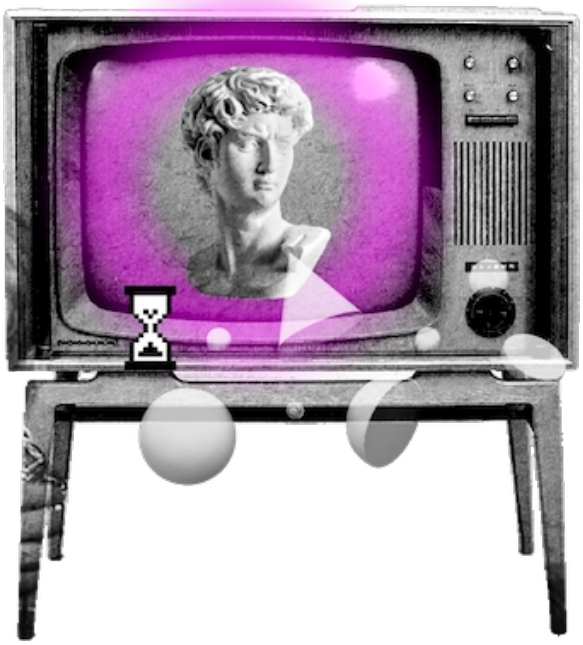
NUCBA

DISEÑO WEB

GIT + GITHUB II



APUNTES

GIT + GITHUB
Branches, merges y Vercel

Branches o Ramas

Las **ramas** en Git son líneas separadas de desarrollo dentro de un mismo repositorio. Cada rama representa una versión independiente del proyecto, lo que permite a los desarrolladores trabajar en **diferentes funcionalidades** o correcciones de errores **al mismo tiempo** sin afectar la rama principal.

Estas son útiles porque permiten una **mayor flexibilidad y organización en el desarrollo de software**.

En resumen, entre las principales ventajas de utilizar **ramas** podemos mencionar:

1. **Desarrollo de características**: Cada nueva característica puede ser desarrollada en una rama separada, lo que permite trabajar en varias funcionalidades al mismo tiempo sin interferir entre ellas.

2. **Corrección de errores**: Las ramas pueden ser utilizadas para identificar y corregir errores específicos en un proyecto. De esta manera, el trabajo de corrección de errores no afecta la rama principal del proyecto.

3. **Experimentación**: Las ramas pueden ser utilizadas para realizar pruebas y experimentos con nuevas soluciones sin afectar el código principal.

4. **Integración de contribuciones externas**: Las ramas pueden ser utilizadas para integrar contribuciones externas, como pull requests en un proyecto open-source, de forma aislada antes de fusionarlas con la rama principal.

Comandos para trabajar con ramas

A continuación vamos a ver los comandos más comunes para trabajar con ramas en Git.

Crear ramas - Git Branch *rama*

Para crear una nueva rama en Git, se puede usar el comando **"git branch"** seguido del nombre de la nueva rama.

```
PC@DESKTOP-JSA96B6 MINGW64 ~
$ git branch *nombre-de-la-rama*
```

Listar ramas - Git Branch

Para ver una lista de todas las ramas en un repositorio Git, se puede usar el comando **"git branch"** . Las ramas activas se marcan con un asterisco (*).

```
PC@DESKTOP-JSA96B6 MINGW64 ~
$ git branch
```

Cambiar de rama - Git Checkout *rama*

Para cambiar a una rama en Git, se puede usar el comando **"git checkout"** seguido del nombre de la rama.

```
PC@DESKTOP-JSA96B6 MINGW64 ~
$ git checkout *nombre-de-la-rama*
```

Crear rama y cambiar a la rama creada - Git Checkout -b *rama*

Para crear una rama y al mismo tiempo cambiar a esa rama creada, podemos utilizar el comando **“git checkout -b”** seguido por el nombre de la rama.

```
PC@DESKTOP-JSA96B6 MINGW64 ~
$ git checkout -b *nombre-de-la-rama*
```

Borrar una rama - Git Branch -d *rama*

Para borrar una rama en Git, se puede usar el comando **"git branch -d"** seguido del nombre de la rama.

```
PC@DESKTOP-JSA96B6 MINGW64 ~
$ git branch -d *nombre-de-la-rama*
```

Vincular una rama nueva con el repositorio remoto

Para vincular la rama local con un repositorio remoto, se puede usar el comando **"git push -u origin nombre-de-la-rama"**. En este caso, **"origin"** es el nombre predeterminado del repositorio remoto y **"nombre-de-la-rama"** es el nombre de la rama local.

Importante: Esto funcionará siempre y cuando se hayan realizado los demás pasos necesarios para subir algo a **Github** (add y commit).

GIT Merge

Imaginemos que estamos trabajando en un proyecto de software con un equipo de desarrolladores.

Cada uno de ellos puede estar trabajando en una rama separada, pero eventualmente es necesario combinar esos cambios en una sola rama para tener una versión consolidada del proyecto. **Este proceso de fusión se conoce como "merge".**

Cuando se realiza un **merge**, Git **compara los cambios de ambas ramas** y **crea una nueva versión con todos los cambios combinados**.

¿Qué es un conflicto?

Un **conflicto de merge** ocurre cuando los cambios realizados en una rama son **mutuamente excluyentes** con los cambios realizados en otra rama. Por ejemplo, si un desarrollador modifica una línea de código en una rama y otro desarrollador modifica la misma línea en otra rama, Git no sabe cuál versión es la correcta y genera un conflicto de merge.

Para resolver un conflicto de merge, es necesario **abrir el archivo en conflicto** y editarlo manualmente para incluir los cambios deseados. Luego, se debe confirmar los cambios resolviendo el conflicto y **agregando los cambios al repositorio con un commit**.

