

## Git

Git es un sistema de control de versiones distribuido ampliamente utilizado para el seguimiento de cambios en proyectos de software y colaboración en desarrollo de software. Fue creado por Linus Torvalds en 2005 y desde entonces se ha convertido en una herramienta fundamental en el mundo del desarrollo de software.

Las principales características de Git incluyen:

**Control de Versiones:** Git permite realizar un seguimiento de los cambios en el código fuente y otros archivos a lo largo del tiempo. Cada cambio se registra con un mensaje descriptivo, lo que facilita la comprensión de por qué se realizó un cambio en particular.

**Distribuido:** Git es un sistema de control de versiones distribuido, lo que significa que cada desarrollador tiene una copia completa del repositorio en su máquina local. Esto permite que los desarrolladores trabajen de forma independiente y realicen cambios sin necesidad de estar siempre conectados a un servidor central.

**Ramas (Branches):** Git permite crear ramas separadas en el repositorio para trabajar en características o correcciones de errores específicos sin afectar la rama principal (generalmente llamada "master" o "main"). Esto facilita la colaboración en equipo y la gestión de versiones paralelas del software.

**Fusiones (Merges):** Cuando se completa una característica o corrección de errores en una rama, es posible fusionar esos cambios de regreso a la rama principal utilizando operaciones de fusión. Esto permite mantener un historial limpio y ordenado del proyecto.

**Historial Detallado:** Git almacena un historial completo de cambios, lo que permite rastrear quién realizó un cambio, cuándo se realizó y por qué se realizó. Esto es útil para la auditoría y la resolución de problemas.

**Gestión de Conflictos:** Cuando dos o más personas intentan realizar cambios en la misma parte de un archivo, Git puede detectar y gestionar conflictos, lo que permite a los desarrolladores resolverlos de manera manual.

**Integración con Plataformas de Colaboración:** Git se utiliza comúnmente en combinación con plataformas de colaboración en línea como GitHub, GitLab y Bitbucket, que proporcionan servicios adicionales como seguimiento de problemas, solicitudes de extracción y una forma centralizada de alojar repositorios Git.

## Comandos

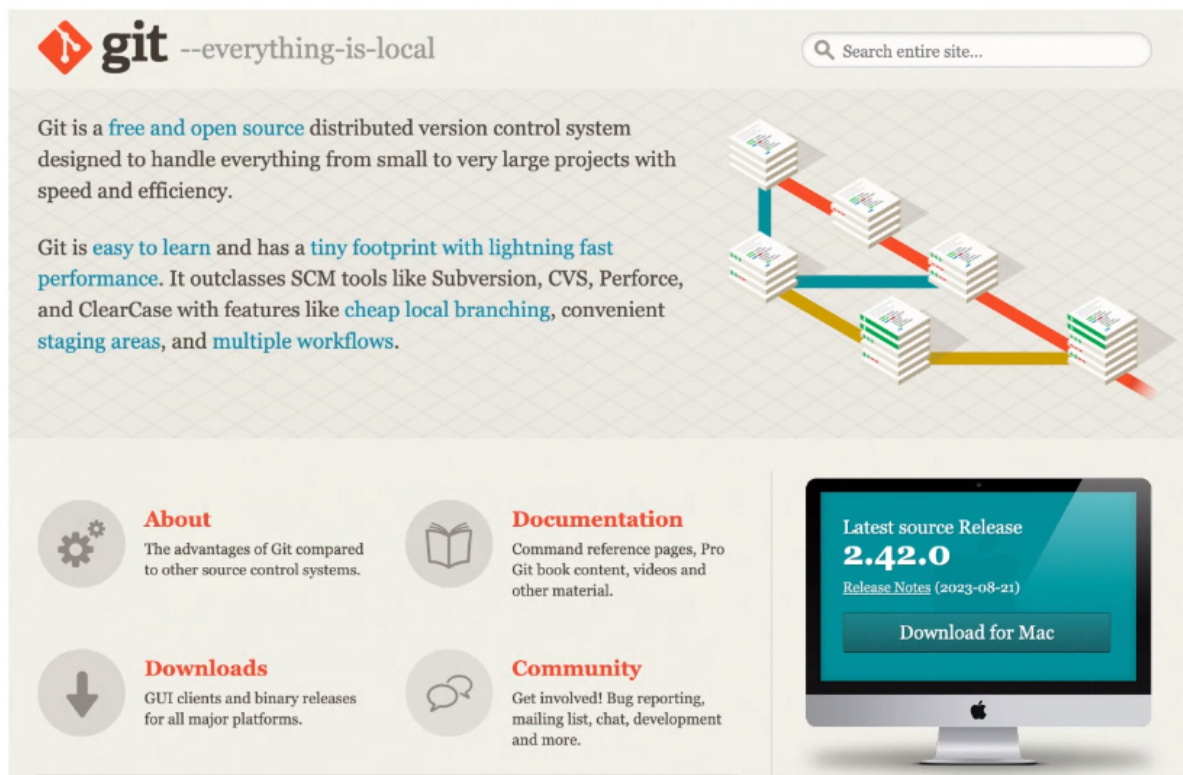
Git es un sistema de control de versiones ampliamente utilizado que te permite realizar un seguimiento de los cambios en tus proyectos de software. Aquí tienes una lista de comandos Git básicos que te ayudarán a trabajar con Git:

1. **git init:** Inicializa un nuevo repositorio Git en un directorio vacío o existente.
2. **git clone [URL]:** Clona un repositorio Git existente desde una URL remota a tu sistema local.
3. **git add [archivo(s)]:** Agrega cambios al área de preparación (staging) para que puedan incluirse en el próximo commit.
4. **git status:** Muestra el estado actual de tu repositorio, incluyendo archivos sin seguimiento, cambios pendientes de commit y archivos en el área de preparación.
5. **git diff:** Muestra las diferencias entre el directorio de trabajo y el área de preparación.
6. **git commit -m "[Mensaje del commit]":** Crea un nuevo commit con los cambios en el área de preparación y un mensaje descriptivo.
7. **git branch:** Muestra una lista de todas las ramas en el repositorio. La rama actual se resalta con un asterisco.
8. **git checkout [nombre de la rama]:** Cambia a otra rama. Puedes crear una nueva rama usando -b seguido del nombre de la nueva rama.
9. **git merge [nombre de la rama]:** Fusiona los cambios de una rama en la rama actual.
10. **git pull:** Recupera los cambios desde el repositorio remoto y los fusiona en la rama actual.
11. **git push:** Envía los cambios locales al repositorio remoto.
12. **git log:** Muestra un registro de commits en la rama actual.
13. **git remote -v:** Muestra una lista de los repositorios remotos configurados y sus URL.
14. **git fetch:** Recupera información de los repositorios remotos sin fusionar cambios.

15. **git remote add [nombre] [URL]**: Agrega un nuevo repositorio remoto con un nombre específico.
16. **git branch -d [nombre de la rama]**: Elimina una rama local.

Estos son algunos de los comandos Git más comunes. Puedes obtener más información sobre cada comando y sus opciones ejecutando `git [comando] --help` en tu terminal o consultando la documentación oficial de Git.

<https://git-scm.com/>



The screenshot shows the Git website homepage. At the top left is the Git logo (a red diamond with a white 'G') followed by the text "git --everything-is-local". To the right is a search bar with the placeholder text "Search entire site...". Below the header, there is a paragraph describing Git as a "free and open source distributed version control system" designed for projects of all sizes. To the right of this text is a diagram showing a network of stacks of papers connected by colored lines (red, blue, yellow), representing a distributed version control system. Below the main text, there are four sections with icons and titles: "About" (gears icon), "Documentation" (book icon), "Downloads" (downward arrow icon), and "Community" (speech bubbles icon). Each section has a brief description. On the right side, there is a large monitor displaying the "Latest source Release 2.42.0" with a "Download for Mac" button and a link to "Release Notes (2023-08-21)".

**git** --everything-is-local

Search entire site...

Git is a **free and open source** distributed version control system designed to handle everything from small to very large projects with speed and efficiency.

Git is **easy to learn** and has a **tiny footprint with lightning fast performance**. It outclasses SCM tools like Subversion, CVS, Perforce, and ClearCase with features like **cheap local branching**, convenient **staging areas**, and **multiple workflows**.

**About**  
The advantages of Git compared to other source control systems.

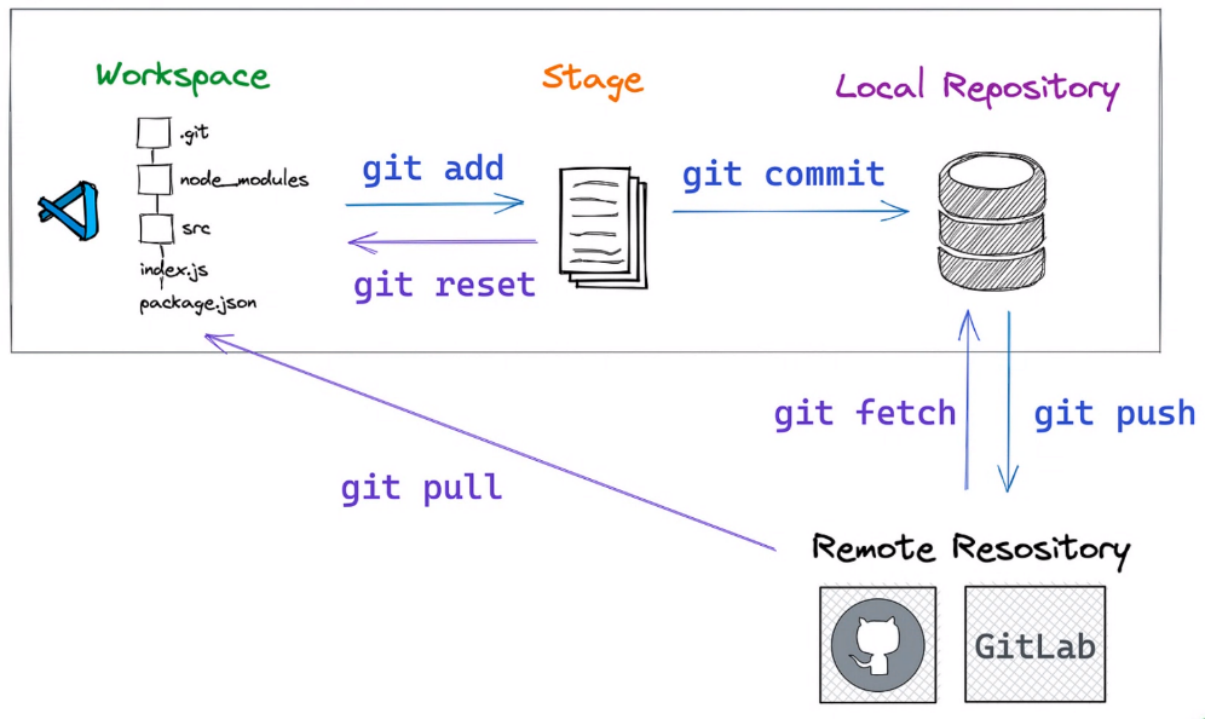
**Documentation**  
Command reference pages, Pro Git book content, videos and other material.

