COMMITS CON SU AUTOR, FECHA-HORA Y CÓDIGO HASH

```
git log --reflog
```

DEVOLVER UN ARCHIVO A LA ÚLTIMA VERSIÓN GUARDADA

```
git checkout -- index.html
```

DEVOLVER UN ARCHIVO A UNA VERSIÓN EN PARTICULAR

```
git checkout hash_commit index.html
```

DEVOLVER UN PROYECTO A UNA VERSIÓN EN PARTICULAR

```
git checkout hash_commit
```

ELIMINAR UN ARCHIVO DEL STAGING AREA Y DEJAR DE RASTREARLO

```
git rm --cached index.html
```

SACAR UN ARCHIVO DEL STAGING AREA

```
git reset index.html
```

DIFERENCIA ENTRE EL DIRECTORIO DE TRABAJO Y LO QUE SE PODRÍA AGREGAR AL ÁREA DE INTERCAMBIO

```
git diff index.html
```

DIFERENCIA ENTRE EL DIRECTORIO DE TRABAJO Y LO QUE SE HA AGREGADO AL ÁREA DE INTERCAMBIO

```
git diff --cached index.html
```

DIFERENCIA ENTRE EL ÚLTIMO COMMIT Y UN COMMIT EN PARTICULAR

```
git diff hashdelcommit
```

CREAR UNA RAMA

```
git branch nombrenuevarama
```

CONSULTAR RAMAS

```
git branch
```

MOSTRAR EL HISTORIAL DEL REPOSITORIO EN FORMA DE GRAFO INCLUYENDO TODAS LAS RAMAS

```
git log --graph
```

CAMBIARSE A UNA RAMA

```
git checkout nombrerama
```

FUSIONAR RAMAS

```
git merge nombrerama
```

INTEGRAR CAMBIOS DE UNA RAMA EN OTRA DE MANERA SECUENCIAL

```
git rebase
```

ELIMINAR UNA RAMA QUE YA HA SIDO FUSIONADA

```
git branch -d nombrerama
```

ELIMINAR UNA RAMA SIN IMPORTAR QUE YA HAYA SIDO FUSIONADA O NO

```
git branch -D nombrerama
```

PARA AGREGAR ARCHIVOS O CARPETAS AL PROYECTO QUE SERÁN IGNORADOS POR GIT, SE CREA

UN ARCHIVO LLAMADO `.gitignore` EN ÉL, SE PONEN LOS NOMBRES DE LOS ARCHIVOS CON SU EXTENSIÓN O LOS NOMBRES DE LAS CARPETAS, LOS CUALES NO HARÁN PARTE DEL CONTROL DE CAMBIOS. EL ARCHIVO `.gitignore` SE PUEDE AGREGAR AL CONTROL DE CAMBIOS ASÍ:

```
git add .gitignore
```

REMOTO

CREAR UNA CONEXIÓN A UN REPOSITORIO REMOTO

```
git remote add <name> <url>
```

VERIFICAR SI EXISTE UN REPOSITORIO REMOTO

```
git remote -v
```

ELIMINAR UNA RAMA REMOTA

```
git push origin --delete rama_a_borrar
```

REMOVER UN REPOSITORIO REMOTO DE NUESTRA CONFIGURACIÓN LOCAL DE GIT

```
git remote remove nombrereporemoto
```

ENVIAR CAMBIOS A UN REPOSITORIO REMOTO

```
git push nombrereporemoto nombrerama
```

FUSIONAR CAMBIOS REMOTOS A NIVEL LOCAL

```
git pull nombrereporemoto nombrerama
```

GitHub:

- Plataforma basada en la nube que utiliza Git
- Aloja repositorios de código
- Proporciona herramientas adicionales para colaboración y gestión de proyectos

Ventajas de GitHub:

- Facilita la colaboración global
- Proporciona visibilidad y descubrimiento de proyectos
- Ofrece herramientas como issues, pull requests y acciones de CI/CD

Uso y diferencias entre Git y GitHub:

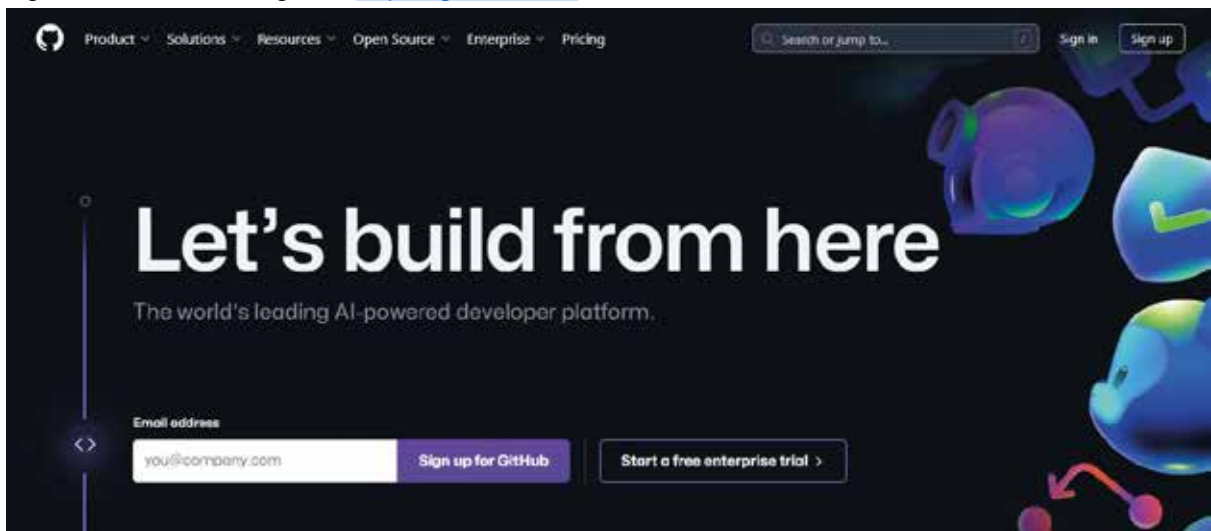
- Git se usa principalmente a través de la línea de comandos o GUI locales
- GitHub se usa a través de su interfaz web o aplicaciones de escritorio
- Git maneja el control de versiones, GitHub añade características sociales y de colaboración

