Guía de Git Cómo trabajar en equipo en proyectos

Aprende a usar Git para controlar versiones, colaborar con otros desarrolladores y mantener tu código organizado.

Edison Achalma

2023-02-16

Introducción

¿Estás interesado en aprender los fundamentos de Git y GitHub? ¡Has llegado al lugar perfecto! En este blog, te presentaremos una guía completa que podrás utilizar como referencia diaria.

Tanto si estás comenzando tu viaje en el control de versiones como si deseas mejorar tus habilidades en Git y aprovechar al máximo GitHub, encontrarás aquí una guía clara y concisa para dar tus primeros pasos.

Los sistemas de control de versiones, como Git, son imprescindibles en las prácticas. Estos sistemas te permiten realizar un seguimiento de los cambios en tu código fuente, revertir a versiones anteriores y crear ramas para experimentar con nuevas ideas o funcionalidades.

Hoy en día, los repositorios de Git albergan muchos proyectos de software, y plataformas como GitHub, GitLab y Bitbucket facilitan la colaboración y el intercambio de código entre desarrolladores.

En esta guía, te mostraremos cómo instalar y configurar Git en GNU Linux (Ubuntu). Exploraremos dos métodos diferentes de instalación, cada uno con sus propios beneficios, para que puedas elegir el que mejor se adapte a tus necesidades específicas.

> Recuerda que es esencial tener un buen dominio de Git y GitHub para colaborar eficientemente en proyectos y aprovechar todas sus funcionalidades. ¡Vamos a sumergirnos en el fascinante mundo de Git y GitHub!

Entendiendo cómo funciona Git

Git es el sistema de control de versiones (SCV) de código abierto más utilizado, diseñado para rastrear los cambios realizados en los archivos. Tanto empresas como programadores confían en Git para colaborar en el desarrollo de software y aplicaciones.

Un proyecto en Git se compone de tres elementos principales: **el directorio de trabajo, el área de preparación y el directorio Git.**

El directorio de trabajo es donde agregas, borras y editas tus archivos. Luego, los cambios se preparan (indexan) en el área de preparación. Una vez que confirmas tus cambios, se guarda una instantánea de los mismos en el directorio Git.

Git está disponible para múltiples plataformas, incluyendo Linux, Windows y Mac. Esto hace que Git sea accesible para todos. Aunque el software puede tener una curva de aprendizaje pronunciada, existen numerosos tutoriales disponibles para ayudarte a dominarlo.

Así que, independientemente de tu sistema operativo, puedes aprovechar los beneficios de Git y aprovechar su poderoso conjunto de características. Con el tiempo y la práctica, te convertirás en un experto en el uso de Git para gestionar tus proyectos de manera eficiente y colaborar de forma efectiva con otros desarrolladores.

Comandos básicos de Git

Aquí te presentamos algunos comandos básicos de Git que es importante que conozcas:

1. `git init`: Inicializa un nuevo repositorio de Git en la carpeta actual.
2. `git clone [url]`: Clona un repositorio existente en la carpeta actual.
3. `git add [file]`: Agrega un archivo al área de preparación (stage) para ser incluido en el próximo commit.
4. `git commit -m "[message]"`: Realiza un commit (guarda un punto de referencia) con un mensaje que describe los cambios realizados.
5. `git status`: Muestra el estado actual del repositorio, incluyendo los archivos modificados y los que están pendientes de commit.
6. `git log`: Muestra un historial de todos los commits realizados en el repositorio.
7. `git diff`: Muestra las diferencias entre los cambios realizados y el último commit.
8. `git branch`: Muestra una lista de todas las ramas existentes en el repositorio.
9. `git checkout [branch]`: Cambia a una rama específica.
10. `git merge [branch]`: Combina los cambios de una rama específica con la rama actual.
11. `git config --global user.email "tu-email@example.com"`
12. `git config --global user.name "tu-usuario-GitHub"`

Estos son solo algunos de los comandos básicos de Git. También existen muchos otros comandos avanzados disponibles para realizar tareas más complejas, como trabajar con ramas remotas y manejar conflictos.

Instalación y Configuración de Git

Git es una herramienta fundamental para el control de versiones y colaboración en proyectos de desarrollo de software. Aquí te explicaremos cómo instalar y configurar Git en Ubuntu.

Instalación de Git con paquetes predeterminados

Si deseas una instalación rápida y estable de Git, puedes utilizar los paquetes predeterminados. Si buscas la versión más reciente o necesitas funciones específicas, puedes optar por la instalación desde la fuente.

En primer lugar, verifica si Git ya está instalado en tu Ubuntu ejecutando el siguiente comando en la terminal:

```
git --version
```

Si obtienes una salida similar a la siguiente, significa que Git ya está instalado:

```
git version 2.34.1
```

Si Git no está instalado, puedes utilizar el administrador de paquetes APT de Ubuntu para instalarlo:

1. Abre una terminal.
2. Actualiza la lista de paquetes disponibles con el siguiente comando:

   ```
   sudo apt update
   ```

3. Instala Git utilizando el siguiente comando:

   ```
   sudo apt install git
   ```

4. Verifica la instalación ejecutando el comando

   ```
   git --version
   ```

Una vez que Git esté instalado, es recomendable configurarlo según tus necesidades. Sigue estos pasos para configurar Git en Ubuntu:

1. Establece tu nombre de usuario en Git con el comando:

```
git config --global user.name "Tu Nombre"
```

2. Establece tu dirección de correo electrónico en Git con el comando:

```
git config --global user.email "tu.correo@example.com"
```

3. Para verificar la configuración actual de Git, utiliza el comando:

```
git config --list
```

También es recomendable configurar un editor de texto para escribir los mensajes de commit. Puedes hacerlo con el siguiente comando, reemplazando **"nano"** o **"vim"** por el editor de texto de tu preferencia:

```
git config --global core.editor "nano"
```

Con estos pasos, habrás instalado y configurado Git en tu sistema Ubuntu. Ahora estás listo para aprovechar todas las ventajas que ofrece esta poderosa herramienta de control de versiones en tus proyectos de desarrollo.

Instalación de Git desde la fuente

Si estás buscando una forma más flexible de instalar Git y quieres tener la versión más reciente, puedes compilar el software desde la fuente. Aunque este método requiere más tiempo y no se integrará con el administrador de paquetes, te permitirá personalizar las opciones de instalación.

