

TEMA II

TRABAJAMOS CON JAVASCRIPT

ÍNDICE

I	...Y APARECIÓ JAVASCRIPT	3
I.1	Historia.....	3
I.2	Node.js.....	4
I.3	Lenguajes preprocesados.....	4
II	LA CAJA DE HERRAMIENTAS DEL FRONT-END DEVELOPER	5
II.1	Navegadores.....	6
II.2	Entornos para el desarrollo web	7
II.3	Instalación de Visual Studio Code	9
II.4	Uso de Visual Studio Code.....	9
II.5	Recomendaciones de trabajo y configuración.....	10
II.6	Node.js	11
III	MATERIAL EMPLEADO EN LA ELABORACIÓN TEMA	17
III.1	Bibliografía consultada y recomendada	17
III.2	Codificación	18

I ...Y APARECIÓ JAVASCRIPT

I.1 HISTORIA

Fue en 1995 cuando JavaScript fue desarrollado por *Brendan Eich* (actualmente CEO de Mozilla) para el navegador *Netscape Navigator* con el nombre de **Mocha**. Más tarde, se renombró a **LiveScript** quedando finalmente como *JavaScript*.

El nombre de JavaScript se debió a que Netscape añadió compatibilidad con Java en su navegador (en este tiempo Java empezó a hacerse tremendamente popular). Además, Netscape fue adquirida por *Sun Microsystems*, propietaria de la marca Java.



En este sentido se suele dar la **confusión de pensar que Java y JavaScript son lo mismo** o considerar a este último una extensión del primero, lo cual no es cierto. JavaScript se diseñó con una sintaxis similar al **lenguaje C** y aunque **adopta nombres y convenciones del lenguaje Java**, éste último no tiene relación con *JavaScript* ya que tienen semánticas y propósitos diferentes.

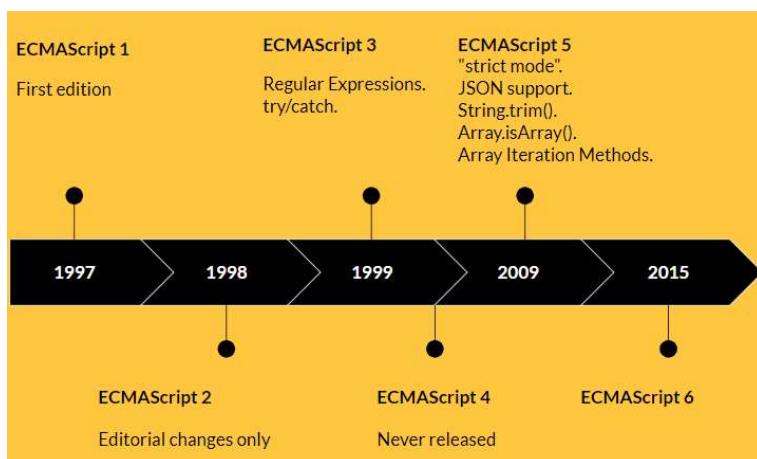
Por aquél entonces Microsoft ya estaba usando su propio JavaScript, **al que llamo JScript para evitar problemas relacionados con la marca**, pero no pudo evitar otros problemas surgidos por las incompatibilidades que su versión de JavaScript tenía con otros navegadores diferentes al Explorer.

Para evitar esas incompatibilidades, en 1997 el W3C promovió la creación del comité **TC39** por la **ECMA** (*European Computer Manufacturers Association*), el cual se encargará de **gestionar las especificaciones de este lenguaje de script** (da igual el nombre que reciba) **en el documento ECMA-262**. Desde este momento, tanto *JavaScript* como *JScript* deberán ser compatibles con dicho documento.

A raíz de esto se diseñará el estándar del DOM, *Document Object Model* que incorporaron las versiones *Internet Explorer 6*, *Netscape Navigator*, *Opera 7* y *Mozilla Firefox* desde la primera versión del DOM para, de esta manera, evitar incompatibilidades entre los navegadores.



Es a partir de entonces cuando los estándares de JavaScript se regirán por el ECMAScript. En 1999 se estandariza la versión 3 de JavaScript que se mantuvo vigente hasta hace relativamente poco.



Hubo algunos intentos de lanzar una versión 4, pero la que finalmente se estandarizó hasta ahora es la versión 5 de ECMAScript aprobada en 2011 (respetado incluso por Microsoft con su *JScript*).

Finalmente, en el año 2015 apareció la norma ECMAScript6 (conocida como ES6 o ECMAScript 2015 o ES2015) que

mejoró mucho el código JavaScript al dotarlo de elementos avanzados de otros lenguajes.