Para crear un nuevo repositorio en GitHub:

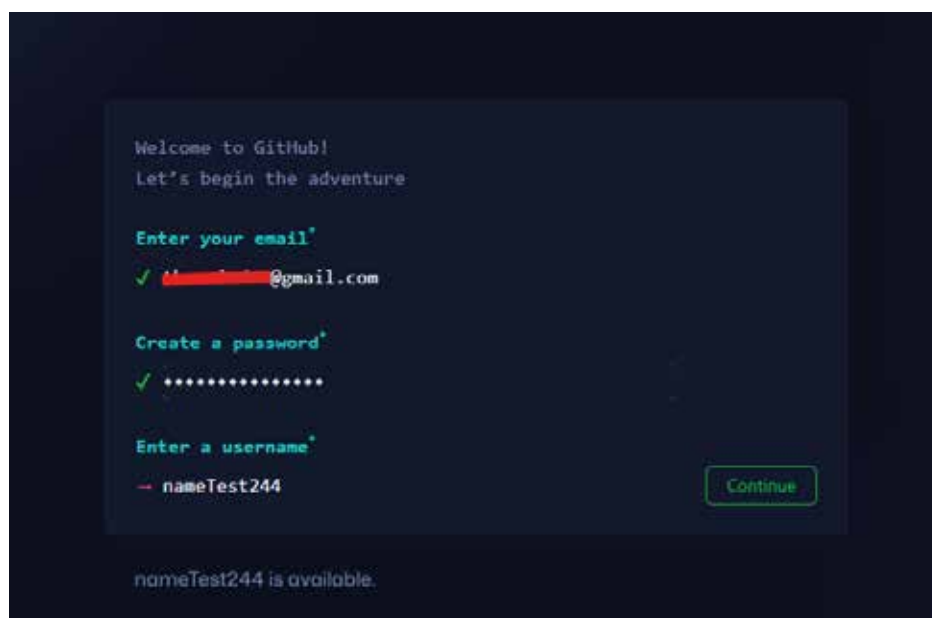
Primero nos registramos en <https://github.com/>

Para registrarnos en github

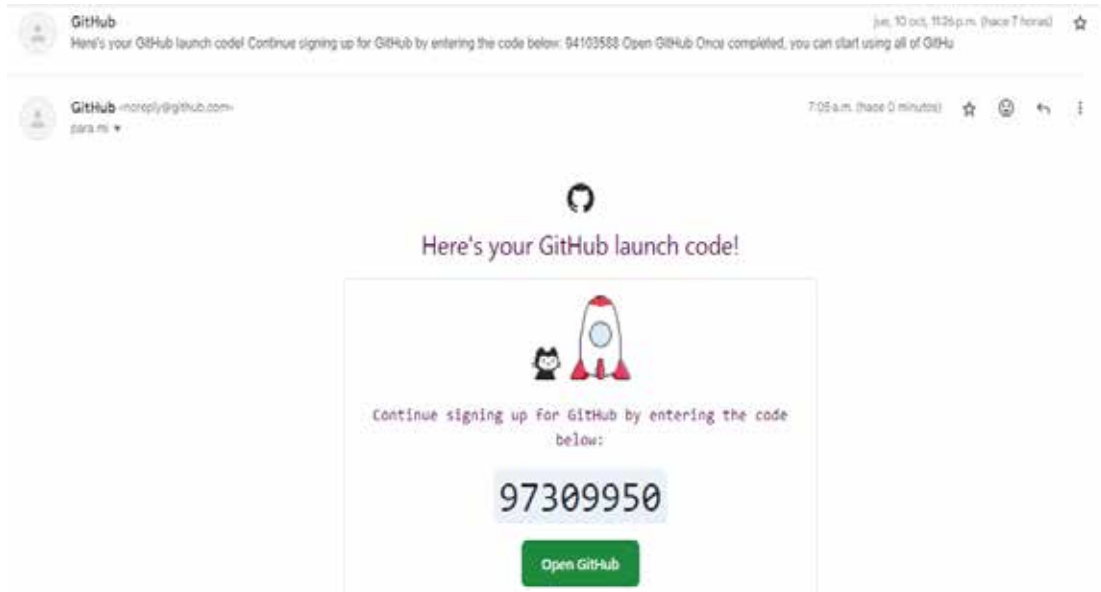
1. Ingresamos al sitio de github <https://github.com/>



