



< Técnico en  
**DESARROLLO DE SOFTWARE** >

*Programación Avanzada*

(CC BY-NC-ND 4.0)  
International

Attribution-NonCommercial-NoDerivatives 4.0



## **Atribución**

Usted debe reconocer el crédito de una obra de manera adecuada, proporcionar un enlace a la licencia, e indicar si se han realizado cambios. Puede hacerlo en cualquier forma razonable, pero no de forma tal que sugiera que tiene el apoyo del licenciante o lo recibe por el uso que hace.



## **No Comercial**

Usted no puede hacer uso del material con fines comerciales.



## **Sin obra derivada**

Si usted mezcla, transforma o crea un nuevo material a partir de esta obra, no puede distribuir el material modificado.

No hay restricciones adicionales - Usted no puede aplicar términos legales ni medidas tecnológicas que restrinjan legalmente a otros hacer cualquier uso permitido por la licencia.

<http://creativecommons.org/licenses/by-nc-nd/4.0/>



# *Programación Avanzada*

## *Semana I*

### 1. Conociendo las herramientas

#### Visual Studio Code

Es un Editor de Código Fuente, desarrollado por Microsoft, es un proyecto de código abierto, es ligero y se puede extender con el uso de diferentes plugins, por lo que es muy versátil.

Algo interesante para el tema que tenemos a mano es que Visual Studio Code (En adelante VSC) fue desarrollado usando Electron.js

Se puede descargar desde aquí <https://code.visualstudio.com/download> .

#### Node.js

Es un entorno de JavaScript, (de ahí la terminación js) que permite ejecutar código de JS fuera de los navegadores web. Es un entorno asíncrono y orientado a eventos.

El primer término significa que Node es capaz de ejecutar código no secuencial, es decir código que no necesariamente se va a ejecutar de principio a fin como está escrito, que sea asíncrono significa que puede haber muchas funciones que se ejecutan al mismo tiempo y que estas pueden terminar antes o después no en el orden en el que se llamaron.

El segundo término significa que la forma en la que se va a ejecutar las funciones de un programa de Node es por distintos eventos que puedan ocurrir al igual que JS ¿qué significa esto? que las distintas funciones y procesos se van a ejecutar como respuesta a los diferentes eventos que ocasione, o el usuario, o las mismas aplicaciones. Por ejemplo, cuando alguien hace clic en un botón, tenemos una función preparada para manejar el evento clic, cuando alguien escribe algo y presiona enter, tenemos una función preparada también para esto, cuando el navegador hace una solicitud, también tenemos una función preparada para responder.

Se puede descargar desde aquí <https://nodejs.org/es/download/>

## Electron.js

Es un framework que trabaja sobre Node, haciendo uso de las características de Node, así como del diseño de las interfaces por medio de HTML y CSS nos permite crear aplicaciones de escritorio. estas aplicaciones son multiplataforma, es decir no importa si estamos utilizando Windows, Mac o Linux, nuestras aplicaciones funcionarán igualmente.

Una de las grandes ventajas de esta herramienta es que, al trabajar con JS, HTML y CSS, es capaz de integrarse con otras herramientas y frameworks diseñados para el desarrollo Web, cómo Express, Angular, React o incluso Bootstrap.

Para utilizar electron se agrega como un paquete a un proyecto de node, lo veremos más adelante.

## ¿Cómo funciona electron?

Primero hablemos de JS, es un lenguaje que funciona sobre las páginas web, creando contenido dinámico, manejando peticiones, cargando archivos, en resumidas cuentas funciona sobre el lado del cliente, es decir el visitante de la página web.

Node trabaja sobre JS, pero lo expande para que pueda funcionar sobre los servidores, es decir sobre quien nos provee el contenido de la web. Node es un lenguaje dirigido por eventos, lo que busca es poder responder a las distintas peticiones que puedan realizarse, o a las distintas situaciones que pueda afrontar el servidor.

Ahora, que es electron, electron utiliza Node, entonces también utiliza JS, electron es usado para desarrollar aplicaciones de escritorio.

¿como así? JS es un lenguaje de red, y Node es un lenguaje de servidor, entonces ¿cómo es que Electron desarrolla aplicaciones de escritorio?

Lo que hace electron es que las aplicaciones las desarrolla sobre Chromium, este es un navegador web, de código abierto, es una versión de Google Chrome.

Entonces Electron desarrolla aplicaciones como que fueran páginas web y luego lo que hace es ponerlas sobre una instancia de Chromium independiente que nos da como resultado una aplicación sobre un navegador web.

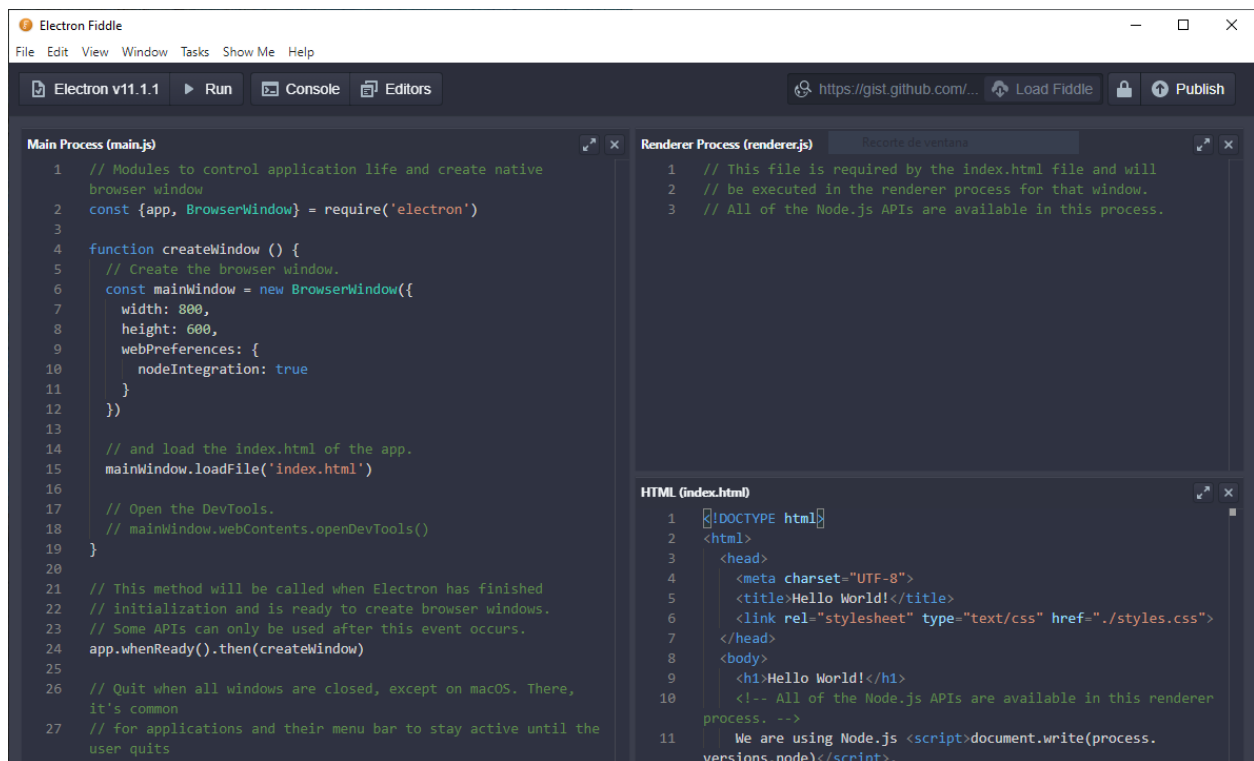
La idea general de Electron es que se trabaja cómo se trabaja la programación web, como estamos basados en Node, HTML y CSS, casi todo lo que hacemos acá se puede traducir a una aplicación web basada en una página web y un servidor

