PROMETEO

Unidad 1: Entorno y herramientas para el desarrollo web

La primera sesión se centra en establecer un entorno de desarrollo completo y profesional. Aprenderemos a instalar y configurar las tres herramientas fundamentales que todo desarrollador web moderno debe dominar: Instalación de Node.js y configuración de npm, configuración de Git para control de versiones, instalación de VSCode con extensiones esenciales y configuración del flujo de trabajo optimizado.

Sesión 1: Instalación de Node.js, Git y VSCode con extensiones clave

Fundamentos

Configurar un entorno de desarrollo profesional es el primer paso, y el más crucial, para construir cualquier proyecto web moderno. No se trata solo de tener un lugar donde escribir código, sino de establecer un flujo de trabajo eficiente, consistente y escalable. Los tres pilares de este entorno son **Node.js**, **Git** y **Visual Studio Code (VSCode)**.

Node.js

Node.js es un entorno de ejecución de JavaScript del lado del servidor, pero en el desarrollo frontend su papel es diferente y fundamental: actúa como el motor que impulsa todo nuestro ecosistema de herramientas. Gracias a Node.js, podemos usar **npm (Node Package Manager)**, el gestor de paquetes más grande del mundo, para instalar y gestionar las miles de librerías y utilidades que optimizan nuestro trabajo, desde frameworks como React o Vue hasta herramientas de testing, empaquetado y optimización.

Git

Git es el sistema de control de versiones distribuido estándar de la industria. Permite a los desarrolladores rastrear cada cambio realizado en el código a lo largo del tiempo. Esto significa que puedes volver a una versión anterior si algo sale mal, comparar cambios, y, lo más importante, colaborar con otros desarrolladores en el mismo proyecto sin pisarse el trabajo. Es como una máquina del tiempo para tu código.

Visual Studio Code

Visual Studio Code (VSCode) se ha convertido en el editor de código por defecto para la mayoría de los desarrolladores web. Es gratuito, de código abierto, ligero y extremadamente potente gracias a su vasto ecosistema de extensiones. Dos de las extensiones más esenciales para empezar son **Live Server**, que crea un servidor de desarrollo local y actualiza automáticamente el navegador cada vez que guardamos un cambio, eliminando la necesidad de recargar manualmente la página, y **Prettier**, un formateador de código que aplica un estilo consistente a todo tu proyecto, asegurando que el código sea legible y profesional sin esfuerzo.

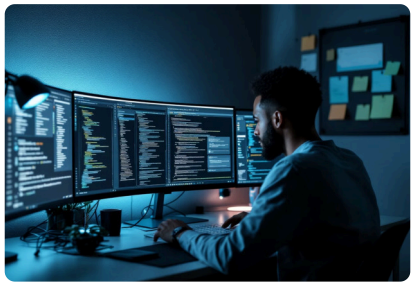
Esquema Visual

Este esquema visualiza los tres componentes fundamentales de un entorno de desarrollo web moderno. **Node.js** actúa como la base que nos da acceso a un ecosistema de herramientas a través de **npm**. **Git** es el sistema que gestiona el historial y la colaboración del proyecto. Finalmente, **VSCode** es nuestro espacio de trabajo, el editor donde escribimos el código, potenciado por **extensiones** como Live Server y Prettier que automatizan tareas y mejoran drásticamente la productividad.

Caso de Estudio: Microsoft y la Revolución de Visual Studio Code

El Contexto:

A principios de la década de 2010, el panorama de los editores de código estaba fragmentado. Los desarrolladores usaban una variedad de herramientas, desde editores de texto simples como Sublime Text hasta entornos de desarrollo integrados (IDEs) pesados y a menudo de pago como WebStorm o los productos de la propia Microsoft, como Visual Studio.



La Estrategia:

En 2015, Microsoft sorprendió al mundo del desarrollo con el lanzamiento de Visual Studio Code. En lugar de seguir su estrategia tradicional de software de pago y cerrado, apostaron por un modelo radicalmente diferente: un editor de código gratuito, de código abierto, multiplataforma (Windows, Mac, Linux) y, sobre todo, ligero y rápido. Su verdadera genialidad fue construirlo sobre una arquitectura extensible, permitiendo a la comunidad crear y compartir miles de extensiones a través de su Marketplace.