Antes de comenzar, verifica la versión actualmente instalada de Git ejecutando el siguiente comando: `git --version`. Si Git ya está instalado, obtendrás un resultado similar a este: `git version 2.34.1`.

Asegúrate de tener instalado el software necesario para compilar Git. Puedes hacerlo actualizando el índice de paquetes locales y luego instalando las dependencias relevantes. Ejecuta los siguientes comandos:

```
sudo apt update
sudo apt install libz-dev libssl-dev libcurl4-gnutls-dev libexpat1-dev gettext cmake gcc
```

Una vez instaladas las dependencias, crea un directorio temporal y accede a él. Aquí es donde descargarás el archivo tarball de Git. Ejecuta los siguientes comandos:

```
mkdir tmp
cd /tmp
```

Desde el sitio web oficial de Git, navega hasta la lista de tarballs disponibles en `https://mirrors.edge.kernel.org/pub/sc` y descarga la versión que desees utilizar. Por ejemplo, si quieres descargar la versión 2.34.1, puedes ejecutar el siguiente comando:

```
curl -o git.tar.gz https://mirrors.edge.kernel.org/pub/software/scm/git/git-2.34.1.tar.gz
```

Descomprime el archivo tarball ejecutando el siguiente comando:

```
tar -zxf git.tar.gz
```

A continuación, accede al nuevo directorio de Git con el siguiente comando:

```
cd git-*
```

Ahora, puedes crear el paquete e instalarlo ejecutando los siguientes comandos:

```
make prefix=/usr/local all
sudo make prefix=/usr/local install
```

Una vez completado el proceso, sustituye la shell actual para utilizar la versión de Git recién instalada ejecutando el siguiente comando:

```
exec bash
```

Para verificar que la instalación se haya realizado correctamente, comprueba la versión de Git nuevamente ejecutando el comando `git --version`. Deberías obtener un resultado similar a este: `git version 2.34.1`.

> ¡Con Git instalado correctamente, ahora puedes continuar con la configuración y aprovechar todas las funcionalidades que ofrece esta poderosa herramienta de control de versiones en tus proyectos!

Configuración de Git

Una vez que hayas elegido la versión de Git con la que estás satisfecho, es importante configurar Git para que los mensajes de confirmación que generes contengan la información correcta y te respalden a medida que desarrollas tu proyecto de software.

La configuración de Git se realiza a través del comando `git config`. Específicamente, debemos proporcionar nuestro nombre y dirección de correo electrónico, ya que Git inserta esta información en cada confirmación que realizamos. Podemos agregar esta información ejecutando los siguientes comandos:

```
git config --global user.name "Your Name"
git config --global user.email "your.email@example.com"
```

Para verificar los elementos de configuración que hemos creado, podemos ejecutar el siguiente comando:

```
git config --list
```

La información que ingreses se almacenará en el archivo de configuración de Git. Si deseas editarlo manualmente con el editor de texto de tu elección (en este tutorial utilizaremos nano), puedes ejecutar el siguiente comando:

```
nano ~/.gitconfig
```

En el archivo `~/.gitconfig`, encontrarás los siguientes contenidos:

```
[user]
    name = Your Name
    email = your.email@example.com
```

Para salir del editor de texto, presiona CTRL + X, luego Y y finalmente ENTER.

Existen muchas otras opciones de configuración que puedes ajustar, pero estas dos son esenciales. Si omites este paso, es probable que veas mensajes de advertencia al realizar una confirmación con Git. Esto implica más trabajo para ti, ya que tendrás que revisar las confirmaciones anteriores y corregir la información.

Al configurar Git correctamente, aseguras que tus confirmaciones tengan la información adecuada y facilitas el seguimiento y control de los cambios en tu proyecto de software.

Cómo Obtener y Configurar tus Claves SSH para Git y GitHub

Si estás utilizando GitHub sin configurar una clave SSH, te estás perdiendo de una gran comodidad. Piensa en todo el tiempo que has gastado ingresando tu correo electrónico y contraseña en la consola cada vez que haces un commit, podrías haberlo utilizado para programar.

Pero ya no más. Aquí tienes una guía rápida para generar y configurar una clave SSH con GitHub, para que nunca más tengas que autenticarte de forma convencional.

Verificar la existencia de una clave SSH

En primer lugar, verifica si ya has generado claves SSH para tu máquina. Abre una terminal y ejecuta el siguiente comando:

```
ls -al ~/.ssh
```

Si ya has generado las claves SSH, deberías ver una salida similar a esta:

```
-rw-------  1 usuario usuario  1766 Jul  7  2018 id_rsa
-rw-r--r--  1 usuario usuario   414 Jul  7  2018 id_rsa.pub
-rw-------  1 usuario usuario 12892 Feb  5 18:39 known_hosts
```

Si tus claves ya existen, continúa con la sección **Copia tu clave pública de SSH** a continuación.

Si no ves ninguna salida o si el directorio no existe (recibes un mensaje de "No such file or directory"), entonces ejecuta el siguiente comando:

```
mkdir $HOME/.ssh
```

A continuación, genera un nuevo par de claves con el siguiente comando:

```
ssh-keygen -t rsa -b 4096 -C achalma_pinguino@gmail.com
```

Presiona Enter para crear el archivo con el nombre predeterminado y luego ingresa una clave para proteger el archivo (por ejemplo, "pepito69").

Ahora verifica que tus claves existan con el comando `ls -al ~/.ssh` y asegúrate de que la salida sea similar a la mencionada anteriormente.

Nota: Las claves SSH siempre se generan como un par de claves públicas (id_rsa.pub) y privadas (id_rsa). Es extremadamente importante que **nunca reveles tu clave privada** y que **solo uses tu clave pública** para autenticarte en servicios como GitHub.

Agrega tu clave SSH a ssh-agent

ssh-agent es un programa que se inicia cuando te conectas y almacena tus claves privadas. Para que funcione correctamente, debe estar en ejecución y tener una copia de tu clave privada.

Primero, asegúrate de que **ssh-agent** se está ejecutando ejecutando el siguiente comando:

```
eval "$(ssh-agent -s)" # para Mac y Linux
```

o:

```
eval ssh-agent -s
```

```
ssh-agent -s # para Windows
```

A continuación, agrega tu clave privada a **ssh-agent** con el siguiente comando:

```
ssh-add ~/.ssh/id_rsa
```

Copia tu clave pública de SSH

Después, necesitarás copiar tu clave pública de SSH al portapapeles.

