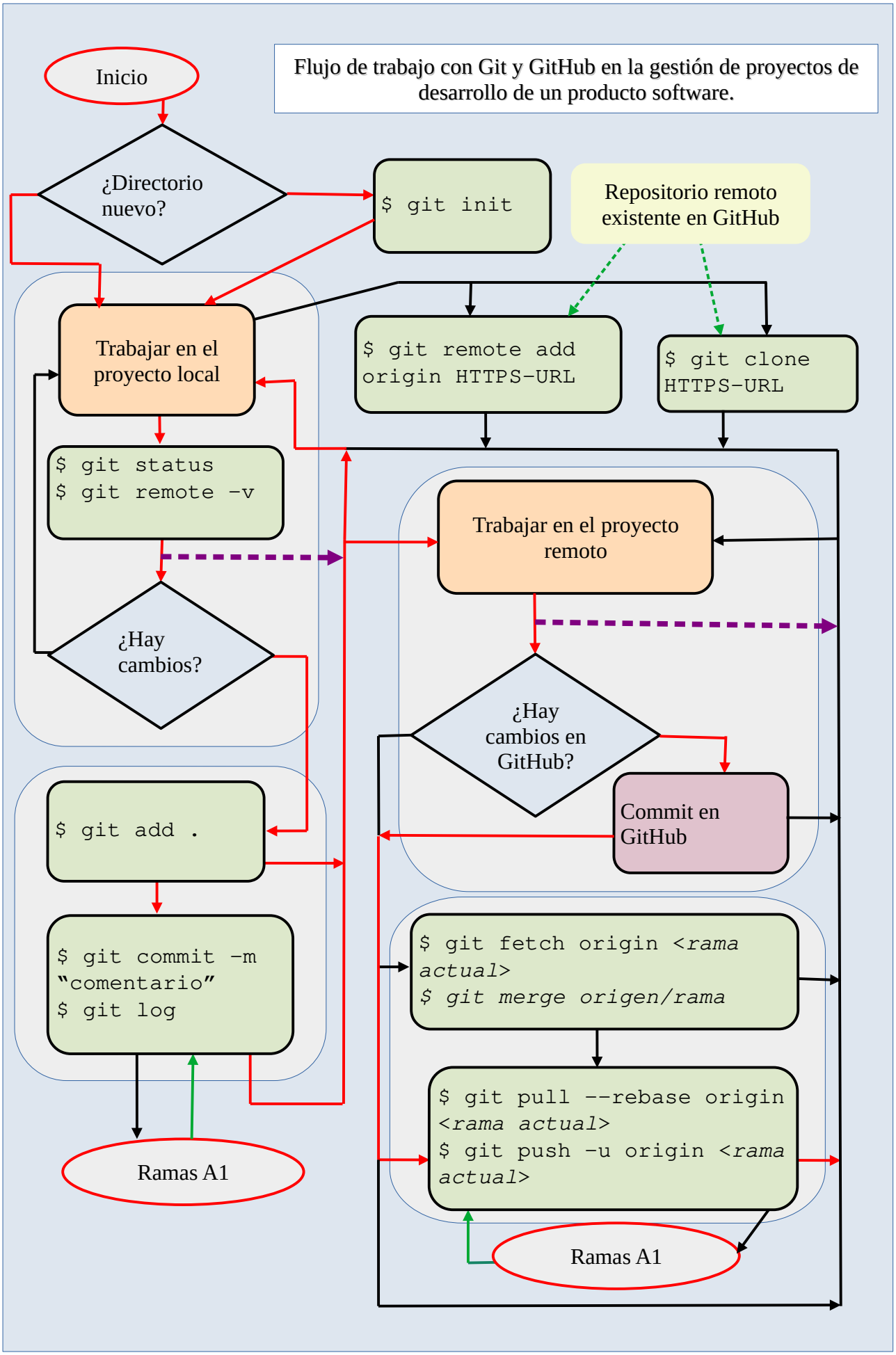


Autor: Carlos Privitera con ayuda de ChatGPT – Enero del 2023

Gestión de proyectos con Git y GitHub

Instalar Git en una computadora Linux para gestionar repositorios de forma local

La página para instalar Git: https://git-scm.com/downloads

Instalar desde Synaptic

Instalar desde la terminal

En el caso del Sistema Operativo Debian y derivados ejecutar:

apt-get install git

Para saber la versión de Git instalada ejecutar:

$ git --version

Después de instalar Git, hay que configurar los datos de la cuenta GitHub

En una Terminal hay que e indicar el nombre de usuario de GitHub y la dirección de correo del usuario vinculada a la cuenta en GitHub:

$ git config --global user.email SU-CORREO-ELECTRONICO
$ git config --global user.name "SU-NOMBRE-DE-USUARIO-DE-GITHUB"

Configurando un Tokens para autentificar en GitHub

Para configurar el token de autenticación de GitHub en Visual Studio Code y poder publicar un branch, puedes seguir estos pasos:

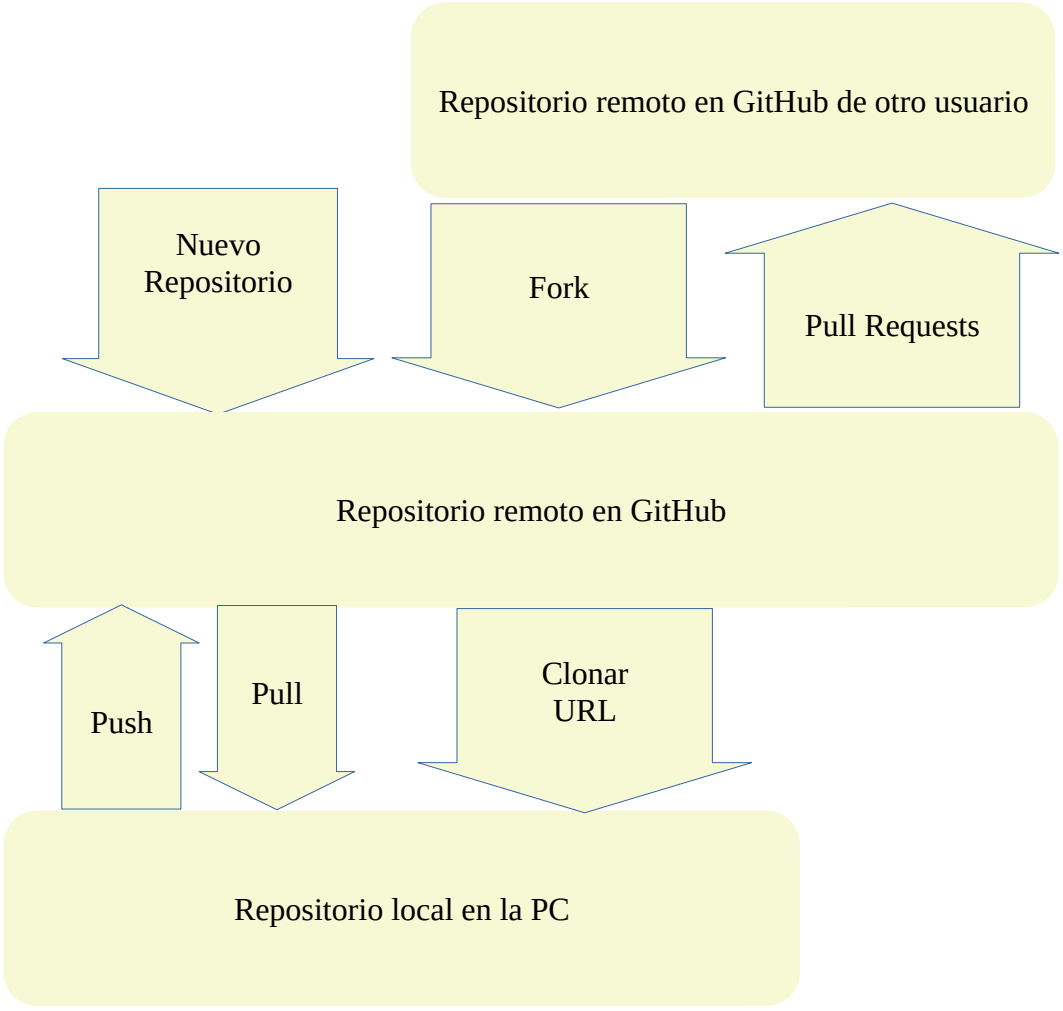
1. Genera un token de autenticación de GitHub desde tu cuenta de GitHub en la sección "Settings" > "Developer settings" > "Personal access tokens". Asegúrate de otorgar permisos adecuados para el token.
2. Abre Visual Studio Code y haz clic en la pestaña "Source Control" en la barra lateral izquierda.
3. Selecciona el botón de engranaje en la parte superior derecha del panel de control de origen y selecciona "Git: Clone" y escribir la URL cuando lo solicite, el comando en la terminal sería el siguiente $ git clone HTTPS-URL. Otra forma de hacer lo mismo sería el comando Git para asociar un repositorio remoto con uno local $ git remote add origin HTTPS-URL.
4. En la ventana de entrada, ingresa la URL del repositorio que deseas clonar, este paso no hace falta si se usó el comando Git para asociar el repositorio remoto con el repositorio local.
5. Haz clic en "Clone from GitHub", se abrirá el navegador y se te solicitará que inicies sesión en tu cuenta de GitHub, este paso no hace falta si se usó el comando Git, $ git remote add origin HTTPS-URL, para asociar el repositorio remoto con el repositorio local.
6. Cuando hayas iniciado sesión, se te pedirá que otorgues permisos a Visual Studio Code para acceder a tu cuenta de GitHub, en realidad conviene tener la página de GitHub abierta todo el tiempo y antes de comenzar a usar los comandos Git.
7. Ahora, en Visual Studio Code, ve a la pestaña "Source Control" y haz clic en el botón "Publish Changes".

8. Si este es el primer push que haces desde Visual Studio Code, se te solicitará que proporciones el nombre de usuario y la contraseña de tu cuenta de GitHub. En su lugar, ingresa tu token de autenticación generado en el paso 1 y haz clic en "OK".
9. Visual Studio Code ahora debería estar configurado para usar tu token de autenticación de GitHub para publicar cambios, push, en el repositorio remoto o hacer pull a tu repositorio local.