Adicionalmente, como Electron está basado en Node, se pueden utilizar herramientas desarrolladas para node como lo son Angular o React, Express o herramientas para desarrollo web como Bootstrap

## 2. Electron Fiddle

Es una herramienta que nos permite crear aplicaciones de electron sin necesidad de instalar las dependencias. Es una forma de no empezar desde cero, y lo mejor es que es posible exportar nuestro trabajo para reutilizarlo en cualquier otro lugar.

Al iniciar el programa podemos ver tres secciones en el editor.



```
Electron Fiddle
File Edit View Window Tasks Show Me Help

Electron v11.1.1 Run Console Editors
https://gist.github.com/... Load Fiddle Publish

Main Process (main.js)
1 // Modules to control application life and create native browser window
2 const {app, BrowserWindow} = require('electron')
3
4 function createWindow () {
5   // Create the browser window.
6   const mainWindow = new BrowserWindow({
7     width: 800,
8     height: 600,
9     webPreferences: {
10       nodeIntegration: true
11     }
12   })
13
14   // and load the index.html of the app.
15   mainWindow.loadFile('index.html')
16
17   // Open the DevTools.
18   // mainWindow.webContents.openDevTools()
19 }
20
21 // This method will be called when Electron has finished
22 // initialization and is ready to create browser windows.
23 // Some APIs can only be used after this event occurs.
24 app.whenReady().then(createWindow)
25
26 // Quit when all windows are closed, except on macOS. There,
27 // it's common
28 // for applications and their menu bar to stay active until the
29 // user quits
30 app.on('window-all-closed', () => {
31   // On macOS it's common for applications and their menu bar
32   // to stay active until the user quits explicitly with Cmd + Q
33   if (process.platform !== 'darwin') {
34     app.quit()
35   }
36 })
37 app.on('activate', () => {
38   // On macOS it's common for applications and their menu bar
39   // to stay active until the user quits explicitly with Cmd + Q
40   if (process.platform === 'darwin') {
41     createWindow()
42   }
43 })
```

```
Renderer Process (renderer.js)
1 // This file is required by the index.html file and will
2 // be executed in the renderer process for that window.
3 // All of the Node.js APIs are available in this process.
```

```
HTML (index.html)
1 <!DOCTYPE html>
2 <html>
3 <head>
4 <meta charset="UTF-8">
5 <title>Hello World!</title>
6 <link rel="stylesheet" type="text/css" href="../styles.css">
7 </head>
8 <body>
9 <h1>Hello World!</h1>
10 <!-- All of the Node.js APIs are available in this renderer
11 process. -->
12 We are using Node.js <script>document.write(process.
13 versions.node)</script>
```

El panel grande de la izquierda representa el llamado main process, es el que dice a la aplicación como se debe comportar al iniciar, al cerrar, etc. Controla el ciclo de vida de la aplicación, sin embargo, no controla el contenido de la ventana, de esto se encarga el renderer process.

La siguiente sección, el panel a la derecha en la parte de arriba, es el renderer process, éste es un proceso que maneja la parte visual de la aplicación, es decir cuando la aplicación ya tiene contenido en este proceso vamos a colocar los manejadores de eventos que controlan el comportamiento de la aplicación.

Finalmente, el panel de abajo a la derecha es el HTML, en él vamos a describir el contenido de la ventana de la aplicación, aquí colocamos los botones, los textos e imágenes.

## Manejo de paquetes con Node (npm)

npm es el manejador de paquetes de Node ¿Qué significa esto? Es una herramienta que nos permite instalar, actualizar, gestionar o eliminar paquetes o librerías que queramos utilizar con Node.

Una gran ventaja de utilizar esta herramienta es que nos permite gestionar automáticamente las dependencias que pueden surgir al descargar e instalar librerías.

A la hora de agregar módulos a un proyecto de node, nos va a aparecer una carpeta llamada node\_modules, en esta carpeta es donde se encuentran todos los archivos que requieren los paquetes y las librerías para ser utilizadas

Viene incluido con la instalación de Node, podemos verificarlo con el comando `npm -v`

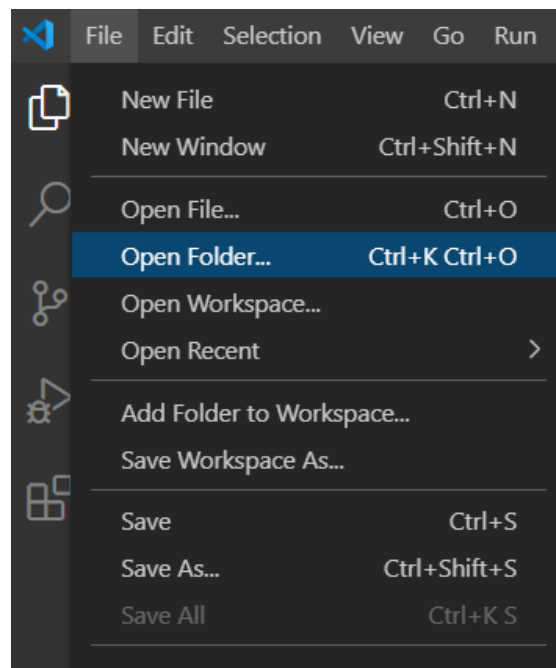
### 3. Creando un proyecto en Node

Vamos a empezar creando una carpeta en la que vamos a tener todos nuestros proyectos. Luego dentro de esa carpeta vamos a crear otra carpeta.

