DigitalHouse >
Coding School

Full Stack Web Development

índice

> ¿Qué es Git?

Es un **software** de **control** de **versiones** que **registra** los **cambios** realizados sobre un archivo o conjunto de archivos a lo largo del **tiempo**. De esta forma, podemos recuperar y tener **acceso** a versiones específicas cuando queramos.

> ¿Por qué usar un sistema de control de versiones?

Usar un sistema de control de versiones (VSC), te permite **revertir archivos** y **proyectos enteros** a un estado anterior, **comparar cambios** a lo largo del tiempo, ver **quién modificó** por última vez algo que puede estar causando un problema, quién introdujo un error y cuándo, y mucho más.

> Instalación de Git

- Ir a la web oficial y descargar el ejecutable.
- Ejecutar el archivo que descargamos.
- Si tu sistema operativo es Windows, además de instalarse Git, se instalará en tu máquina una terminal llamada **Git Bash**.
- Una vez instalado Git, estará disponible el comando `git` para correr en la terminal.
- Para verificar que la instalación se haya realizado correctamente, abrir una terminal y correr el comando `git --version`.

> ¿Qué es GitHub?

GitHub es un **sitio web** en donde podemos **almacenar** los **archivos** y **proyectos** de programación de manera **gratuita**. Para poder hacer uso de sus beneficios solo hace falta crearse una cuenta en la plataforma.

> ¿Qué es un repositorio?

Es el lugar donde se irán **almacenando** los **archivos** de nuestro proyecto. En GitHub podemos tener la cantidad de proyectos que queramos, en donde a **cada proyecto** le corresponderá **un repositorio**.

> Tipos de repositorios

Los repositorios que se alojan en GitHub los llamamos repositorios **remotos**, mientras que a los que se alojan en nuestra PC los llamamos repositorios **locales**. Es necesario crear un **vínculo** entre ambos para poder mantener **actualizados** los archivos locales que están conectados a ese repositorio en la nube.

> Crear un repositorio remoto (GitHub)

- Una vez iniciada la sesión en GitHub, hacer click en el ícono **+** sobre la barra principal y elegir la opción **New repository**.
- Veremos en pantalla un formulario.
- Completar únicamente el nombre que le queremos dar a nuestro repositorio. Podemos nombrarlo como queramos, pero debe ser un nombre que no hayamos usado para otro repositorio.
- Ir hacia abajo de todo y apretar el botón **Create Repository**.

> Crear un repositorio local (PC)

- Crear una carpeta en nuestra PC que será donde alojaremos nuestro proyecto. Esta carpeta será nuestro **repositorio local**.
- Dentro de la carpeta, abrir una terminal y correr el comando `git init`.
- Este comando **inicializa** un **repositorio local** en la carpeta del proyecto.

> Agregar nuestra identidad al repositorio local

Para que Git pueda hacer un **completo seguimiento** de los **cambios** realizados, necesitamos decirle al repositorio quiénes somos.

- Abrir una terminal en la ubicación de nuestro repositorio local.
- Correr el comando `git config user.name "nombreDeUsuario"`, en donde, entre comillas, debemos escribir nuestro nombre de usuario tal cual aparezca en GitHub.
- Para verificar que ingresamos bien nuestro nombre de usuario, correr el comando `git config user.name` y presionar *enter*.
- Correr el comando `git config user.email "nombre@email.com"`, en donde, entre comillas, debemos escribir el e-mail con el que nos registramos en GitHub.
- Para verificar que ingresamos bien nuestro e-mail, correr el comando `git config user.email` y presionar *enter*.

Para configurar nuestra identidad de manera global y no tener que estar aclarando siempre nuestro e-mail y nombre de usuario, agregar la palabra `--global`.

- `git config --gobal user.name "nombreDeUsuario"`.
- `git config --gobal user.email "nombre@email.com"`.

> Conectar repositorio local con repositorio remoto

Para que nuestro repositorio local sepa a dónde queremos subir nuestros archivos hace falta especificarlo.

- Tener creado previamente un repositorio en GitHub.
- Ir a la ubicación del mismo y copiar la URL.
- Escribir el comando `git remote add origin`.
- Pegar la **URL** después de la palabra *origin* (dejando un espacio de por medio) y presionar *enter*.
- Para verificar que el paso anterior se ejecutó correctamente, correr el comando `git remote -v`. Deberíamos ver en la terminal la palabra `origin` seguida de la URL.