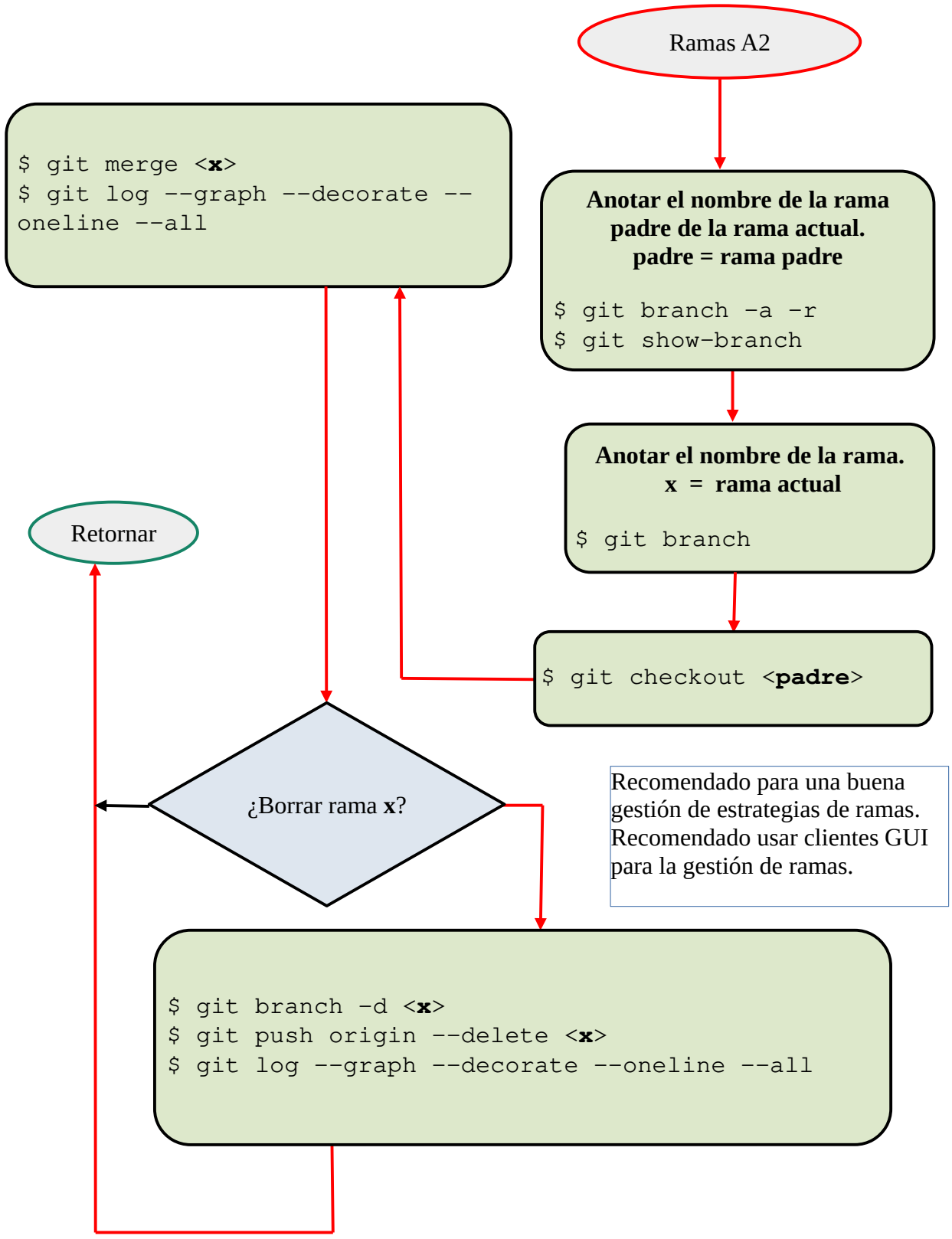
Es muy probable que antes de trabajar con VSCode convenga abrir el sitio de GitHub y el correo de Microsoft y tenerlos abiertos con las credenciales correctas, ya que VSCode necesitará verificar que el Explorador Web está autenticado antes de ejecutar comandos Git. Es conveniente verificar que dicha autenticación permanezca activa cuando ha pasado mucho tiempo.

En GitHub hay dos forma de tener un nuevo repositorio

1. Creando un nuevo repositorio
2. Haciendo un Fork de un repositorio existente en otra cuenta de GitHub



Gestión de Merge en Git y GitHub



Comandos básicos de Git

```
$ git init
```

El comando "git init" se utiliza para inicializar un repositorio de Git en una carpeta local, para comenzar con el comando git init hay que crear una carpeta nueva y dentro de la carpeta ejecutar el comando git init. Este comando crea una nueva carpeta llamada ".git" en la carpeta actual, que es donde Git almacena toda la información del repositorio, como los add y commits. El directorio .git se puede borrar, simplemente hay que ejecutar el comando git init nuevamente para poner bajo el control de versiones el directorio actual.

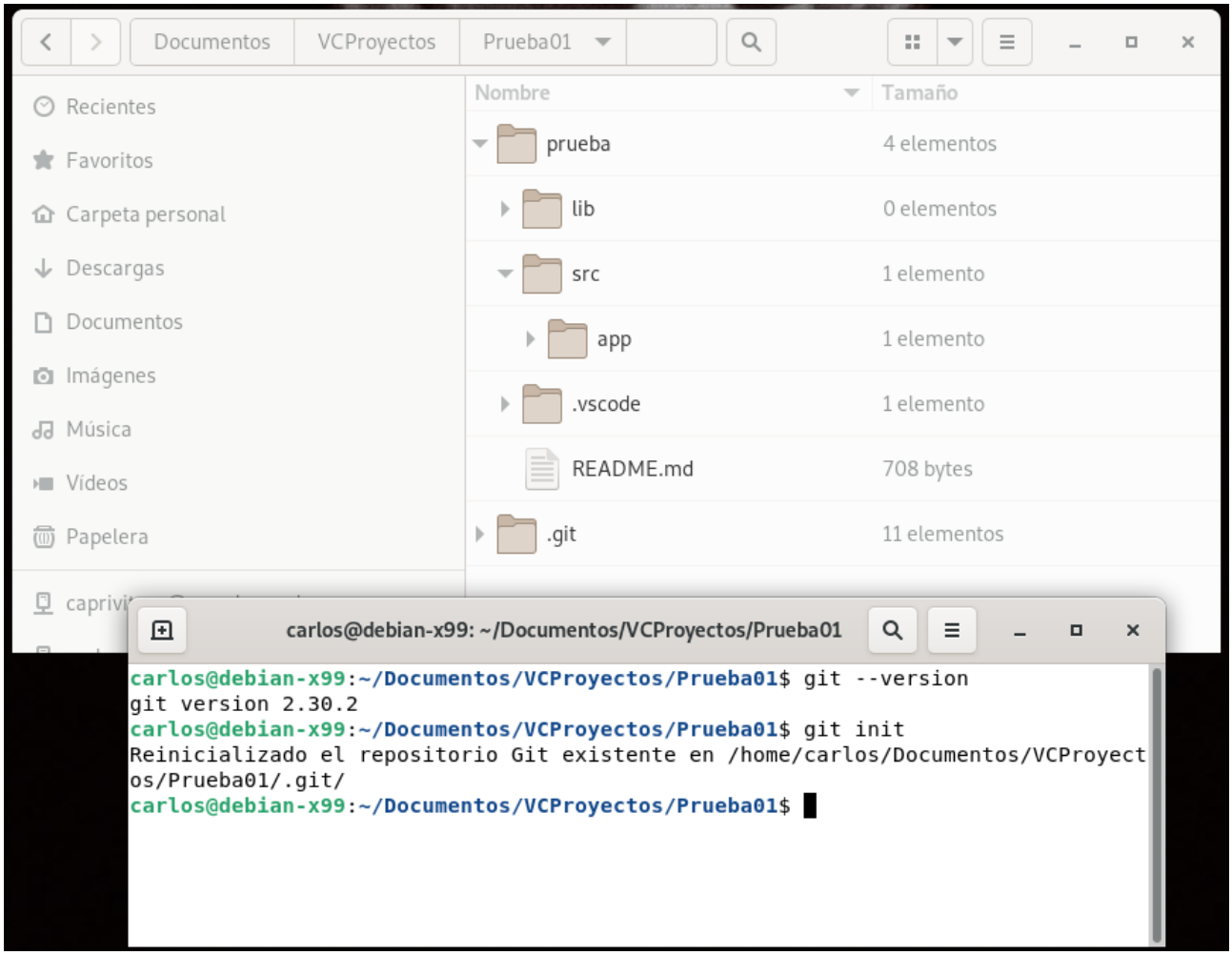


Figura 1: Ejemplo del comando git init en Linux

En la figura se ha ejecutado el comando git --version para ver la versión del programa Git instalado en el Sistema Operativo. Luego se ejecuta el comando git init para comenzar el control de versiones en el directorio actual.

Una salida típica del comando `$ git int`

```
ayuda: Using 'master' as the name for the initial branch. This default branch name
ayuda: is subject to change. To configure the initial branch name to use in all
ayuda: of your new repositories, which will suppress this warning, call:
ayuda:
ayuda: git config --global init.defaultBranch <name>
ayuda:
```

```
ayuda: Names commonly chosen instead of 'master' are 'main', 'trunk' and
ayuda: 'development'. The just-created branch can be renamed via this command:
ayuda:
ayuda: git branch -m <name>
```

En dicha ayuda Git nos explica el uso de los comandos:

- `$ git config --global init.defaultBranch <name>`
- `$ git branch -m <name>`

Una vez que se ejecuta "git init" en una carpeta, el repositorio de Git está listo para usar, y ya es factible rastrear cambios de archivos y hacer add, commits. Es importante mencionar que el comando solo inicializa el repositorio en el directorio donde se ejecuta, no crea una copia del repositorio en esa carpeta.

Los IDE. (Entornos de Desarrollo Integrado) y los editores de código tienen la funcionalidad de ejecutar los comandos Git desde su interfaz y gestionar el control de versiones de forma integrada y con el pluging adecuado es factible trabajar con Git de forma gráfica.