El proyecto de ejemplo que vamos a crear es una calculadora simple, entonces vamos a nombrar esta carpeta calculadora.

Luego podemos abrir la carpeta en VSC haciendo click derecho sobre la carpeta y vamos a elegir la opción de abrir con Visual Studio Code

También se puede abrir la carpeta desde VSC en el menú `File > Open Folder`





Nos interesa abrir la carpeta para que podamos visualizar todos los archivos.

Ahora, tenemos VSC abierto, y podemos ver en el panel de la izquierda que estamos trabajando en nuestra carpeta

Lo primero que tenemos que hacer es abrir una terminal o una consola de comandos.(dentro de la carpeta) Visual Studio Code tiene una terminal integrada que podemos utilizar.

Ahí vamos a colocar el comando...

```
npm init
```

Este comando hace uso de npm (el manejador de paquetes de node) para inicializar el archivo que describe el proyecto, también podemos usar...

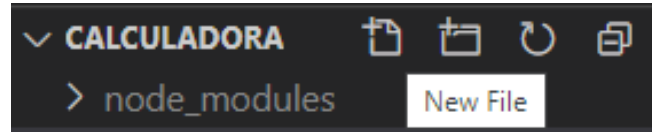
```
npm init -y
```

para iniciar el archivo package.json con los valores por defecto

## ¿Qué es el archivo package.json?

Es el archivo que describe el proyecto, el nombre, los autores, las dependencias, es muy similar a un archivo pom.xml

Ahora vamos a necesitar 4 archivos diferentes, en el panel izquierdo podemos agregar archivos nuevos, vamos a nombrarlos de la siguiente manera:



*main.js*

*index.html*

*index.js*

*style.css*

También vamos a ir al archivo package.json y vamos a cambiar el nombre del archivo principal, de index.js a main.js. Adicionalmente vamos a colocar un script de inicio para que se pueda inicializar la aplicación de electron con npm.

Tenemos que colocar el script

```
"start": "electron ."
```

El archivo de package.json debería verse así

```
{  
  "name": "calculadora",  
  "version": "1.0.0",  
  "description": "",  
  "main": "main.js",  
  "scripts": {  
    "start": "electron ."  
  },  
}
```

```
"author": "",  
"license": "ISC"  
}
```

podemos quitar el script "test" del json ya que no estamos realizando pruebas y finalmente, vamos a usar el siguiente comando en la línea de comandos

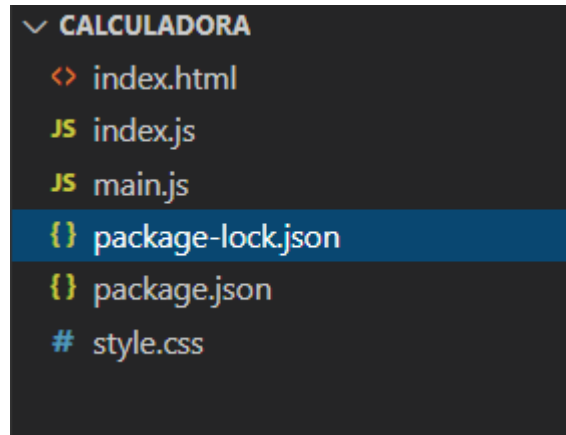
```
npm i --save-dev electron
```

Esto va a instalar la dependencia de electron en nuestro proyecto.

Veamos qué significa cada parte de este comando:

- npm: invoca la instrucción por medio del manejador de paquetes
- i es una forma abreviada de decir instalar, la palabra install también funciona pero es más común usar i por brevedad
- --save-dev significa que esta dependencia solo nos va a servir mientras estemos desarrollando, a la hora de empaquetar o distribuir la aplicación esta no va a estar incluida. ¿cuáles librerías se van a instalar normalmente y cuáles se tienen que instalar con --save-dev? Eso depende de cada librería, y por eso tenemos que leer las instrucciones de cada una, por ejemplo en la [guía de instalación](#) de electron nos dice que usemos save-dev. Podemos notar que cambiar el orden de las instrucciones de instalación: el nombre del paquete se puede colocar antes de la instrucción de save-dev o viceversa

Al final nuestra estructura de proyecto se debería ver así:



Ahora vamos a trabajar sobre los diferentes archivos para darle forma a nuestro proyecto.

## El archivo HTML

### ¿Qué son los archivos HTML?

Son archivos que describen el contenido de una página web. En ellos encontramos todos los elementos, como imágenes, bloques de texto, botones, tablas y listas, están estructurados de una forma jerárquica, y por medio de etiquetas.

En general hay 2 etiquetas obligatorias, el head, que describe el contenido de la página y el body que contiene el contenido relacionado al HTML se encuentra el DOM, o Document Object Model, es un API que muestra el contenido del archivo HTML, nos será de mucha utilidad para programar los eventos relacionados al archivo HTML

Veamos algunas de las etiquetas que podemos esperar encontrar en un archivo html

<i>etiqueta</i>	<i>descripcion</i>
-----------------	--------------------

<code>&lt;DOCTYPE html&gt;</code>	Le dice al navegador o aplicación que tipo de archivo es. Nos dice el formato y las etiquetas que podemos esperar encontrar en el archivo
<code>&lt;html&gt;&lt;/html&gt;</code>	Es la etiqueta principal de un archivo html, contiene todas las demás etiquetas dentro del archivo. Es la base de todos los archivos html
<code>&lt;head&gt;&lt;/head&gt;</code>	contiene información sobre el documento, como el idioma del contenido, el alfabeto de caracteres que utiliza, metadata sobre el archivo, como tamaño del mismo o palabras clave
<code>&lt;body&gt;&lt;/body&gt;</code>	es la etiqueta que va a contener todo el contenido de la pagina, lo que seran imágenes, textos, enlaces, scripts o botones, entre otros

Junto a estas etiquetas podemos encontrar otras más que son usadas para describir el contenido de las páginas como, por ejemplo

<i>etiqueta</i>	<i>descripcion</i>
<code>&lt;title&gt;&lt;/title&gt;</code>	Sirve para mostrar el título de la pagina o aplicacion
<code>&lt;script&gt;&lt;/script&gt;</code>	Describe un script o una secuencia de

	<p>comandos a ejecutar.</p> <p>Puede ser tanto un script directamente descrito en la etiqueta, o referencia a un script externo</p>
<code>&lt;div&gt;&lt;/div&gt;</code>	<p>es un contenedor, no tiene significado particular</p>
<code>&lt;h1&gt;&lt;/h1&gt;</code>	<p>Es una forma de denotar un heading o un título, empezando en h1, tenemos varios tipos de importancia en los títulos, h2, h3, h4, h5 y h6.</p>
<code>&lt;input &gt;</code>	<p>Es una etiqueta que define un campo de entrada, puede ser un campo de texto, una checkbox, o un grupo de radio buttons. Esta etiqueta no tiene etiqueta de cierre</p>
<code>&lt;button&gt;&lt;/button&gt;</code>	<p>Define un botón sobre el cual podemos hacer clic</p>

Además de estas existen muchas otras etiquetas, podemos encontrar una referencia completa en [w3schools](https://www.w3schools.com/) o en la página de [mozilla](https://developer.mozilla.org/).