Han aparecido, hasta el momento de escribir estos apuntes, las versiones siete (o ECMAScript 2016), ocho (o ECMAScript 2017), nueve (o ECMAScript 2018) y diez (o ECMAScript 2019) y acaba de ser liberada la 11 o ECMAScript 2020, todos ellos con ligeros cambios sobre el ES6 (<https://es.wikipedia.org/wiki/ECMAScript>) y no totalmente soportados por todos los navegadores.

Queda ya poco de la idea original de JavaScript como un lenguaje para añadir efectos y animaciones a los sitios web; desde que en 2005 apareció Gmail y su uso de la tecnología AJAX, la popularidad de JavaScript no ha dejado de crecer haciendo que este evolucione hasta convertirse en un lenguaje multipropósito.

I.2 NODE.JS

En 2009, Ryan Dahl creó **Node.js**. Node es un entorno de ejecución para JavaScript en el servidor a partir del motor V8 de renderizado que utiliza el navegador Chrome de Google.

Node facilita la creación de aplicaciones de servidor altamente escalables siendo muy popular para el desarrollo de Microservicios, APIs, aplicaciones web Full-stack, isomórficas, etc...

Su comunidad es muy grande, y su sistema de paquetes y librerías NPM (*Node Package Manager*), hoy día también orientadas para JavaScript del lado cliente), ha superado los 150.000 módulos, convirtiéndolo en el más extenso de todos por delante de *Java*, *Ruby*, *PHP*, etc...



I.3 LENGUAJES PREPROCESADOS

La idea de un lenguaje preprocesado es la de un lenguaje que mejore el original y que, en nuestro caso, nos permita escribir JavaScript de forma más rápida y eficaz.



Ningún navegador, de forma nativa, es capaz de interpretar el código de un lenguaje preprocesado, para ello el código deberá ser convertido a *JavaScript estándar*. Esto se consigue con un software especial conocido simplemente como **preprocesador**.

Bajo esta idea nacieron:

- **CoffeeScript**. Ya un poco en retroceso, permitió dotar a JavaScript de una sintaxis inspirada en *Python*.
- **TypeScript**. Lenguaje creado por Microsoft que amplía JavaScript para que reconozca tipos de datos avanzados, variables fuertemente tipadas, etc. Es el lenguaje base del exitoso framework *Angular* de Google.
- **Dart**. Desarrollado por Google para modernizar JavaScript.
- **Elm**. Para programar en un lenguaje puramente funcional.
- ...

Para concluir, decir que hoy en día JavaScript se utiliza en múltiples entornos: *Frontend*, *Backend*, aplicaciones isomórficas (aquella que comparte todo (o casi todo) su código entre el cliente y el servidor), microcontroladores, *Internet of Things*, *apps wearables*, etc... convirtiéndose en el lenguaje de programación del presente.

Toda la documentación y referencia sobre JavaScript se puede encontrar en el sitio web de desarrolladores de Mozilla (<https://developer.mozilla.org/es/docs/Web/JavaScript>), muy recomendable de visitar cuando se tienen dudas sobre cómo se usa o implementa una función u objeto determinado.

Actualmente JavaScript es una **marca registrada** de *Oracle Corporation*, y es usado con licencia por los productos creados por *Netscape Communications* y entidades actuales, como la *Mozilla Foundation*.

