Guía de Git

Esta guía está diseñada para que aprendas los fundamentos de Git y GitHub a través de la práctica. Sigue los pasos en orden para crear un proyecto, controlar sus versiones y entender el flujo de trabajo colaborativo.

Parte 1: Conceptos Fundamentales

- **Control de Versiones:** Es un sistema que registra los cambios realizados en un archivo o conjunto de archivos a lo largo del tiempo. Esto te permite revertir a versiones específicas, comparar cambios y ver quién modificó qué. Reemplaza la necesidad de tener archivos como `proyecto_v1.doc`, `proyecto_final.doc`, etc.
- **Git:** Es el software de control de versiones distribuido. Es una herramienta que se instala y se ejecuta en tu computadora para gestionar el historial de cambios de tu proyecto. Al ser "distribuido", cada persona que trabaja en el proyecto tiene una copia completa del historial en su máquina local.
- **GitHub:** Es una plataforma web que aloja repositorios de Git en la nube. Funciona como un servidor central para tu código, facilitando la colaboración, la copia de seguridad y la visibilidad de tus proyectos. Git es la herramienta, GitHub es el servicio.

Parte 2: Configuración Inicial

2.1. Instalación de Git

Primero, verifica si ya tienes Git instalado abriendo una terminal (o Git Bash en Windows) y ejecutando:

```
git --version
```

Si recibes una respuesta con un número de versión, puedes saltar al siguiente paso. De lo contrario, instálalo desde git-scm.com. Acepta las opciones de instalación por defecto.

2.2. Configuración de Identidad

Git necesita saber quién eres para registrar tus cambios. Configura tu nombre de usuario y correo electrónico. **Es importante que el correo electrónico sea el mismo que usarás para tu cuenta de GitHub.**

```
git config --global user.name "Tu usuario"
git config --global user.email "tu_correo@ejemplo.com"
```

El indicador `--global` establece esta configuración para todos los repositorios en tu sistema.

2.3. Creación de Cuenta en GitHub

Ve a github.com y regístrate para obtener una cuenta gratuita. Este será el lugar donde alojarás tus proyectos de forma remota.

Parte 3: El Flujo de Trabajo Básico de un Proyecto

En esta sección, crearemos un repositorio, lo conectaremos a tu máquina local y guardaremos tus primeros cambios.

3.1. Creación de un Repositorio Remoto en GitHub

1. Inicia sesión en GitHub. En la esquina superior derecha, haz clic en el ícono `+` y selecciona **New repository**.
2. Asigna un nombre al repositorio, por ejemplo, `mi-proyecto-git` .
3. Asegúrate de que esté configurado como **Public**.
4. **Importante:** Marca la casilla que dice **Add a README file**. Esto inicializa el repositorio con un archivo, lo cual simplifica el proceso de clonación.
5. Haz clic en **Create repository**.

3.2. Clonar el Repositorio (`git clone`)

Clonar es el proceso de descargar una copia de un repositorio remoto a tu máquina local.

1. En la página de tu nuevo repositorio en GitHub, haz clic en el botón verde **< > Code**.
2. Copia la URL HTTPS que aparece.
3. Abre tu terminal, navega al directorio donde deseas guardar tu proyecto (ej. `cd Documents/Projects`) y ejecuta el siguiente comando:

```
git clone URL_QUE_COPIASTE
```

Esto creará una carpeta llamada `mi-proyecto-git` en tu directorio actual. Ingresa a esa carpeta:

```
cd mi-proyecto-git
```

3.3. Guardar Cambios Localmente (`git add` y `git commit`)

Este es el ciclo fundamental de trabajo en Git.

1. **Realiza un cambio.** Crea un nuevo archivo o realiza cambios en uno existente.
2. **Verifica el estado.** Pregúntale a Git qué ha cambiado en el proyecto:

```
git status
```

Git te informará que hay un archivo "Untracked" (no rastreado).
3. **Prepara los cambios (`git add`).** Debes indicarle a Git qué cambios específicos quieres incluir en tu próximo guardado. Este paso se conoce como "staging" o poner en el área de preparación.

```
git add <nombre_del_archivo>
```

Si deseas agregar todos los archivos modificados, puedes usar `git add .` .
4. **Confirma los cambios (`git commit`).** Ahora, guarda los cambios preparados en el historial local de Git. Cada `commit` es una "instantánea" o un punto de guardado en la historia de tu proyecto. Es obligatorio incluir un mensaje descriptivo.

```
git commit -m "Mensaje descriptivo del cambio"
```

3.4. Sincronizar Cambios con GitHub (`git push`)

Hasta ahora, tus `commits` solo existen en tu máquina local. Para subirlos a GitHub, utiliza el comando `push` .

```
git push origin main
```

- **`push`** : Es el comando para enviar commits.
- **`origin`** : Es el nombre por defecto del repositorio remoto (el de GitHub desde donde clonaste).
- **`main`** : Es el nombre de la rama principal que estás actualizando.