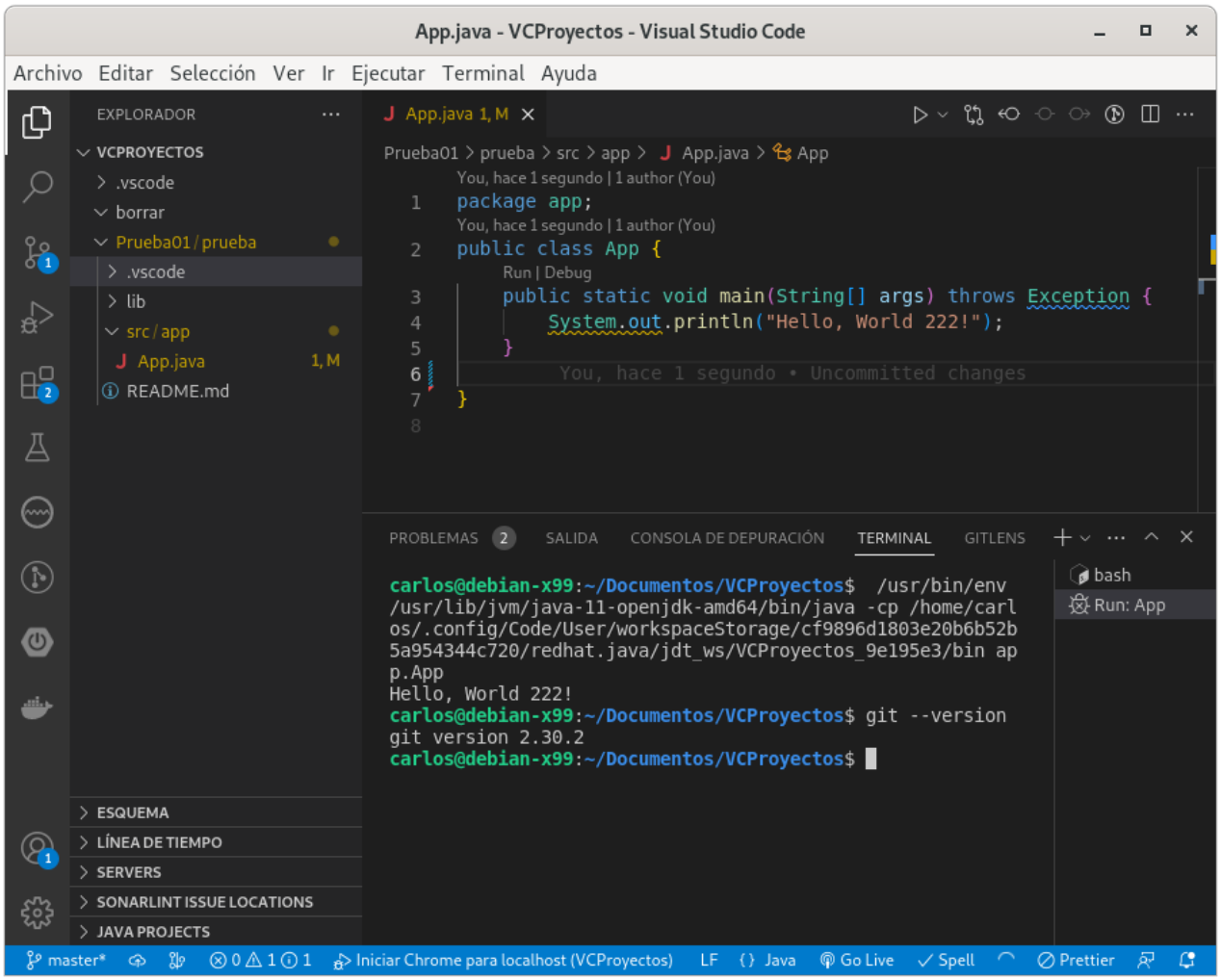


Figura 2: Ejemplo del control de versiones con Git en el editor de código VSCode

En resumen: `$ git init` es utilizado para inicializar un repositorio de Git en una carpeta específica, creando una nueva carpeta ".git" donde se almacenará toda la información del

repositorio y permitiendo así comenzar a hacer seguimiento de los cambios en los archivos en esa carpeta.

Recordar que en una Terminal hay que e indicar el nombre de usuario de GitHub y la dirección de correo del usuario vinculada a su cuenta en GitHub:

```
$ git config --global user.email SU-CORREO-ELECTRONICO
$ git config --global user.name "SU-NOMBRE-DE-USUARIO-DE-GITHUB"
```

Recordar que hay que tener las credenciales de GitHub: correo, clave de usuario y un tokens, y que hay que abrir la página de GitHub con las credenciales adecuadas antes de comenzar a usar los comandos Git. Los entornos IDE en muchos casos piden las credenciales GitHub para ejecutar algunos comandos Git, también puede pasar que la clave de usuario de GitHub es remplazada por el tokens obtenido en GitHub.

```
$ git clone
HTTPS-URL
```

El comando "git clone URL" se utiliza para crear una copia exacta de un repositorio remoto en tu computadora local. El comando "git clone" toma la URL del repositorio remoto como argumento y descarga todo el contenido del repositorio, incluyendo toda la historia de commits y ramas, en una nueva carpeta con el nombre del repositorio.

La URL del repositorio remoto se puede encontrar en la página del repositorio en el sitio web de GitHub o en la plataforma similar donde se encuentra el repositorio. Al ejecutar "git clone URL" se crea una nueva carpeta con el nombre del repositorio en el directorio actual y se descarga todo el contenido del repositorio remoto en esa carpeta.

Además de descargar el repositorio, al ejecutar "git clone" se establece una conexión entre el repositorio local y el remoto, lo que permite enviar y recibir cambios fácilmente. Una vez clonado el repositorio, puedes trabajar con él de la misma manera que con cualquier otro repositorio local, haciendo commits, creando ramas, etc.

!Importante¡
El clonado de un repositorio remoto se debe hacer sobre un directorio local vacío. Si el directorio local no está vacío puede suceder que se pierdan archivos. Para recuperar archivos después de una operación de fusionar o sincronizado de un repositorio remoto con uno local hay que ejecutar el siguiente comando:

```
$ git reflog
```

El comando `git reflog` ayudará a identificar los últimos commit locales realizados, para poder regresar el repositorio local al estado de uno de esos commit realizados en el pasado.

```
$ git checkout abc1234 -- .../ruta-repositorio-git/repositorio-git-a-recuperar
```

El comando `git checkout <commit n> -- <ruta a recuperar>` restablecerá los archivos borrados y dejará el repositorio en el estado del último commit n.

```
$ git remote add
origin HTTPS-URL
```

El comando "git remote add" es utilizado para agregar un repositorio remoto al repositorio local, un repositorio local puede estar asociado a varios repositorios remotos. Un repositorio remoto es una copia del

repositorio que se encuentra en un servidor externo, como GitHub, GitLab, o Bitbucket. El comando "git remote add" se utiliza para vincular el repositorio local con un repositorio remoto, para que sea posible enviar y recibir cambios entre ambos repositorios. Agregar un repositorio remoto es opcional, normalmente se lo utiliza para compartir y trabajar de forma colaborativa con otros programadores.

El comando "git remote add origin URL" se utiliza para agregar un nuevo repositorio remoto llamado "origin" y vincularlo con la URL especificada. La palabra "origin" es un nombre convencional para el primer repositorio remoto que se agrega, pero puedes darle el nombre que desees. La URL es la dirección del repositorio remoto en internet, por ejemplo "https://github.com/username/repository.git"

Una vez que se ha agregado el repositorio remoto, es posible utilizar comandos como "git push" o "git pull" para enviar o recibir cambios entre el repositorio local y el remoto.

En resumen, el comando "git remote add origin URL" se utiliza para vincular un repositorio local con un repositorio remoto, mediante la adición de un nuevo repositorio remoto llamado "origin" y especificando la URL del repositorio remoto, permitiendo así enviar o recibir cambios entre el repositorio local y remoto.

¿Cual es la diferencia entre origin y master y main?

"origin" es el nombre que se le da al repositorio remoto donde se almacena el código de tu proyecto. Es decir, es el lugar donde se encuentra el código de tu proyecto que se ha subido a un servidor remoto, como GitHub, GitLab, Bitbucket, entre otros. Cuando clonas un repositorio de un servidor remoto, se crea un enlace entre tu repositorio local y el repositorio remoto, y se le da el nombre "origin" por defecto.

"master" o “main”es el nombre de la rama principal de tu repositorio local. Es la rama por defecto que se crea cuando inicializas un repositorio en Git. Esta rama se utiliza para almacenar el código estable y funcional de tu proyecto. Puedes crear otras ramas para desarrollar nuevas características o corregir errores, y luego fusionarlas con la rama "master" una vez que se hayan completado y probado.

En resumen, "origin" es el repositorio remoto donde se encuentra el código de tu proyecto, mientras que "master" es la rama principal de tu repositorio local donde se encuentra la versión estable de tu proyecto. Puedes hacer cambios en tu repositorio local, crear nuevas ramas y fusionarlas con la rama "master", y luego enviar tus cambios al repositorio remoto en "origin" mediante comandos como "git push".

¿Cuál es la diferencia entre el comando `git clone` y el comando `git remote add`?

La diferencia entre el comando "git clone" y el comando "git remote add" es que el primero se utiliza para crear una copia local de un repositorio remoto, mientras que el segundo se utiliza para agregar un repositorio remoto existente a un repositorio local existente.

"git clone" es utilizado para crear una copia exacta de un repositorio remoto en tu computadora local, incluyendo toda la historia de commits y ramas. Al ejecutar "git clone" se crea una nueva carpeta con el nombre del repositorio, y dentro de ella se encuentra todo el contenido del repositorio remoto. Al clonar un repositorio también se establece una conexión con el repositorio remoto original, lo que permite enviar y recibir cambios fácilmente.

"git remote add" se utiliza para agregar un repositorio remoto existente a un repositorio local existente. Es decir, si ya tienes un repositorio local y quieres vincularlo con un repositorio remoto existente, puedes utilizar "git remote add" para establecer esa conexión. Es importante mencionar que no se descarga ninguna copia del repositorio remoto al utilizar este comando, solo se establece una conexión entre el repositorio local y el remoto.

¿Cuál es el procedimiento para clonar un repositorio remoto?

El procedimiento para clonar un repositorio remoto es el siguiente:

Asegúrate de tener Git instalado en tu computadora. Abrir una terminal o consola en tu computadora y crear una nueva carpeta y entrar en ella. Utiliza el comando "git clone" seguido de la URL del repositorio remoto que deseas clonar. La URL del repositorio se puede encontrar en la página del repositorio en el sitio web de GitHub o en la plataforma similar donde se encuentra el repositorio. El comando sería algo así como "$ git clone https://github.com/usuario/repositorio.git". Presiona Enter para ejecutar el comando. Git descargará una copia del repositorio remoto en una nueva carpeta con el nombre del repositorio en el directorio actual.

Una vez que la descarga se ha completado, puedes acceder a la carpeta del repositorio clonado y trabajar con los archivos como lo harías con cualquier otro repositorio local, por ejemplo ejecutar el comando $ git status.

Además de descargar el repositorio, al ejecutar "git clone" se establece una conexión entre el repositorio local y el remoto, lo que permite enviar y recibir cambios fácilmente.

Una vez dentro del repositorio clonado, puedes ejecutar comandos de Git como "git pull" para actualizar la copia local con los cambios del repositorio remoto, o "git push" para enviar tus cambios al repositorio remoto.

Puedes trabajar con el repositorio clonado de la misma manera que con cualquier otro repositorio local, haciendo commits, creando ramas, etc.

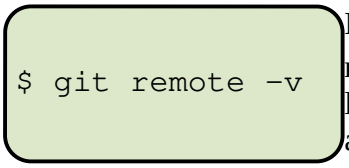
No es necesario inicializar un directorio antes de usar el comando git clone. El comando git clone es una forma común de obtener una copia de un repositorio Git existente en un servidor remoto. Cuando se ejecuta git clone, Git crea una copia completa del repositorio remoto en el directorio de destino especificado y establece automáticamente el repositorio local para seguir la rama por defecto del repositorio remoto.

Es importante mencionar que al clonar un repositorio, solo se descarga una copia de los archivos tal y como se encuentran en ese momento, no se actualizarán automáticamente a medida que se hagan cambios en el repositorio remoto, para esto necesitarás ejecutar el comando "git pull" para actualizar tu copia local con los cambios del repositorio remoto.

```
$ git status
```

El comando "`git status`" es uno de los comandos básicos de Git que se utiliza para ver el estado actual del repositorio local. Este comando muestra información sobre los archivos que se han modificado, eliminado o agregado desde el último commit, así como también información sobre el branch, (rama), en el que te encuentras actualmente y cual sería el branch al que estaría apuntando si hicieras un commit en ese momento. Ejecutando "`git status`" en la consola, se mostrará información sobre los archivos que están en seguimiento (tracked) y los que no lo están, los archivos modificados que aún no se han confirmado, y cualquier otra información

relevante sobre el estado actual de tu repositorio. Esta información es útil para saber qué cambios se han realizado, qué cambios están pendientes de confirmación y si estás trabajando en el branch correcto.