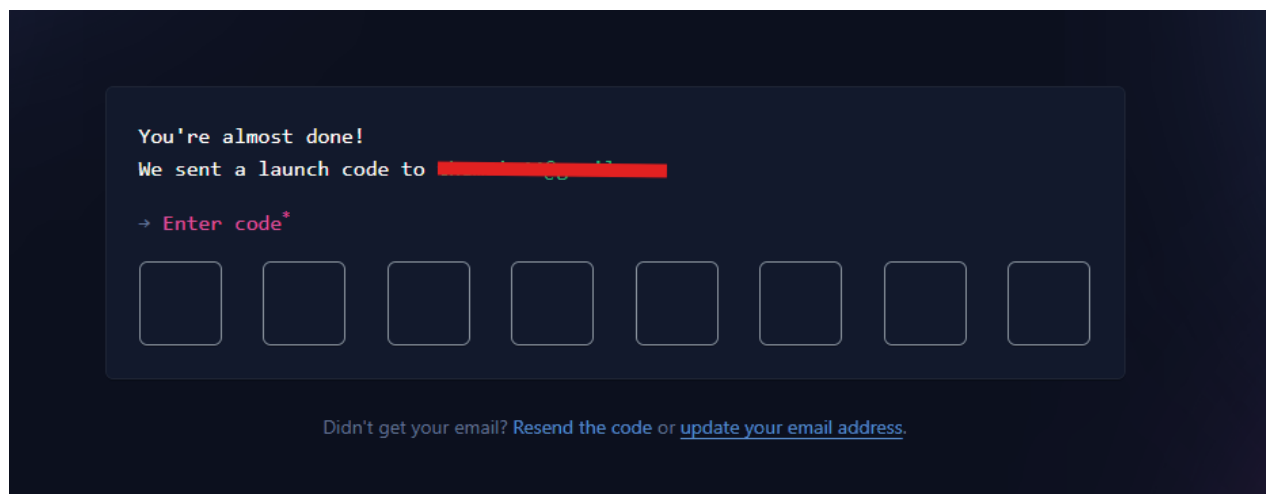
2. Presionamos “**Sign Up**” ingresamos los datos que este pide como correo, contraseña y nombre de usuario



3. Solucionaremos el captcha, y nos llegará un código al correo para verificar la cuenta

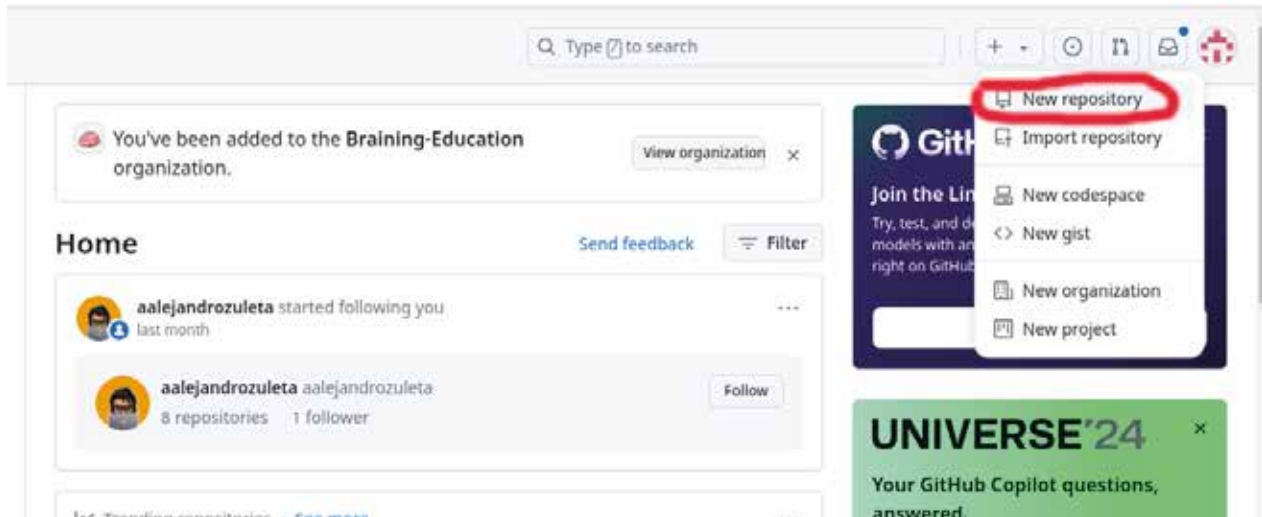


copiamos el código y lo pegamos en la siguiente ventana en GitHub



4. Luego nos redirigirá automáticamente a iniciar sesión donde ingresamos usuario y contraseña

Luego de iniciar sesión, crearemos un repositorio en github:



Asignamos el nombre al repositorio, lo crearemos privado para que sólo nosotros y nuestros colaboradores tengan acceso a él

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository](#).

Required fields are marked with an asterisk (*).

Owner * gargaritah / Repository name * primer-repo
primer-repo is available.

Great repository names are short and memorable. Need inspiration? How about [curly-potato](#) ?

Description (optional)

☐ Public
Anyone on the internet can see this repository. You choose who can commit.

☒ Private
You choose who can see and commit to this repository.

Initialize this repository with:
☐ Add a README file
This is where you can write a long description for your project. [Learn more about READMEs](#).

