

UD2. PREPARACIÓN DEL ENTORNO DE TRABAJO

1. LA CAJA DE HERRAMIENTAS DEL ENTORNO DE TRABAJO	2
2. NAVEGADORES	2
3. EDITORES DE CÓDIGO	4
3.1. AYUDAS DE EDITORES DE CÓDIGO	4
3.2. TIPOS DE EDITORES DE CÓDIGO	5
3.3. INSTALACIÓN DE VISUAL STUDIO CODE	6
3.3.1. INSTALACIÓN	7
3.5.1. CONFIGURACIÓN DEL ENTORNO	9
3.5.2. EXTENSIONES	9
4. NODE.JS Y NPM	10
4.1. ¿QUÉ ES NODE.JS?	10
4.2. INSTALACIÓN DE NODE.JS	10
4.2.1. EL PROBLEMA DE LAS VERSIONES DE NODE	10
4.3. NVM	12
5. GIT	13
5.1. SISTEMAS DE CONTROL DE VERSIONES Y GIT	13
5.2. INSTALACIÓN DE GIT	14

1.LA CAJA DE HERRAMIENTAS DEL ENTORNO DE TRABAJO

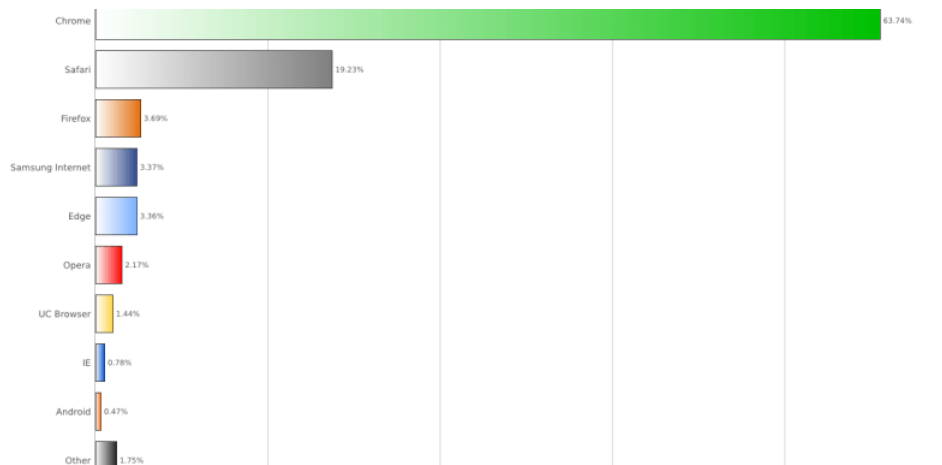
En general un desarrollador de aplicaciones web front-end necesitará:

- **Varios navegadores.** Si indicamos varios es porque no debería bastarnos con probar nuestro código en un solo navegador.
- Un **editor de código.** Es la primera herramienta necesaria. Elegir un buen editor es la primera gran decisión. Junto con el navegador, es lo mínimo que necesitamos para trabajar. Es recomendable utilizar **Visual Studio Code**.
- Un **intérprete independiente** de JavaScript. Al decir independiente queremos decir que permita ejecutar JavaScript fuera del navegador. Eso permite abordar y aprender las bases del lenguaje de forma más rápida. **Node.js** es la opción lógica en este aspecto, además aporta el sistema de gestión de paquetes **npm (Node Package Manager)**.
- Gestor de paquetes para instalar herramientas y componentes. El uso de **frameworks** (herramienta que proporciona componentes listos para usar o soluciones personalizadas para acelerar el desarrollo.) y componentes como **JQuery, React, Vue o Angular** entre muchos otros, así como herramientas de gestión de tareas como **Gulp o Grunt**, precompiladores como **Babel, Sass o TypeScript** y otros muchos componentes y elementos, se suele gestionar gracias a la ayuda de los gestores de paquetes; **npm** es el más popular porque está integrado en **Node.js** pero ay otras alternativas como, por ejemplo, **yarn**.
- Sistemas de control de versiones. Se trata de software integrado, normalmente, en la línea de comandos del sistema. Permite poder tener diferentes versiones de nuestro código para poder acudir a ellas cuando deseemos. Actualmente, **Git** es el sistema más popular.
- Repositorio, con **sistema de control de versiones integrado en la nube**. Se trata de un servicio que permita alojar nuestro código manteniendo las diferentes versiones que están siendo gestionadas por nuestro sistema de control de versiones. Indudablemente, **GitHub** es el servicio más popular.

2.NAVEGADORES

Todo desarrollador necesita instalar varios navegadores. La probabilidad de que un usuario use Chrome como navegador es muy alta (más de un 60%) actualmente y hay que tener en cuenta, que muchos navegadores (como **Edge u Opera**) se basan en el proyecto **Chromium** de Google. Por lo tanto, es vital que probemos nuestras aplicaciones en ese navegador.

Pero hay un porcentaje importante de personas que utilizan otros navegadores como Mozilla **Firefox** o Apple **Safari**. Cuantos más navegadores instalemos para probar nuestra aplicación más seguros estaremos de que nuestra aplicación funciona perfectamente.