El comando `git remote –v` muestra la lista de los repositorios remotos asociados con el repositororio local de Git. Cada entrada en la lista incluye el nombre corto del repositorio remoto, la dirección URL asociada y si la URL está configurada para leer (fetch) o para leer y escribir (push). Esto es útil para verificar con qué repositorios remotos está asociado tu repositorio local y con qué URL se accede a ellos.

Aquí hay un ejemplo de la salida después de ejecutar `git remote –v` en un repositorio local:

```
origin  https://github.com/username/repo.git (fetch)
origin  https://github.com/username/repo.git (push)
```

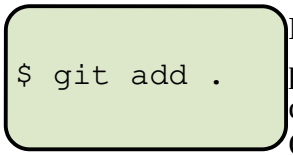
En este ejemplo, el nombre corto del repositorio remoto es origin, y su URL es https://github.com/username/repo.git. Además, se puede acceder tanto para leer como para escribir al repositorio remoto origin.

¿Cómo saber la URL del repositorio remoto en Git?

- Hay varias formas de conocer la URL del repositorio remoto en Git. Utilizando el comando "git remote -v". Este comando muestra todos los repositorios remotos vinculados con el repositorio local, junto con sus URLs. $ git remote -v
- Utilizando el comando "git config --get remote.origin.url". Este comando muestra la URL del repositorio remoto "origin" vinculado con el repositorio local. $ git config --get remote.origin.url
- Revisando el archivo ".git/config" en el directorio raíz del repositorio local: Este archivo contiene toda la configuración del repositorio, incluyendo la URL del repositorio remoto.
- Revisando en la plataforma donde se encuentra alojado el repositorio, por ejemplo en GitHub, en la sección de "settings" del repositorio.

Si estás utilizando alguna interfaz gráfica como SourceTree, puedes ver la URL del repositorio en la sección de "Repository" o "Remotes"

En resumen, existen varias formas de conocer la URL del repositorio en git, como utilizando los comandos "git remote -v" o "git config --get remote.origin.url", revisando el archivo ".git/config" o en la plataforma donde se encuentra alojado el repositorio.



El comando "`git add`" es uno de los comandos básicos de Git que se utiliza para agregar archivos al área de "staging" (preparación) antes de hacer un commit. Cuando se realizan cambios en los archivos en el repositorio local, Git no los rastrea automáticamente. El comando "`git add`" se utiliza para informar a Git que debe rastrear ciertos cambios en los archivos y agregarlos al área de preparación. El comando "`git add .`" se utiliza para agregar todos los archivos modificados y nuevos en el directorio actual y sus subdirectorios al área de preparación. Es importante mencionar que este comando no confirma los cambios, solo los prepara para ser confirmados con el comando commit.
En resumen "`git add .`" permite agregar todos los cambios realizados en el repositorio del directorio actual y sus subdirectorios al área de preparación, para que estén listos para ser confirmados en un commit posterior.

```
$ git commit –m
"comentario"
$ git log
```

El comando "`git commit`" es uno de los comandos básicos de Git que se utiliza para confirmar y registrar los cambios realizados en el repositorio. Un commit es una instantánea de los archivos en el repositorio en un momento específico, y permite volver a esa versión en cualquier momento. Cuando se ejecuta el comando "`git commit`", Git toma una instantánea de los archivos que se encuentran en el área de preparación (staging) y los almacena en la historia del repositorio. Cada commit tiene un mensaje asociado que describe los cambios realizados en esa versión. Es importante mencionar que antes de hacer un commit, es necesario agregar los archivos al área de preparación con el comando "`git add`" o "`git add .`" para que sean aceptados por el próximo commit.

En resumen, el comando "git commit" se utiliza para confirmar y registrar los cambios realizados en el repositorio, tomando una instantánea de los archivos en el área de preparación y registrando esa versión en la historia del repositorio, con un mensaje que describe los cambios realizados en esa versión.

```
$ git log
```

El comado `git log` es un comando que muestra el historial de confirmaciones (commit) en un repositorio Git. Cada confirmación representa un conjunto de cambios en el código que se han guardado y etiquetado con un mensaje describiendo las modificaciones realizadas. El comando `git log` muestra una lista de confirmaciones, con información detallada para cada una, como el autor, la fecha, el mensaje de confirmación y el hash de la confirmación (también conocido como identificador único). Esta información es útil para revisar el historial de cambios en un proyecto y ver quién ha hecho qué y cuándo. Por ejemplo, ejecutar `git log` en el repositorio local, y se puedrá ver una lista de todas las confirmaciones realizadas en el repositorio, con información detallada para cada una. Se puede usar opciones adicionales con `git log` para filtrar y personalizar la salida, como mostrar solo las confirmaciones realizadas por un autor específico, o mostrar solo las confirmaciones de un rango de fechas determinado.

```
$ git pull --rebase origin
<rama actual>
```

El comando "git pull" es utilizado para descargar cambios del repositorio remoto y combinarlos con el repositorio local. El comando "git pull" es una combinación de dos comandos: "git fetch" y "git merge", que se utilizan para descargar los cambios del repositorio remoto y combinarlos con el repositorio local respectivamente.

El comando "git pull --rebase origin master" es una variante del comando "git pull" que utiliza la opción "--rebase" para combinan los cambios. En lugar de crear un nuevo commit con los cambios descargados, "git pull --rebase" coloca los cambios descargados encima de los cambios locales, manteniendo un historial más limpio.

¿Qué diferencia hay entre pull y pull to Upstream?

El comando git pull fusiona los cambios del repositorio remoto con el repositorio local en el que te encuentras trabajando. Por defecto, git pull fusiona los cambios del branch remoto rastreado (por lo general origin/master) con el branch local actual.

Por otro lado, git pull --rebase también fusiona los cambios del repositorio remoto con el repositorio local, pero lo hace reorganizando los commits locales para que se apliquen después de los cambios del repositorio remoto. Esto puede ayudar a mantener un historial de cambios más limpio y fácil de seguir.

git pull y git pull --rebase se usan para actualizar tu rama local con respecto al estado actual del repositorio remoto.

Por otro lado, git pull upstream master es una variación del comando git pull que se utiliza para fusionar los cambios de otro repositorio remoto que no es el por defecto (origin). En este caso, upstream es el nombre del repositorio remoto y master es el nombre del branch remoto que se quiere fusionar. Este comando es útil cuando se trabaja en un fork de un repositorio, ya que permite actualizar el fork con los cambios del repositorio original (upstream) sin afectar al branch origin/master del fork.

En resumen, mientras que git pull y git pull --rebase se utilizan para actualizar tu rama local a partir del repositorio remoto que estás siguiendo (origin por defecto), git pull upstream master se utiliza para actualizar tu rama local a partir de un repositorio remoto diferente (upstream) y un branch remoto específico (master).

```
$ git fetch origin <rama actual>
$ git merge origen/rama
```

El comando "git fetch" se utiliza para descargar los cambios del repositorio remoto sin combinarlos automáticamente con el repositorio local. Es decir, "git fetch" descarga los cambios del repositorio remoto, pero no los aplica automáticamente al repositorio local. El comando "git fetch origin" es una variante del comando "git fetch" que especifica de qué repositorio remoto se deben descargar los cambios.

Un ejemplo de uso sería:

```
$ git fetch origin
```

En este caso, se descargarán los cambios del repositorio remoto "origin" pero no se combinaran con el repositorio local, es decir, no se actualizará el repositorio local con esos cambios. Esto permite revisar los cambios antes de aplicarlos al repositorio local, o simplemente tener una copia de los cambios en caso de que se necesite.

También puedes especificar la rama remota, para descargar solo los cambios de esa rama específica. Por ejemplo:

```
$ git fetch origin develop
```

En este caso se descargarán los cambios de la rama remota "develop" del repositorio remoto "origin" pero no se combinaran automáticamente con la rama local "develop". Es importante mencionar que para poder trabajar con los cambios descargados con "git fetch" es necesario crear una rama local con el mismo nombre de la rama remota, y hacerle un merge con la rama remota descargada.

Es importante destacar que después de ejecutar git fetch origen develop, es necesario realizar un comando adicional para fusionar los cambios descargados en la rama actual del repositorio local,

como por ejemplo git merge o git rebase. Este paso es necesario para aplicar los cambios descargados en el trabajo actual del programador.

En resumen, "git fetch origin" se utiliza para descargar los cambios del repositorio remoto "origin" sin combinarlos automáticamente con el repositorio local, permitiendo revisar los cambios antes de aplicarlos. También se puede especificar una rama remota para descargar solo los cambios de esa rama específica, pero para trabajar con los cambios descargados es necesario crear una rama local y hacerle un merge con la rama remota descargada.

Además, "git fetch" también permite descargar todas las ramas remotas y tags, para esto se puede utilizar el comando "git fetch --all". Este comando descargará todos los cambios de todas las ramas remotas y tags, y creará una rama local para cada una de ellas con el mismo nombre de la rama remota descargada.

```
$ git fetch --all
```

Este comando descargará todos los cambios de todas las ramas remotas y tags del repositorio remoto "origin" y creará una rama local para cada una de ellas con el mismo nombre de la rama remota descargada.

En resumen "git fetch" es una herramienta muy útil para descargar los cambios del repositorio remoto sin combinarlos automáticamente con el repositorio local. Puede trabajar con ramas específicas o con todas las ramas y tags, y permite revisar los cambios antes de aplicarlos al repositorio local.

Gestión de conflictos antes de hacer un git pull

Por ejemplo, puedes utilizar el comando "git diff" para comparar los cambios entre la rama local y la rama remota descargada con "git fetch":

git diff rama_local rama_remota

o en su defecto

git diff origin/rama_remota

de esta forma podrás ver los cambios entre ambas ramas y resolver cualquier conflicto antes de hacer el "git pull". Una vez resueltos los conflictos, puedes hacer un "git pull" para combinar los cambios del repositorio remoto con el repositorio local.

En resumen, el uso de "git fetch" antes de "git pull" permite controlar errores antes de aplicar los cambios del repositorio remoto al repositorio local, permitiendo revisar y resolver conflictos antes de combinar los cambios.

En conclusión. Usted decide si utiliza cualquiera de las tres secuencia de comandos Git a la hora de descargar cambios de un repositorio remoto:

1. fetch, merge o rebase, add y commit
2. fetch y pull
3. pull

Después de ejecutar el comando git pull, en la mayoría de los casos no es necesario realizar un git add ni un git commit, ya que git pull ya fusiona automáticamente los cambios descargados con la rama local y crea un nuevo commit del merge si es necesario.

Sin embargo, en algunos casos, es posible que necesites hacer un git add y un git commit después de ejecutar git pull. Por ejemplo, si la fusión automática genera conflictos de fusión que debes resolver manualmente, deberás editar los archivos para solucionar los conflictos y luego agregar los cambios con git add y crear un nuevo commit con git commit para completar la fusión.