II LA CAJA DE HERRAMIENTAS DEL FRONT-END DEVELOPER

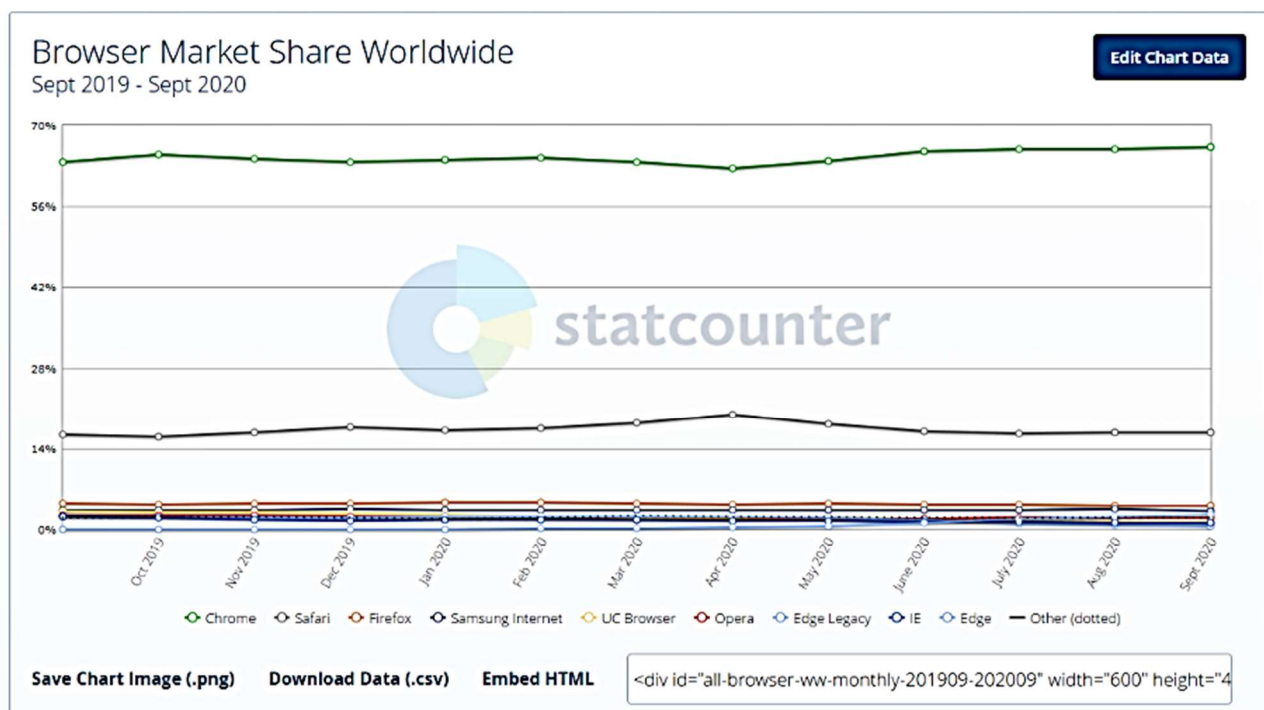
Las herramientas que nos permitirán conseguir un entorno de trabajo que nos permita desarrollar aplicaciones de forma cómoda podrían ser:

- ✓ **Varios navegadores**. Para tener varios entornos de pruebas.
- ✓ **Un editor de código**. Junto al navegador, es lo mínimo que necesitamos para trabajar. Por aclamación popular vamos a usar el *Visual Studio Code*, aunque os animo a probar otros como *Sublime Text*, *Brackets*, *notepadqq*, *atom*...

- ✓ **Un intérprete independiente** que nos permita probar JavaScript fuera del navegador (p.e. *Node.js* que aporta el *sistema de gestión de paquetes npm*).
- ✓ **Gestor de paquetes para instalar herramientas y componentes.** Frameworks, herramientas de gestión de tareas, precompiladores... se pueden gestionar gracias a la ayuda de estos gestores. *Npm* es el más popular (viene integrado en *Node.js*).
- ✓ **Sistema de control de versiones.** Permite tener diferentes versiones de nuestro código para acudir a ellas cuando deseemos. Actualmente *Git* es el más popular.
- ✓ **Repositorio con sistema de control de versiones integrado en la nube.** Servicio que permite alojar nuestro código manteniendo las diferentes versiones que están siendo gestionadas por nuestro sistema de control de versiones. *GitHub* es el servicio más popular, aunque también tenemos *Gitlab* o *bitBucket*.

II.1 NAVEGADORES

En el momento de escribir estos apuntes, Google Chrome es claramente el navegador más utilizado en todo el mundo. La probabilidad de que un usuario utilice *Google Chrome* como navegador es muy alta (más de un 60%) y hay que tener además en cuenta, que muchos otros navegadores (como *Microsoft Edge* u *Opera*) se basan en el proyecto *Chromium* de Google.



Pero no hay que olvidar que un porcentaje importante de personas utilizan otros navegadores como *Apple Safari* o *Mozilla Firefox*. Así que cuantos más navegadores instalemos para probar nuestra aplicación, más seguros estaremos de que funcione correctamente. Como posibles navegadores a instalar en nuestro sistema tenemos:

- **Microsoft Edge** (Antiguo *Internet Explorer*). Diseñado como un navegador ligero con un motor de renderizado de código abierto construido según los estándares web.
- **Mozilla Firefox**. Navegador web libre y de código abierto desarrollado por la *Corporación Mozilla* y la *Fundación Mozilla*. Usa el motor *Gecko* para renderizar páginas webs, el cual implementa actuales y futuros estándares web. Posee una versión para desarrolladores: *Firefox Developer Edition*.
- **Google Chrome**. Desarrollado por Google y compilado con base en varios componentes e infraestructuras de desarrollo de aplicaciones (frameworks) de código abierto, como el motor de renderizado *Blink*. Está disponible gratuitamente bajo condiciones específicas del software privativo o cerrado. Su versión para desarrolladores es el *Google Chrome Dev*.
- **Safari**. *Safari* es un navegador web de código cerrado desarrollado por *Apple Inc.* Está disponible para OS X, *iOS* (el sistema usado por el *iPhone*, el *iPod touch* y *iPad*) y *Windows* (sin soporte desde el 2012).
- **Opera**. Navegador web creado por la empresa noruega *Opera Software*. Usa el motor de renderizado *Blink*. Tiene versiones para escritorio, smartphones y tablets.