> ¿Qué es un commit?

Cada vez que subimos archivos (nuevos o modificados) a un repositorio remoto, se suben en forma de un pequeño **paquete de archivos**. Cada paquete tiene una **fecha de creación** (*timestamp*) y un **autor**.

Es a través de los **commits** que vamos a poder hacer el seguimiento de los cambios que se van realizando en los proyectos, ya que cada uno de ellos genera un **punto cronológico** en la línea del tiempo del proyecto.

> ¿Qué significa que un archivo está en seguimiento?

Cuando enviamos un archivo al repositorio, estamos diciéndole a Git que queremos hacerle un **seguimiento** al mismo a través del tiempo.

Es decir, queremos que se guarde el **estado actual** de ese archivo para que cada vez que hagamos un cambio nuevo y lo enviemos, podamos **comparar estados** y ver cómo estaba en determinado momento. **Seguirlo** a lo largo del proyecto.

> Subiendo archivos a un repositorio remoto

Para poder subir nuestros archivos a la nube, debemos seguir los siguientes pasos:

- Abrir una terminal en la ubicación de nuestro repositorio local.
- Correr el comando `git status` para ver el **estado** de nuestros archivos (aquellos en rojo son los archivos que aún no están en seguimiento).
- Correr el comando `git add .` para indicar que queremos **agregar todos** los archivos al repositorio.
- Para **agregar** un **solo archivo**, correr el comando `git add archivo.extension` en donde deberemos indicar tanto el nombre como la extensión del archivo.
- Correr el comando `git status` para ver el **estado** de nuestros archivos (aquellos en verde son los archivos que serán agregados al repositorio, por lo tanto estarán en seguimiento).

- Para **confirmar** que queremos subir de manera definitiva aquellos archivos que agregamos, correr el comando `git commit -m "mensaje"`, en donde, entre comillas, deberemos escribir, en lo posible, un mensaje corto que resuma el trabajo que estamos subiendo.
- Para **enviar** los archivos al repositorio remoto correr el comando `git push origin master`.

> Actualizando archivos de un repositorio local

Para poder actualizar los archivos de nuestro repositorio local con respecto a los que están en el repositorio remoto, debemos correr el comando `git pull origin master`.

> Clonando archivos de un repositorio remoto

Para **descargar** por primera vez un repositorio remoto a nuestra máquina, tendremos que **clonarlo**.

- **Abrir** una terminal en la ubicación en donde queramos clonar el proyecto.
- Copiar la **URL** del repositorio que queremos clonar.
- Escribir el comando `git clone`.
- Pegar la **URL** después de la palabra clone (dejando un espacio de por medio) y presionar enter.

> Resolviendo conflictos

Una de las grandes ventajas de Git y de GitHub es que **muchas personas** pueden trabajar en el **mismo proyecto** en paralelo. En este escenario, es muy posible que dos o más personas modifiquen el mismo archivo y es ahí cuando suelen aparecen los **conflictos**.

> Detectando el conflicto

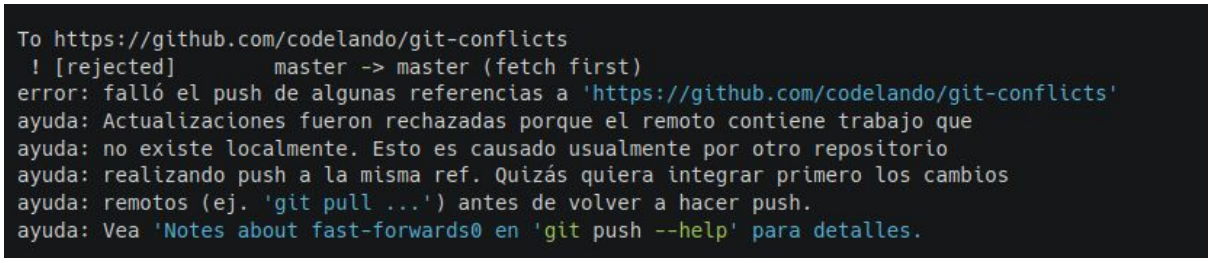
Imaginemos que estamos trabajando los estilos de un sitio web, en el archivo `styles.css`. Al mismo tiempo, otro de los miembros del equipo —vamos a llamarla Juana— decide hacer una modificación **en ese mismo archivo** y sube inmediatamente sus cambios al repositorio:

```
body {
  background-color: yellow;
  font-family: monospace;
}
```