Para Linux o Mac, puedes imprimir el contenido de tu clave pública en la consola con el siguiente comando:

```
cat ~/.ssh/id_rsa.pub # Linux
```

Aparecerá una cadena de números y letras. Si al final está el correo que registraste antes, debes borrarlo antes de pegarlo en GitHub.

Luego, resalta y copia el resultado.

Para Windows, simplemente ejecuta el siguiente comando:

```
clip < ~/.ssh/id_rsa.pub # Windows
```

Agrega tu clave SSH pública a GitHub

Accede a la página de **configuración** de tu cuenta de GitHub aquí y haz clic en el botón "New SSH key".

A continuación, asigna un título descriptivo a tu clave y pégala en el campo correspondiente a tu clave pública (id_rsa.pub).

Por último, prueba la autenticación con el siguiente comando:

```
ssh -T git@github.com
```

Si has seguido correctamente todos estos pasos, deberías ver el siguiente mensaje:

```
Hi tu_usuario! You've successfully authenticated, but GitHub does not provide shell access.
```

o

```
Warning: Permanently added the ECDSA host key for IP address '140.82.114.3' to the list of known h
Hi achalmed! You've successfully authenticated, but GitHub does not provide shell access.
```

Para obtener más información sobre SSH, puedes consultar la documentación.

CLI de GitHub

En este tutorial, te mostraré cómo instalar y configurar la CLI de GitHub en Linux para facilitar tus operaciones en GitHub directamente desde la terminal.

Paso 1: Instalación de la CLI de GitHub en Linux

Para comenzar, debemos instalar la CLI de GitHub en Linux. Ejecuta los siguientes comandos en tu terminal:

```
sudo apt install gh # versión 2.12.1+dfsg1-1
```

Paso 2: Autenticación con GitHub

Una vez que hayas instalado la CLI de GitHub, puedes autenticarte con tu cuenta de GitHub. Ejecuta el siguiente comando:

```
gh auth login
```

Aparecerán varias opciones. A continuación, elige las siguientes opciones:

- ¿En qué cuenta quieres iniciar sesión? GitHub.com
- ¿Cuál es tu protocolo preferido para las operaciones de Git? SSH
- ¿Deseas generar una nueva clave SSH para agregar a tu cuenta de GitHub? Sí
- Ingresa una frase de contraseña para tu nueva clave SSH (opcional) ****** (usa tu propia frase de contraseña)
- Título para tu clave SSH: achalmagit
- ¿Cómo te gustaría autenticar la CLI de GitHub? Iniciar sesión con un navegador web

Luego, copia el código proporcionado y pégalo en tu navegador para completar la conexión con GitHub.

> ¡Listo! Ahora estás autenticado con éxito y puedes usar la CLI de GitHub para trabajar con tus repositorios en la terminal.

Paso 3: Utilizar la CLI de GitHub

Una vez que hayas completado la autenticación, puedes comenzar a utilizar la CLI de GitHub en la terminal. Simplemente ingresa el código de contraseña (******) cuando se te solicite para realizar operaciones en tus repositorios de GitHub.

Para obtener más información sobre la CLI de GitHub y todas sus capacidades, puedes consultar la documentación oficial.

> ¡Disfruta utilizando la CLI de GitHub para simplificar tus interacciones con GitHub desde la terminal!

Crear un Repositorio

En este tutorial, te mostraré cómo crear un nuevo repositorio GIT local utilizando el comando `git init`. Este proceso te permitirá iniciar un repositorio para tu proyecto y realizar un seguimiento de los cambios a lo largo del tiempo.

Paso 1: Inicializar un nuevo repositorio

Para iniciar un nuevo repositorio GIT, simplemente ejecuta el siguiente comando en tu terminal:

```
git init
```

Esto creará un nuevo repositorio GIT vacío en tu directorio actual.

Si deseas especificar un nombre para tu proyecto al crear el repositorio, puedes utilizar el siguiente comando:

```
git init [nombre del proyecto]
```

> Recuerda que este paso solo se realiza una vez al inicio del proyecto.

Comandos en Visual Studio Code

En Visual Studio Code, puedes interactuar con GIT directamente desde la interfaz de usuario. Aquí hay algunos comandos clave que puedes utilizar:

1. Abre el terminal integrado en Visual Studio Code desde la carpeta de tu proyecto.
2. Ejecuta `git init` para inicializar el repositorio.
3. Utiliza `git add` seguido del nombre de archivo para agregar archivos individuales al repositorio `git add index.html`.
4. Si deseas agregar todos los archivos modificados y no rastreados, puedes ejecutar `git add .`.

5. Haz un commit de los cambios utilizando el comando `git commit -m "mensaje del commit"`.
6. Si aún no tienes una rama principal (main), puedes crearla con el comando `git branch -M main`.
7. Para vincular tu repositorio local a un repositorio remoto en GitHub, utiliza el comando `git remote add origin [URL del repositorio]`.
8. Finalmente, puedes enviar tus cambios al repositorio remoto utilizando `git push -u origin main`. Asegúrate de ingresar tu nombre de usuario y contraseña correctamente.

También puedes importar código desde otro repositorio o iniciar un repositorio con código de proyectos Subversion, Mercurial o TFS.

Recuerda que es importante configurar GIT previamente y configurar la autenticación utilizando claves SSH o GitHub CLI, según tus preferencias.

¡Ahora estás listo para crear y gestionar repositorios con GIT en Visual Studio Code!

Clonar un Repositorio con Git

El comando `git clone` se utiliza para copiar un repositorio, ya sea desde un servidor remoto o desde una ubicación local. Si el repositorio está en un servidor remoto, se utiliza la siguiente sintaxis: `git clone nombredeusuario@host:/ruta/al/repositorio`. En cambio, si el repositorio se encuentra en una ubicación local, se utiliza: `git clone /ruta/al/repositorio`.

¿Qué es git clone?

El comando `git clone` se utiliza para apuntar a un repositorio existente y hacer una copia del mismo en otra ubicación. Este comando creará un nuevo directorio, lo configurará para utilizar Git y copiará los archivos del repositorio en él. Sin clonar un repositorio Git, no podrás realizar cambios en él ni contribuir con tu trabajo.