<https://www.funinformatique.com/es/ranking-de-navegadores-web/>

Es complicado tener instalados todos los navegadores. Además, hay que tener en cuenta que hay navegadores que usan el mismo motor (**Edge**, **Opera** y **Chrome**) y que, en nuestro sistema, no podremos instalar todos los navegadores; por ejemplo, **Samsung Internet** es el navegador de los Smart TV de la marca Samsung, **Safari** solo funciona con sistemas Apple y **UC Browser** no está disponible en sistemas de escritorio. Al final hay que pensar que el estándar del lenguaje, poco a poco es aceptado por todas las marcas. Nuestra recomendación mínima a instalar es:

- **Google Chrome**. Al ser el más instalado.
- **Mozilla Firefox**. Es el 2º más utilizado y además el único rival real de Chrome al tener un motor diferente, tanto para interpretar JavaScript como para renderizar HTML y CSS.
- **Internet Explorer**, para saber cómo ven las páginas los usuarios que no tengan un sistema al día. Es desaconsejable utilizar este navegador hoy en día, pero hay muchos usuarios todavía que lo utilizan y por ello hay que probar, al menos, cómo verán la aplicación estos usuarios para saber si debemos avisar o no que nuestra aplicación no funciona bien en este navegador.
- **Apple Safari**, si es posible, ya que solo funciona en dispositivos de la marca Apple.
- Navegadores específicos de dispositivos, si es posible, como Smart TV, móviles diferentes, etc.

Muchos fabricantes ofrecen versiones de navegadores en pruebas. Estos navegadores especiales se usan para probar características recientes que todavía no están implementadas en los navegadores que utilizan los usuarios. La idea general es la siguiente:

- Google proporciona los siguientes navegadores (aparte del Chrome definitivo)
 - ✓ **Canary**. Versión inestable para pruebas. Es el primer navegador en el que Google incorpora las novedades que vayan apareciendo en HTML, CSS y JavaScript
 - ✓ **Dev**. Versión para programadores, es una versión más estable que la anterior. Las novedades que son aún muy inestables, no se implementan aquí.
 - ✓ **Beta**. Versión casi definitiva, se suele lanzar 15 días antes del Chrome final.
- Firefox utiliza la misma idea:
 - ✓ **Nightly**. Versión para probar novedades. ES en esta versión donde primero aparecen las características últimas de los lenguajes.

- ✓ **Developer Edition.** Más estable y con capacidades especiales para desarrolladores. Sirve para probar las novedades sin los problemas de posible inestabilidad de la anterior.
- ✓ **Beta.** Versión previa a la definitiva para confirmar la buena implementación de nuevas características.
- Microsoft Edge tiene versiones equivalentes a las de **Chrome: Canary, Dev y Beta.**
- Apple Safari tiene 2 versiones: **Technology Preview** (equivalente a Canary) y **Beta.**

Puede ser conveniente instalar alguno de estos navegadores (por ejemplo, **Google Chrome Dev** y **Firefox Developer Edition**) si queremos ir probando nuevas características para que, cuando se aprueben realmente en el navegador definitivo) podamos incorporarlas a nuestros proyectos.

3. EDITORES DE CÓDIGO

3.1. AYUDAS DE EDITORES DE CÓDIGO

El editor de código es el primer software que todo desarrollador debe instalar para trabajar en el desarrollo de aplicaciones web. El código que se escribe para crear aplicaciones web se almacena en ficheros de texto plano. Esto significa que cualquier herramienta capaz de escribir texto es válida, incluso el Bloc de Notas de Windows, aunque no es lo ideal.

Hay editores creados específicamente para crear código que nos van a facilitar mucho más nuestro trabajo. Las facilidades y ayudas que se nos proporcionan, difieren según el tipo de editor que utilicemos. A continuación, resumimos las más habituales y prácticas.

- **Coloreado de código.** Facilita enormemente la escritura de código porque se distinguen muy bien las palabras claves del texto normal o de los valores que toman las variables. En la mayoría de editores el coloreado es personalizable y también es común disponer de diferentes temas para elegir coloreados del entorno y código que se ajusten a nuestras preferencias.
- **Facilidad para seleccionar texto.** Los procesadores de texto (como Word) siempre han dispuesto de herramientas y combinaciones de teclado que facilitan la selección de texto. En el caso de los editores de código, estas facilidades se ajustan mejor a las necesidades de los programadores porque seleccionan sabiendo que lo que tenemos es código y no texto normal.
- **Navegación avanzada.** A veces los desarrolladores generan archivos de cientos o miles de líneas, moverse por archivos tan grandes es difícil. Por ello hay ayudas que ofrecen los editores como:
 - ✓ Expandir y comprimir bloques de código, para ocultar o mostrar texto y ver mejor el que nos interesa
 - ✓ Mapa que muestra los elementos involucrados en el código para acceder al instante al código de un elemento en concreto.
 - ✓ Mini mapas del código que muestran una miniatura del código para, a través de ella, acceder más rápido a la zona que nos interesa.
 - ✓ Ir directamente a la línea de código o una zona concreta de código a través del nombre de una función u objeto.
- **Búsqueda y reemplazo.** Estas son 2 operaciones fundamentales. En muchos editores además se permite buscar no solo en el documento actual, sino en cualquier archivo de nuestro

proyecto de trabajo (que puede estar formado por decenas de archivos). Además, se suelen proporcionar herramientas de reemplazo de nombres inteligentes. Un ejemplo suele ser el caso de un programador que quiere cambiar el nombre de una variable que aparece continuamente en el código.