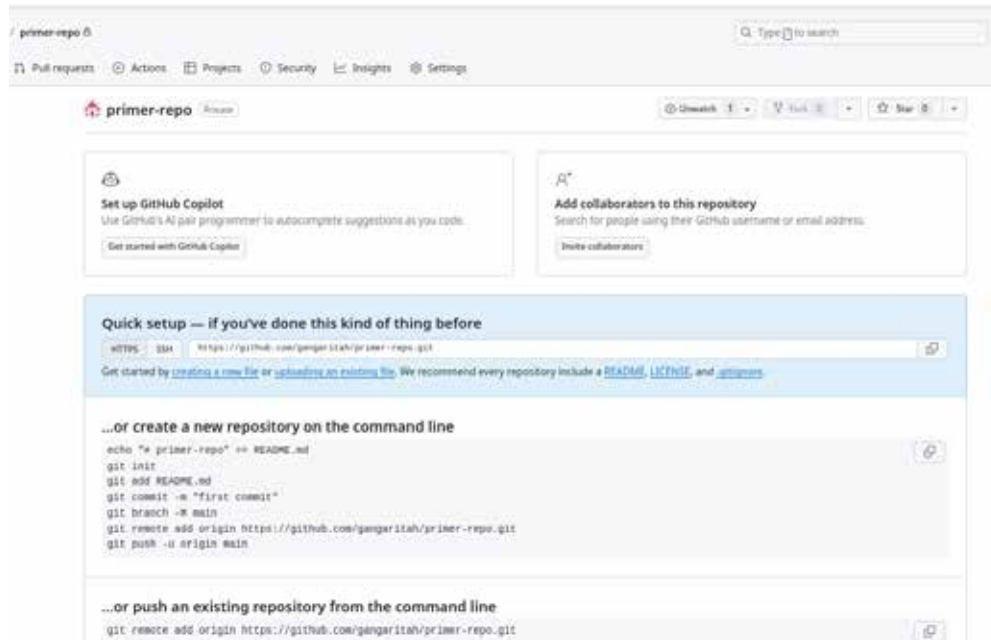
Add .gitignore
.gitignore template: None
Choose which files not to track from a list of templates. [Learn more about ignoring files](#).

Choose a license
License: None
A license tells others what they can and can't do with your code. [Learn more about licenses](#).

① You are creating a private repository in your personal account.

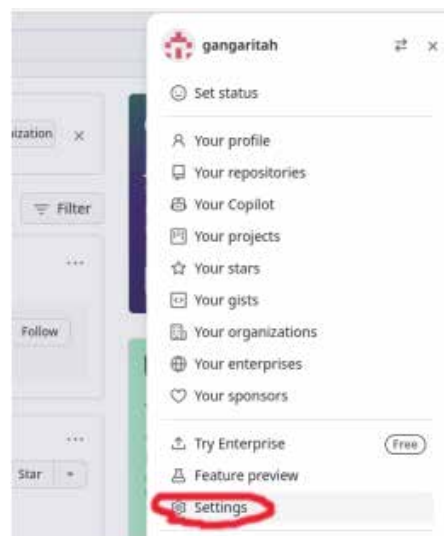
Create repository

Ya tenemos nuestro primer repositorio en Github

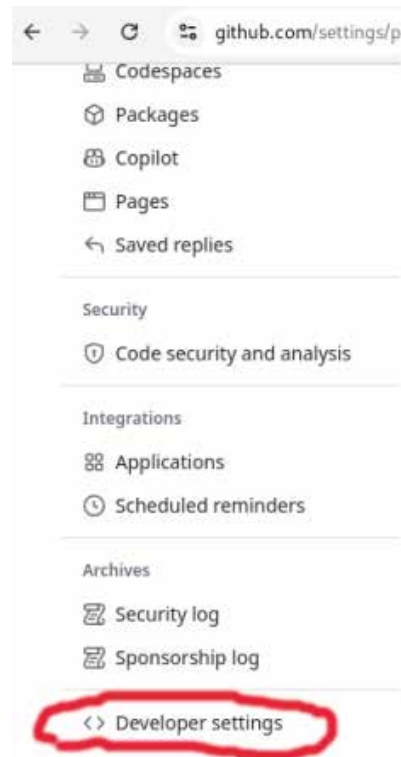


Ahora iniciaremos sesión en git

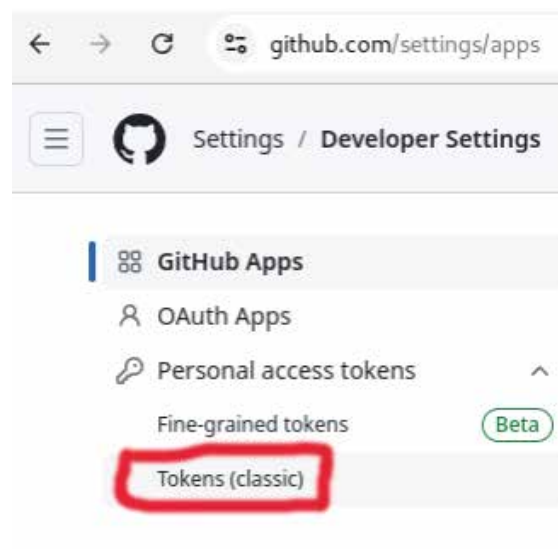
1. La manera de autenticarnos que usaremos en GitHub es usando un Token de acceso.
 - a. Para obtener nuestro Token debemos:
 - b. Hacemos login con GitHub
 - c. Vamos a configuración



d. Configuración de desarrollador



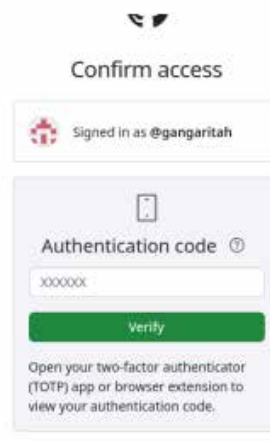
e. Seleccionamos el tipo de Token



f. Generamos un nuevo token



- g. En caso de que nos pida un código como segundo factor de autenticación, podemos usar la app Authenticator de Google la cual encontramos en PlayStore o AppStore de Apple