Por otro lado, después de ejecutar git fetch y luego fusionar los cambios descargados con git merge, generalmente es necesario hacer un git add y un git commit para confirmar los cambios en la rama local. Esto se debe a que git fetch solo descarga los cambios del repositorio remoto a la rama remota correspondiente, mientras que la fusión real de los cambios en la rama local se realiza con git merge, lo que requiere una confirmación con git add y git commit.

!Importante¡

Es posible que durante el proceso de fusión (merge) al hacer el pull, cause la eliminación accidental de archivos. Para evitar esto, es recomendable seguir algunas buenas prácticas:

- Hacer commits frecuentes y con mensajes descriptivos.
- Antes de hacer un pull, asegúrate de haber hecho un commit o stash de todos los cambios que has hecho en tu rama.
- Revisa cuidadosamente los cambios que se van a fusionar durante el proceso de pull.
- Si hay conflictos, resolverlos cuidadosamente, verificando que no se borren o modifiquen accidentalmente archivos importantes.
- Si no estás seguro de lo que estás haciendo, haz una copia de seguridad de tus archivos importantes antes de hacer cualquier cambio.

Siguiendo estas prácticas, puedes minimizar la posibilidad de errores y pérdida de datos durante el desarrollo en equipo con repositorios remotos que necesitan funcionarse con repositorios locales.

¿Qué diferencia hay entre Fetch y Fetch to Upstream?

git fetch y git fetch upstream son comandos de Git que descargan cambios desde el repositorio remoto, pero la diferencia principal es el lugar donde se almacenan estos cambios.

Cuando se ejecuta git fetch, Git descarga los cambios del repositorio remoto y los almacena en el branch remoto-local temporal. Los cambios no se aplican al branch local actual, sino que se almacenan en el branch remoto-local que corresponde al branch local actual.

Por otro lado, cuando se ejecuta git fetch upstream, Git descarga los cambios del repositorio remoto y los almacena en una rama separada en el repositorio local, que se puede fusionar más tarde con la rama principal. Esto permite tener una copia local separada de los cambios remotos, lo que permite una mejor gestión de los cambios y una mayor flexibilidad para fusionar los cambios.

Git descarga los cambios del repositorio remoto y los almacena en un branch local temporal llamado FETCH_HEAD. Este branch contiene una copia exacta del estado del repositorio remoto, pero no está disponible para la edición y modificación. Por lo tanto, se debe fusionar los cambios en un branch local para integrarlos en el repositorio local. Entonces, el proceso sería:

Ejecutar git fetch para descargar los cambios del repositorio remoto y almacenarlos en el branch local temporal FETCH_HEAD.

Crear y cambiar al branch local que se desea actualizar con los cambios del repositorio remoto, utilizando el comando git checkout.

Fusionar los cambios del branch temporal FETCH_HEAD en el branch local utilizando el comando git merge.

Resolver cualquier conflicto que pueda surgir durante la fusión de los cambios.

Confirmar los cambios en el branch local y, si se desea, subirlos al repositorio remoto con el comando git push.

```
$ git push –u origin <rama actual>
```

El comando "git push -u origin master" es una variante del comando "git push" que utiliza la opción "-u" (o "--set-upstream") para vincular el repositorio local con una rama específica del repositorio remoto. El término "origin, master" especifica a qué repositorio y rama remota se enviarán los cambios. En este caso, "origin" es el nombre del repositorio remoto y "master" es el nombre de la rama en el repositorio remoto.

La opción "-u" indica que la rama local debe ser vinculada con la rama remota especificada. Esto significa que después de ejecutar "git push -u origin master" una vez, puedes usar simplemente "git push" en el futuro para enviar tus cambios a la rama remota "master" en el repositorio remoto "origin". Además te permite llevar un seguimiento de la rama remota a la que estas enviando tus cambios.

En resumen "git push -u origin master" se utiliza para enviar los cambios realizados en la rama local "master" al repositorio remoto "origin" especificando que se establezca una vinculación entre la rama local y la remota, para facilitar el envío de cambios futuros y llevar un seguimiento de la rama remota a la que estas enviando tus cambios.

Cómo Sincronizar un repositorio local antiguo con un repositorio nuevo de GitHub

Muchas veces estamos trabajando en un repositorio local con un proyecto Java o C++ gestionado por Git, en algún momento queremos subir nuestro viejo repositorio local a un nuevo repositorio GitHub, para hacer eso tenemos que crear un nuevo repositorio en GitHub y hacer como mínimo un commit en dicho repositorio remoto, normalmente se hace sobre el archivo README.md.

Escenario:

1. Un repositorio local con archivos antiguos inicializado con Git y con los comandos add y commit realizados correctamente, es decir no hay cambios para confirmar.
2. Un repositorio remoto creado recientemente y con al menos un commit en el archivo README.md.

Para sincronizar tu repositorio local con el remoto en GitHub, puedes seguir los siguientes pasos:

Agrega el repositorio remoto a tu repositorio local utilizando el siguiente comando:

```
$ git remote add origin <URL_del_repositorio_remoto>
```

Reemplaza `<URL_del_repositorio_remoto>` por la URL del repositorio remoto, La URL del repositorio GitHub se obtiene en la página de GitHub al momento de abrir el repositorio en un botón verde ubicado a la derecha de la página.

Verifica que el repositorio remoto se ha agregado correctamente ejecutando el siguiente comando:

```
$ git remote –v
```

Esto mostrará la lista de los repositorios remotos configurados.

Descarga los cambios del repositorio remoto en el repositorio local ejecutando el siguiente comando:

```
$ git pull –– rebase origin master–o–main
```

Este comando descarga los cambios en la rama principal (master o main según sea el caso) del repositorio remoto y los fusiona con la rama principal de tu repositorio local, si hay conflictos los resuelve de forma automática. Hay que tener en cuenta que este comando funciona si sabemos que no hay conflictos para resolver por un humano, es garantía que no hay conflictos ya que no hay archivos en común, el repositorio remoto se ha creado recientemente y el único archivo que tiene es el README.md. Si sabemos que habrá conflictos para resolver por un humano, el comando `$ git pull –– rebase origin master–o–main` deberá ser remplazado por los comandos `fetch y merge`, estos comandos gestionarán con ayuda de un humano los conflictos entre los dos repositorios.

Enviar los cambios de tu repositorio local al repositorio remoto ejecutando los siguientes comandos:

```
$ git add .
$ git commit –m "Mensaje de confirmación de cambios"
$ git push origin master–o–main
```

El primer comando agrega todos los cambios realizados en tu repositorio local. El segundo comando confirma los cambios con un mensaje. Y el tercer comando envía los cambios a la rama principal del repositorio remoto en GitHub.

El comando push pedirá el nombre del usuario GitHub y el password para la cuenta GitHub. El password debe ser un token que debe gestionarse en la página de GitHub.

Esto muestra la ejecución del comando push

```
$ git push –u origin master

Username for 'https://github.com': tu–nombre–de–usuario–GitHub
Password for 'https://carlosprivitera@github.com': tokens–GitHub

Enumerando objetos: 63, listo.
Contando objetos: 100% (63/63), listo.
Compresión delta usando hasta 48 hilos
Comprimiendo objetos: 100% (51/51), listo.
Escribiendo objetos: 100% (62/62), 55.02 KiB | 5.00 MiB/s, listo.
Total 62 (delta 21), reusado 0 (delta 0), pack–reusado 0
```

```
remote: Resolving deltas: 100% (21/21), done.
To https://github.com/.../tu-repositorioGitHub.git
   ed88cf0..bd306d8  master -> master
Rama 'master' configurada para hacer seguimiento a la rama remota 'master'
de 'origin'.
```

Con estos comandos, tu repositorio local y el remoto estarán sincronizados.

Qué hacer si al fusionar o sincronizar los repositorios local y remoto presentan conflictos entre sus archivos

Los conflictos pueden ser muchos

1. Dos miembros del equipo trabajan sobre un mismo archivo, uno en forma local y el otro en forma remota, al intentar sincronizarlos no se podrá combinar ambos archivos. Este problema re resuelve con comunicación efectiva entre los miembros del equipo de trabajo.
2. El repositorio local y remoto tienen dos archivos con el mismo nombre y contenidos distintos, el equipo de trabajo no tiene una estrategia de trabajo clara, este es el motivo para crear los workspace de trabajo para los miembros del equipo de trabajo.
3. Los commit de los repositorios locales y remotos no están relacionados, esto pasa la primera vez que se intenta sincronizar ambos repositorios, normalmente se resuelven con **rebase**.

Si sabemos que habrá conflictos para resolver, pero estos conflictos **no son un problema** podemos dejar que Git los resuelva por nosotros con el comando **rebase** de Git. Ejecutar el siguiente comando para resolver conflictos de forma automática:

`$ git pull --rebase origin <rama actual>`

Si sabemos que habrá conflictos para resolver por un humano, el comando `$ git pull --rebase origin master-o-main` deberá ser remplazado por los comandos `fetch` y `merge`, estos comandos gestionarán los conflictos entre los dos repositorios con intervención de un humano.

Para resolver los tipos de errores que debe solucionar un humano, seguir los siguientes pasos:

1. Verifica que estás en la rama correcta ejecutando el comando git branch. Si no estás en la rama correcta, cámbiate a ella con `git checkout <nombre-de-la-rama>`.
2. Ejecuta el comando `git fetch origin` para obtener los cambios remotos.
3. Luego, ejecuta el comando `git merge origin/master`. Esto fusionará la rama remota con la rama local.
4. Si hay conflictos de fusión, resolverlos manualmente y luego se realiza un commit con `git commit -m "Mensaje de commit"`.
5. Finalmente, haz un push con `git push origin <nombre-de-la-rama>` para enviar los cambios locales al repositorio remoto.

Con estos comandos, tu repositorio local y el remoto estarán sincronizados. Hay que hacer un push regularmente para mantener sincronizados los repositorios. Lamentablemente si se hacen cambios en el repositorio remoto se pueden perder archivos locales al hacer `pull, merge, rebase` o

cualquier combinación de comando Git que intenten traer el repositorio remoto al repositorio local. Por tal motivo es importante el manejo de ramas y el fusionado de ramas de forma local.

¿Cómo cambiar el nombre de la rama principal en Git?