Es importante tener en cuenta que es necesario **ser cuidadoso al resolver conflictos de merge** ya que los errores en la resolución pueden tener **graves consecuencias** para la integridad del código. Por esta razón, es recomendable **revisar cuidadosamente** los cambios antes de confirmarlos y **trabajar en equipo** para resolver conflictos de manera **eficiente y segura**.

Tipos de merging

Hay tres tipos principales de merge en Git: merge **automático**, merge **manual** y merge **con conflicto**.

1. **Merge automático**: Es el tipo más sencillo de merge en el que Git combina los cambios de manera automática sin conflictos. Este tipo de merge se realiza cuando las ramas a fusionar no tienen cambios que entren en conflicto entre sí.

2. **Merge manual**: Es un tipo de merge en el que Git identifica conflictos entre las ramas y requiere la intervención manual del usuario para resolverlos. Este tipo de merge se realiza cuando las ramas a fusionar tienen cambios que entran en conflicto entre sí.

3. **Merge con conflicto**: Es un tipo de merge en el que Git identifica conflictos entre las ramas y no puede resolverlos automáticamente. Este tipo de merge requiere la intervención manual del usuario para resolver los conflictos y completar el merge.

Para realizar un merge en Git, primero se debe cambiar a la rama donde se quiere fusionar los cambios y luego se debe usar el comando **"git merge"** seguido del nombre de la rama con los cambios que se desean fusionar. Git realizará el merge y, en caso de conflicto, indicará los archivos que tienen conflictos y requieren la intervención manual del usuario.

Buenas prácticas para trabajar con ramas y merges

1. **Utilice ramas para organizar su trabajo**: Crear ramas separadas para diferentes funcionalidades o características es una forma eficiente de organizar su trabajo y evitar interrupciones en el código principal.

2. **Mantenga las ramas separadas hasta que estén listas**: Es recomendable mantener las ramas separadas hasta que estén completamente desarrolladas y probadas, antes de fusionarlas en la rama principal.

3. **Realice pruebas antes de fusionar**: Antes de fusionar una rama en la rama principal, es importante realizar pruebas exhaustivas para asegurarse de que el código funciona correctamente y no tiene errores.

4. **Documentación detallada**: Mantenga un registro detallado de los cambios realizados en cada rama y los motivos detrás de ellos, especialmente en el caso de conflictos de merge. Esto ayudará a su equipo a entender las decisiones tomadas y a solucionar problemas en el futuro.

La mayoría de estos consejos tendrán un mayor sentido cuando comiencen a trabajar en equipos de desarrollo, pero siempre está bueno que vayan sabiendo de antemano buenas costumbres a la hora de trabajar con este tipo de herramientas.

Deployar un proyecto

Deployar un proyecto significa **publicar o desplegar una aplicación o sitio web en un servidor o plataforma en línea para que esté disponible públicamente**. Es decir, deployar es el proceso de tomar el código de tu proyecto y colocarlo en un servidor donde se ejecutará y estará disponible para el público.

El proceso de deployar un proyecto web incluye tareas como la **configuración del servidor**, la subida de archivos y la configuración de la base de datos. La forma en que se deploya un proyecto web depende de la tecnología y plataforma utilizadas, pero en general es un proceso que requiere un conocimiento técnico y un proceso detallado.

Por suerte, para personas que recién se están iniciando en este mundo, existen plataformas como **Vercel** que facilitan ese proceso,

¿Qué es Vercel?

Vercel es una plataforma de desarrollo web y hospedaje que permite a los desarrolladores **crear y desplegar fácilmente aplicaciones web y sitios estáticos**. Ofrece una variedad de herramientas y recursos para ayudar a los desarrolladores a optimizar su flujo de trabajo y mejorar la velocidad y la experiencia de carga de sus aplicaciones. Vercel es conocido por su **integración con Git** y su enfoque en el desarrollo de aplicaciones con **tecnologías modernas**, como React y Next.js. También ofrece opciones de hospedaje de alto rendimiento y escalabilidad para proyectos de cualquier tamaño.

Error 404 en Vercel

Un error común con el que nos podemos encontrar al tratar de deployar nuestra primera página en Vercel es el **error 404**.

Este error implica que el servidor no pudo encontrar la página que estás buscando.

Normalmente, este problema se debe a un problema con el archivo **index.html**.

Este archivo es el archivo principal que se carga cuando un usuario accede a la raíz de tu sitio web en Vercel. Si el archivo **index.html** no existe o está mal configurado, puede causar un error 404.

Para solucionar esto, debemos asegurarnos de **dos cosas**:

1. **Tiene que existir si o si un archivo index.html en nuestro proyecto**. Vercel lo tomará como **punto de entrada a la aplicación** y por lo tanto será la primera página que intentará renderizar. Por ejemplo, si estuviéramos haciendo una página institucional con varias sub-páginas (y en consecuencia, varios archivos html), debemos asegurarnos que la **landing page (página principal) se realice en el archivo index.html.** En caso de no existir este archivo, Vercel nos arroja el error.

2. **El archivo index.html debe estar en la carpeta raíz de nuestro proyecto.** No debemos colocarlo dentro de subcarpetas que están en nuestro proyecto, debemos ponerlo en el nivel principal, conocido como la raíz del proyecto.

Obviamente, estas cosas son configurables, pero es algo por lo cual no deben preocuparse ahora que recién están dando sus primeros pasos en la carrera.

Nucba tip: *Sigan investigando sobre los temas aquí expuestos. Practiquen, apoyense en la documentación oficial y sobre todo mantengan la constancia y la curiosidad a medida que vayan aprendiendo cosas nuevas.*

#HappyCoding 🚀