Google Authenticator

Google LLC

3.9★
516 k opiniones

100 M+
Descargas

Apto para todo público

Instalar en más dispositivos

Compartir



- h. Luego creamos el Token. Indicamos para qué es el token, el tiempo de expiración y los permisos.

New personal access token (classic)

Personal access tokens (classic) function like ordinary OAuth access tokens. They can be used instead of a password for Git over HTTPS, or can be used to [authenticate to the API over Basic Authentication](#).

Note

Token de prueba

What's this token for?

Expiration *

30 days

The token will expire on Wed, Sep 25 2024

Select scopes

Scopes define the access for personal tokens. [Read more about OAuth scopes](#).

<input checked="" type="checkbox"/> repo	Full control of private repositories
<input checked="" type="checkbox"/> repo:status	Access commit status
<input checked="" type="checkbox"/> repo_deployment	Access deployment status
<input checked="" type="checkbox"/> public_repo	Access public repositories
<input checked="" type="checkbox"/> repo:invite	Access repository invitations
<input checked="" type="checkbox"/> security_events	Read and write security events

i. Generamos el token, click en Generate token

- ☐ write:ssh_signing_key Write public user SSH signing keys
- ☐ read:ssh_signing_key Read public user SSH signing keys

Generate token

Cancel

j. Copiamos el token obtenido

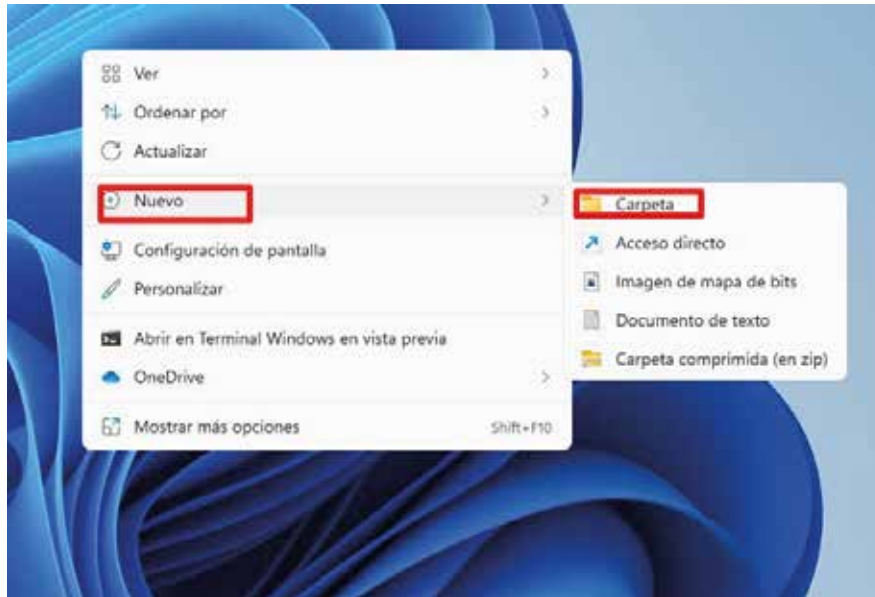
Tokens you have generated that can be used to access the [GitHub API](#).

Make sure to copy your personal access token now. You won't be able to see it again!

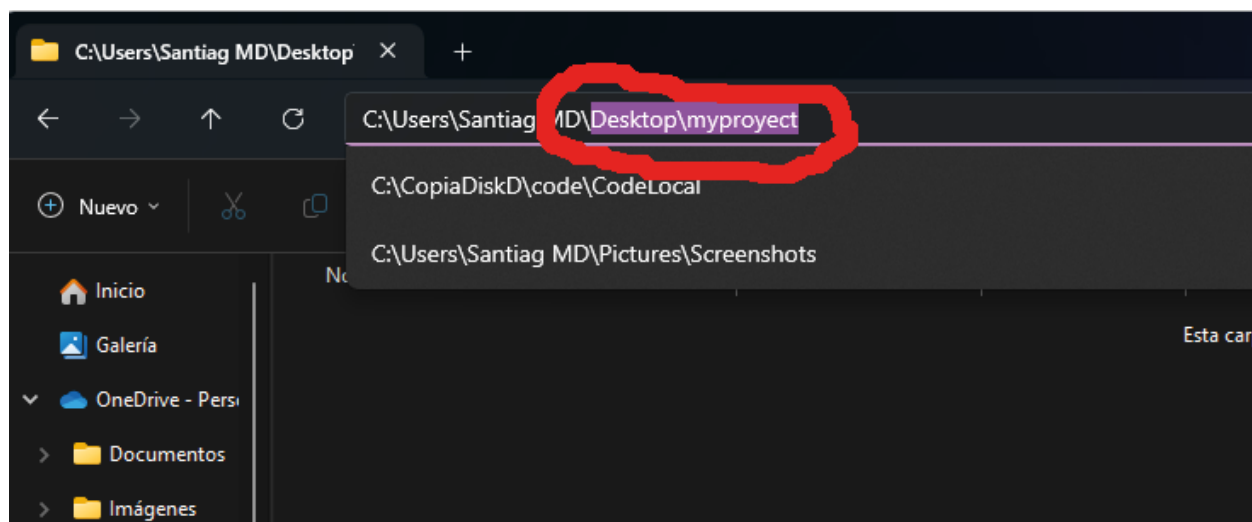
✓ ghp_tRscEGqHpb0vMZt [redacted]  [Delete](#)