- **Autocorrección al escribir.** Todos los programadores cometemos errores continuamente al escribir código. Muchos editores son capaces de marcar errores con un subrayado rojo, por ejemplo. Esto ahorra muchísimo tiempo.
- **Uso de abreviaturas.** Son los llamados **Snippets** (trozos de código que sabemos que usamos a menudo y que asociamos con una abreviatura). Por ejemplo, podemos conseguir que cuando escribamos `cl` y pulsemos el tabulador, el editor lo cambie por `console.log`.
- **Visualización del resultado.** La mayoría de editores tienen integrada en sus menús y opciones la posibilidad de lanzar el resultado de nuestro código en el navegador. Algunos tienen posibilidades avanzadas, como crear en caliente un servidor web que permita poder examinar los cambios en la aplicación a la vez que los vamos escribiendo en el código.
- **Integración de herramientas.** Se trata de que el editor integre, sin salir del editor, herramientas de uso habitual de los programadores como el acceso al terminal del sistema, control de versiones de código, publicación del código, herramientas de test y depuración, etc.
- **Adición y extensiones y/o plugins.** Esto permite añadir funcionalidades que, inicialmente, no están disponibles en el editor y que podemos añadir a voluntad para crear un entorno de trabajo a medida.

3.2. TIPOS DE EDITORES DE CÓDIGO

- **Editores de texto multipropósito.** Son editores de texto especializados en facilitar la escritura de código fuente para diversos lenguajes. Son capaces de adaptarse al tipo de código escrito. Por ejemplo, si escribimos código **Javacript** el coloreado del texto se adapta a este lenguaje; si escribimos código **PHP**, el coloreado será distinto porque reconoce la sintaxis particular de este lenguaje.

Suelen ser muy rápidos y ligeros. Pero su principal ventaja es que son ideales también para otros fines, aparte de escribir aplicaciones web. Por ejemplo, editar los archivos de configuración de un sistema o escribir texto puro.

Aunque disponen, al menos de base, menos herramientas que los editores más especializados, podemos, a través de plugins y extensiones, incorporar muchas herramientas interesantes.

Editores de este tipo son **Sublime Text**, **Notepad++**, **TextMate** y **Geany** (editor del sistema Ubuntu).

- **Editores ligeros especializados en desarrollo web.** La frontera es muy difusa con los anteriores y, a veces, es difícil diferenciarlos. Su funcionamiento es parecido: son editores rápidos y ligeros y aumentan sus prestaciones instalando extensiones y plugins. La diferencia es que, por defecto, ya vienen preparados con herramientas y elementos que nos facilitan el código de las aplicaciones web HTML, CSS y JavaScript.

Rivalizan con los anteriores y, claramente, pueden ser igual de versátiles. A esta categoría pertenecen los editores **Visual Studio Code de Microsoft**, **Atom de GitHub**, **Brackets de Adobe** y también editores como **Codekit** y **Coda** disponibles solo en entorno *Macintosh*.

- **Entornos de desarrollo integrados (IDE).** Es software mucho más grande en tamaño y con opciones que aglutinan todas las necesidades de los programadores. Su ventaja es que no se echa en falta ninguna herramienta, ya que incorporan todo tipo de posibilidades, extras y ayudas.

A cambio, son entornos de trabajo más pesados, tardan mucho en ejecutarse y suelen ser de pago.

En el ámbito de la programación front-end algunos IDEs populares son **WebStorm**, **Komodo**, **Visual Studio** o **Netbeans**, siendo WebStorm el más utilizado actualmente.

- **Editores WYSIWYG.** No son los ideales para el desarrollo de aplicaciones web, pero pueden complementar el trabajo del desarrollador. Suelen incluir un editor de código, pero su finalidad es crear aplicaciones web viendo al instante el resultado visual. Es decir, son editores visuales de páginas web.

Pueden proporcionar un punto de entrada para el diseño inicial de la aplicación y luego el código escribirlo en otro entorno o bien en el editor incorporado. El más famoso software de este tipo es **Adobe Dreamweaver**. Otros son: el vistoso **CoffeeCup**, el ligero y fácil (y gratuito) **Mobirise** o el clásico **SeaMonkey** (de la fundación **Mozilla**).

- **Editores de código online.** Se trata de aplicaciones web que permiten la escritura de código y previsualizan el resultado al instante. Suelen incluir cierto espacio de almacenamiento para guardar el código en la nube y así acceder al mismo desde cualquier máquina. Su ventaja fundamental es que no requieren instalar nada en nuestra máquina para comprobar el funcionamiento del código.