Para cambiar el nombre de la rama principal en Git, seguir estos pasos:

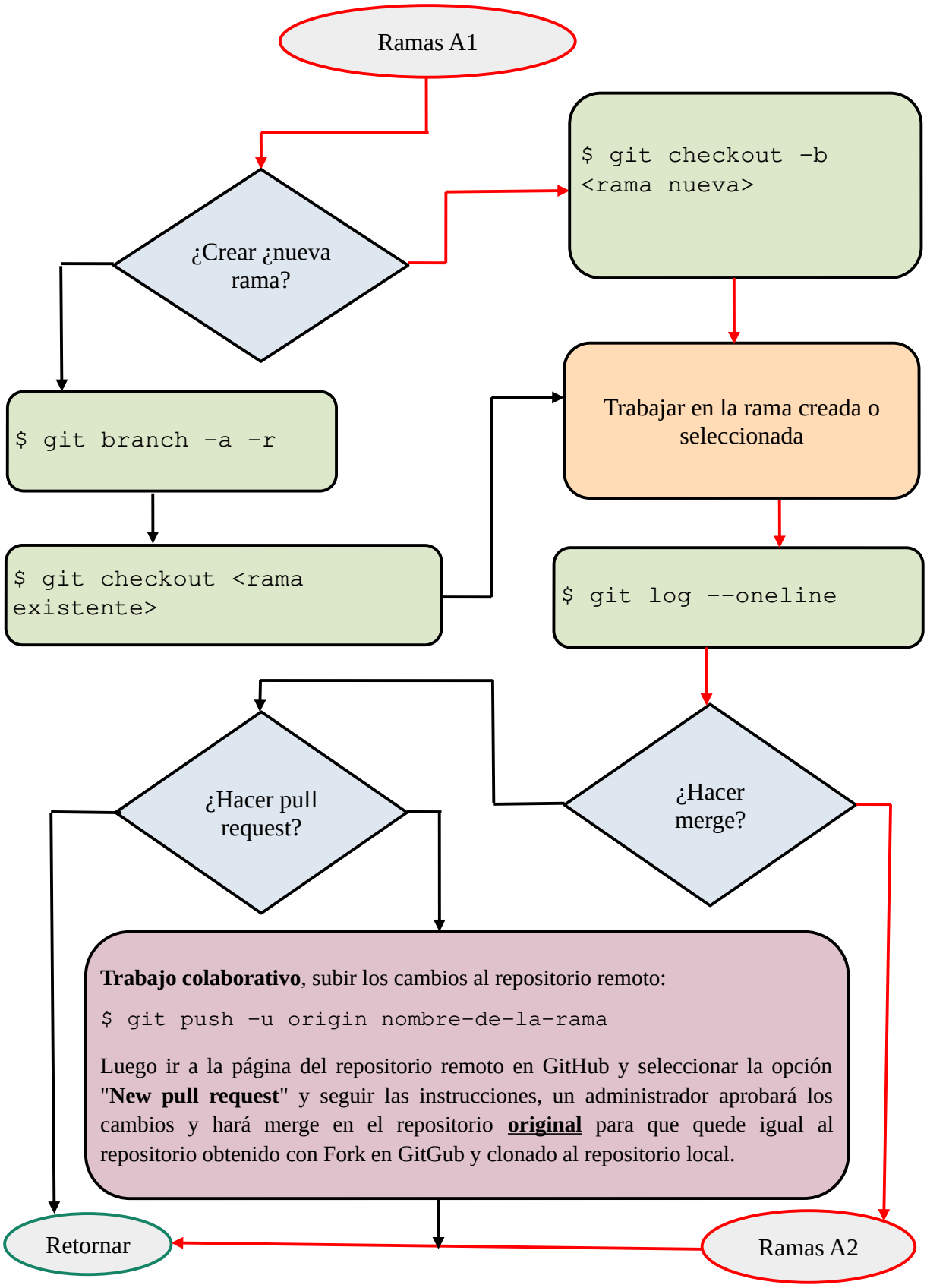
1. Seleccionar la rama a renombrar. Verificar el nombre con el comando: `$ git branch`.
2. Crea una nueva rama con el nombre deseado utilizando el comando `$ git branch nuevo_nombre`.
3. Mueve todos los cambios de la rama antigua a la nueva utilizando el comando `$ git reset --hard nuevo_nombre`. Recordar que no se está moviendo contenido o archivos, en realidad se está creando un puntero.
4. Elimina la rama antigua con el comando `$ git branch -D antiguo_nombre`.
5. Haz que la nueva rama sea la rama principal con el comando `$ git branch -m nuevo_nombre`.
6. Ten en cuenta que estos pasos debes hacerlos en tu computadora local y luego subir los cambios a tu repositorio remoto usando `$ git push --set-upstream origin nuevo_nombre` si es que estas trabajando en un repositorio compartido y aun no se ha realizado la subida.

¿Qué diferencia hay entre Push y Push to Upstream?

Al usar push -u o push --set-upstream, estás estableciendo la rama de destino predeterminada para futuros push. Esto significa que la próxima vez que quieras hacer push a esa rama, no necesitarás especificar la rama de destino. En lugar de eso, solo necesitarás ejecutar el comando git push.

Además, también es útil para colaboradores en proyectos de Git, ya que pueden ver fácilmente la rama que cada colaborador está siguiendo y evita errores comunes como enviar accidentalmente cambios a la rama equivocada.

Trabajando con ramas en Git



```
$ git branch –a –r
```

El comando git branch -a -r muestra todas las ramas remotas y locales en un repositorio git. El parámetro -a muestra todas las ramas, tanto las locales como las remotas. El parámetro -r muestra sólo las ramas remotas.

Otro comando para dar información similar sería git remote show, el cual muestra información sobre todos los repositorios remotos configurados en un repositorio git. Incluye información sobre las ramas remotas y las últimas actualizaciones.

En resumen, el comando git branch -a -r muestra todas las ramas remotas y locales en un repositorio git, mientras que el comando git remote show muestra información sobre todos los repositorios remotos configurados en un repositorio git, incluyendo información sobre las ramas remotas y las últimas actualizaciones.

```
$ git checkout –b
<rama nueva>
```

El comando "git checkout -b <new-branch>" se utiliza en Git para crear una nueva rama y cambiarse a ella en un solo paso. La opción "-b" se utiliza para indicar que se va a crear una nueva rama. Al ejecutar este comando, Git creará una nueva rama con el nombre especificado por "<new-branch>", y luego cambiará el HEAD (puntero) a la nueva rama. Esto significa que todos los nuevos cambios y confirmaciones que realices después de ejecutar el comando se agregarán a la nueva rama.

¿Qué diferencia hay entra git branch nueva_rama y git checkout -b nueva_rama?

La principal diferencia entre los dos comandos es que "git branch nueva_rama" simplemente crea una nueva rama con el nombre "nueva_rama" en el repositorio local, mientras que "git checkout -b nueva_rama" crea una nueva rama con el nombre "nueva_rama" y automáticamente cambia el puntero de la rama actual a esta nueva rama. En otras palabras, "git checkout -b" crea una nueva rama y se posiciona en ella, mientras que "git branch" solo crea una nueva rama.

```
$ git checkout <rama
existente>
```

El comando "git checkout <branch>" se utiliza en Git para cambiar de rama. Al ejecutar este comando, Git cambiará el HEAD (puntero) a la rama especificada por "<branch>", lo que significa que se actualizará el directorio de trabajo para reflejar el contenido de esa rama. Por ejemplo, si deseas cambiar a una rama llamada "mi-rama", puedes ejecutar el siguiente comando:

```
$ git checkout mi–rama
```

Esto cambiará el HEAD a la rama "mi-rama" y actualizará el directorio de trabajo para reflejar el contenido de esa rama. A partir de ahora, cualquier cambio que realices y confirmes se realizará en la rama "mi-rama", en lugar de la rama actual o la rama anterior al cambio de rama.

También es posible crear una nueva rama y cambiar a ella en un solo paso utilizando la opción "-b" como en el siguiente ejemplo:

```
$ git checkout –b mi–nueva–rama
```

Esto creará una nueva rama llamada "mi-nueva-rama" y te cambiará a ella. A partir de ahora, cualquier cambio que realices y confirmes se realizará en la nueva rama, en lugar de la rama actual.

```
$ git log --oneline
```

El comando "git log --oneline" muestra un historial simplificado de los compromisos (commits) realizados en el repositorio Git actual. Al agregar la opción "--oneline", se mostrará una sola línea de información para cada confirmación en lugar de la salida detallada que se obtiene sin esta opción. Cada línea de la salida de este comando mostrará el hash corto (abreviado) del commit y el mensaje de confirmación (commit message) asociado, todo en una sola línea. La salida de este comando es útil para obtener una vista rápida y general del historial de confirmaciones en el repositorio.

Por ejemplo, al ejecutar el siguiente comando: `$ git log --oneline`

Se mostrará un historial simplificado de todos los compromisos realizados en el repositorio Git actual. Cada línea de salida mostrará un hash abreviado de siete caracteres del commit, seguido del mensaje de confirmación asociado, como este:

```
1a2b3c4 commit message 1
5d6e7f8 commit message 2
9g8h7i6 commit message 3
```

Esta salida puede ser útil para identificar rápidamente las confirmaciones más recientes o para buscar una confirmación específica en el historial de Git.

En realidad, la traducción correcta del término "commit" en Git al español es "confirmación", "registro" o "registro de cambios", en lugar de "compromiso". Sin embargo, es posible que en algunos contextos se use el término "compromiso" para referirse a los "commits".

El término "commit" en Git se refiere a un registro de cambios en el repositorio Git, que se realiza mediante la confirmación de un conjunto de cambios realizados en los archivos del proyecto. Cada "commit" se registra con un mensaje que describe los cambios realizados, la fecha en que se realizó el "commit" y un hash que lo identifica de forma única.

La importancia de los "commits" en Git radica en que proporcionan un historial completo y detallado de los cambios realizados en el repositorio. Esto facilita el seguimiento de la evolución del proyecto, la identificación de problemas y la recuperación de versiones anteriores del código en caso de ser necesario.

En resumen, aunque la traducción correcta del término "commit" en Git al español es "confirmación" o "registro de cambios", en algunos contextos se puede utilizar el término "compromiso" para referirse a ellos, aunque es menos común.

Qué es esto? git config --global init.defaultBranch <name>

git config --global init.defaultBranch <name> es un comando de Git que se utiliza para establecer el nombre de la rama principal predeterminada al crear un nuevo repositorio. El parámetro --global indica que esta configuración se aplicará a todos los repositorios en el sistema, mientras que

init.defaultBranch especifica el nombre de la configuración y <name> es el nombre de la rama que deseas que sea la rama principal predeterminada.

Por ejemplo, si quieres que todos los nuevos repositorios que crees tengan una rama principal llamada "main" en lugar de "master", podrías ejecutar el comando git config --global init.defaultBranch main. A partir de ese momento, cada vez que utilices el comando git init para crear un nuevo repositorio, Git creará automáticamente una rama "main" en lugar de "master".

Ten en cuenta que esta configuración solo afecta a nuevos repositorios creados después de establecerla, no afecta a repositorios existentes.

Qué es esto? git branch -m <name>

git branch -m <name> es un comando de Git que se utiliza para renombrar una rama existente. El parámetro -m indica que estás renombrando una rama existente, y <name> es el nuevo nombre que le estás dando a la rama.

Por ejemplo, si quieres renombrar una rama llamada "antiguo_nombre" a "nuevo_nombre", podrías ejecutar el comando git branch -m antiguo_nombre nuevo_nombre.

Ten en cuenta que al usar este comando solo estas renombrando la rama local, si quieres que también se refleje en el repositorio remoto debes de usar git push origin --delete antiguo_nombre y luego git push origin nuevo_nombre.