Vamos a crear una interfaz para la calculadora, primero la base que contiene las etiquetas más elementales...

```
<!DOCTYPE html>

<html>

  <head>

    <title>Calculadora</title>

  </head>

  <body>


    <script>


    </script>

  </body>

</html>
```

Dejaremos una etiqueta script para luego vincular el archivo que vamos a utilizar.

Ahora colocaremos algunos div para identificar la calculadora y colocaremos algunos elementos que nos servirán para mostrar el resultado

Dentro de la etiqueta body

```
<div id="calculadora">

  <div class="numeroDisplay" id="numeroActual">0</div>

  <div class="numeroDisplay" id="resultado">0</div>

  <div id="botones" >

    <button class="botonCalc" id="botonUno">1</button>

    <button class="botonCalc" id="botonDos">2</button>

    <button class="botonCalc" id="botonTres">3</button>
```

```
</div>
```

```
</div>
```

```
<script>
```

```
</script>
```

Colocamos dos div en donde vamos a mostrar el valor actual que se está ingresando y el valor del resultado.

Luego colocamos un div más para identificar dónde van a ir los botones y dentro del div empezamos a colocar todos los botones que vamos a utilizar

Así para todos los botones numéricos y algunas operaciones

El contenido completo se vería así:

```
<body>
```

```
<div id="calculadora">
```

```
<div class="numeroDisplay" id="numeroActual">0</div>
```

```
<div class="numeroDisplay" id="resultado">0</div>
```

```
<div id="botones" >
```

```
<button class="botonCalc" id="botonUno">1</button>
```

```
<button class="botonCalc" id="botonDos">2</button>
```

```
<button class="botonCalc" id="botonTres">3</button>
```

```
<button class="botonCalc" id="botonCuatro">4</button>
```

```
<button class="botonCalc" id="botonCinco">5</button>
```

```
<button class="botonCalc" id="botonSeis">6</button>
```

```
<button class="botonCalc" id="botonSiete">7</button>
```



```
<button class="botonCalc" id="botonOcho">8</button>

<button class="botonCalc" id="botonNueve">9</button>

<button class="botonCalc" id="botonCero">0</button>


<button class="botonOp" id="suma">+</button>

<button class="botonOp" id="resta">-</button>

<button class="botonOp" id="division">/</button>

<button class="botonOp" id="mult">x</button>

</div>

</div>


<script type="text/javascript" src="./index.js"></script>

</body>
```

Colocamos una etiqueta script en el archivo html. Podemos recordar que se pueden usar archivos externos al HTML, indicando la ruta de estos en el atributo src de una etiqueta script.

Con eso ya estaría los componentes generales de la calculadora

## El renderer Script

¿Qué son los scripts?

un archivo de javascript, es un archivo de código contiene variables y funciones, y más importante eventos, en un archivo de javascript podemos colocar funciones que

se ejecutarán cuando se cumpla un evento en particular. cuando se coloca el cursor sobre un elemento, cuando se presiona una tecla del teclado.

Es posible manejar las propiedades del DOM desde el archivo de JS. agregar elementos, modificarlos o incluso eliminarlos

El script render es el script relacionado a la pagina HTML, en electron lo conoceremos como render porque es el encargado de renderizar o de manejar la representación gráfica de las páginas de la aplicación. Este funciona exactamente igual que un script de JS en una página web por lo que podemos acceder a los elementos de HTML.

El elemento al que queremos acceder es al DOM, este es un objeto que representa el archivo HTML, y nos provee métodos y propiedades para manipularlo.

Por ejemplo, para acceder al botonUno, podemos hacer lo siguiente

```
document.getElementById('botonUno')
```

Esto nos da una referencia a el botonUno. Ahora podemos: Guardarlo en una variable

```
var boton1 = document.getElementById('botonUno')
```

o colocarle un eventListener

```
document.getElementById('botonUno').addEventListener('click', clickUno)
```

El eventListener es una función que espera a que ocurra un evento sobre el elemento especificado, en este caso estamos esperando el evento click sobre el botonUno, y cuando ocurra vamos a llamar a la funcion clickUno.

Esto se puede realizar también con el método onclick de la siguiente forma:

```
document.getElementById('botonUno').onclick = clickUno
```

Ahora vamos a obtener los campos de valor actual y resultado que es donde irán nuestros números e inicializamos unas variables que nos permitirán escribir los resultados a pantalla.

```
var actualElemento = document.getElementById('numeroActual')  
var resultadoElemento = document.getElementById('resultado')  
  
var actual = "  
var resultado = 0
```

La variable actual será un String porque nos interesa concatenar o combinar los números, no sumarlos, el resultado si sera un numero porque vamos a sumar

La función clickUno se vería así:

```
function clickUno(){  
    actual += 1  
    actualElemento.innerHTML = actual  
}
```

Ahora, al hacer click el el botonUno, se actualiza el número actual en el div correspondiente

Claro, los números solos no hacen una calculadora, vamos a agregarle funcionamiento tambien al boton de suma :

```
document.getElementById('suma').addEventListener('click', clickSuma)

function clickSuma(){

    if(actual != ""){

        resultado += parseInt(actual)

        actual = ""

        actualElemento.innerHTML = '0'

        resultadoElemento.innerHTML = resultado

    }

}
```

En el caso de la suma, tenemos que tener cuidado, como estamos usando un String para concatenar los valores, puede darse el caso de que alguien no haga clic en un número antes de la suma, por lo que el string vacío nos va a dar una suma no válida.

Tenemos que verificar antes de realizar la suma, que el valor sea válido

Agreguemos también funcionalidad al botonDos:

```
document.getElementById('botonDos').addEventListener('click', clickDos)

function clickDos(){
```

```
    actual += 2

    actualElemento.innerHTML = actual
}
```

Así podemos ir agregando funcionalidad a todos los botones.