Los más exitosos de este género de editores son los que están pensados para crear pequeños retazos de código, que pueden incluir código en varios lenguajes como HTML, CSS y JavaScript, para hacer demostración de pequeñas funcionalidades o para acompañar de código a explicaciones y manuales. **JSFiddle**, **CodePen**, **JS Bin** o **CSS Deck** son algunos de los más populares.

3.3. INSTALACIÓN DE VISUAL STUDIO CODE

Visual Studio Code es, quizás, el editor gratuito más utilizado para escribir código. Se trata de un software creado para crear aplicaciones web, pero que se utiliza actualmente para crear todo tipo de aplicaciones gracias al gran número de extensiones que posee.

Está disponible para entornos Windows, MacOS y Linux, lo que le permite ser muy utilizado en todo tipo de entornos.

Entre sus principales ventajas están:

- **La tecnología IntelliSense.** Se trata de una tecnología de Microsoft creada para sus entornos de desarrollo ya hace bastantes años, en especial ha sido uno de los buques insignia de **Visual Studio** (el entorno profesional de Microsoft, que en realidad tiene otra filosofía de trabajo). Esta tecnología es la que permite que el editor nos ayude a escribir aportando sugerencias a medida que escribimos para facilitar la escritura correcta de nuestro código.
- **Depuración.** Permite depurar el código desde el propio editor. Las herramientas de depuración nos permiten hacer pruebas con el código para facilitar la detección de errores.
- **Control de versiones mediante Git.** El control de versiones está integrado en el propio entorno. Tiene una integración total de comandos **Git** que veremos más adelante.
- **Extensiones.** *Visual Studio Code* dispone de un gran número de extensiones que permiten hacer que el editor integre todo lo necesario para trabajar cómodos con él. Inicialmente *Visual Studio Code* tiene preinstaladas pocas extensiones para que nosotros seleccionemos las que nos interesan más.

3.3.1. INSTALACIÓN

Desde la página <https://code.visualstudio.com/download> podemos seleccionar la versión dependiendo de nuestro sistema, que deseamos descargar.

Una vez descargado el instalador, seguiremos sus pasos y podremos utilizar el editor para trabajar.




❑ ACTIVIDAD 1 - INSTALACIÓN DE VISUAL STUDIO CODE



<https://code.visualstudio.com/>

3.4. USO DE VISUAL STUDIO CODE

Posee un menú en la parte superior que contiene la mayoría de opciones de trabajo. El área principal de trabajo la ocupará el archivo en el que estemos trabajando.

En el lado izquierdo aparece un menú vertical con 5 iconos que nos permiten cambiar la vista principal. Estos son:

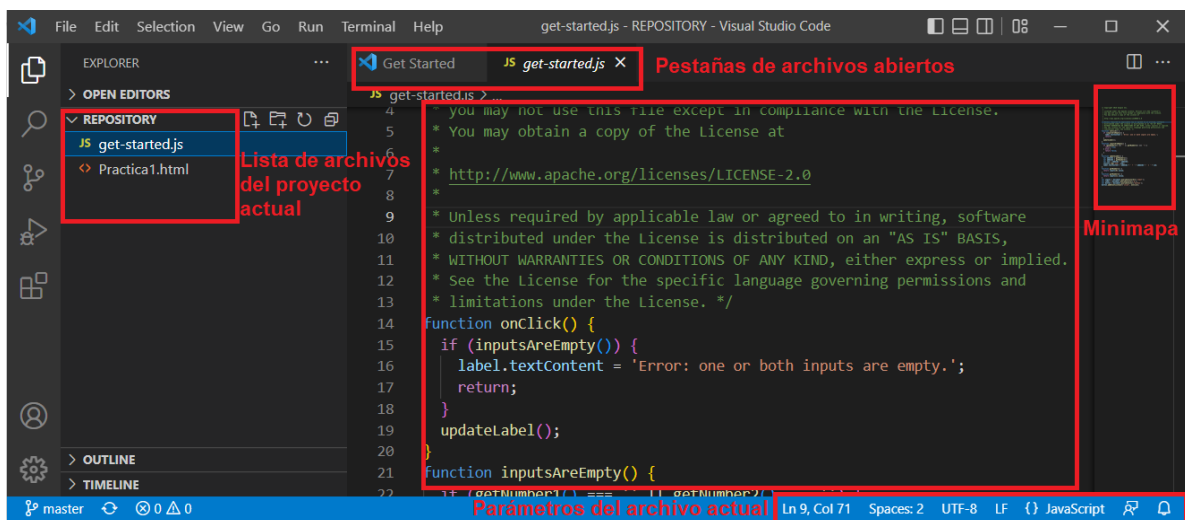
ICONO	USO
	Vista de explorador. Permite ver los archivos y carpetas del proyecto de trabajo actual. Desde esta vista podemos abrir los archivos deseados, borrarlos, cambiar el nombre, moverlos, copiarlos, etc.
	Buscar. Sirve para buscar textos por el archivo actual o incluso por otros archivos. Dispone 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.
	Extensions. Permite añadir y quitar nuevos componentes al editor.