El Resultado:

El éxito fue abrumador. VSCode se convirtió rápidamente en el editor de código más popular del mundo, según encuestas como la de Stack Overflow. Empresas de todos los tamaños, desde startups ágiles hasta gigantes tecnológicos como Google, Amazon y Facebook, lo adoptaron como su herramienta principal. VSCode no solo unificó el entorno de trabajo para millones de desarrolladores, sino que también demostró el poder del software de código abierto y centrado en la comunidad, cambiando la percepción de Microsoft dentro del ecosistema del desarrollo.

Herramientas y Consejos

1

Usa un gestor de versiones de Node (nvm)

En lugar de instalar Node.js directamente, utiliza una herramienta como nvm (Node Version Manager) para Mac/Linux o nvm-windows para Windows. Esto te permitirá tener varias versiones de Node.js instaladas en tu sistema y cambiar entre ellas con un simple comando (nvm use 18). Es increíblemente útil cuando trabajas en diferentes proyectos que pueden tener requisitos de versión distintos.

2

Configura Prettier para que formatee al guardar

La verdadera magia de Prettier se desata cuando se ejecuta automáticamente. En los ajustes de VSCode (puedes abrirlos con Ctrl + ,), busca "Format On Save" y activa la casilla. A partir de ese momento, cada vez que guardes un archivo (Ctrl + S), Prettier lo formateará según las reglas del proyecto, ahorrándote tiempo y discusiones sobre estilos de código.

3

Domina la terminal integrada de VSCode

VSCode incluye una terminal integrada muy potente (puedes abrirla con Ctrl + `). Acostúmbrate a usarla para ejecutar tus comandos de Git, npm y otras herramientas de línea de comandos. Tener tu código y tu terminal en la misma ventana agiliza enormemente el flujo de trabajo y te mantiene enfocado

Mitos y Realidades

❌ Mito: "Con el bloc de notas es suficiente para escribir HTML."

→ **FALSO.** Aunque técnicamente es posible escribir código HTML en cualquier editor de texto plano, esta práctica es contraproducente en un entorno profesional. Un editor como VSCode ofrece características esenciales que mejoran significativamente la productividad y la calidad del código: resaltado de sintaxis que facilita la lectura, autocompletado inteligente que previene errores tipográficos, detección de errores en tiempo real, y extensiones especializadas que proporcionan funcionalidades avanzadas como formateo automático, linting, y integración con herramientas de desarrollo.

❌ Mito: "Node.js es solo para hacer backend (servidores)."

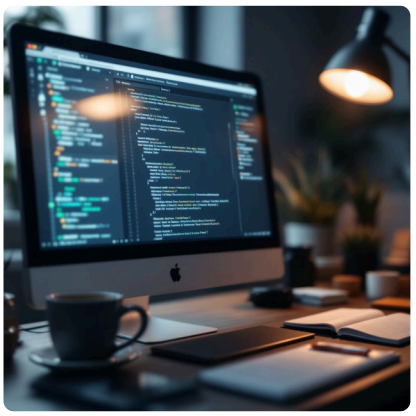
→ **FALSO.** Aunque Node.js nació como un entorno de ejecución para JavaScript en el servidor, en el desarrollo frontend moderno es absolutamente fundamental. Node.js permite ejecutar el ecosistema completo de herramientas de desarrollo web: gestores de paquetes como npm o yarn, bundlers como Webpack o Vite, transpiladores como Babel, preprocesadores de CSS como Sass, herramientas de testing como Jest, y frameworks de desarrollo como Create React App o Vue CLI. Sin Node.js, el desarrollo web moderno sería imposible.

Resumen Final

Aunque Node.js nació como un entorno de ejecución para JavaScript en el servidor, en el desarrollo frontend moderno es absolutamente fundamental. Node.js permite ejecutar el ecosistema completo de herramientas de desarrollo web: gestores de paquetes como npm o yarn, bundlers como Webpack o Vite, transpiladores como Babel, preprocesadores de CSS como Sass, herramientas de testing como Jest, y frameworks de desarrollo como Create React App o Vue CLI. Sin Node.js, el desarrollo web moderno sería imposible.