Hasta el momento, el archivo se vería así:

```
document.getElementById('botonUno').addEventListener('click', clickUno)
document.getElementById('botonDos').addEventListener('click', clickDos)
document.getElementById('suma').addEventListener('click', clickSuma)
var actualElemento = document.getElementById('numeroActual')
var resultadoElemento = document.getElementById('resultado')
var actual = ""
var resultado = 0
function clickUno(){
    actual += 1
    actualElemento.innerHTML = actual
}
function clickDos(){
    actual += 2
    actualElemento.innerHTML = actual
}
function clickSuma(){
    if(actual != ""){
        resultado += parseInt(actual)
```

```
    actual = "  
  
    actualElemento.innerHTML = '0'  
  
    resultadoElemento.innerHTML = resultado  
  }  
}
```

## El main script

Este es el archivo que dice como se ejecuta la aplicación, que hace cuando se inicia y como se comporta al cerrar. Este archivo no tiene contacto directo con los HTML o con las hojas de estilo.

Lo que hacemos en este script, es, primero, importar algunas características de electron con las siguiente instrucción:

```
const {app, BrowserWindow} = require('electron')
```

Luego vamos a construir la ventana, con ayuda de una función y la propiedad BrowserWindow que acabamos de importar.

```
function createWindow(){  
  
  const ventana = new BrowserWindow({  
  
    width : 350,  
  
    height : 250,  
  
    webPreferences: {  
  
      nodeIntegration: true,
```

```
contextIsolation: false  
  
}  
  
})  
  
ventana.loadFile('index.html')  
  
}
```

esta función hace lo siguiente

- crea una instancia de BrowserWindow
  - Le pasamos parámetros colocándolos entre llaves dentro del paréntesis  
BrowserWindow ({})
- los parámetros que le pasamos son
  - width o el ancho de la ventana
  - height o el alto de la ventana
  - webPreferences, tiene dos propiedades que básicamente nos permite usar todas las funciones de node dentro de nuestra página web. No es la manera correcta, veremos como se hace correctamente más adelante. Por simplicidad, lo usaremos así por el momento.
- finalmente vamos a cargar el archivo index.html a la ventana que acabamos de crear

Finalmente vamos a llamar un método de app, la otra propiedad que importamos:

```
app.whenReady().then(createWindow)
```

Esta línea va a esperar que el entorno de app termine de cargar, y luego va a llamar nuestra función que crea la ventana.

En este punto podemos ejecutar nuestra aplicación (desde la terminal o línea de comandos) con la instrucción `npm start`

## Los archivos CSS

¿Qué son los archivos CSS?

El css es un archivo que contiene propiedades visuales que describen la apariencia de los distintos objetos del HTML

Describe propiedades como lo son color de texto, fuente de letras, colores de fondos y mucho más.

El archivo que creamos con anterioridad, también hay que vincularlo al HTML para poder usarlo, lo haremos colocando la siguiente línea dentro de la etiqueta `<head>`.

```
<link rel="stylesheet" href="./style.css"/>
```

Lo que dice aquí es que vamos a colocar, o enlazar, una hoja de estilos, y el href indica la ruta donde se encuentra la hoja en cuestión.

Ahora ya podemos empezar a crear estilos para nuestro archivo HTML.

Lo primero que vamos a hacer es colocar los números en forma de cuadrícula, esto lo haremos colocando una clase al div que tiene id botones, que se llamara "grid-container"

```
<div id="botones" class="grid-container">
```



y luego en el archivo style.css colocamos los estilos que nos permitirán mostrar los botones como cuadrícula.

```
.grid-container{  
    display: grid;  
    grid-template-columns: auto auto auto auto;  
}
```

La primera propiedad nos dice que la forma en la que queremos desplegar los elementos que encontremos dentro de esta clase es en forma de cuadrícula.

la segunda define la cantidad de columnas que se encuentran en la cuadrícula, y podemos colocar también el ancho de cada columna. auto significa que se ajusta al contenido de estas.

Nótese que también podríamos haber colocado estas propiedades directamente asociadas al id de botones:

```
#botones{  
    display: grid;  
    grid-template-columns: auto auto auto auto;  
}
```

Con esto ahora nos aparecen los botones en forma de cuadrícula, pero los vamos a reorganizar para que se muestran en una forma más parecida a los teclados. Es decir, 7, 8 y 9 en la fila de arriba, 4 5 y 6 en la de en medio y 1, 2 y 3 en la de abajo.

```
<div id="botones" class="grid-container">

  <button class="botonCalc" id="botonSiete">7</button>
  <button class="botonCalc" id="botonOcho">8</button>
  <button class="botonCalc" id="botonNueve">9</button>
  <button class="botonOp" id="suma">+</button>
  <button class="botonCalc" id="botonCuatro">4</button>
  <button class="botonCalc" id="botonCinco">5</button>
  <button class="botonCalc" id="botonSeis">6</button>
  <button class="botonOp" id="resta">-</button>
  <button class="botonCalc" id="botonUno">1</button>
  <button class="botonCalc" id="botonDos">2</button>
  <button class="botonCalc" id="botonTres">3</button>
  <button class="botonOp" id="division">/</button>

  <div></div>

  <button class="botonCalc" id="botonCero">0</button>

  <div></div>

  <button class="botonOp" id="mult">x</button>

</div>
```

También vamos a agregar unos div junto al botonCero para colocar espacios y que esté alineado

Con estos tres elementos, el JS, el HTML y el CSS tenemos una separación de funciones, de contenido, comportamientos y apariencia

La ventaja de separar estos elementos, ya la hemos visto antes, nos permite tener la lógica separada, si queremos, por ejemplo cambiar los colores de la página no es necesario buscar cada elemento en el HTML, si queremos cambiar, el comportamiento de los botones y de las áreas de texto, se puede hacer desde el JS. Y estos son intercambiables y combinables, es decir, podemos cambiar totalmente los componentes, el HTML, el CSS o el JS, o podemos agregar nuevos archivos JS o CSS que contengan comportamientos o estilos nuevos sin afectar el comportamiento de los que ya teníamos definidos.

## Usando otros paquetes de Node

Igualmente que con maven, podemos buscar paquetes útiles para nuestra aplicación en <https://www.npmjs.com>

Vamos a utilizar un paquete llamado [mousetrap](#), que nos facilita obtener los eventos del teclado, con el fin de darle el funcionamiento esperado a la calculadora.

