Trabajando con Git y GitHub

¡Comencemos!

¿Qué aprenderás?

- Aplicar el procedimiento de habilitación de una página web con Github Pages, para que el contenido esté disponible públicamente.

Introducción

Ahora que hemos aprendido a trabajar con los comandos básicos de GitHub, abordaremos el concepto central de ramas o branch. Manejar esto, nos permitirá hacer un mejor uso de los recursos colaborativos de esta herramienta. Por ejemplo, podemos definir versiones de prueba e intentar distintas soluciones para un mismo problema sin afectar al código final.

Veremos además cómo utilizar GitHub pages y qué son los dominios personalizados, para permitir que los visitantes puedan ver nuestra página. Con todos estos recursos, ya seremos capaces de publicar sitios web y compartir contenido con el mundo.

¡Vamos con todo!

Git branch

Ahora que ya conocemos Git y GitHub y su uso básico, vamos a profundizar más en sus herramientas.

Una de las principales ventajas de git es la utilización de branch (ramas). Primero definiremos una rama simplemente como un camino donde está nuestro código, mientras que la definición técnica se refiere a un branch como un apuntador a donde van los commits que realizamos.

Cada vez que inicializamos git, de forma automática se genera un branch main, que es donde se guardarán los nuevos commits.

Git nos da la posibilidad de generar más de una rama para poder almacenar nuestros cambios, de modo que podamos probar nuevas funcionalidades y ordenar el trabajo colaborativo de forma más ordenada y sin dañar el trabajo ya realizado.

Conociendo las ramas del proyecto

Veámoslo paso a paso. Al iniciar git en una carpeta, de forma automática se genera la rama main. Podremos conocer las ramas que tiene nuestro proyecto con el comando:

```
git branch
```

Nos mostrará en consola todos los branch del proyecto. Como solo tenemos una rama, nos aparecerá `main`.

¿Cuándo generar una nueva rama?

Imaginemos que estamos trabajando en un equipo y necesitamos que un compañero haga cambios en el código. Posiblemente necesite modificar muchas cosas para su nueva funcionalidad.

Si solo utilizamos una rama, estos cambios quedarán sobre lo que ya estaba funcionando, lo que hará difícil saber qué es lo nuevo y solventar algún posible problema.

Para estos casos será mejor segmentar los flujos de trabajo con ramas: una para la aplicación ya funcional en la rama main y una nueva para las otras funcionalidades.

Para crear una nueva branch utilizaremos el siguiente comando:

```
git branch nueva_branch
```

Una vez creado, no cambiará de forma automática a la nueva rama, sino que nos quedaremos en la rama original. Para cambiarnos debemos hacerlo con el comando:

```
git checkout nueva_branch
```

De esta forma nos cambiamos a la nueva branch y podemos utilizar todos los comandos de git ya conocidos (git add, commit, etc.).

Así es cómo podemos lograr registrar los cambios de la nueva funcionalidad, sin afectar los de la rama main.

Si queremos crear una branch y situarnos enseguida en ella debemos ejecutar el siguiente comando:

```
git checkout -b otra_branch
```

Veamos con el siguiente comando las ramas que hemos creado:

```
git branch
```

Git Stash

Existe una manera de reservar o "apartar" las modificaciones que estamos haciendo en nuestra rama actual para realizar una tarea emergente y asegurarnos que no perderemos los cambios que hemos realizado, para esto podemos optar por ocupar el comando:

```
git stash
```

Al ejecutarlo, veremos que todos nuestros cambios han desaparecido, sin embargo, lo que sucedió es que fueron diferidos a un área en paralelo de nuestro historial de commits al cual podemos acceder con el siguiente comando:

```
git stash list
```

Como verás, al ejecutarlo se muestra un listado que incluye los stash que hayamos hecho en nuestra rama en conjunto con un hash generado que podremos ocupar junto al siguiente comando:

```
git stash apply
```

Para volver a unir el directorio actual con los cambios que estábamos trabajando y que fueron apartados del commit en el que nos encontrábamos.

Este comando es muy útil para desarrollar sin temor a perder nuestras modificaciones, no obstante se recomienda usarlo en casos puntuales donde necesitemos de forma urgente solucionar problemas que fueron detectados mientras estamos trabajando en una nueva versión de nuestro proyecto.

Git Rebase

Cuando estamos trabajando en un equipo de desarrollo y varios usuarios manipulan el mismo repositorio en paralelo, es normal que se generen muchas ramas y muchos commits que dificultan la lectura del historial general, puesto que habrán modificaciones agrupadas en una rama A que serán unidas a una rama B a partir del último cambio generado.