Sesión 2: Uso de GitHub, conexión SSH y repositorios para proyectos web

La segunda sesión se enfoca en llevar nuestro código a la nube y establecer un flujo de trabajo colaborativo profesional. GitHub se ha convertido en el estándar de facto para alojar código y colaborar en proyectos de desarrollo.



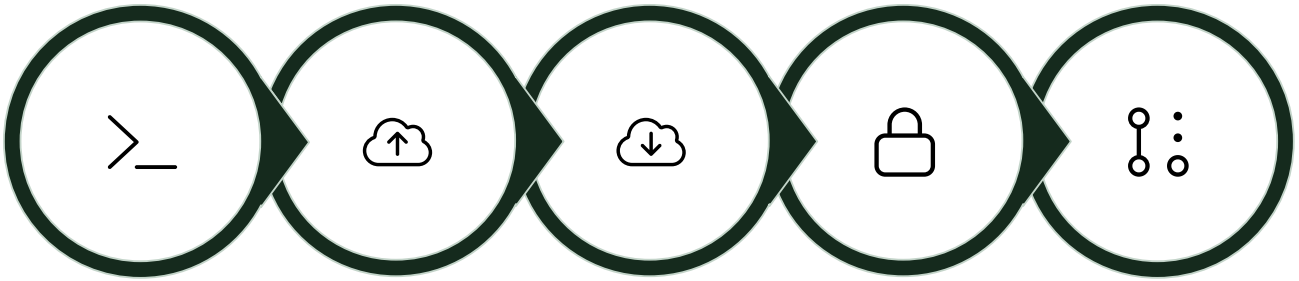
- Configuración de cuenta GitHub y repositorios
- Generación y configuración de claves SSH
- Flujo de trabajo Git: commit, push, pull
- Colaboración mediante Pull Requests
- Gestión de ramas y buenas prácticas

Fundamentos

Para establecer una comunicación segura entre nuestro ordenador y GitHub, utilizamos una **clave SSH (Secure Shell)**. En lugar de usar un nombre de usuario y una contraseña cada vez que nos comunicamos con GitHub, generamos un par de claves criptográficas: una clave privada que guardamos en nuestro ordenador y una clave pública que subimos a nuestra cuenta de GitHub. Cuando intentamos conectarnos, GitHub utiliza este par de claves para verificar nuestra identidad, un método mucho más seguro y conveniente que las contraseñas tradicionales. Si Git es la herramienta que rastrea los cambios en nuestro ordenador, **GitHub** es la plataforma social y colaborativa donde vive nuestro código en la nube. GitHub es una plataforma web que aloja repositorios de Git, permitiendo a los desarrolladores no solo guardar una copia de seguridad de su trabajo, sino, lo que es más importante, colaborar con otros en proyectos de cualquier tamaño. Un **repositorio** en GitHub es el equivalente a una carpeta de proyecto en nuestro ordenador, pero con superpoderes: historial de cambios, gestión de problemas (Issues), revisión de código (Pull Requests) y mucho más.

1	2	3
Trabajar en local	**Confirmar cambios**	**Subir a GitHub**
Escribir código y hacer cambios en el proyecto en nuestro propio ordenador.	Usar git commit para guardar una "instantánea" de los cambios realizados, acompañándola de un mensaje descriptivo.	Usar git push para enviar esos cambios confirmados desde nuestro repositorio local al repositorio remoto en GitHub, actualizando la versión en la nube y haciéndola accesible para otros colaboradores.

Esquema Visual



Caso de Estudio: El Código Fuente de Linux y su Migración a GitHub

El Contexto:

Git fue creado en 2005 por Linus Torvalds, el creador de Linux, específicamente para gestionar el desarrollo del kernel del sistema operativo, un proyecto de código abierto masivo con miles de colaboradores en todo el mundo. Sin embargo, durante más de una década, el código fuente de Linux se alojó en servidores propios y su gestión era compleja, lo que suponía una barrera de entrada para nuevos desarrolladores.