Desde la terminal de VSC vamos a instalar el paquete

```
npm i mousetrap
```

Luego en el archivo index.js lo importamos

```
const mousetrap = require('mousetrap')
```

Finalmente podemos agregar eventos a las teclas del teclado al usar la aplicación de esta manera:

```
mousetrap.bind('1',clickUno)  
mousetrap.bind('2',clickDos)  
mousetrap.bind('+',clickSuma)
```

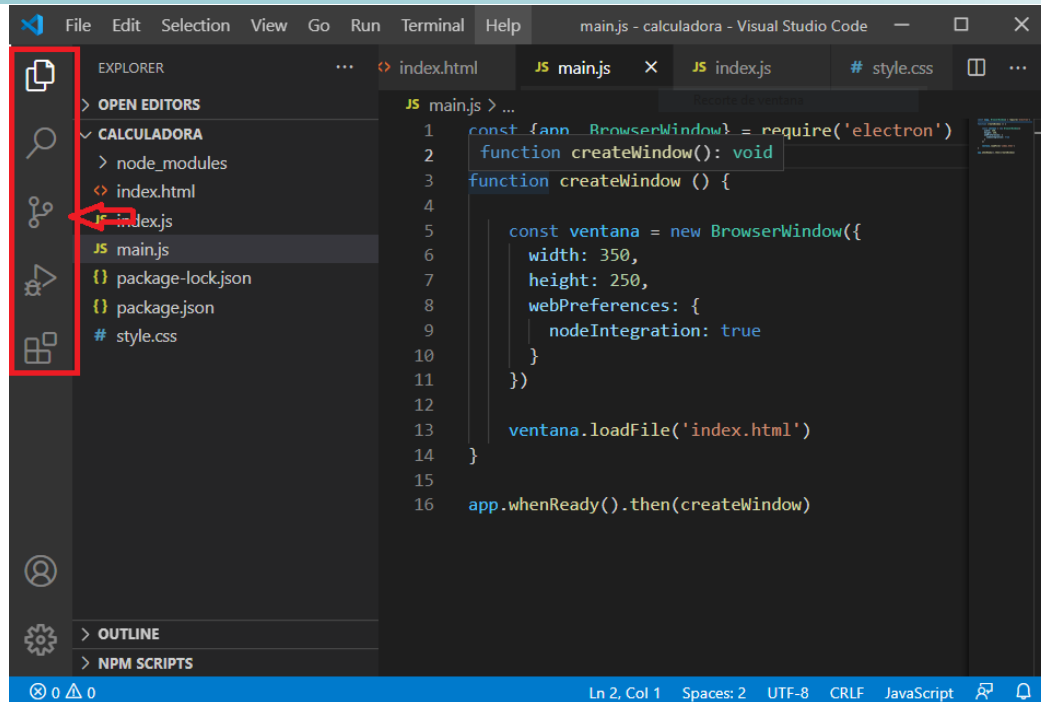
#### 4. Agregando control de versiones desde el editor

Agregando control de versiones al IDE

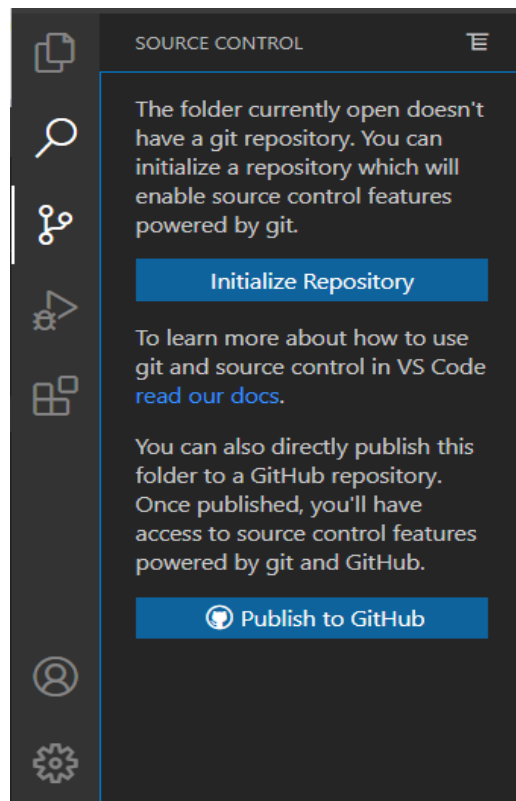
Ya sabemos cómo trabajar con control de versiones, agregar un repositorio para llevar el historial de nuestros proyectos, sin embargo lo hemos realizado por medio de la línea de comando, lo que puede resultar algo tedioso, aprender y repetir los mismos pasos cada vez que queremos realizar un cambio.

La mayoría de los IDEs modernos tienen la capacidad de gestionar el control de versiones por nosotros, nos permite manejar los add, commit, push y pulls que nos puedan interesar realizar, sin necesidad de recordar los comandos o de usar la línea de comandos.

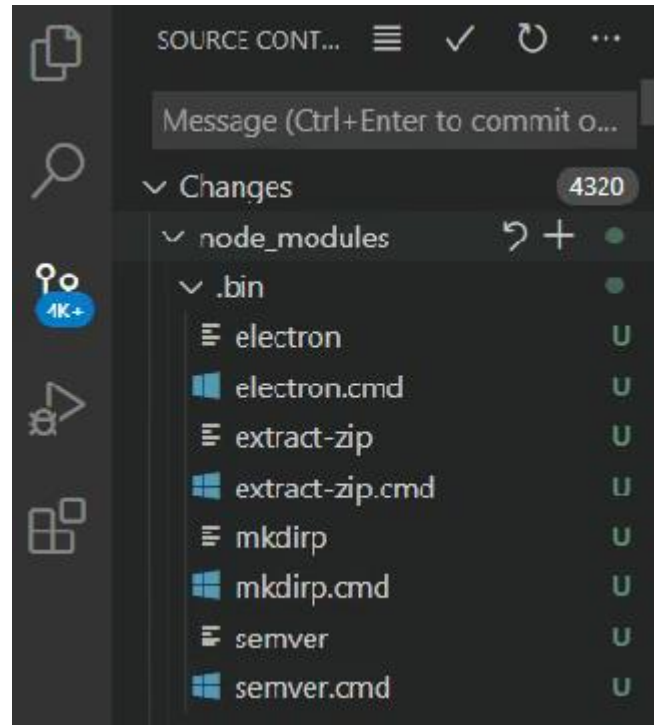
Otra ventaja que tiene VSC es que nos muestra el contexto de lo que estamos trabajando, la branch en la que estamos, la cantidad de cambios que tenemos, incluso nos muestra sobre el editor las líneas del archivo que han sido modificadas. Para agregar Control de Versiones a VSC tenemos que hacer clic en el tercer icono de la barra de menú izquierda:



Luego hacemos clic en el botón de inicializar repositorio (Esto es equivalente a hacer un git init)



El problema es que esto luego nos va a mostrar una gran cantidad de cambios



El problema es que está contando todos los archivos de las librerías que agregamos, y no queremos eso.

Para solucionarlo vamos a agregar un archivo llamado .gitignore

## ¿Qué es el gitignore?

Es un archivo especial que funciona con el sistema de control de versiones de git.