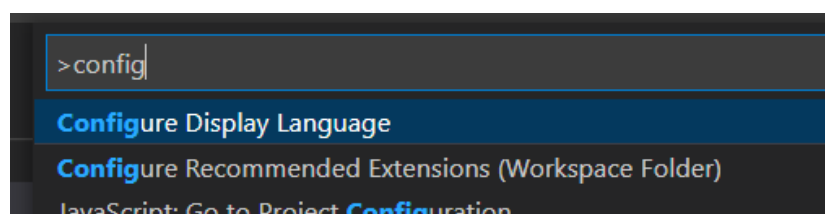
Cuando se trabaja en una aplicación web, todo el código de la misma se coloca dentro de un mismo directorio. Para que *Visual Studio Code* trabaje con ese directorio, deberemos elegir **Archivo – Abrir Carpeta (File – Open Folder)** en los menús y después elegir el directorio raíz de nuestro trabajo. Lo habitual es al menos haber creado el directorio de trabajo.

Una vez abierto el directorio podremos crear más archivos y directorios y realizar todas las operaciones que deseemos sobre los archivos. Si estamos en el explorador de archivos (primer botón de la barra lateral) aparecerá nuestro directorio y su contenido (si es que hay contenido), arrojando el botón al nombre del directorio aparecen botones para crear archivos y carpetas

El área principal lo ocupará el archivo en el que estemos trabajando. Hacer doble clic en el archivo lo abre y se muestra su contenido en el área principal. Podemos abrir más y veremos como cada archivo abierto ocupa una pestaña en la zona superior.



Bastará con escribir el código y guardar los cambios para poder trabajar. Una opción muy interesante es acudir a la paleta de comandos mediante el menú **Ver - Paleta de comandos (View – Command palette)** o pulsar la combinación de teclas **Ctrl+ May+ P**. Esto muestra una ventana que nos permite escribir y buscar cualquier opción disponible en *Visual Studio Code*. Escribiremos **config** y de las opciones que nos aparezcan elegiremos la primera que tenga que ver con el idioma y seleccionaremos **Inglés**.



En la imagen se puede apreciar que no hace falta escribir el término entero, Visual Studio Code va buscando el comando apropiado a medida que escribimos. Además, se nos muestra, si es posible, la combinación de teclas equivalente.

Para conocer más opciones, la documentación oficial de Microsoft sobre Visual Studio Code la encontramos en:

<https://code.visualstudio.com/docs>

ACTIVIDAD 2 ---- INICIÁNDOSE CON VISUAL STUDIO CODE

3.5. CONFIGURACIÓN DE VISUAL STUDIO CODE

3.5.1. CONFIGURACIÓN DEL ENTORNO

La instalación de Visual Studio Code permite empezar a trabajar con el editor directamente, sin embargo, es conveniente personalizar el entorno para poder trabajar de forma más cómoda y eficiente.

La configuración atañe a diversos aspectos como: el tipo de letra y tamaño, a la apariencia general, al funcionamiento de los componentes del editor, a la manera de responder del editor al escribir en cada lenguaje, a decidir cuánta ayuda queremos que nos preste el editor, etc.

IMAGEN

Realmente hay 2 tipos de ajustes:

- **Ajustes de usuario (user settings).** Afectan a todo el funcionamiento de Visual Studio Code independientemente del proyecto en el que estemos trabajando.
- **Ajustes del espacio de trabajo (workspace settings).** Afectan solo al proyecto (espacio de trabajo) actual. Estos ajustes tienen preferencia sobre los del usuario.

Para llegar a los ajustes deberemos elegir los menús: Archivo-Preferencia-Ajustes (File-Preferences-Settings)

La configuración realmente se graba en un archivo en formato JSON, si son ajustes de usuario se graban en el directorio de configuración de usuario de nuestro sistema y si son de espacio de trabajo se graban en el directorio `.vscode` que se creará en el directorio raíz de nuestro proyecto.

3.5.2. EXTENSIONES

El panel de extensiones de Visual Studio Code permite agregar nuevos elementos al editor para mejorar sus prestaciones. También permite quitar aquellos que no nos interesen. Cuantas más extensiones instalemos, más posibilidades tendrá el editor, pero también, más lento se ejecutará. Las extensiones permiten personalizar el editor a nuestra forma de trabajar.

Hay extensiones que permiten usar las combinaciones de teclas de otros editores con los que estemos más familiarizados, otras permiten integrar nuevos componentes o herramientas, otras

manejar nuevos lenguajes, etc. En realidad, hay extensiones para mejorar o modificar casi cada aspecto de Visual Studio Code.

Para gestionar las extensiones de nuestro editor deberemos pulsar el botón de extensiones (último botón del panel de botones izquierdo). Aparece un nuevo panel que nos permitirá ver y modificar las extensiones instaladas y también instalar nuevas a través de un panel de búsqueda que nos permite buscarlas por Internet.

ACTIVIDAD 3 ---- EJEMPLO DE CONFIGURACIÓN

4.NODEJS Y NPM

4.1. ¿QUÉ ES NODEJS?

Node.js es un 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.

Además, es fundamental para utilizar el gestor de paquetes **npm**, que permite instalar numerosas utilidades en el sistema que nos van a ayudar en nuestro trabajo de desarrolladores.