Cómo Clonar un Repositorio Git

Git es un sistema de control de versiones ampliamente utilizado en el mundo empresarial. Antes de poder comenzar a trabajar en un repositorio, es necesario clonarlo en tu computadora local. En este artículo, te mostraré cómo clonar un repositorio Git en Ubuntu. Estos pasos son aplicables para clonar repositorios desde plataformas populares como GitHub, Bitbucket, GitLab y otras basadas en Git.

Clonar un Repositorio Remoto

Supongamos que deseas clonar un repositorio remoto desde GitHub, Bitbucket u otra plataforma en la nube hacia tu máquina local.

1. Abre la terminal y navega hasta la ubicación donde deseas que se copie el repositorio, por ejemplo:

```
cd /home/ubuntu/
```

2. Cada repositorio remoto de Git tiene una URL única. Inicia sesión en tu plataforma de desarrollo preferida, como GitHub, y copia la URL de tu repositorio.
3. Utiliza el siguiente comando `git clone` seguido de la URL del repositorio para clonarlo en tu máquina local. Por ejemplo:

```
sudo git clone https://github.com/usuario/repositorio
```

Asegúrate de reemplazar “usuario” con tu nombre de usuario de GitHub y “repositorio” con el nombre de tu repositorio.

4. Se te pedirá la contraseña para la autenticación, después de lo cual Git descargará automáticamente una copia de tu repositorio en el directorio de trabajo actual.

Clonar en una Carpeta Específica

Si deseas clonar el repositorio en una carpeta específica, puedes utilizar el siguiente comando:

```
sudo git clone <repositorio> <directorio>
```

Por ejemplo, supongamos que deseas clonar tu repositorio en la carpeta `/home/desarrollador`:

```
sudo git clone https://github.com/usuario/repositorio /home/desarrollador
```

> Ahora has aprendido cómo clonar un repositorio Git en tu máquina local utilizando el comando `git clone`. ¡Comienza a trabajar en tu repositorio clonado y realiza cambios en él!

Clonar un Repositorio Superficialmente

Si necesitas clonar un repositorio grande con un historial extenso de confirmaciones, el proceso puede llevar mucho tiempo. Sin embargo, en esos casos, existe la opción de realizar un clon superficial en el cual puedes especificar las últimas "n" confirmaciones que deseas clonar. Esto resultará en un proceso mucho más rápido y ocupará menos espacio en tu sistema.

Aquí tienes la sintaxis para realizar un clon superficial, donde "n" representa el número de confirmaciones más recientes que deseas clonar:

```
sudo git clone --depth=n <repo>
```

Por ejemplo, si deseas clonar solamente la última confirmación de tu repositorio, puedes utilizar el siguiente comando:

```
sudo git clone --depth=1 https://github.com/test_user/demo.git
```

De manera similar, si deseas clonar las últimas 10 confirmaciones de tu repositorio, puedes ejecutar este comando:

```
sudo git clone --depth=10 https://github.com/test_user/demo.git
```

Clonar una Rama Específica de Git

Si únicamente deseas clonar una rama específica (por ejemplo, "working") en lugar de todo el repositorio, puedes utilizar la opción `-branch` en el comando `git clone`:

```
git clone --branch=working https://github.com/test_user/demo.git
```

> ¡Y eso es todo! Como puedes ver, clonar un repositorio Git en Ubuntu es bastante sencillo. Ahora puedes utilizar estas técnicas para realizar clones superficiales o clonar ramas específicas según tus necesidades.

Utilizar Git y GitHub para subir proyectos

Para comenzar a gestionar tus proyectos utilizando Git y GitHub, es necesario realizar algunos pasos. En este artículo, te mostraremos cómo subir un proyecto desde tu computadora a GitHub y también te enseñaremos a utilizar los principales comandos a través de la terminal de Linux.

Crear un nuevo repositorio

En primer lugar, debes crear un nuevo repositorio en la página web de GitHub. Dirígete a la página "New Project" y proporciona un nombre para tu repositorio, una descripción y selecciona si deseas que el repositorio sea público o privado (recuerda que los repositorios privados requieren una suscripción). Una vez que hayas completado todos los campos, haz clic en "Create Repository" y tu repositorio estará listo.

Subir un proyecto

Para subir un proyecto, primero debes crear un proyecto Git localmente en tu computadora. Para ello, ve al directorio raíz de tu proyecto a través de la terminal utilizando el siguiente comando:

```
cd ruta/al/archivo
```

Una vez que estés en el directorio del proyecto, inicializa Git ejecutando el siguiente comando:

```
git init
```

Verás una confirmación de que el proyecto Git ha sido creado.

Ahora debes agregar los archivos al repositorio Git. Puedes hacerlo utilizando una de las siguientes opciones, según si deseas agregar un archivo específico o todos los archivos existentes en el proyecto:

```
git add .  # Añade todos los archivos existentes en la carpeta al proyecto Git
git add nombredelarchivo.extension  # Añade únicamente el archivo especificado al proyecto
```

Es importante que identifiques todos los cambios que realices en tu proyecto con un comentario. Puedes hacerlo utilizando el siguiente comando:

```
git commit -m 'comentario'
```

Ahora estás listo para subir el proyecto a GitHub. Para hacerlo, debes agregar el repositorio remoto ejecutando el siguiente comando en la terminal:

```
git remote add origin git@github.com:achalmed/achalmaedison.web.git
```

Asegúrate de reemplazar “achalmed” con tu nombre de usuario en GitHub y “achalmaedison.web.git” con el nombre de tu repositorio previamente creado. Una vez que hayas ingresado los datos correctos, presiona Enter y podrás subir el proyecto ejecutando el siguiente comando:

```
git push -u origin master
```

Ahora tu proyecto estará cargado en tu cuenta de GitHub. Puedes verificarlo accediendo a la página del proyecto.

En caso de que encuentres un error llamado “fatal: remote origin already exists” durante el paso anterior, debes ejecutar el siguiente comando:

```
git remote rm origin
```

Luego, repite el proceso desde el principio para agregar el repositorio remoto correctamente.

> Con estos pasos, has aprendido cómo utilizar Git y GitHub para subir tus proyectos y comenzar a gestionarlos desde allí. ¡Aprovecha al máximo estas herramientas para colaborar y controlar las versiones de tus proyectos de manera efectiva!

Observa tu Repositorio

git status

El comando `git status` muestra una lista de los archivos que han sido modificados junto con los archivos que están pendientes de ser preparados o confirmados.