Ahora pasaremos a crear nuestra carpeta en donde crearemos el programa

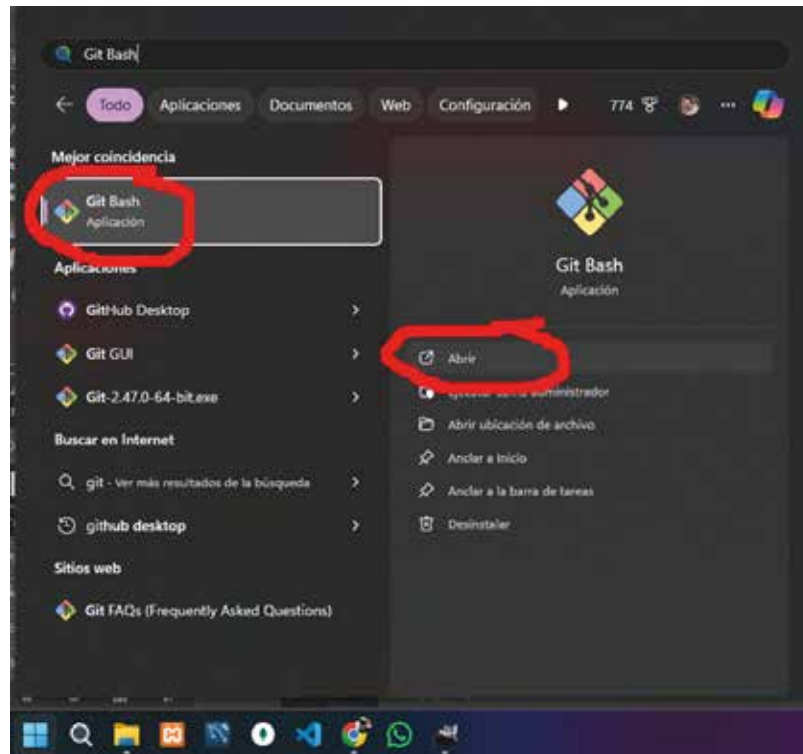
1. Crearemos la carpeta presionando **click derecho** en el escritorio seguido de **Nuevo y Carpeta** y nombraremos a la carpeta **myproyect**



2. Ahora daremos **doble click** en la carpeta creada lo cual nos llevará dentro de la carpeta luego copiamos la ruta resaltada en morado con las teclas **ctrl +c**



3. Luego buscaremos Git Bash y lo abrimos



4. Luego ingresamos **cd ruta_copiada** con **ctrl + v** pegaremos la ruta copiada anteriormente

```
MINGW64:/c/Users/Santiago M X + v
Santiago MD@DESKTOP-RHTCD9R MINGW64 ~
$ cd Desktop/myproyect/

Santiago MD@DESKTOP-RHTCD9R MINGW64 ~/Desktop/myproyect
$
```

5. luego iniciamos un repositorio git con **git init**
6. Ahora abriremos Visual Studio Code con el comando **code** . o lo podemos abrir desde menú inicio de Windows, cualquiera de las dos opciones está bien.

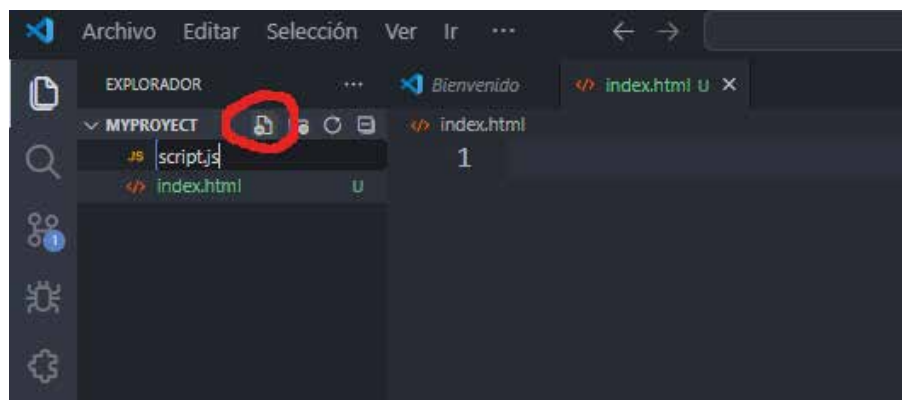

```
MINGW64: c:/Users/Santiago Iv x + v

Santiag MD@DESKTOP-RHTCD9R MINGW64 ~
$ cd Desktop/myproject/

Santiag MD@DESKTOP-RHTCD9R MINGW64 ~/Desktop/myproject
$ git init
Initialized empty Git repository in C:/Users/Santiago MD/Desktop/myproject/.git/

Santiag MD@DESKTOP-RHTCD9R MINGW64 ~/Desktop/myproject (master)
$ code .|
```