Lo más común es que con git agreguemos todos los archivos de nuestra carpeta y los enviamos al sistema de control de versiones.

El problema es que si tenemos alguna carpeta o algunos archivos que no queramos enviar, entonces tenemos que seleccionar manualmente los archivos que si queremos, esto se puede volver muy tedioso en caso de que los archivos están distribuidos en varias carpetas

un gitignore es un archivo que le dice a git, que archivos no ha de incluir en los push al control de versiones. Esto, vamos a ver, es muy útil, para no incluir archivos de, por ejemplo contraseñas o configuraciones, librerías y archivos que se usan durante el desarrollo, o en el caso de Node, las carpetas de Node\_modules, que incluyen los archivos de las librerías

Incluir estos archivos puede resultar muy pesado y acabaríamos con un proyecto muy grande, en particular como Node tiene un gestor de paquetes (npm) es posible descargar la definición del proyecto y luego dejar que el gestor de paquetes se encargue de descargar las librerías necesarias.

Es muy importante agregar el gitignore al trabajar con proyectos grandes.

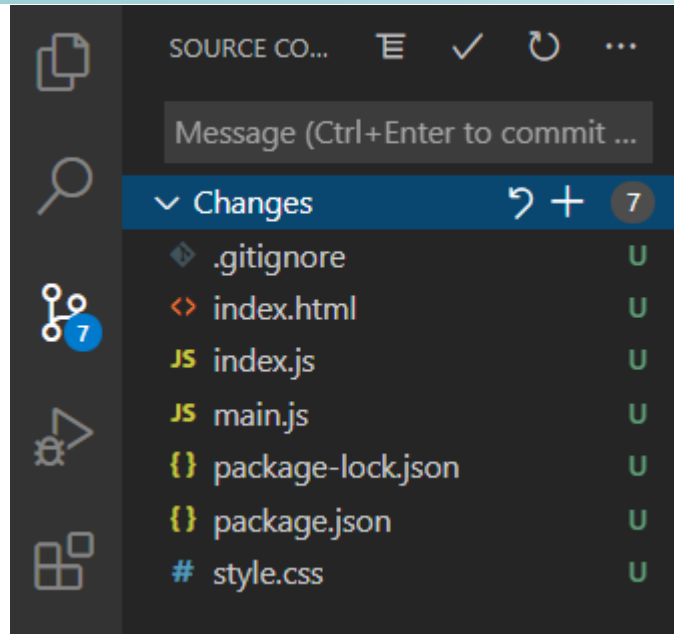
Agregamos un archivo nuevo y le colocamos de nombre .gitignore y dentro de él colocamos una sola línea.

```
node_modules/
```

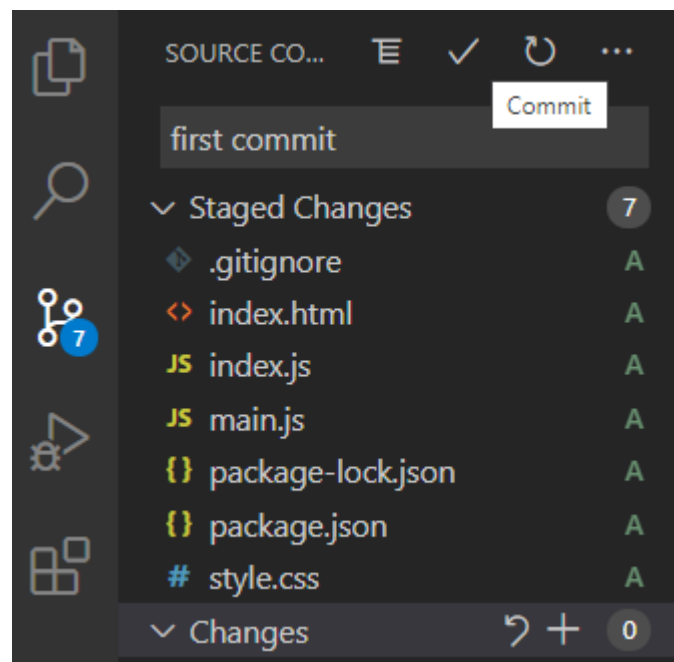
esto le dirá al control de versiones que no incluya ningún archivo que se encuentre en la carpeta de node\_modules, esto reducirá la cantidad de archivos de más de 4,000 a solamente 7.

Ahora sobre el menú que indica el nombre de la carpeta vamos a hacer clic en el signo de + (esto es equivalente a un git add . )

# < Técnico en DESARROLLO DE SOFTWARE >

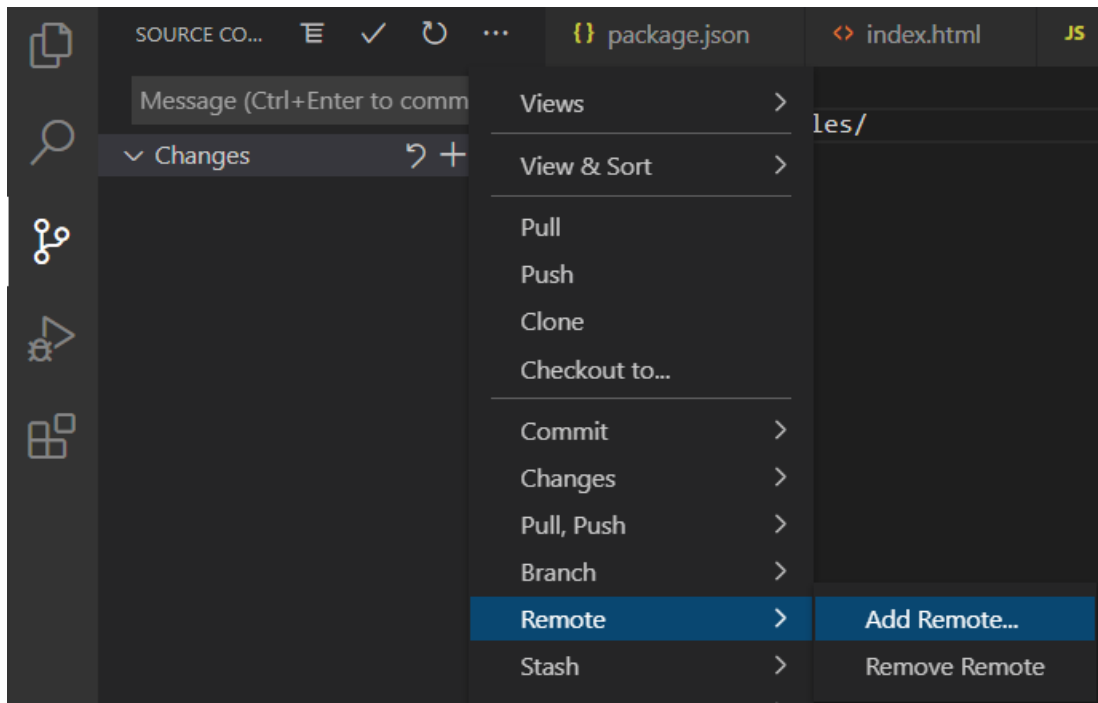


Ahora colocamos un comentario de "first commit" en el campo de texto y hacemos clic en el icono de cheque de arriba (esto es equivalente a un git commit -m "first commit")