Para listar los archivos no sincronizados, puedes utilizar `git status -s`.

Verifica el estado actual del repositorio. Lista los archivos nuevos o modificados que aún no han sido confirmados: `git status`

git diff

- `git diff`: Muestra los cambios en archivos que aún no han sido preparados. Se utiliza para hacer una lista de todos los conflictos.
- `git diff --base <nombre-archivo>`: Permite ver los conflictos en relación al archivo base.
- `git diff <rama-origen> <rama-destino>`: Se utiliza para ver los conflictos entre ramas antes de fusionarlas.
- `git diff --cached`: Muestra los cambios en los archivos preparados (staged).
- `git diff HEAD`: Muestra todos los cambios en archivos, tanto preparados como no preparados.
- `git diff commit1 commit2`: Muestra los cambios entre dos identificadores de confirmación.

  Muestra las diferencias de los cambios realizados y que no han sido agregados a un commit.

git blame [archivo]

`git blame [archivo]` lista las fechas y autores de los cambios realizados en un archivo.

git show

- `git show`: Se utiliza para mostrar información sobre cualquier objeto de Git.
- `git show [confirmación]:[archivo]`: Muestra los cambios de un archivo para un identificador de confirmación y/o archivo específico.

git log

`git log`: Muestra una lista de los cambios realizados (commits), cuántas copias hay en el repositorio y el número de commits. Muestra el historial completo de cambios.

`git log -p [archivo/directorio]`: Muestra el historial de cambios para un archivo o directorio, incluyendo las diferencias.

`git log --oneline`

`git log --oneline --decorate --all --graph --since=2018-12-04`

`git log` se utiliza para ver el historial del repositorio, mostrando detalles específicos de cada confirmación. Al ejecutar el comando, obtendrás una salida similar a esta:

```
achalmaubuntu\@hp-pavilion:\~/Documents/GitHub/achalmed\$ git log

commit 7e320e8bc6939626195a83def24f91308683a87e (HEAD -> main, origin/main, origin/HEAD)
Author: achalmed <achalmaedison@outlook.com>
Date:   Sun Jun 4 14:02:57 2023 -0500

    update
```

Trabajando con Ramas

En este artículo, exploraremos las diferentes operaciones relacionadas con las ramas en Git. Resaltaré los puntos clave de cada comando:

Listar y crear ramas

- **Listar ramas locales**: El comando `git branch` muestra todas las ramas locales presentes en el repositorio. Puedes utilizarlo para obtener una lista de las ramas existentes.
- **Listar todas las ramas**: Para ver tanto las ramas locales como las remotas, puedes usar `git branch -av`. Esto proporcionará una lista completa de todas las ramas del repositorio.
- **Cambiar a una rama**: Si deseas cambiar a una rama específica y actualizar tu directorio de trabajo, utiliza `git checkout [mi_rama]`.
- **Crear una nueva rama**: Utiliza `git branch [nuevo_nombre_rama]` para crear una nueva rama con el nombre especificado.
- **Eliminar una rama**: Si deseas eliminar una rama, utiliza `git branch -d [nombre_rama]`. Esto eliminará la rama especificada.
- **Ver todas las ramas**: Para obtener una lista de todas las ramas creadas en el proyecto, puedes utilizar `git branch -a`.

Crear ramas para pruebas de versión

Si necesitas crear una rama para realizar pruebas de versión u otras modificaciones, sigue estos pasos:

1. **Crear una nueva rama**: Utiliza `git branch [Rama2]` para crear una nueva rama basada en la rama actual.
2. **Ver los commits**: Utiliza `git log --oneline` para mostrar todos los commits realizados en la rama.
3. **Indicar las ramas**: Puedes utilizar `git branch` para ver las ramas del proyecto y determinar en qué rama te encuentras actualmente.
4. **Cambiar a otra rama**: Utiliza `git checkout Rama2` para cambiar al flujo de trabajo de otra rama, por ejemplo, una rama de prueba.

Una vez que hayas realizado las modificaciones de prueba, sigue estos pasos:

5. **Agregar los cambios**: Utiliza `git add .` para agregar los archivos modificados, incluyendo los cambios realizados en la rama de prueba.
6. **Realizar el commit**: Utiliza `git commit -m "Saludos, agregado a la rama de prueba"` para realizar el commit de los cambios.
7. **Ver los commits**: Utiliza `git log --oneline` para mostrar todos los commits realizados en la nueva rama.

Para regresar a la rama principal (Master o main), sigue estos pasos:

1. **Cambiar a la rama principal**: Utiliza `git checkout [master]` para cambiar nuevamente al flujo de trabajo de la rama principal. Ten en cuenta que los cambios realizados en la rama de prueba no estarán presentes aquí.

Si deseas ver los cambios en los archivos, minimiza la consola de Git.

Para enviar finalmente los archivos al repositorio en línea, escribe el siguiente comando:

1. **Enviar los cambios**: Utiliza `git push origin [Rama2]` para enviar los cambios al repositorio en línea. También puedes reemplazar `[Rama2]` con `main` si esa es la rama principal en tu repositorio en línea, como GitHub.

   ¡Recuerda que trabajar con ramas te permite experimentar y realizar pruebas sin afectar la rama principal del proyecto!

Fusionando ramas con `git checkout [archivo]`

El comando `git checkout [archivo]` es muy útil para crear y navegar entre ramas en Git. Aquí están los puntos clave que debes tener en cuenta:

- **Crear una nueva rama y cambiar automáticamente a ella**: Utiliza `git checkout -b [nombre-rama]` para crear una nueva rama y cambiar inmediatamente a ella.
- **Cambiar entre ramas**: Para cambiar de una rama a otra, simplemente utiliza `git checkout [nombre-rama]`.

Cuando se trata de fusionar ramas, sigue estos pasos:

1. **Cambiar a la rama de destino**: Antes de fusionar las ramas, asegúrate de estar en la rama de destino (por ejemplo, la rama `master`). Utiliza `git checkout [branch_b]` para cambiar a esa rama.
2. **Fusionar la rama**: Utiliza `git merge [branch_a]` para fusionar la rama `branch_a` en la rama actual (`branch_b`). Esto combinará los cambios realizados en `branch_a` con la rama actual.