Ahora, si refrescas la página de tu repositorio en GitHub, verás el archivo o los archivos que has subido.

3.5. Obtener Cambios del Repositorio Remoto (`git pull`)

Si otra persona (o tú mismo desde la interfaz de GitHub) realiza cambios en el repositorio remoto, necesitas traer esos cambios a tu copia local.

1. Ve a tu repositorio en GitHub, abre el archivo `README.md` y haz clic en el ícono de lápiz para editarlo. Añade una línea de texto y guarda los cambios.
2. De vuelta en tu terminal, ejecuta el comando `pull` para descargar y fusionar los cambios remotos:

```
git pull origin main
```

Ahora, si abres el archivo `README.md` en tu computadora, verás los cambios que hiciste en la web.

Parte 4: Ramas y Fusiones (El Flujo de Trabajo Colaborativo)

Las ramas (branches) son la característica más poderosa de Git. Permiten trabajar en nuevas funcionalidades o solucionar errores en un entorno aislado sin afectar la línea principal de desarrollo (la rama `main`).

4.1. Creación de una Nueva Rama (`git checkout`)

Imagina que quieres añadir una nueva sección a tu página web. En lugar de trabajar directamente en `main`, creas una nueva rama.

```
git checkout -b <nombre-de-la-rama>
```

- **`checkout -b`** : Es un atajo que crea (`-b`) una nueva rama y se cambia (`checkout`) a ella en un solo paso.

Puedes ver todas las ramas de tu proyecto con `git branch`. El asterisco (`*`) indicará en qué rama te encuentras.

4.2. Trabajar en la Rama

Ahora que estás en la rama `<nombre-de-la-rama>`, todos tus cambios estarán aislados aquí.

1. Modifica el archivo.
2. Usa el flujo de trabajo `add` y `commit` para guardar tus cambios en esta rama:

```
git add <archivo_modificado>
git commit -m "<Mensaje describiendo los cambios>"
```

Estos cambios solo existen en la rama `<nombre-de-la-rama>` y no son visibles en la rama `main`.

4.3. Fusionar la Rama (`git merge`)

Una vez que has terminado el trabajo en tu rama y todo funciona correctamente, el siguiente paso es integrar (fusionar) esos cambios de vuelta a la rama principal.

1. Primero, regresa a la rama `main` :

```
git checkout main
```

2. Ahora, ejecuta el comando `merge` para traer los cambios desde la otra rama:

```
git merge <nombre-de-la-rama>
```

Git aplicará los commits de la rama de contacto a la rama `main` . Si abres `<archivo_modificado>` , verás que ahora contiene los cambios realizados en la rama `<nombre-de-la-rama>` .

4.4. Limpieza y Sincronización

- Una vez que una rama ha sido fusionada, generalmente se elimina para mantener el repositorio limpio:

```
git branch -d <nombre-de-la-rama>
```

- Finalmente, no olvides subir los cambios fusionados a GitHub:

```
git push origin main
```

Parte 5: Archivos `.gitignore`

A veces, hay archivos o carpetas que no deseas que Git rastree, como archivos temporales, configuraciones locales o dependencias. Para evitar que estos archivos se incluyan en los commits, puedes crear un archivo llamado `.gitignore` en la raíz de tu repositorio.
Dentro de este archivo, puedes listar los nombres o patrones de los archivos y carpetas que deseas ignorar. Por ejemplo:

```
*.log
node_modules/
dist/
.DS_Store
```

Resumen del Flujo de Trabajo

1. `git pull` : Siempre empieza actualizando tu repositorio local.
2. `git checkout -b nombre-rama` : Crea una rama para trabajar en una nueva tarea.
3. **Trabaja:** Haz tus cambios, crea archivos, etc.
4. `git add .` : Prepara todos tus cambios.
5. `git commit -m "Mensaje"` : Guarda los cambios en la rama.
6. `git checkout main` : Regresa a la rama principal.
7. `git merge nombre-rama` : Integra tus cambios.
8. `git push origin main` : Sube el resultado a GitHub.