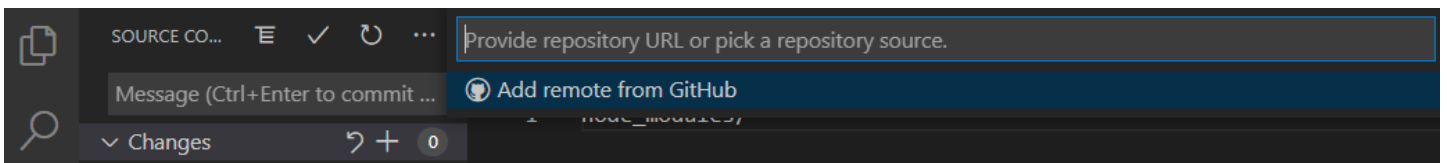




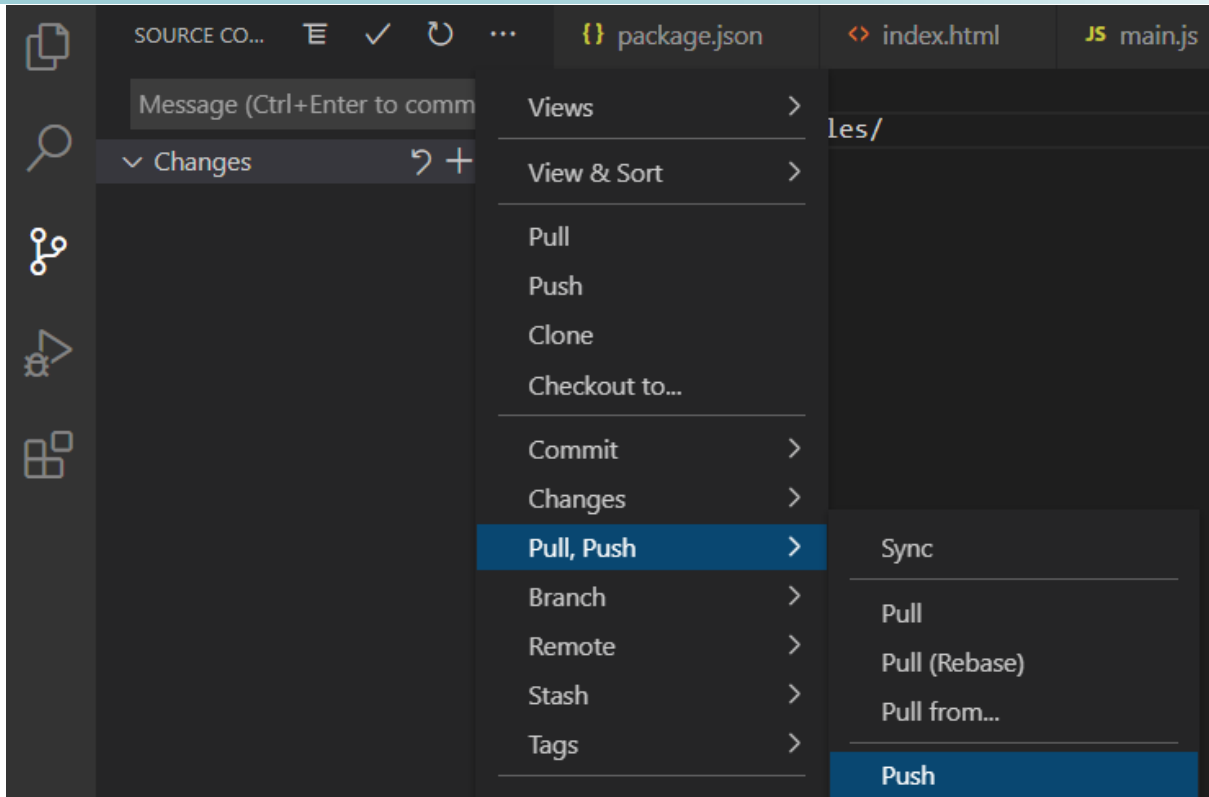
Ahora solo falta subir el proyecto al repositorio, hacemos clic en el icono de los tres puntos, y luego vamos a la opción de Remote > Add Remote...



Nos va a aparecer un campo de texto en el cual podemos colocar el URL del repositorio:



Ahora solo queda hacer un push, nuevamente en el menú de los tres puntos:



En este paso puede que nos pidan credenciales, las podemos ingresar y darle permiso a la aplicación para interactuar con github y listo, el repositorio habrá sido actualizado exitosamente.

## Contenido Adicional

- Guías y Tutoriales de Electron  
<https://www.electronjs.org/docs/README>  
<https://www.electronjs.org/docs/tutorial>
- Componentes de HTML  
[https://developer.mozilla.org/es/docs/Learn/HTML/Introduccion\\_a\\_HTML/iniciar](https://developer.mozilla.org/es/docs/Learn/HTML/Introduccion_a_HTML/iniciar)
- El lenguaje de JavaScript  
<https://eloquentjavascript.net>
- CSS garden, ¿qué se puede hacer con CSS?  
<http://www.csszengarden.com>
- Conceptos básicos de CSS en español  
<https://riptutorial.com/es/css>
- Videos, HTML y CSS para principiantes  
<https://www.youtube.com/watch?v=rbyYtrNUxg4>  
<https://www.youtube.com/watch?v=W6GTDfrWjXs>

---

## *Descargo de responsabilidad*

La información contenida en este documento descargable en formato PDF o PPT es un reflejo del material virtual presentado en la versión online del curso. Por lo tanto, su contenido, gráficos, links de consulta, acotaciones y comentarios son responsabilidad exclusiva de su(s) respectivo(s) autor(es) por lo que su contenido no compromete al área de e-Learning del Departamento GES o al programa académico al que pertenece.

El área de e-Learning no asume ninguna responsabilidad por la actualidad, exactitud, obligaciones de derechos de autor, integridad o calidad de los contenidos proporcionados y se aclara que la utilización de este descargable se encuentra limitada de manera expresa para los propósitos educativos del curso.