Por motivos de facilidad de depuración de nuestro software vamos a usar principalmente el entorno del *Mozilla Firefox*. El motivo de usarlo es la gran cantidad de herramientas para depuración que posee incluso en su versión estándar (está disponible una versión que amplía las herramientas de desarrollo llamada *Firefox Developer Edition*).

II.2 ENTORNOS PARA EL DESARROLLO WEB

Para programar con JavaScript se necesita un editor de texto. Muchos lenguajes de programación se encuentran soportados con **IDE** (*Integrated Development Environment*) para escribir código más rápido, acceder a la documentación y evitar errores.

En el caso de JavaScript no importa tanto el editor de texto que se utiliza como un entorno donde podamos probar y depurar de forma rápida y cómoda nuestro código. Afortunadamente, los navegadores web (*Mozilla Firefox*, *Google Chrome*, ...) ya llevan incorporados de forma predeterminada depuradores y entornos de desarrollo que ayudan a la programación.

Es útil encontrar atajos de teclado, opciones y posibilidades para sacar el máximo provecho y rendimiento al programa. La situación perfecta sería trabajar con dos pantallas: una para el código y herramientas de depuración y otra para ver el resultado. Por lo general, a la hora de elegir un editor esperamos las siguientes facilidades:

- ✓ **Elección de lenguaje de programación.** En nuestro caso, *JavaScript*, *html* o *css*.
- ✓ **Coloración sintáctica.** Lo normal y deseable es tener diferentes temas para elegir colorados del entorno y código que se ajusten a nuestras preferencias.
- ✓ **Facilidad para seleccionar texto.** Tenemos que saber, de una sola pasada, qué es código y qué texto normal.
- ✓ **Herramientas avanzadas de búsqueda y reemplazo.** Tanto individual como en grupo (por ejemplo, seleccionar todas las variables con un cierto nombre).
- ✓ **Navegación avanzada.** Bloques de código, minimapas con una miniatura del código, acceder a una línea de código a través del nombre de una función o un objeto...
- ✓ **Autocorrección al escribir** (aunque hay que acostumbrarse).
- ✓ **Abreviaturas o snippets** (por ejemplo, que cuando escribamos *cl* y pulsamos el tabulador, el editor lo cambie por *console.log*).
- ✓ **Visualización del resultado.** Si evitamos tener que estar actualizando el navegador, mejor.
- ✓ **Nuevas funcionalidades con instalación de extensiones (plugins).**

Entre los editores que puede utilizar destacamos:

- **Editores de texto multipropósito.** Facilitan la escritura de código fuente para diversos lenguajes (*Sublime Text*, *Notepad++*, *TextMate*...). A veces tienen la opción de instalar extensiones y plugins.
- **Editores ligeros especializados en desarrollo web.** Como el anterior pero con la diferencia de que suelen venir ya preparados con herramientas que mejoran las prestaciones (*Visual Studio Code* de Microsoft, *Atom* de GitHub, *Brackets* de Adobe...)
- **Entornos de desarrollo integrados (IDE).** Entornos de trabajo más pesados y normalmente de pago. Incluyen una gran cantidad de herramientas para cubrir cualquier necesidad de los programadores. (*Visual Studio*, *Netbeans*, ...).
- **Editores WYSIWYG.** Editores visuales de páginas web que complementan la labor del desarrollador (*Dreamweaver*, *CoffeeCup*...)
- **Editores de código online.** Se trata de aplicaciones web que permiten la escritura de código y previsualizan el resultado al instante. Su ventaja es que no requieren instalar nada en nuestra máquina para comprobar el funcionamiento del código. (*JSFiddle*, *CodePen*, *FreeCodeCamp*...).