Para lograr un historial más limpio con menos ramas y menos commits distribuidos, git nos ofrece el siguiente comando:

```
git rebase otra_branch
```

Al ser ejecutado, se posicionarán todos los commits de la rama actual en la punta de la rama que mencionemos en el rebase, de esta manera se estará guardando un historial lineal y mucho más cómodo de leer.

La alternativa de este comando es el popular merge que podemos también utilizar para unir dos ramas, no obstante al utilizar git merge se generan commits que representan el punto de unión y para muchos desarrolladores esto es innecesario y contraproducente.

GitHub Pages

¿Qué es GitHub pages?

Ahora que ya sabemos un poco cómo utilizar git y GitHub, vamos a conocer una utilidad para mostrar nuestros sitios construidos con HTML, CSS y JavaScript.

Se llama GitHub pages y renderiza nuestro código de una rama específica en un dominio y espacio dentro de GitHub.

Esto será de utilidad para poder mostrar nuestro trabajo de forma gratis, sin tener que recurrir a dominios o servidores externos.

Subiendo una página a GitHub pages

Podemos utilizar un proyecto simple o el mismo que venimos utilizando. Lo importante es que tengamos un repositorio en GitHub ya creado y, por supuesto, hayamos subido los cambios al repositorio remoto.

Hay dos formas de realizar la subida a GitHub pages. La primera es manual a través de la consola:

Crearemos la rama:

```
git branch gh-pages
```

Recuerda que el nombre debe ser "gh-pages".

Ahora, pasaremos todos los commits de la rama main a la nueva rama, para luego subir los cambios al repositorio remoto.

```
git checkout gh-pages
```

```
git merge main
```

```
git push origin gh-pages
```

Finalmente, visitamos la página (reemplaza con tu nombre de usuario y nombre de tu repositorio donde corresponde).

```
https://TuNombreDeUsuario.GitHub.io/Nombre_de_tu_repositorio/
```

Podremos ver la página almacenada en GitHub pages, que es completamente online. Se la puedes mostrar a quien quieras.

La segunda forma de hacer un despliegue de sitios web en este servicio es a través de la interfaz gráfica de Github, entrando al repositorio encontraremos un botón desplegable para agregar archivos a una rama y posteriormente arrastrar los mismos desde de el explorador del sistema operativo, así como se muestra en las siguientes imágenes.

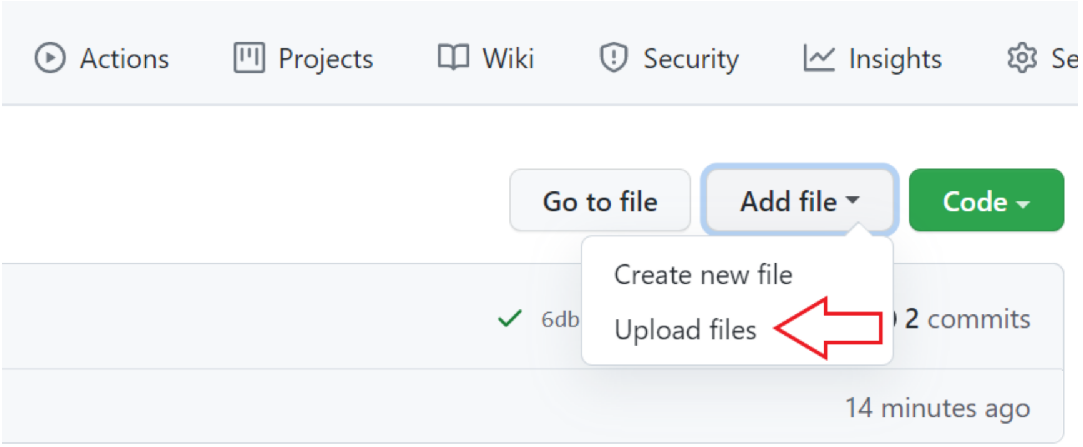

Imagen 1. Subida de archivos desde Github.com.
Fuente: Desafío Latam

Resumen

- En esta unidad hemos aprendido a trabajar con la terminal y sus comandos principales. Además, analizamos la importancia de versionar nuestro código y utilizar recursos tanto locales como remotos para gestionar proyectos.

- Estas herramientas son clave para trabajar desarrollando aplicaciones, pues nos permiten resguardar nuestro progreso, hacer pruebas de concepto sin impactar el código principal y lo que es más relevante aún, trabajar en equipos con responsabilidades diversas.