Es importante destacar que al fusionar ramas, podrían surgir conflictos. Sin embargo, si estás utilizando Visual Code, resolver los conflictos será más sencillo gracias a sus herramientas de resolución de conflictos integradas.

Para realizar la fusión de ramas correctamente, sigue estos pasos:

1. **Cambiar a la rama principal**: Asegúrate de estar en la rama principal (por ejemplo, `master`) antes de realizar la fusión. Utiliza `git checkout [master]` para cambiar a la rama principal.
2. **Realizar la fusión**: Utiliza `git merge [Rama2]` para fusionar la rama `Rama2` en la rama principal.

Recuerda que fusionar ramas te permite combinar los cambios realizados en diferentes ramas, lo que facilita el trabajo colaborativo y la incorporación de nuevas características a tu proyecto.

¡Experimenta con diferentes ramas y disfruta de la flexibilidad y el poder de Git!

Incorporando cambios con `git rebase [nombre_rama]`

El comando `git rebase [nombre_rama]` es una herramienta poderosa que nos permite incorporar los cambios de una rama en otra. Aquí están los puntos clave que debes conocer:

- **Incorporar cambios de una rama**: Utiliza `git rebase [nombre_rama]` para traer los cambios de la rama especificada y aplicarlos en la rama actual. Esto puede ser útil para mantener un historial de cambios más limpio y lineal.

Etiquetando commits con `git tag [versiones]`

Las etiquetas en Git nos permiten marcar puntos específicos en la historia de nuestro proyecto. Aquí están los puntos clave que debes tener en cuenta:

- **Etiquetar el commit actual**: Utiliza `git tag [nombre_tag]` para crear una etiqueta en el commit actual. Por ejemplo, puedes usar `git tag v0.0.1` para marcar la versión v0.0.1.
- **Etiquetar un commit específico**: Si deseas etiquetar un commit específico, puedes usar `git tag [nombre_tag]/commit_SHA1`, donde `commit_SHA1` es el identificador único del commit.

Para acceder al commit donde se encuentra una etiqueta, utiliza el siguiente comando:

```
git checkout [nombre_tag]
```

Las etiquetas son útiles para marcar puntos de lanzamiento o versiones importantes en tu proyecto. Los desarrolladores las utilizan para indicar hitos significativos, como v1.0 o v2.0. Por ejemplo, puedes crear una etiqueta v1.1.0 en el commit específico utilizando el siguiente comando:

```
git tag v1.1.0 [insertar-ID-commit-aquí]
```

Ejemplos

1. **Creación de una etiqueta de versión:** Utiliza el comando `git tag 20-06-2023v1.0.0 -m "Versión 1 del proyecto"` para crear una etiqueta que represente una versión específica de tu proyecto.
2. **Subir etiquetas en línea:** Utiliza el comando `git push origin master --tags` para enviar las etiquetas al repositorio remoto y compartirlas con otros colaboradores.

   Las etiquetas son una forma conveniente de marcar y acceder a puntos importantes en tu historial de cambios. ¡Aprovecha su potencial y mantén un seguimiento claro de las versiones de tu proyecto!

Personalizando la configuración con `git config`

El comando `git config` te permite personalizar la configuración de Git según tus necesidades. Aquí tienes algunos puntos clave:

- **Agregar un alias**: Puedes utilizar el comando `git config --global alias.lodag 'log --oneline --decorate --all --graph'` para crear un alias. Los alias son atajos que te permiten ejecutar comandos largos y complejos de una manera más sencilla.
- **Ver la lista de alias**: Si deseas ver la lista de alias creados, utiliza el comando `git config --global --get-regexp alias`. Esto te mostrará todos los alias definidos en la configuración global de Git.
- **Eliminar un alias**: Si ya no necesitas un alias, puedes eliminarlo utilizando el comando `git config --global --unset alias.trololo`, donde "trololo" es el nombre del alias que deseas eliminar.

Además de los alias, `git config` también te permite establecer otras configuraciones específicas del usuario, como el correo electrónico y el nombre de usuario. Aquí tienes algunos ejemplos:

- **Establecer el correo electrónico**: Utiliza el comando `git config --global user.email tuemail@ejemplo.com` para configurar tu dirección de correo electrónico. El uso de la opción `--global` indica que esta configuración se aplicará a todos los repositorios locales.
- **Configuración local**: Si deseas utilizar diferentes direcciones de correo electrónico para diferentes repositorios, puedes usar el comando `git config --local user.email tuemail@ejemplo.com`. Esto establecerá la configuración solo para el repositorio actual.

Personalizar la configuración de Git según tus preferencias te permite trabajar de manera más eficiente y adaptar Git a tus necesidades específicas. ¡Aprovecha las opciones de `git config` para optimizar tu flujo de trabajo!

Realiza cambios en Git

Cuando trabajas con Git, es importante saber cómo hacer cambios y preparar tus archivos correctamente. Aquí tienes algunos puntos clave:

`git add [archivo]`

- **Preparar un archivo específico**: Utiliza el comando `git add [archivo]` para preparar un archivo en particular y agregarlo al área de preparación. Por ejemplo, si deseas indexar el archivo `temp.txt`, ejecuta el siguiente comando:

  ```
  git add temp.txt
  ```

- **Preparar todos los archivos modificados**: Si has realizado cambios en varios archivos y deseas prepararlos todos, utiliza el comando `git add .`. Esto sincronizará y preparará todos los archivos modificados para el siguiente commit.

- **Actualizar los cambios**: Si ya has preparado algunos archivos y deseas actualizar los cambios realizados, puedes utilizar el comando `git add -u`. Esto agregará al área de preparación todos los archivos que hayan sido modificados o eliminados.

Ejemplos

- **Agregar un archivo específico**: Si deseas agregar el archivo `index.html` al área de preparación, ejecuta el siguiente comando:

```
git add index.html
```

- **Actualizar todos los cambios**: Si has realizado modificaciones en varios archivos y deseas prepararlos todos, utiliza el siguiente comando:

```
git add .
```

Cuando hayas realizado tus cambios y preparado los archivos, sigue los siguientes pasos para confirmarlos y enviarlos al repositorio en línea:

1. Verifica la rama actual utilizando el comando `git branch`.
2. Utiliza `git add .` para agregar todos los archivos modificados al repositorio local. También puedes usar `git add [nombreDelArchivo]` para agregar un archivo específico.
3. Genera un commit para confirmar los cambios utilizando el comando `git commit -m "Aquí va el mensaje"`. Asegúrate de proporcionar un mensaje descriptivo que explique los cambios realizados.
4. Finalmente, envía los archivos al repositorio en línea utilizando el comando `git push origin [rama]`. La rama puede ser `master` o `main`, dependiendo de tu configuración.