Node.js permite practicar los fundamentos de JavaScript sin necesidad de crear una página web. 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.

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.

4.2. INSTALACIÓN DE NODEJS

La instalación de Node.js depende del entorno en el que estemos, pero, para Windows podemos acceder a la página oficial de node.js (<https://nodejs.org>) y descargar el instalador apropiado (la propia página detecta nuestro sistema). Para Ubuntu, ejecutaremos los siguientes comandos:

```
$ sudo apt update
```

```
$ sudo apt install nodejs
```

Una vez descargado el instalador hay que ejecutarlo como administradores y seguir los pasos indicados en la actividad.

ACTIVIDAD 4 ----INSTALACIÓN DE NODEJS

4.2.1. EL PROBLEMA DE LAS VERSIONES DE NODE

Como ocurre en mucho otro software (especialmente si es de código abierto), Node.js tiene versiones con diferentes tipos de soporte. Actualmente se distinguen 2 versiones principales:

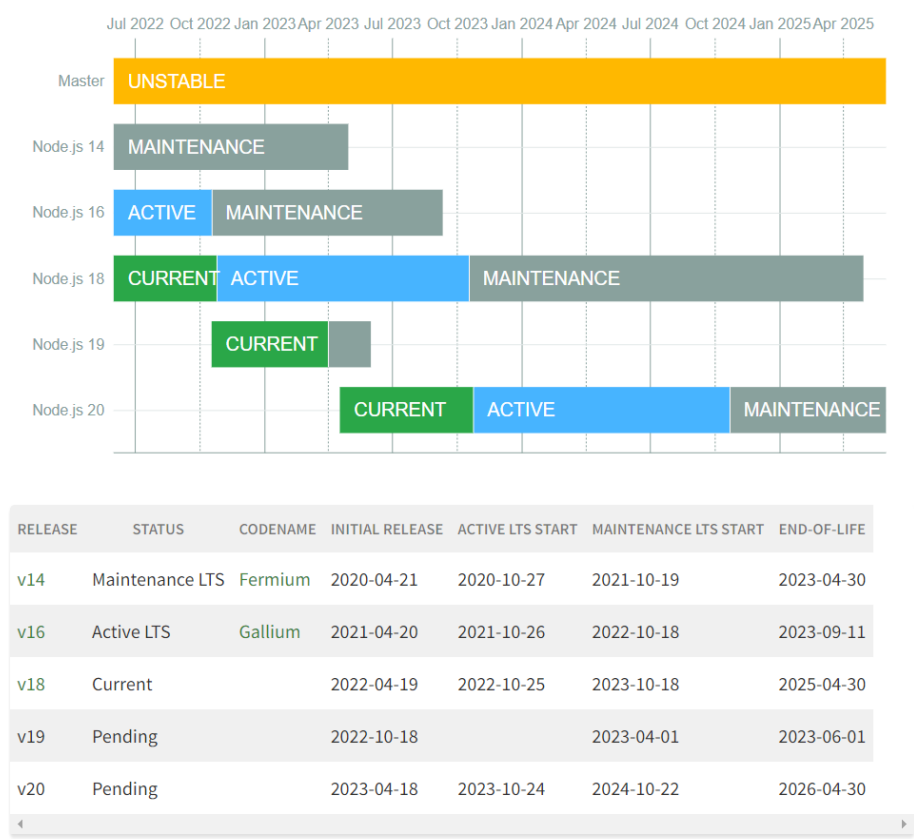
- **LTS. Long Term Support**, Soporte a Largo Plazo. Se trata de la versión de la que se asegura ya estabilidad y que, además, dispondrá de mucho tiempo de ciclo de vida. Es decir, habrá actualizaciones durante mucho tiempo desde su descarga. Es una versión ya final a la que no se añadirán nuevas características de alto nivel. Es la recomendada, en general, para ser instalada por los desarrolladores.
- **Current**. Versión actual. Incorpora las últimas novedades del lenguaje, pero no se considera estable. Puede seguir renovándose e incorporando nuevas funcionalidades. Suele ser cargada para poder probar y aprender el uso de las nuevas capacidades de JavaScript que se han añadido a Node.js.

La versión LTS, es la que se debe descargar.

Ahora bien, cada cierto tiempo aparecen nuevas versiones. Node.js pasa del estado **Current** al estado **Active** (Activo) cada versión en un ciclo de desarrollo predecible y que se puede consultar en <https://nodejs.org/en/about/releases>

Cuando una versión pasa al estado **Maintenance** (Matenimiento) significa que ya no se renueva y que se da soporte solo cuando se detecten problemas y **bugs** en esa versión.

Suele ser un problema el actualizar node.js, porque implica desinstalar la versión anterior e instalar la nueva.



Calendario de versiones de node.js indicando la previsión de versiones activas, en mantenimiento e inestables.

4.3. NVM

Existe una utilidad que permite administrar de forma cómoda la gestión de las versiones de node.js. Se trata de **nvm (Node Version Manager)**.

Cuando esta utilidad está instalada, disponemos del comando **nvm** en el terminal del sistema (independientemente del terminal que estemos usando). Con este comando podremos instalar varias versiones de Node.js (para hacer pruebas o para probar la compatibilidad de nuestras aplicaciones) a la vez.