II.3 INSTALACIÓN DE VISUAL STUDIO CODE

Si seguimos los pasos indicados en el enlace

<https://ubunlog.com/visual-studio-code-editor-codigo-abierto-ubuntu-20-04/> veremos que los dos modos más eficientes para instalarlo son:

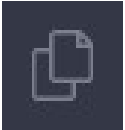




1. Por línea de comando y paquetes *snap*:

- `sudo apt update` (recomendado)
- `sudo apt upgrade` (opcional)
- `sudo snap install --classic code`

2. A través de la GUI. Para ello entramos en *Software de Ubuntu* y buscamos *Visual Studio Code*. Lo instalamos.

II.4 USO DE VISUAL STUDIO CODE

En lugar de explicar el funcionamiento del *Visual Studio Code* (podríamos dedicar todo un curso a ello) lo mejor es que la simple práctica sea nuestro tutorial. De todos modos, podríamos destacar el *Panel de Navegación* (menú vertical del lado izquierdo):

ICONO	USO
	Vista de explorador. Permite ver los archivos y carpetas del proyecto de trabajo actual. Desde aquí podemos abrir los archivos deseados, borrarlos, cambiarles el nombre, etc.
	Buscar. Sirve para buscar textos por el archivo actual o incluso por otros archivos. Dispone de opciones muy avanzadas para realizar esa búsqueda.
	Control de código fuente. Permite el trabajo con el sistema de control de versiones de forma integrada en el editor.
	Debug. Acceso a la herramienta integrada de depuración de programas. La depuración facilita la corrección y detección de errores.
	Extensiones. Permite añadir y quitar nuevos componentes al editor.



Gitlens. Interesante extensión que nos permite navegar y explorar nuestro repositorio Git.

II.5 RECOMENDACIONES DE TRABAJO Y CONFIGURACIÓN

Antes de verlas hay que recordar que siempre hay opciones que se adaptan mejor o peor a cada desarrollador. Escojamos siempre aquél método de trabajo en el que nos sintamos más cómodos.

- ✚ **Carpeta de trabajo.** Una vez creada la abriremos desde el VSC desde **File → Open Folder**, luego podemos crear las subcarpetas que deseemos.
- ✚ **Ficheros.** Podemos crear nuevos ficheros iniciando el menú contextual desde la carpeta que queremos que lo contenga.
- ✚ **Terminal del VSC.** Conviene tenerlo activado para no tener que estar cambiando el foco de la ventana del VSC al terminal de Linux. Lo puedes activar con **Ctrl+`**
- ✚ **Emmet.** VS Code tiene instalada la extensión *Emmet* por defecto, gracias a ella nos será más fácil escribir el código. Simplemente (esto funciona en ficheros HTML) escribiremos el cierre de admiración (!) y pulsamos tabulador. Se escribirá el código base de la página.

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6   <title>Document</title>
7 </head>
8 <body>
9
10 </body>
11 </html>
```

- ✚ **Cambiar temas.** Muy interesante si prefieres colores con mayor contraste para diferenciar mejor los textos de las palabras propias del código. **File → Preferences → Color Theme**

🔧 **Otras extensiones recomendadas.** Personalmente me gusta trabajar con las siguientes extensiones:

- ✓ **Auto Rename Tag.** Cuando modificamos el nombre de una etiqueta HTML (o XML) se modifica también la etiqueta de cierre.
- ✓ **Open in browser.** Desde el menú contextual en un archivo del panel de navegación podremos elegir el navegador con el que se abrirá el archivo. Muy interesante para probar nuestro código en diferentes navegadores. Por supuesto, el navegador debe estar instalado.
- ✓ **Live Server.** Imprescindible. Procediendo como en la extensión anterior elegiremos la opción **Open with Live Server**. Cada vez que hagamos cambios y guardemos se actualizará automáticamente el navegador.
- ✓ **Spanish Language Pack for VSC** (opcional, deberá reiniciar VSC).
- ✓ **JS-CSS-HTML Formatter.** Muy cómodo, da formato al texto eliminando espacio, ajustando sangrías, eliminando saltos de línea sobrantes, etc...
- ✓ **GitLens – Git supercharged.** Muy útil herramienta Git para gestionar nuestro repositorio en la nube sin necesidad de introducir comandos git (modificaciones, ramas, ficheros, ...).
- ✓ **Material Icon Theme.** Ofrece iconos más característicos del tipo de fichero con el que trabajamos.
- ✓ **Rainbow Brackets.** Sistema de control de paréntesis por colores.
- ✓ **Otros**
 - Auto Close Tag
 - Can i use
 - HTML CSS Support
 - HTML Snippets
 - JavaScript (ES6) code snippets
 - Markdown all in one
 - Markdownlint

II.6 NODE.JS

Software que nos permite interpretar código JavaScript. Es muy interesante su instalación porque nos permite poder utilizar JavaScript fuera del navegador y así programar otro tipo de aplicaciones. La compatibilidad de node.js con los estándares es altísima y eso hace que aprender a trabajar con estructuras de datos y otros elementos avanzados del lenguaje JavaScript, sea más cómoda.

Aparte de esto, node.js tiene capacidad para crear código JavaScript back-end, además de capacitar a nuestra máquina para funcionar como un servidor web. Es más, node.js es uno de los servidores web más populares actualmente.

II.6.1 Instalamos Node.js

Usando el sistema de empaquetado snap y siguiendo los pasos de la página <https://ubunlog.com/nodejs-npm-instalacion-ubuntu-20-04-18-04/> (puede que antes sea necesario un *sudo apt update*):