Si deseas verificar el estado de la transacción después de cada instrucción, puedes utilizar el comando `git status`. Esto te mostrará información sobre los archivos preparados, los cambios pendientes y el estado actual del repositorio.

> ¡Recuerda que hacer cambios y preparar correctamente tus archivos es fundamental para mantener un flujo de trabajo eficiente con Git!

Realizar commits en Git y fusionar ramas

Cuando trabajas con Git, es esencial entender cómo realizar commits y fusionar ramas. Aquí tienes algunos puntos clave:

`git commit`

- El comando `git commit` crea una instantánea de los cambios realizados y los guarda en el directorio Git.
- Utiliza `git commit -m "El mensaje que acompaña al commit va aquí"` para agregar un mensaje descriptivo que explique los cambios realizados en el commit.

Recuerda que los cambios confirmados no se envían automáticamente al repositorio remoto.

- Si deseas agregar y hacer commit de todos los archivos rastreados en el historial versionado al mismo tiempo, utiliza `git commit -am "mensaje del commit"`.

Ejemplos

- Ejecuta `git commit -m "Comienzo del proyecto"` para crear un commit con el mensaje "Comienzo del proyecto".
- Si deseas agregar y hacer commit de todos los archivos modificados, utiliza `git commit -am "Párrafo y tamaño de fuente"`.

`git add . && git commit -m "Cambionumeros"`

- Para agregar y hacer commit de los cambios en una sola línea, utiliza el comando `git add . && git commit -m "Cambionumeros"`.

`git merge [nombre_rama]`

- El comando `git merge [nombre_rama]` fusiona una rama con otra rama activa.

Descripción

Supongamos que tenemos las ramas "develop" y "feature", y queremos integrar la rama "feature" en "develop". Sigue estos pasos:

1. Cambia a la rama "develop" utilizando `git checkout develop`.
2. Ejecuta `git merge feature` para fusionar la rama "feature" en la rama "develop".

`git revert --m 1 SHA1_merge`

- `git revert -m 1 SHA1_merge` revierte un merge específico especificando el SHA1 del merge que deseas revertir.

`git revert HEAD`

- `git revert HEAD`
- `git revert --no-commit HEAD`
- `git revert --no-commit HEAD~1`
- `git revert --continue`

Descripción

El comando `git revert` revierte un commit en la rama actual. Puedes revertir dos o más commits juntos utilizando el mismo comando de revert para que se tomen como uno solo.

Recuerda que realizar commits y fusionar ramas correctamente es fundamental para mantener un historial versionado limpio y una colaboración efectiva en Git.

Retrocediendo en el tiempo con Git: `git reset`

Cuando trabajamos en un proyecto de Git, es posible que necesitemos retroceder en el tiempo y regresar a un commit anterior. Para ello, el comando `git reset` es muy útil. Veamos cómo se utiliza:

- `git reset SHA1_commit/[nombre_rama]`
- `git reset --soft SHA1_commit/[nombre_rama]`
- `git reset --hard SHA1_commit/[nombre_rama]`

Descripción

El comando `git reset` nos permite regresar a un commit anterior y deshacer cambios en nuestro proyecto. Dependiendo de las opciones que utilicemos, los cambios se manejan de diferentes formas:

- `git reset SHA1_commit/[nombre_rama]`: Regresa a un commit o rama anterior y deja los cambios fuera del área de preparación.
- `git reset --soft SHA1_commit/[nombre_rama]`: Regresa a un commit o rama anterior y deja los cambios hechos en el commit que se va a quitar listos en el área de preparación.
- `git reset --hard SHA1_commit/[nombre_rama]`: Regresa a un commit o rama anterior y elimina por completo los cambios, dejando en blanco el área de preparación y el directorio de trabajo.

Es importante tener en cuenta que esta operación no afectará a otras ramas, por lo que puedes realizar nuevos commits a partir de ese punto sin modificar otras ramas del proyecto.

Adicionalmente, el comando `git reset --hard HEAD` nos permite resetear el índice y el directorio de trabajo al último estado de confirmación.

Conocer y utilizar `git log` con sus diferentes opciones nos permitirá manejar correctamente la creación de ramas, los movimientos entre ellas y los avances y retrocesos entre commits.

Regresando a un commit anterior en Git

A veces, necesitamos retroceder en el tiempo a un punto anterior en nuestro proyecto. Para ello, podemos utilizar el comando `git checkout` de la siguiente manera:

1. Utiliza el comando `git log --oneline` para ver la estructura de los últimos commits.
2. Ejecuta `git checkout [766abcd]` para regresar a un commit previo.

Existen diferentes formas de retroceder en el tiempo a commits anteriores. Además de `checkout`, también podemos utilizar `reset` con los atributos `soft` o `hard`.

- `git reset --soft [568abcj]`: Retrocede a un commit previo manteniendo los cambios.
- `git reset --soft HEAD~`: Deshace solo el último commit.
- `git reset --hard [789abcd]`: Elimina permanentemente los cambios realizados después de un commit específico.

Si deseamos eliminar los cambios después del último commit, podemos utilizar `git reset --hard HEAD~`. También es posible utilizar el comando `stash` para descartar los cambios antes de retornar a un commit.

> Recuerda tener precaución al utilizar estos comandos, ya que pueden modificar el historial del proyecto y afectar a otros colaboradores. Siempre es recomendable crear una copia de seguridad o consultar con tu equipo antes de realizar cambios significativos en el repositorio.

Synchronize

El proceso de sincronización en Git es fundamental para mantener actualizado el repositorio local con los cambios realizados en el repositorio remoto. A continuación, se presentan algunos comandos clave para sincronizar el repositorio local con el remoto.

`git fetch`

El comando `git fetch` permite al usuario obtener todos los objetos de un repositorio remoto que no se encuentran actualmente en el directorio de trabajo local. Esto no fusiona automáticamente los cambios con el repositorio local, solo los descarga y los hace accesibles para su revisión.