La instalación de **nvm** en un sistema Linux o MacOS es muy sencilla, basta con usar nuestro gestor de paquetes habitual e instalar el paquete **nvm** de la misma forma que se instala cualquier otro paquete.

En el caso de Windows, la instalación es un poco más compleja. Debemos acudir al repositorio disponible en <https://github.com/coreybutler/nvm-windows> donde se nos informa de la instalación en Windows. Además, en <https://github.com/coreybutler/nvm-windows/releases> podemos descargar el instalador de **nvm** que deseemos.

Realmente hay matices en el funcionamiento de **nvm** para Windows y el **nvm** normal (Linux/Mac), pero, en general, funcionan muy parecido.

Debemos desinstalar cualquier versión de node.js para poder aprovechar toda la potencia de **nvm** a la hora de gestionar las diferentes versiones de node.

Los comandos de **nvm** más interesantes son:

- **nvm install versión**. Instala la versión de node.js que indiquemos. Si la versión la cambiamos por la palabra **latest**, se instala la última estable
- **nvm ls**. Muestra las versiones de node.js disponibles para instalar. Coloca un asterisco en la versión que se utiliza actualmente como versión por defecto.
- **nvm ls available**. Muestra las versiones de node.js disponibles para instalar. En ambiente Linux este comando es **nvm ls-remote**
- **nvm use versión**. Hace que la versión indicada sea la que funcione como versión por defecto.
- **nvm uninstall versión**. Desinstala la versión de node.js indicada.
- **nvm root**. Muestra la ruta del sistema en la que se almacenan físicamente las diferentes versiones de node. Js.
- **nvm -v**. Muestra la versión instalada del propio **nvm**.

ACTIVIDAD 5 --- INSTALACIÓN DE NVM

4.4. NPM

Es fácil confundir **nvm** con **npm**. Ambos son comandos en el sistema relacionados con node.js. Mientras **nvm** es una utilidad para gestionar diferentes versiones de node.js en el mismo sistema, **npm** es una utilidad de instalación de paquetes software.

El administrador de paquetes npm permite instalar todo tipo de utilidades que facilitan mucho el trabajo del desarrollador front-end (también el de otro tipo de desarrolladores). Los paquetes que instalemos los podemos poner a disposición de todo el sistema si usamos el modificador `-g`. De otro modo, el paquete se instala para el directorio en el que estemos. Este segundo método se usa cuando queremos instalar paquetes específicamente para un proyecto.

En nuestro caso la mayoría de paquetes que necesitemos se instalarán con el modificador `-g`, porque queremos que sean utilidades disponibles en cualquier parte del sistema.

Así, algunos comandos a conocer de npm son (se usa en todos ellos el modificador `-g` para referirnos a modificadores globales):

- **npm ls -g.** Muestra los paquetes globales instalados.
- **npm install *paquete* -g.** Instala un paquete en el sistema. Se puede indicar más de un paquete si los separamos con espacios.
- **npm uninstall *paquete* -g.** Desinstala el paquete indicado.
- **npm root.** Muestra la ruta del sistema en la que se almacenan los paquetes descargados.
- **npm search *texto*.** Busca en los repositorios de npm en Internet nombres de paquetes que contengan el texto indicado.
- **npm -v.** Muestra la versión instalada de npm.

ACTIVIDAD 6 --- INSTALACIÓN DE PAQUETES GLOBALES CON NPM.

5.GIT

5.1. SISTEMAS DE CONTROL DE VERSIONES Y GIT

Hay una situación temida por los programadores. Se trata de disponer de un software que funciona correctamente, pero al que tenemos que añadir una funcionalidad o modificar parte de su comportamiento. Cuando intentamos hacer las modificaciones resulta que nuestro programa ya no funciona como antes y tenemos problemas para detectar cuál es el problema. Ante esta situación lo ideal sería volver a la situación anterior.

Este es uno de los ejemplos en los que un sistema de control de versiones nos proporciona su utilidad. Otro típico y muy habitual, ocurre en el caso de que varios programadores estén colaborando para crear un determinado proyecto y trabajan en paralelo y desean mezclar las modificaciones de ambos para que el proyecto refleje ambos trabajos.

En definitiva, 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.

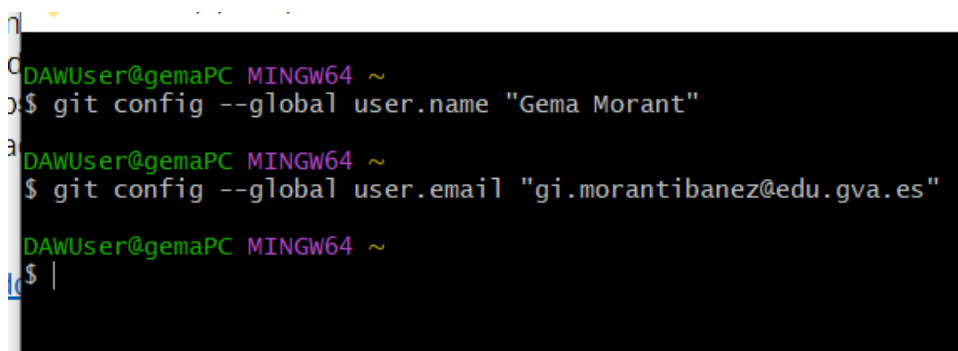
La cuestión del trabajo en equipo se soluciona mediante ramas en las que el desarrollo del proyecto sigue distintos caminos (cada uno con sus versiones o instantáneas) que podremos mezclar a voluntad.