```
➤ sudo snap install node --channel=14/stable --classic
```

Aunque también lo podemos encontrar en el repositorio de Ubuntu.

II.6.2 Comprobación de instalación

Abriremos un terminal y escribimos el comando **node**. Aparecerá el *prompt* de node (símbolo >). Como prueba podemos escribir la siguiente orden que mostrará por pantalla el mensaje “Hola caracola”:

```
profesor@profesor-VirtualBox:~/Desktop/PracticasDWE$ node  
> console.log("Hola caracola");  
Hola caracola
```

Podemos probar códigos más complejos:

```
profesor@profesor-VirtualBox:~/Desktop/PracticasDWE$ node  
> for(let i=0;i<=10;++i) console.log(`Numero: ${i}`);  
Numero: 0  
Numero: 1  
Numero: 2  
Numero: 3  
Numero: 4
```

Para salir pulsaremos dos veces **Ctrl+c**

También podemos ejecutar un archivo JavaScript, para ello, suponiendo el archivo *script.js*:

```
profesor@profesor-VirtualBox:~/Desktop/PracticasDWE$ node script.js  
Hola caracola  
profesor@profesor-VirtualBox:~/Desktop/PracticasDWE$
```

II.7 SISTEMA DE CONTROL DE VERSIONES Y GIT

Imaginemos estas dos situaciones:

- ❖ Queremos añadir una funcionalidad o modificar parte del comportamiento de nuestro programa y nos encontramos con que este ya no funciona como antes y además, no somos capaces de localizar el problema en el código. Ante esta situación lo ideal sería volver a la situación anterior.

- ❖ Dos programadores están colaborando para crear un determinado proyecto trabajando en paralelo y desean mezclar las modificaciones de ambos para que el proyecto refleje sus trabajos.

Un sistema de control de versiones nos permite poder hacer instantáneas de la situación de un proyecto en un momento dado, para regresar a dicha instantánea cuando lo consideremos necesario.

En el caso del trabajo en equipo, el sistema de control de versiones nos permite trabajar mediante ramas en las que el desarrollo del proyecto sigue distintos caminos (cada uno con sus versiones o instantáneas) que podremos mezclar a voluntad. Actualmente el sistema de control de versiones más exitoso es Git, creado en 2005 por **Linus Torvals**.

II.7.1 Instalamos Git

El modo más sencillo es desde el administrador de paquetes apt:

- `sudo apt update` (si todavía no lo has hecho)
 - `sudo apt install git`
- Comprobamos la instalación con:
- `git --version`

II.7.2 Configuración inicial

Una de las primeras cosas que debemos hacer después de la instalación es configurar nuestro nombre de usuario y dirección de correo electrónico. Git asocia tu identidad con cada **commit** que hagas:

- `git config --global user.name "Nuestro nombre"`
 - `git config --global user.email "tudireccion@dominio.com"`
- Verificamos los cambios
- `git config --list`

II.7.3 Comenzar un proyecto Git

Normalmente realizar un nuevo proyecto partirá de una carpeta o directorio raíz que contendrá todos los archivos del proyecto. Para que dispongamos de control de versiones basado en Git sobre esa carpeta, deberemos entrar en esa carpeta desde el terminal y escribir **git init**:

```
profesor@profesor-VirtualBox:~/Desktop/PracticasDWE$ git init
Initialized empty Git repository in /home/profesor/Desktop/PracticasDWE/.git/
profesor@profesor-VirtualBox:~/Desktop/PracticasDWE$
```

II.7.4 Resumimos el funcionamiento de Git

Se ha creado un directorio oculto llamado **.git** en el que se almacenarán los archivos que el sistema git necesita para controlar las versiones de nuestro proyecto (recuerda que el comando anterior solo se necesitará escribir una vez).

A partir de ese momento nuestro proyecto puede tener tres estados:

1. **Working Directory**. Estado en el que hemos hecho modificaciones a los archivos.
2. **Staging**. Se han hecho cambios y se han marcado los que queremos que se tengan en cuenta para la siguiente confirmación (commit). Pasaremos a este estado escribiendo:

```
➤ git add .
```

En la orden anterior hemos indicado que todos los cambios realizados en cualquier archivo (comodín “.”) se tendrán en cuenta.

3. **Repositorio**. En este estado los cambios marcados ya están grabados en una instantánea. Para ello escribiremos:

```
➤ git commit -m "Realizada área de login"
```

Gracias a este comando grabaremos una versión de nuestro código a la cual se asigna una clave única e irrepetible. Esa clave es la que nos permitirá acudir a la versión cuando deseemos. Siempre deberemos escribir un mensaje aclaratorio de la operación.

II.7.5 Otras operaciones

El estado actual del proyecto lo podemos obtener mediante:

```
➤ git status
```

El listado de las versiones que hemos guardado lo obtendremos con:

```
➤ git log
```

Volver a una versión concreta de un archivo se puede hacer con el comando:

```
➤ git checkout 433ff index.html
```

La línea anterior devolverá el archivo *index.html* al estado que tenía tras la versión asociado al número que comienza con el hexadecimal *43e6f*. Para que se mantengan esos cambios, basta con ejecutar un nuevo **commit**.

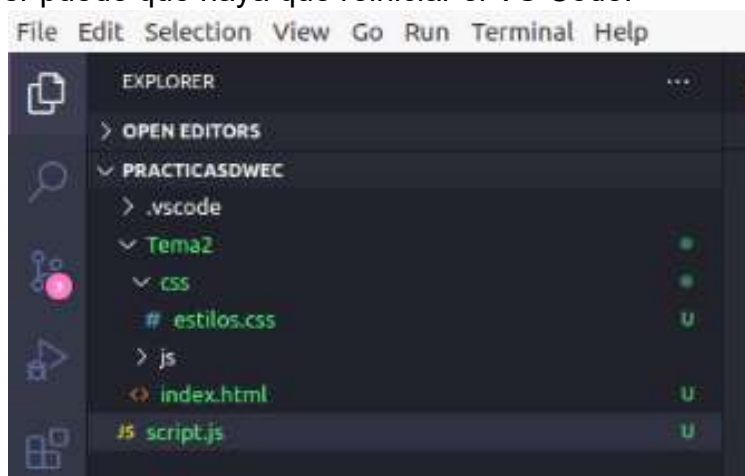
Por último, si queremos devolver nuestro proyecto a un estado anterior, pero eliminando todos los *commits* a partir de ese momento (hay que estar muy seguros de que eso es lo que queremos hacer) tendremos que escribir:

```
➤ git reset -hard 43fff
```

II.7.6 Control de versiones

Fijémonos en el detalle de que el botón dedicado al control de versiones, dentro del propio VS Code, ya se habrá percatado de que usamos Git y nos aparece un aviso por cada archivo con cambios no guardados en repositorios Git.

De no aparecer puede que haya que reiniciar el VS Code.



Si queremos confirmar los cambios tendremos que pulsar en dicho botón, escribir un mensaje que nos recuerde el motivo de las modificaciones y por último pulsar en *Commit*.



Usando la extensión **GitLens** podemos observar los cambios en cada fichero pulsando en **Repositories → Branch Master**. En el siguiente ejemplo he eliminado el elemento párrafo (`<p></p>`). A la izquierda puedes ver la primera versión y a la derecha la segunda.