También es importante mencionar que este comando solo funciona si estas en otra rama distinta a la que estas renombrando, si no es así tendrás que usar git branch -m <name> y luego git checkout <name> para moverte a la rama renombrada

¿Cómo borrar una rama en git local y GitHub al mismo tiempo?

Para borrar una rama en git local y GitHub al mismo tiempo, debes seguir los siguientes pasos:

Asegúrate de estar en la rama principal de la rama a borra (por ejemplo, "master" o rama x) antes de borrar la rama. Puedes cambiar de rama utilizando el comando git checkout.

$ git checkout master

Borra la rama localmente utilizando el comando git branch -d seguido del nombre de la rama.

$ git branch -d <rama x a borrar>

Envía los cambios al repositorio remoto utilizando el comando git push.

git push origin --delete <rama x a borrar>

Esto borrará la rama del repositorio remoto en GitHub.

Es importante mencionar que si la rama que se va a borrar no ha sido mergeada, entonces el comando git branch -d no funcionaría ya que git no te permitiría borrar una rama con cambios no mergeados, en ese caso debes utilizar git branch -D para forzar la eliminación.

En resumen, para borrar una rama en git local y GitHub al mismo tiempo debes asegurarte de estar en la rama principal, borrar la rama localmente con git branch -d, y luego enviar los cambios al repositorio remoto con git push origin --delete seguido del nombre de la rama a eliminar.

La estrategia principal para llevar un proyecto de construcción de un producto software de forma ordenada es borrar las ramas que son fusionadas con otra rama. De todas formas existen otras estrategias a la hora de ejecutar el comandos git merge y git rebase

Sí, existen varias estrategias de merge en git que se utilizan para fusionar ramas. A continuación, te describiré algunas de las estrategias de merge más comunes con ejemplos:

- Fast-forward: Es la estrategia de merge predeterminada en git. Si la rama actual no ha cambiado desde que se creó la rama a fusionar, git puede mover automáticamente el puntero de la rama actual al último commit de la rama a fusionar, sin crear un nuevo commit de merge.
  - `$ git merge feature-branch`
- Recursive: Es la estrategia de merge predeterminada si la rama actual ha cambiado desde que se creó la rama a fusionar. Git intentará resolver automáticamente cualquier conflicto que pueda surgir al fusionar las ramas. Si hay conflictos, debes resolverlos manualmente antes de hacer el merge.
  - `$ git merge --strategy-option recursive feature-branch`
- Octopus: Es una estrategia de merge que se utiliza cuando se fusionan más de dos ramas. Esta estrategia busca un punto común entre las ramas y crea un único commit de merge.
  - `$ git merge feature-branch1 feature-branch2 feature-branch3`
- Resolve: Es una estrategia de merge que se utiliza cuando se quiere forzar un merge y se sabe que git no podrá resolver automáticamente los conflictos. Debe ser utilizada junto con –strategy-option=resolve.
  - `$ git merge --strategy-option=resolve feature-branch`

Otras estrategias: Además de estas estrategias existen otras estrategias como subtree,ours,theirs entre otras.

En resumen, las estrategias de merge en git se utilizan para fusionar ramas, las estrategias más comunes son Fast-forward, Recursive, Octopus y Resolve, cada una tiene un propósito específico y un modo de operar, de

Estrategias de merge comunes con ejemplos:

- git merge: Es la estrategía de merge predeterminada en git. Crea un nuevo commit de merge en la rama actual con las modificaciones de la rama que se está fusionando.
  - `$ git merge feature-branch`
- git merge --squash: Es una estrategia de merge que combina los cambios de la rama que se está fusionando en un único commit en lugar de crear varios commits de merge.
  - `$ git merge --squash feature-branch`
- git rebase: Es una estrategia de merge que mueve los commits de la rama que se está fusionando al final de la rama actual en lugar de crear un nuevo commit de merge. Esto hace que el historial de la rama actual sea más lineal, ya que se "reescribe" la historia de la rama que se está fusionando para que aparezca como si los commits se hubieran realizado en la rama actual desde el principio.
  - `$ git rebase feature-branch`

En resumen, git merge es la estrategia de merge predeterminada, con git merge --squash se combinan los cambios de la rama que se está fusionando en un único commit, y con git rebase se mueven los commits de la rama que se está fusionando al final de la rama actual, esto hace que el historial de la rama actual sea más lineal.

Nombres específicos de las estrategias merge y rebase .

- git merge es conocido como "merge estándar" o "merge tradicional"
- git merge --squash se conoce como "squash merge" o "merge aplastado"
- git rebase se conoce como "rebase merge" o "rebase de rama"

Es importante mencionar que cada una de estas estrategias tiene sus ventajas y desventajas dependiendo del uso que se le quiera dar, y cada una de estas estrategias funciona mejor en situaciones específicas. Por ejemplo, el merge estándar crea un nuevo commit de merge, lo que permite ver fácilmente los cambios realizados en cada merge y el squash merge combina los cambios en un solo commit, lo que hace que el historial sea más limpio y fácil de seguir.

Beneficios y problemas de las estrategias de merge:

- Merge estándar (git merge):
  - Beneficios:
    - Es la estrategia de merge predeterminada en git.
    - Es fácil de entender y utilizar.
    - Crea un nuevo commit de merge, lo que permite ver fácilmente los cambios realizados en cada merge.
  - Problemas:
    - El historial de commits puede volverse muy denso y difícil de seguir.
    - Pueden haber conflictos difíciles de resolver.
- Squash merge (git merge --squash):
  - Beneficios:
    - Combina los cambios en un solo commit, lo que hace que el historial sea más limpio y fácil de seguir.
    - Permite hacer una revisión más detallada de los cambios antes de aplicarlos.
  - Problemas:
    - Puede ser más difícil de entender para los desarrolladores que no están familiarizados con esta estrategia de merge.
    - Puede ser difícil de revertir en caso de que se cometa un error.
- Rebase merge (git rebase):
  - Beneficios:
    - Mueve los commits de la rama que se está fusionando al final de la rama actual, lo que hace que el historial de la rama actual sea más lineal.
    - Puede ayudar a evitar conflictos al evitar que los cambios se solapen.
  - Problemas:
    - Puede ser más difícil de entender para los desarrolladores que no están familiarizados con esta estrategia de merge.
    - Puede ser difícil de revertir en caso de que se cometa un error.
    - Puede causar problemas de integridad de datos si se publican cambios antes de rehacer la rama actual.
    - Puede ser difícil de trabajar en un repositorio compartido si varios desarrolladores están trabajando en la misma rama.
    - Puede crear problemas de integridad de datos si se publican cambios antes de rehacer la rama actual.

Una estrategia de merge depende de las necesidades y preferencias, y de cómo te gustaría manejar tu historial de commits.

Es importante entender las diferencias entre estas estrategias para poder elegir la más adecuada para tu proyecto y equipo. Es recomendable también documentar las estrategias de merge que utilizas en tu proyecto para que otros desarrolladores puedan entender cómo se están manejando los cambios.

Que comando git me muestra la rama raiz de la rama actual.

Para mostrar la rama raíz (o rama principal) de la rama actual en git, se puede usar el comando

```
$ git rev-parse --abbrev-ref --symbolic-full-name @{u}
```

Este comando te muestra la rama a la cual está apuntando la rama remota de la rama actual.

El parámetro --abbrev-ref muestra solo el nombre abreviado de la rama, en lugar del SHA completo.

El parámetro --symbolic-full-name muestra el nombre completo de la rama, en lugar de la ruta completa desde el directorio raíz del repositorio.

El @{u} hace referencia a la rama remota asociada a la rama actual (upstream branch)

En resumen, el comando `git rev-parse --abbrev-ref --symbolic-full-name @{u}` te muestra la rama raíz (o rama principal) de la rama actual, es decir, a que rama remota está apuntando tu rama actual.

Un comando más sencillo para mostrar la rama raíz (o rama principal) de la rama actual.

```
$ git branch --show-upstream
```

Este comando te mostrará la rama remota asociada a la rama actual.

En resumen, el comando `git branch --show-upstream` es más sencillo que `git rev-parse --abbrev-ref --symbolic-full-name @{u}` para mostrar la rama raíz (o rama principal) de la rama actual, ya que te muestra directamente la rama remota asociada.

Saber la rama padre.

Si estás trabajando de forma local y quieres saber la rama padre de la rama local actual, puedes usar el comando:

```
$ git show-branch <rama actual>
```

Este comando te mostrará un gráfico de las ramas locales y te indicará la rama padre de la rama actual.

También puedes usar el comando:

```
$ git log --oneline --decorate --graph --all
```

Este te mostrará un historial de commits de todas las ramas locales y te indicará en qué rama se hizo el último commit antes de cambiar a la rama actual.

En resumen, puedes usar el comando `git show-branch` o `git log --oneline --decorate --graph --all` para ver la rama padre de la rama local actual. El primero muestra un gráfico de ramas locales y el segundo un historial de commits de todas las ramas locales.

Explícame cómo hacer un pull resquest considerando un escenario de cambios hechos en una rama nueva local y que tienen que ser aprobados por un usuario en GitGHub.

Un pull request (PR) es una solicitud para fusionar cambios desde una rama a otra, generalmente desde una rama de desarrollo a una rama principal (como "master"). Para crear un pull request en GitHub, sigue estos pasos:

Crea una nueva rama en tu repositorio local con los cambios que quieres enviar. Puedes hacerlo con el comando git branch <nombre_de_la_rama>

Cambia a esa rama con git checkout nombre_de_la_rama

Haz un push a esa rama en tu repositorio remoto con git push -u origin <nombre_de_la_rama>

Accede a tu repositorio en GitHub y verás un botón "Compare & pull request" en la página de la rama. Haz clic en él.

En la página siguiente, verifica que la rama base y la rama comparada estén correctamente seleccionadas. Agrega un título y una descripción para describir los cambios que has hecho.

Haz clic en "Create pull request" para crear la solicitud de extracción.

Una vez creado, el pull request se enviará al administrador del repositorio para su revisión y aprobación. Si todo está en orden, el administrador puede fusionar los cambios en la rama base.

Ten en cuenta que algunos repositorios pueden tener configuraciones diferentes en cuanto a quienes pueden crear o aceptar pull requests, así que es importante revisar las políticas del repositorio.

Trabajando con Fork

Para hacer un pull request en un escenario de cambios hechos en una rama nueva local y que tienen que ser aprobados por un usuario en GitHub, sigue estos pasos:

Haz un fork del repositorio original en GitHub. Esto te creará una copia del repositorio en tu cuenta de GitHub.

Clona el repositorio forked en tu computadora local con el comando git clone URL-del-repositorio-forked

Crea una nueva rama local para los cambios que deseas hacer con el comando git branch nombre-de-la-rama

Cambia a la nueva rama con el comando git checkout nombre-de-la-rama

Haz tus cambios y añádelos al área de preparación con git add .

Haz un commit de tus cambios con el comando git commit -m "mensaje del commit"

Empuja tus cambios a tu repositorio forked en GitHub con git push origin nombre-de-la-rama

Vuelve a GitHub y en la página de tu repositorio forked, haz click en el botón "New pull request"

Selecciona la rama que acabas de empujar como la rama base y la rama master del repositorio original como la rama de comparación.