Hay varios posibles sistemas de control de versiones que podemos utilizar. Actualmente el más exitoso es **Git**. Este, fue creado por **Linus Torvals** (conocido por ser la persona que desarrolló Linux) en el año 2005. Git tiene numerosas posibilidades, pero nos centraremos solo en la capacidad de poder grabar versiones de nuestro código a las que podremos acudir cuando queramos.

5.2. INSTALACIÓN DE GIT

Git se puede descargar de <https://git-scm.com/downloads> donde dispondremos de instaladores para todas las versiones.

Una vez instalado dispondremos del comando git en nuestro terminal del sistema. Lo primero que debemos hacer es indicar nuestro nombre con el que se grabará nuestro código, que dará autoría al código que gestionemos con git, así como nuestro email.



```
DAWUser@gemaPC MINGW64 ~  
$ git config --global user.name "Gema Morant"  
DAWUser@gemaPC MINGW64 ~  
$ git config --global user.email "gi.morantibanez@edu.gva.es"  
DAWUser@gemaPC MINGW64 ~  
$ |
```

ACTIVIDAD 7 --- INSTALAR GIT

5.3. FUNCIONAMIENTO BÁSICO DE GIT

Normalmente, un proyecto de trabajo se basa en una carpeta o directorio raíz que contendrá todos los archivos del proyecto. Para que dispongamos del control de versiones basado en Git sobre esa carpeta, deberemos acudir al terminal y situarnos en dicha carpeta. Ahí escribiremos:



```
$ git init
```

Este comando creará un directorio oculto llamado **.git** en el que se almacenarán los archivos que el sistema git necesita para controlar las versiones de nuestro proyecto. El comando solo se escribirá una vez.

A partir de este momento nuestro proyecto puede tener 3 estados:

- **Working Directory.** Estado en el que hemos hecho modificaciones a los archivos, pero no las hemos marcado de ninguna manera para ser tenidos en cuenta.
- **Staging.** Estado de puesta en escena. Se han hecho cambios y se han marcado los que queremos que se tengan en cuenta para la siguiente confirmación. El comando que permite pasar a este estado es **git add**, al que hay que indicar qué archivos se tendrán en cuenta. Si estamos en la raíz de nuestro proyecto y escribimos:

```
$ git add .
```

Entonces todos los cambios realizados en cualquier archivo se tendrán en cuenta.

- **Repositorio.** Con él, los cambios marcados ya están grabados en una instantánea. El comando que hace posible este estado es **git commit**.

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

Tras ese comando se graba una versión de nuestro código a la cual se asigna una clave única e irreplicable. Esa clave es la que nos permite acudir a la versión cuando deseemos.

El estado actual del proyecto lo podemos obtener mediante el comando:

```
$ git status
```

El listado de las versiones que hemos guardado está disponible con este comando:

```
$ git log
```

 (para ver el número de commit, ejecutar `$ git log -oneline`

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

```
$ git checkout 43e6f index.html
```

Este comando devuelve 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**.

Sin indicar archivo alguno, devolveremos nuestro repositorio a ese estado. Sería necesario utilizar ramas para separar esa nueva versión del proyecto.

El comando que devolvería nuestro proyecto a un estado anterior, pero eliminando todos los **commits** a partir de ese momento sería:

```
$ git reset -hard 43e6f
```

ACTIVIDAD 8 --- CREAR VERSIONES CON GIT

5.4. GITHUB. GIT EN REMOTO

Los sistemas de control de versiones se hacen todavía más interesantes cuando se añade la posibilidad de almacenar el código en Internet. En este sentido hay que hablar de GitHub, un servicio que permite almacenar repositorios Git en la nube. GitHub contiene miles de proyectos (la mayoría de código abierto) pensados para que se desarrollen de forma colaborativa. Almacenar nuestros proyectos en GitHub es gratuito y solo requiere darnos de alta.

Una vez de alta, bastará con crear un repositorio en GitHub con el nombre que deseemos. Inicialmente en las cuentas gratuitas solo podíamos crear repositorios públicos, pero desde el año 2019 es posible crear de forma gratuita repositorios privados (con, como mucho, 3 colaboradores).

Hay otros servicios similares a GitHub, como **BitBucket** o **GitLab**, pero GitHub (que actualmente pertenece a Microsoft) es el más popular. En todo caso el uso de repositorios en la nube requiere de estos comandos:

- **git clone URL.** Permite descargar un repositorio en la nube en el directorio en el que estemos situados. Se descargan 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 borrar 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.

ACTIVIDAD 9 --- SUBIR NUESTRO PROYECTO A GITHUB