Nosotros, sin saber que Juana subió esos cambios, seguimos trabajando en nuestro archivo normalmente, pero a diferencia de ella, en la línea 2 le asignamos un fondo azul al body:

```
body {
  background-color: blue;
  font-family: monospace;
}
```

Minutos más tarde, cuando terminamos con nuestros cambios y hacemos el commit y el push la consola nos devuelve un hermoso error que parece estar escrito en élfico antiguo...



```
To https://github.com/codelando/git-conflicts
 ! [rejected]        master -> master (fetch first)
error: falló el push de algunas referencias a 'https://github.com/codelando/git-conflicts'
ayuda: Actualizaciones fueron rechazadas porque el remoto contiene trabajo que
ayuda: no existe localmente. Esto es causado usualmente por otro repositorio
ayuda: realizando push a la misma ref. Quizás quiera integrar primero los cambios
ayuda: remotos (ej. 'git pull ...') antes de volver a hacer push.
ayuda: Vea 'Notes about fast-forwards0 en 'git push --help' para detalles.
```

Pero, **¿qué quiere decir este mensaje?**

Que Juana nos ganó de mano y hay **nuevos cambios** en GitHub y no coincide la versión de lo que tenemos en el repositorio local con lo que está en el repositorio remoto. Git nos dice: "Quizás quiera integrar primero los cambios remotos antes de volver a hacer push".

Por lo tanto, sabiendo que en el repositorio remoto existen cambios que **no tenemos**, el próximo paso será actualizarnos y traernos esos cambios con el comando: `git pull origin master`.

Y ahora, la consola nos devuelve otro mensaje jeroglífico:

```
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 1), reused 3 (delta 1), pack-reused 0
Desempaquetando objetos: 100% (3/3), listo.
Desde https://github.com/codelando/git-conflicts
   e7e0905..3216f3f  master     -> origin/master
Auto-fusionando styles.css
CONFLICTO (contenido): Conflicto de fusión en styles.css
Fusión automática falló; arregle los conflictos y luego realice un commit con el resultado.
```

De este mensaje solo nos importan las tres últimas líneas:

```
Auto-fusionando styles.css
CONFLICTO (contenido): Conflicto de fusión en styles.css
Fusión automática falló; arregle los conflictos y luego realice un commit con el resultado.
```

¿Qué pasó?

Git intentó fusionar nuestros cambios junto con los que trajimos con el pull y no pudo: nosotros y Juana editamos la misma línea y Git no sabe con cuál de los dos cambios se tiene que quedar. **Es nuestro trabajo aclarárselo**.

> Cómo se ve el conflicto

En nuestro editor de código veremos lo siguiente **por cada conflicto que haya en el proyecto**: en este caso sabemos que el conflicto está en `styles.css`.

```
<<<<<<< HEAD
    background-color: blue;
=======
    background-color: yellow;
>>>>>>> 3216f3fd5ca65cfd3252ae76808d8f659a715fa6
```

La **primera parte**, entre <<<<<<< HEAD y ======= será nuestra **versión local**, el cambio que hicimos y queremos subir a GitHub.

La **segunda parte**, entre ======= y >>>>>>> será **lo que nos llega desde GitHub** (en otras palabras, los cambios de Juana).

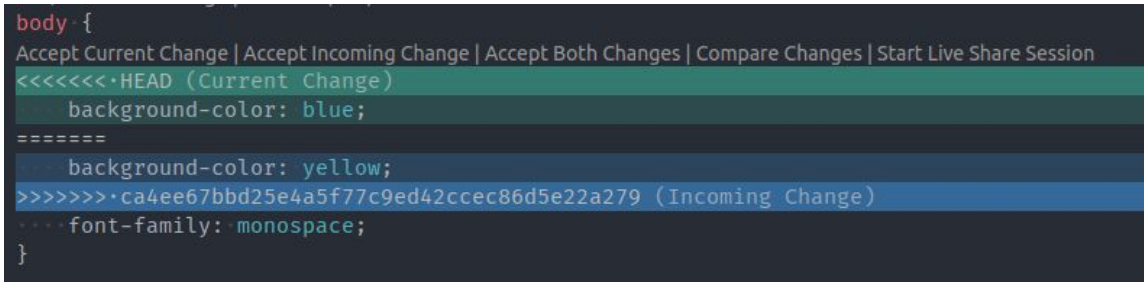
> Resolviendo el conflicto

Por cada uno de estos conflictos tendremos que decidirnos por alguna de estas tres opciones:

- Dejamos nuestro código, versión local.
- Dejamos el código que viene de GitHub.
- Unimos ambos.