La Estrategia:

A pesar de que GitHub existía desde 2008, la comunidad del kernel de Linux era reacia a depender de una plataforma comercial. Sin embargo, la facilidad de uso, las herramientas de colaboración y la popularidad masiva de GitHub se hicieron innegables. En 2017, el repositorio oficial del kernel de Linux finalmente migró a GitHub.

Este esquema visualiza el flujo de trabajo entre el entorno local y GitHub. Los cambios se realizan y se confirman (git commit) en el **repositorio local**. Luego, se suben (git push) al **repositorio remoto** en GitHub. La conexión entre ambos se asegura mediante una **clave SSH**. Para colaborar, otros pueden revisar los cambios a través de un **Pull Request** antes de que se integren en la rama principal. Para obtener los cambios más recientes de otros, se utiliza git pull.

El Resultado: La migración fue un hito que consolidó a GitHub como el centro neurálgico del desarrollo de software de código abierto. Facilitó enormemente el proceso de contribución para miles de desarrolladores, haciendo que el seguimiento de cambios, la revisión de código y la discusión de nuevas características fueran mucho más transparentes y accesibles. Demostró que incluso los proyectos más grandes y complejos del mundo podían beneficiarse de una plataforma de colaboración bien diseñada.

Herramientas y Consejos

1

Crea un archivo .gitignore desde el principio

Antes de hacer tu primer git commit, crea un archivo llamado .gitignore en la raíz de tu proyecto. Este archivo de texto le dice a Git qué archivos y carpetas debe ignorar. Es crucial añadir aquí la carpeta node_modules (que puede contener miles de archivos y nunca debe subirse), así como archivos de configuración del sistema operativo (como .DS_Store en Mac) o archivos de entorno (.env) que contengan claves secretas.

2

Escribe mensajes de commit claros y en imperativo

Un buen mensaje de commit es fundamental para entender el historial del proyecto. El estándar de la industria es escribirlos en modo imperativo, como si dieras una orden. Por ejemplo: "Añade la sección de contacto" en lugar de "He añadido la sección de contacto" o "Añadiendo la sección de contacto". Esto crea un historial limpio y legible.

3

Trabaja siempre en ramas (branches)

Nunca hagas cambios directamente en la rama principal (main o master). Para cada nueva característica o corrección, crea una nueva rama con un nombre descriptivo (ej. git checkout -b feature/user-login). Trabaja en esa rama y, una vez que hayas terminado y probado tus cambios, intégralos en la rama principal a través de un Pull Request. Esto mantiene la rama main siempre estable y funcional.

Mitos y Realidades

❌ Mito: "Git y GitHub son lo mismo."

→ **FALSO.** Esta es la confusión más común. **Git** es el sistema de control de versiones, una herramienta de línea de comandos que se ejecuta en tu ordenador y rastrea los cambios. **GitHub** es una plataforma web, una empresa (propiedad de Microsoft) que aloja repositorios de Git y proporciona una interfaz gráfica y herramientas de colaboración (Pull Requests, Issues, etc.) sobre Git.

❌ Mito: "GitHub es solo para guardar copias de seguridad de mi código."

→ **FALSO.** Si bien tener tu código en la nube es una excelente copia de seguridad, ese no es su propósito principal. El verdadero poder de GitHub reside en la **colaboración**. Permite a equipos de desarrolladores trabajar en el mismo código de forma asíncrona, revisar las contribuciones de los demás antes de integrarlas, gestionar tareas y errores, y mantener un historial completo y transparente del proyecto.

Resumen Final

GitHub es la plataforma estándar para alojar repositorios de Git y colaborar en proyectos. La conexión segura se realiza mediante **claves SSH**. Un flujo de trabajo profesional implica el uso de .gitignore para ignorar archivos innecesarios, escribir mensajes de commit claros y trabajar en ramas para mantener la estabilidad del proyecto. Git es la herramienta, GitHub es la plataforma social que le da un hogar social y colaborativo en la nube.