Revisa los cambios y escribe una descripción detallada de los cambios propuestos.

Haz click en "Create pull request"

El propietario del repositorio original recibirá una notificación de tu pull request y podrá revisar y aprobar o rechazar tus cambios.

Es importante mencionar que el repositorio original debe tener habilitado las opciones de pull request para que puedas hacer una petición.

Revertir un commit

Para revertir un commit en git, puedes usar el comando git revert <hash del commit>. Este comando creará un nuevo commit que anulará los cambios realizados en el commit especificado.

Otra opción es utilizar el comando git reset <hash del commit> seguido de git push -f <nombre del repositorio remoto> <nombre de la rama> para eliminar completamente el commit. Sin embargo, tenga en cuenta que esta acción eliminará permanentemente el commit y todos los cambios realizados en él, por lo que debe ser utilizado con precaución.

Por último, también puedes utilizar el comando git revert -m 1 <hash del commit> si deseas revertir un merge.

Es importante tener en cuenta que en cualquiera de estos casos, si ya has compartido el repositorio con otros usuarios, debes asegurarte de que todos ellos actualicen su copia local antes de continuar trabajando.

Revertir un merge

Para revertir un merge en Git, puedes usar el comando "git revert". Este comando crea un nuevo commit que deshace los cambios del commit especificado. Por ejemplo, si quieres revertir el merge de una rama llamada "feature" en tu rama actual, puedes ejecutar el siguiente comando:

$ git revert -m 1 <hash del merge>

Donde -m 1 indica que se deshaga el merge y el hash del merge es el identificador único del merge que deseas revertir.

Otra forma de revertir un merge es utilizando el comando "git reset". Este comando deshace los cambios de un commit específico y los trae de nuevo al estado del árbol de commits. Por ejemplo, si quieres revertir el merge de una rama llamada "feature" en tu rama actual, puedes ejecutar el siguiente comando:

$ git reset --hard <hash del merge>

Donde --hard indica que se eliminen los cambios realizados en el merge y el hash del merge es el identificador único del merge que deseas revertir.

Cabe mencionar que ambos comandos generan un nuevo commit con los cambios revertidos y si ya se han compartido los cambios a otras personas es recomendable hacer un force push o un push con --force-with-lease

Para revertir un merge, hay varias formas de hacerlo, pero una de las formas más comunes es usando el comando git revert. Este comando crea un nuevo commit que deshace los cambios del commit especificado.

Por ejemplo, si quieres revertir el último merge que se hizo, primero debes identificar el identificador único (SHA) del commit del merge. Puedes hacerlo ejecutando el comando git log y buscando el commit del merge. Una vez que tienes el SHA, puedes ejecutar el comando git revert <SHA>, donde <SHA> es el identificador del commit del merge.

Ejemplo:

$ git log
commit abcdef1234567890
Merge: 123456 789012
Author: John Doe <johndoe@example.com>
Date: Mon Jan 1 12:00:00 2020 -0600

Merge branch 'new-feature' into 'master'

commit 123456abcdef67890
Author: Jane Doe <janedoe@example.com>
Date: Sun Dec 31 12:00:00 2019 -0600

Add new feature

$ git revert abcdef1234567890

Es importante mencionar que al revertir un merge, git crea un nuevo commit que deshace los cambios del merge, pero no elimina el merge de la historia del repositorio.

Otra forma de revertir un merge es utilizando el comando git reset, especificando el commit anterior al merge.

$ git reset --hard commit_anterior_al_merge

Con esto se elimina el merge de la historia, pero también se pierden los cambios realizados en ese merge.

Cabe mencionar que ambas opciones tienen sus pros y contras, por lo que es importante evaluar cuál es la mejor opción en función del escenario específico.

Revertir un pull request

Para revertir un pull request, hay varios pasos que puedes seguir:

Haz un fork del repositorio original en tu cuenta de GitHub.
Clona el fork a tu computadora local.
Utiliza el comando git log para encontrar el commit específico que quieres revertir.
Utiliza el comando git revert seguido del hash del commit. Este comando creará un nuevo commit que revertirá los cambios del commit específico.
Utiliza git push para subir los cambios al fork en tu cuenta de GitHub.
Crea un pull request desde tu fork al repositorio original. Incluye una descripción detallada de los cambios que estás proponiendo y la razón por la cual estás revertiendo el commit original.
Ten en cuenta que revertir un pull request es diferente a rechazarlo. Revertir significa crear un nuevo commit que deshace los cambios del commit original, mientras que rechazar significa simplemente descartar los cambios y no agregarlos al repositorio.

También es importante mencionar que si el pull request ya fue mergeado en el repositorio principal, la forma correcta de revertirlo seria creando un nuevo commit con los cambios contrarios y luego hacer un nuevo merge o un nuevo pull request con ese commit. Es importante que se comunique a todos los miembros del equipo que se esta revertiendo algo y se explique las razones del porque.

Configurar el editor de código por defecto en Git

Para configurar Visual Studio Code como editor de texto predeterminado en git, debes ejecutar el siguiente comando en la terminal:

```
$ git config --global core.editor "code -w"
```

La opción "--global" establece la configuración para todos los repositorios en tu sistema, si deseas configurar solo para un repositorio específico debes eliminar --global.

El comando "code -w" indica a git que utilice Visual Studio Code como editor de texto y que espere a que se cierre el editor antes de continuar con la operación.

Una vez ejecutado el comando, git utilizará Visual Studio Code como editor de texto predeterminado para las operaciones que requieran ingresar un mensaje de commit, crear una nueva rama, etc.

En resumen, para configurar Visual Studio Code como editor de texto predeterminado en git debes ejecutar el comando "git config --global core.editor "code -w"" para establecer la configuración para todos los repositorios en tu sistema.

Para IntelliJ IDEA

```
$ git config --global core.editor "idea -w"
```

Para Gedit en Linux

```
$ git config --global core.editor "gedit -w"
```

Para Visual Code

```
$ git config --global core.editor "code -w"
```

Para ver la configuración realizada ejecutar el siguiente comando

```
$ git config --global -e
```

Algunos ejemplos de configuraciones básicas de git son:

Establecer tu nombre de usuario:

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

Establecer tu dirección de correo electrónico:

```
git config --global user.email "tu_email@ejemplo.com"
```

Habilitar color en la salida de comandos:

```
git config --global color.ui true
```

Establecer un editor de texto predeterminado:

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

Habilitar la firma GPG para commits:

```
git config --global user.signingkey "clave_publica_GPG"
```

Establecer un mecanismo de merge predeterminado:

```
git config --global merge.tool "kdiff3"
```

Establecer un mecanismo de diff predeterminado:

```
git config --global diff.tool "meld"
```

Estos son solo algunos ejemplos de configuraciones básicas que se pueden hacer en Git. Hay muchas otras opciones y configuraciones avanzadas disponibles para personalizar tu experiencia con git.

En resumen, las configuraciones básicas de git incluyen establecer tu nombre de usuario y dirección de correo electrónico, habilitar el color en la salida de comandos, establecer un editor de texto predeterminado, habilitar la firma GPG para commits, establecer un mecanismo de merge y diff predeterminado, entre otras.

Para ver la configuración realizada ejecutar el siguiente comando

```
$ git config --global -e
```

Directorio y archivo de la configuración de Git

La configuración de git se encuentra en archivos de texto plano y están ubicados en diferentes lugares dependiendo del nivel de configuración.

Configuración global: La configuración global se encuentra en el archivo .gitconfig en el directorio home del usuario. Puedes verlo y editarlo utilizando el comando:

```
nano ~/.gitconfig
```

Configuración del repositorio: La configuración del repositorio se encuentra en el archivo .git/config dentro del directorio del repositorio. Puedes verlo y editarlo utilizando el comando:

```
nano .git/config
```

Configuración de la sesión: La configuración de la sesión se encuentra en la memoria, y se utiliza solo durante la sesión actual de git. Puede ser visto utilizando el comando

```
git config --list
```

En resumen, la configuración de git se encuentra en archivos de texto plano y se dividen en tres niveles: configuración global, configuración del repositorio y configuración de la sesión. Puedes ver y editar estas configuraciones utilizando los comandos mencionados anteriormente.

Trabajando con otro programadores

¿Qué pasa si ejecuto el comando git init en un repositorio inicializado previamente?

Si ejecutas el comando git init en un repositorio que ya ha sido inicializado previamente con Git, no sucederá nada. El comando git init se utiliza para inicializar un nuevo repositorio Git en un directorio que aún no está siendo controlado por Git.

Al ejecutar git init en un directorio que ya es un repositorio Git, Git no sobrescribirá ningún archivo o información existente. En cambio, simplemente mostrará un mensaje de error que indica que ya hay un repositorio Git presente en el directorio.

Es importante tener en cuenta que si ya tienes un repositorio Git en un directorio, no necesitas ejecutar el comando git init de nuevo. En su lugar, puedes utilizar los comandos Git regulares para trabajar con el repositorio existente, como git add, git commit, git push, etc.

Comandos Git para trabajar solamente de forma local:

- git init: inicializa un nuevo repositorio Git local.
- git add: agrega archivos al área de preparación (staging area) para prepararlos para el commit.
- git commit: confirma los cambios en el repositorio local.
- git status: muestra el estado actual del repositorio, incluyendo los archivos modificados y los archivos preparados para el commit.
- git log: muestra un registro de los commits realizados en el repositorio.

Comandos Git para trabajar con un repositorio remoto:

- git clone: clona un repositorio remoto en una ubicación local.
- git push: envía los cambios confirmados localmente al repositorio remoto.
- git pull: obtiene los cambios más recientes del repositorio remoto y los fusiona con los cambios locales.

Comandos Git para trabajar con un repositorio que es de otro programador del cual se ha realizado un fork:

- git remote: muestra los repositorios remotos configurados en el repositorio local.
- git fetch: obtiene los cambios más recientes del repositorio remoto pero no los fusiona con los cambios locales.
- git merge: fusiona los cambios obtenidos del repositorio remoto con los cambios locales.
- git rebase: aplica los cambios obtenidos del repositorio remoto al inicio de la línea de tiempo de los cambios locales.

Es importante tener en cuenta que estos grupos no son mutuamente excluyentes, y algunos comandos pueden ser utilizados en más de una forma de trabajo con el repositorio. Por ejemplo, git log puede ser utilizado tanto en un repositorio local como en un repositorio remoto para ver un registro de los cambios realizados en el repositorio.

Comandos de Git que son útiles en varias situaciones:

- git branch: muestra una lista de ramas (branches) en el repositorio y permite crear nuevas ramas.
- git checkout: cambia entre ramas o versiones anteriores del repositorio.
- git stash: guarda temporalmente los cambios no confirmados para trabajar en otra cosa.
- git tag: crea una etiqueta (tag) en un commit específico para identificarlo de manera fácil.

Es importante destacar que Git tiene muchos más comandos que los mencionados anteriormente, y cada uno de ellos tiene muchas opciones y variaciones. Por lo tanto, lo mejor es leer la documentación oficial de Git o buscar tutoriales específicos para aprender más sobre cómo utilizar Git en diferentes situaciones.