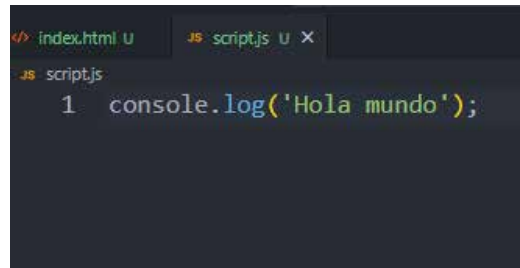
7. Luego creamos los archivos **index.html** y **script.js** con el botón señalado en rojo seguido del nombre del archivo



8. Luego agregamos esto al **index.html**

```
index.html x
index.html > script
1 <script src="script.js"></script>
```

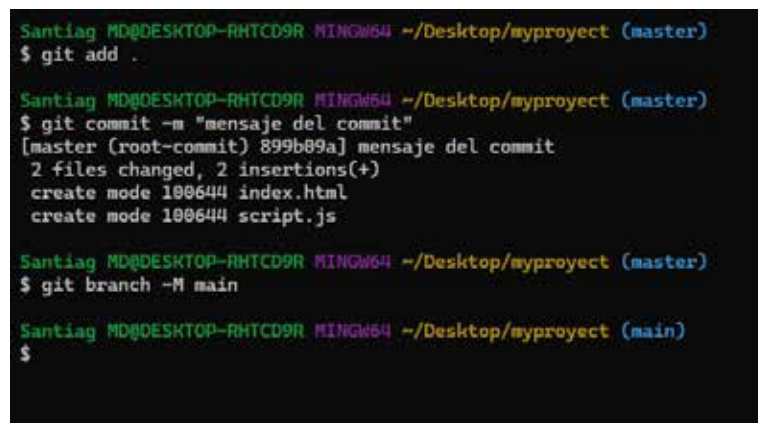
9. Luego agregamos esto al **script.js**



```
index.html U JS script.js U X
JS script.js
1 console.log('Hola mundo');
```

10. Luego en la consola de **git bash** que abrimos en el paso 3 y ubicamos en la carpeta en el paso 4 ingresamos los siguientes comandos:

- git add .**
- git commit -m "mensaje del commit"**
- git branch -M main**



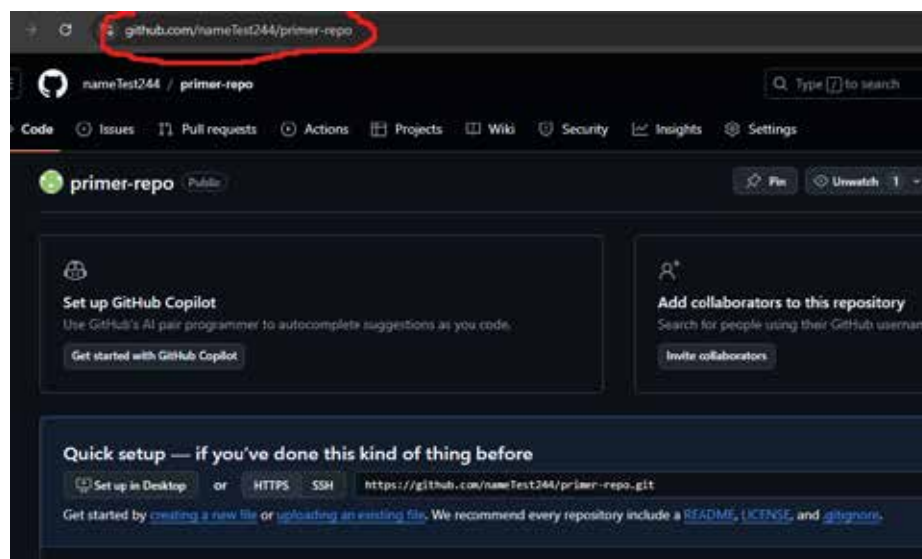
```
Santiago MD@DESKTOP-RHTCD9R MINGW64 ~/Desktop/myproyect (master)
$ git add .

Santiago MD@DESKTOP-RHTCD9R MINGW64 ~/Desktop/myproyect (master)
$ git commit -m "mensaje del commit"
[master (root-commit) 899b89a] mensaje del commit
2 files changed, 2 insertions(+)
create mode 100644 index.html
create mode 100644 script.js

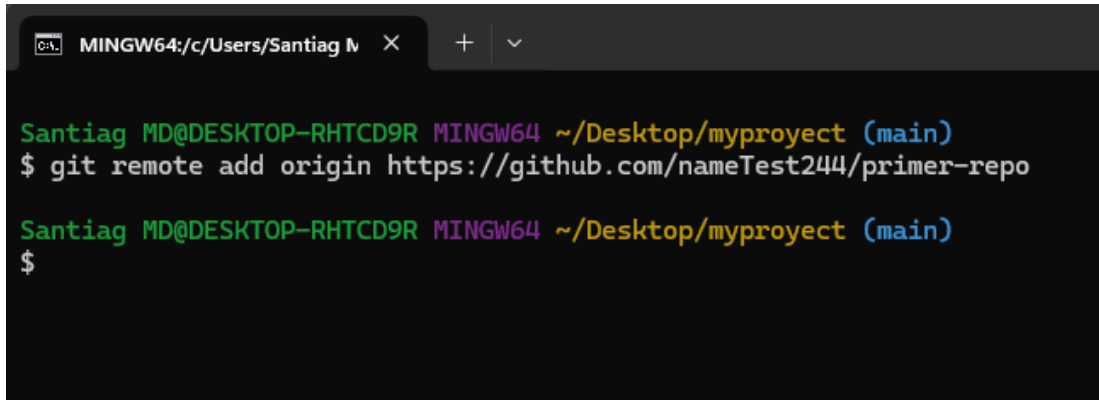
Santiago MD@DESKTOP-RHTCD9R MINGW64 ~/Desktop/myproyect (master)
$ git branch -M main

Santiago MD@DESKTOP-RHTCD9R MINGW64 ~/Desktop/myproyect (main)
$
```