Ejemplo

Para obtener los cambios desde el repositorio en GitHub, puedes ejecutar:

```
git fetch origin
```

`git pull`

El comando `git pull` combina los cambios realizados en el repositorio remoto con el directorio de trabajo local. Básicamente, actualiza las ediciones de línea en el repositorio local. Es una combinación de los comandos `git fetch` y `git merge`.

Ejemplo

Para obtener los cambios y fusionarlos en el repositorio local, puedes utilizar:

```
git pull origin [rama]
```

o simplemente:

```
git pull --rebase
```

`git remote`

El comando `git remote` se utiliza para administrar las conexiones a repositorios remotos.

- `git remote`: Muestra todos los repositorios remotos conectados.
- `git remote -v`: Lista las conexiones junto con sus URLs.

Ejemplo

- Para agregar un repositorio remoto, utiliza el siguiente comando:

```
git remote add [nombre_repo] [URL]
```

- Para eliminar una conexión a un repositorio remoto específico, utiliza:

```
git remote remove [nombre_repo]
```

`git push`

El comando `git push` se utiliza para enviar los cambios locales a la rama principal del repositorio remoto. Permite sincronizar los commits locales con el repositorio remoto.

Ejemplo

Para enviar los cambios al repositorio en línea, puedes utilizar el siguiente comando:

```
git push origin [rama]
```

En caso de que no desees enviar los cambios a la rama principal, reemplaza `[rama]` por el nombre de la rama deseada.

> Recuerda que al redactar tu propio contenido, es fundamental utilizar tus propias palabras y expresiones únicas para evitar cualquier problema de plagio. Las sugerencias proporcionadas son solo ejemplos de cómo puedes mejorar la redacción del texto original.

Finally

En esta sección, exploraremos algunos comandos adicionales de Git que pueden resultar útiles en tu flujo de trabajo. A continuación, destacaré los puntos clave de cada comando:

`git command --help`

El comando `git command --help` te proporciona información detallada sobre el uso y las opciones disponibles para un comando específico de Git. Si tienes dudas, siempre puedes recurrir a `git help` para obtener ayuda.

`git stash`

El comando `git stash` te permite guardar temporalmente los cambios que aún no están listos para ser confirmados. Esto te permite cambiar de tarea o rama sin perder tus modificaciones actuales.

`git ls-tree`

El comando `git ls-tree` muestra información sobre un objeto de árbol en el repositorio. Muestra los nombres y modos de cada ítem, así como el valor blob de SHA-1. Puedes utilizar `git ls-tree HEAD` para ver el árbol actual (HEAD) del repositorio.

`git cat-file`

El comando `git cat-file` se utiliza para ver información sobre un objeto específico en el repositorio. Puedes usar la opción `-p` junto con el valor SHA-1 del objeto para ver su información detallada.

`git grep`

El comando `git grep` te permite buscar frases o palabras específicas en los árboles de confirmación, el directorio de trabajo y el área de preparación. Por ejemplo, para buscar la frase "www.hostinger.com" en todos los archivos, puedes ejecutar `git grep "www.hostinger.com"`.

`gitk`

El comando `gitk` muestra una interfaz gráfica para el repositorio local. Puedes utilizarlo para visualizar la historia del repositorio, las ramas y los commits de una manera más visual e interactiva.

`git instaweb`

El comando `git instaweb` te permite explorar tu repositorio local utilizando la interfaz GitWeb. Puedes configurar diferentes opciones, como el servidor HTTP, para acceder a la interfaz.

`git gc`

El comando `git gc` realiza la limpieza y optimización del repositorio local. Elimina archivos innecesarios y optimiza el almacenamiento para mejorar el rendimiento.

`git archive`

El comando `git archive` te permite crear archivos zip o tar que contienen los archivos de un árbol de repositorio específico. Puedes especificar el formato y el árbol que deseas archivar.

`git prune`

El comando `git prune` elimina los objetos del repositorio que no tienen apuntadores entrantes. Esto ayuda a limpiar el repositorio y eliminar objetos no utilizados.

`git fsck`

El comando `git fsck` realiza una comprobación de integridad del sistema de archivos git y busca objetos corruptos en el repositorio. Es útil para detectar posibles problemas de integridad.

`git rebase`

El comando `git rebase` se utiliza para aplicar cambios de una rama a otra. Permite mover, modificar o combinar commits para mantener una historia de confirmaciones más limpia y estructurada.

`git rm`

El comando `git rm` se utiliza para eliminar archivos del repositorio de Git. Puedes especificar los archivos que deseas eliminar y luego confirmar los cambios.

Borrar archivos/carpetas del repositorio

En este apartado, exploraremos cómo eliminar archivos o carpetas en un repositorio utilizando los comandos de Git. A continuación, resaltaré los puntos clave de cada paso:

1. **Actualizar los cambios:** Antes de borrar archivos o carpetas, es importante asegurarse de que todos los cambios estén actualizados en el repositorio. Puedes hacerlo utilizando el siguiente comando:

   ```
   git add -u
   ```

   Este comando actualizará todos los cambios realizados en el repositorio.

2. **Realizar un commit:** Después de asegurarte de que los cambios están actualizados, debes crear un commit para registrar la eliminación de los archivos o carpetas innecesarios. Puedes utilizar el siguiente comando:

   ```
   git commit -m "Elimino archivos innecesarios"
   ```

   Recuerda proporcionar un mensaje descriptivo para el commit.

3. **Subir los cambios:** Finalmente, debes enviar los cambios al repositorio en línea. Utiliza el siguiente comando para realizarlo:

   ```
   git push origin master
   ```

   Ten en cuenta que "master" puede ser reemplazado por "main" si esa es la rama principal en tu repositorio.

Si deseas eliminar un archivo específico, utiliza el siguiente comando:

```
git rm miarchivo.php
```

En caso de querer eliminar una carpeta completa, puedes utilizar el siguiente comando:

```
git rm -r micarpeta
```

Recuerda que después de ejecutar el comando `git rm`, debes realizar un commit y luego enviar los cambios al repositorio utilizando `git push`.

Conclusión

Aprender los comandos básicos de Git es fundamental para los desarrolladores, ya que les permite gestionar fácilmente el código fuente de sus proyectos. Aunque pueda llevar algo de tiempo recordarlos todos, nuestra hoja de trucos de Git puede ser de gran utilidad.

¡Practica estos comandos de Git y aprovecha al máximo tus habilidades de desarrollo! ¡Te deseamos mucho éxito!

Edison Achalma