Sin importar cuál de los tres caminos tomemos, no tenemos que olvidarnos de borrar las tres líneas de texto que nos agrega Git para identificar la zona del conflicto.

Si estamos trabajando con un editor moderno, como es VS Code, lo más probable es que comprenda este formato y nos dé alguna opción para resolver el conflicto con un **solo click**:



Para algunos conflictos, Git podrá determinar cómo unificar los cambios de manera **automática**. En esos casos todo este trabajo será hecho automáticamente.

Una vez que Git termine de fusionar todos los cambios, la consola nos devolverá un mensaje como este:

Lo que nos pide Git es que agreguemos, si así lo deseamos, un mensaje que explique por qué ese merge fue necesario. Si no escribimos nada, por defecto quedará registrado el mensaje de arriba: "Merge branch...".

Para salir de la consola sin escribir un mensaje en particular, tendremos que apretar `:q` , o `Ctrl+x` .

Si nos interesa aprender cómo escribir un mensaje, podemos investigar por nuestra cuenta los editores de texto **Nano** y **Vim**.

> Subiendo los archivos

Una vez que hayamos quedado conformes con el cambio, vamos a repetir el proceso como si se tratara de un nuevo cambio. Es decir, será necesario volver a **agregar** el archivo con `git add` , en este caso `styles.css`, incluirlo en un commit y subirlo al servidor con un `git push` .

> Evitando conflictos

Una buena manera de evitar conflictos es mantener los commit relativamente pequeños y subir al repositorio frecuentemente. De esta manera, tenemos menos probabilidades de que ocurran conflictos y de que, si ocurren, sean **pequeños**.

Otra manera un poco más avanzada es el uso de ramas para trabajar en paralelo a la versión principal del proyecto. Los invitamos a investigarlas por su cuenta.

Conflictos: flujo completo

- Tenemos listos los cambios que queremos subir al repositorio.
- Hacemos el add, el commit y el push.
- La consola nos devuelve un mensaje de error. No se pudo hacer el push porque en el repositorio remoto existen cambios que todavía no tenemos en nuestro repositorio local y no coinciden las versiones.
- Hacemos un pull para actualizar nuestro repositorio local y traer esos cambios.
- Si no hay conflictos entre lo que teníamos y lo que trajimos, Git va a fusionar todos los cambios de forma automática.
    - En este caso, la consola nos va a pedir que escribamos un mensaje de commit para explicar el merge que acaba de suceder. Podemos escribir un mensaje, o salir de la consola apretando las teclas `:q` o `Ctrl+x`.
    - Solo nos queda hacer el push para mandar nuestros cambios al repositorio remoto.
- Si no se puede resolver de forma automática, la consola va a devolver un nuevo mensaje dejándonos saber que existen conflictos que tenemos que resolver a mano antes de poder subir nuestros cambios al repositorio remoto.
- Vamos al editor de texto para ver cuáles son los archivos conflictuados.
- Git nos deja saber la zona del conflicto de la siguiente manera:
  <<<<<<< HEAD
  Todo lo que esté entre estas dos líneas serán *nuestros* **cambios locales**, *lo que estuvimos trabajando y queremos integrar con el resto del proyecto*.
  =======
  Todo lo que esté entre estas dos líneas serán los cambios que trajimos desde el repositorio remoto con el pull.
  >>>>>>> 3216f3fd5ca65cfd3252ae76808d8f659a715fa6
- Por cada conflicto que encontremos, tendremos que decidir si:
    - Nos quedamos con nuestro cambio y borramos lo que trajimos.
    - Nos quedamos con lo que trajimos y borramos nuestro cambio.
    - Nos quedamos con los dos.

- Resolvemos el conflicto tomando alguno de estos tres caminos y borramos las líneas de texto que agregó Git: <<<<<<< HEAD, ======= y **>>>>>>> 3216f3fd5ca65cfd3252ae76808d8f659a715fa6**
- Una vez resueltos todos los conflictos, hacemos el add, el commit y el push para subir estos nuevos cambios al repositorio remoto.

¡Hasta la próxima!