11. De el repositorio anteriormente creado en **GitHub** copiaremos su **URL**

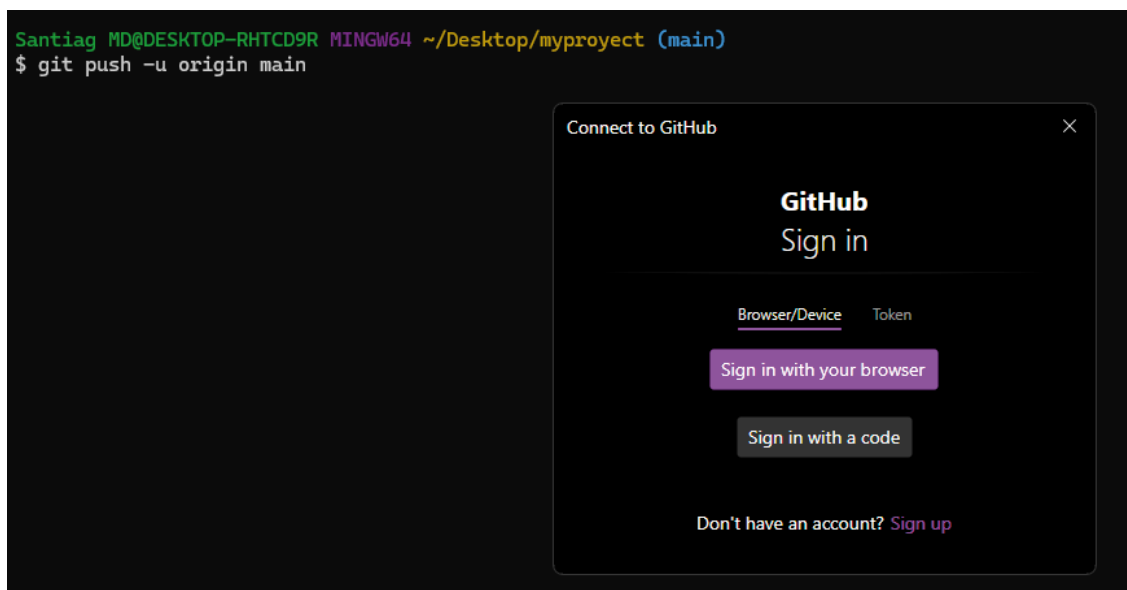


12. Luego conectamos al repositorio remoto en **GitHub** con el siguiente comando **git remote add origin url_copiada_del_repo_github**

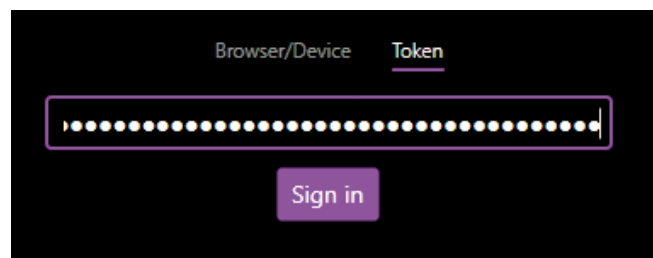


```
MINGW64:/c/Users/Santiago Iv X + v
Santiago MD@DESKTOP-RHTCD9R MINGW64 ~/Desktop/myproyect (main)
$ git remote add origin https://github.com/nameTest244/primer-repo
Santiago MD@DESKTOP-RHTCD9R MINGW64 ~/Desktop/myproyect (main)
$
```

13. Luego subiremos el código al repositorio **GitHub** con el siguiente comando **git push -u origin main** y nos pedirá el token que generamos anteriormente, seleccionamos la opción **Token**



14. Lo ingresamos y presionamos **Sign in**



15. Y el código se habrá subido a **GitHub**

```
Santiago MD@DESKTOP-RHTCD9R MINGW64 ~/Desktop/myproyect (main)
$ git push -u origin main
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 2 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (4/4), 312 bytes | 156.00 KiB/s, done.
Total 4 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To https://github.com/nameTest244/primer-repo
 * [new branch]      main -> main
branch 'main' set up to track 'origin/main'.
```

```
Santiago MD@DESKTOP-RHTCD9R MINGW64 ~/Desktop/myproyect (main)
$
```

16. Ingresamos a nuestro repositorio en GitHub y nuestro código se habrá subido