```

2 <html lang="es-es">
3
4 <head>
5   <meta charset="UTF-8">
6   <meta name="viewport" content="width=device-width, initial-scale=1">
7   <title> Prueba</title>
8   <link rel="stylesheet" href="css/estilos.css">
9 </head>
10
11 <body>
12   <h1>Mi primera página en <strong>Visual Studio</strong>
13   <p>un párrafo</p>
14 </body>
15
16 </html>
  
```

Por último, la forma de volver a un commit (instantánea) anterior usando **GitLens** sería ir al apartado *branches* y buscar el mensaje de *commit* al que deseamos regresar. Simplemente hacemos clic en **Switch to commit** del menú contextual del mensaje para que el código cambie y aparezca tal cual estaba en ese commit.

II.7.7 Git en Remoto

Lo más interesante de las herramientas **git** viene cuando se nos da la posibilidad de almacenar el código en Internet. Tenemos varios servicios a elegir: **GitLab**, **Bitbucket** y **GitHub** son los más conocidos aunque hay bastante para escoger.

GitHub (que actualmente pertenece a Microsoft) es de los más populares si no el que más. En todo caso el uso de repositorios en la nube requiere de estos comandos:

- 📁 **Git clone URL**. Descarga un repositorio en la nube en el directorio en el que estamos situados. Se descargarán también todas las versiones del mismo. Si el repositorio es privado se nos requerirá introducir nuestro usuario y contraseña.
- 📁 **Git remote**. Permite mostrar los servicios remotos asociados a nuestro proyecto actual.
- 📁 **Git remote alias URL**. Establece un servicio remoto asociado a nuestro proyecto. La dirección del repositorio remoto es la indicada con su URL. El alias es el nombre familiar que le damos al repositorio remoto.
- 📁 **Git remote rm alias**. Quita la asociación con el repositorio remoto indicado. No borra realmente el repositorio, lo que hace es dejar de asociar ese repositorio remoto a nuestro proyecto.
- 📁 **Git push alias rama**. Publica el repositorio actual en el remoto indicado por su alias. Indica también la rama que subimos, si no hemos usado ramas, la rama habitual se llama master.
- 📁 **Git pull alias rama**. Descarga el contenido del repositorio remoto y lo mezcla con el repositorio actual.

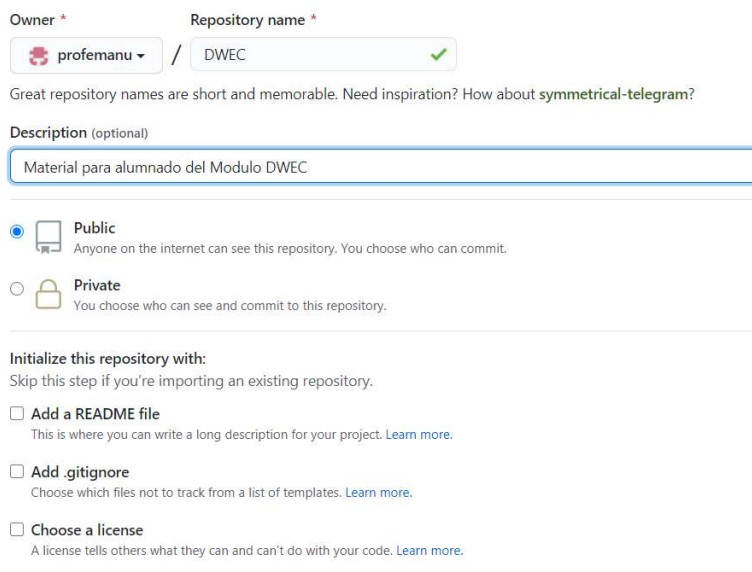
II.7.8 GitHub

Crea una cuenta GitHub. Si ya la tienes en GitLab prueba a instalar la extensión GitLab WordFlow para integrar la cuenta en VS Code.

Pon nombre al repositorio y elige público o privado si quieres que sea oculto o no.

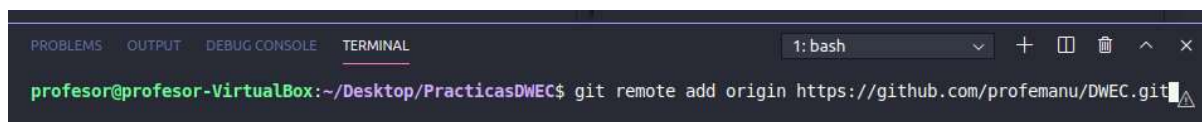
Crea el repositorio.

Copia el primer comando que aparece (el que empieza por **git remote...**) y ejecútalo en el terminal.



...or push an existing repository from the command line

```
git remote add origin https://github.com/profemanu/DWECC.git
git branch -M main
git push -u origin main
```







A partir de ese momento, cada vez que confirmemos los cambios, podremos subirlos al repositorio remoto. Para ello, deberemos ir al **panel de Navegación** → **Control de código fuente** y hacer clic en el botón con tres puntos y elegir **Publicar rama (Branch → Publish branch)**.







Si todo va bien, podremos ver la publicación en el apartado de remotos (*remotes*) en el panel de *GitLens*.







III MATERIAL EMPLEADO EN LA ELABORACIÓN DEL TEMA

III.1 BIBLIOGRAFÍA CONSULTADA Y RECOMENDADA

-  *Desarrollo web en entorno cliente*. Juan Carlos Moreno Pérez, Edit. Síntesis
-  *Desarrollo web en entorno cliente con JavaScript*. Jorge Sánchez Asenjo, Edit. Garceta
-  *Aprendiendo JavaScript: Desde cero*. Azaustre, Carlos.
-  <https://platzi.com/>

-  <https://developer.mozilla.org/es/docs/Web/JavaScript>
-  <https://www.w3schools.com/>
-  <http://es.wikipedia.org>
-  Blogs de desarrollo y programación

III.2 CODIFICACIÓN

-  <https://www.freecodecamp.org/>
-  <https://codepen.io/>
-  https://www.tutorialspoint.com/online_javascript_editor.php
-  Visual Studio Code
-  Sublime Text
-  Notepad ++