**Downloads**  
GUI clients and binary releases for all major platforms.

**Community**  
Get involved! Bug reporting, mailing list, chat, development and more.

Latest source Release  
**2.42.0**  
[Release Notes \(2023-08-21\)](#)  
Download for Mac

## Local



## GitHub → Trabaja en conjunto con Git

GitHub es una plataforma en línea que se utiliza para alojar, gestionar y colaborar en proyectos de desarrollo de software utilizando el sistema de control de versiones Git. Fue fundada en 2008 por Tom Preston-Werner, Chris Wanstrath y PJ Hyett y se ha convertido en una de las plataformas más populares para la colaboración y el desarrollo de software a nivel mundial. En 2021, GitHub fue adquirida por Microsoft.

Aquí hay algunas características y funciones clave de GitHub:

**Alojamiento de Repositorios:** GitHub permite a los desarrolladores alojar repositorios Git en línea de forma gratuita o mediante planes de pago. Los repositorios pueden ser públicos (accesibles para cualquiera) o privados (accesibles solo para colaboradores autorizados).

**Seguimiento de Problemas:** GitHub proporciona un sistema de seguimiento de problemas (issue tracking) que permite a los equipos de desarrollo gestionar y realizar un seguimiento de problemas, solicitudes de características y correcciones de errores. Los usuarios pueden comentar, etiquetar y asignar problemas a otros miembros del equipo.

**Solicitudes de Extracción (Pull Requests):** Las solicitudes de extracción son una característica clave de GitHub que permite a los desarrolladores proponer cambios en un repositorio y solicitar que esos cambios se fusionen en la rama principal del proyecto. Esto facilita la revisión de código y la colaboración entre miembros del equipo.

**Colaboración en Equipo:** GitHub es ampliamente utilizado para la colaboración en proyectos de código abierto y privados. Los equipos pueden trabajar juntos en un proyecto, realizar comentarios en el código, realizar pruebas y llevar un registro de todas las contribuciones.

**Integración con Herramientas de Desarrollo:** GitHub se integra con una amplia variedad de herramientas y servicios de desarrollo, como servicios de integración continua (CI/CD), herramientas de automatización, IDEs (Entornos de Desarrollo Integrados) y más. Esto facilita la automatización de tareas y flujos de trabajo de desarrollo.

**Gestión de Versiones:** GitHub almacena un historial completo de cambios en un repositorio, lo que permite rastrear quién realizó cambios, cuándo se realizaron y por qué. Los desarrolladores pueden realizar un seguimiento de versiones anteriores y comparar diferencias entre versiones.

**Documentación y Wiki:** GitHub permite a los proyectos mantener documentación detallada y wikis para proporcionar información adicional sobre el proyecto, instrucciones de instalación, guías de contribución y más.

**Seguridad:** GitHub proporciona herramientas de seguridad, como escaneo de vulnerabilidades, para ayudar a los desarrolladores a mantener sus proyectos seguros y actualizados.

## Issues (Problemas):

¿Qué son?: Los "issues" o problemas son entradas que se utilizan para realizar un seguimiento de tareas, problemas, solicitudes de características o cualquier otro elemento que requiera atención en un proyecto de desarrollo de software.

Características clave:

Pueden utilizarse para informar sobre errores (bugs), proponer mejoras o nuevas características.

Los issues suelen contener detalles, comentarios y etiquetas para facilitar la organización y la colaboración.

Se pueden asignar a miembros del equipo o colaboradores específicos.

Los issues también pueden estar vinculados a pull requests para llevar un registro de los cambios relacionados con un problema específico.

## Pull Request (Solicitud de Extracción):

¿Qué es?: Una pull request es una propuesta que un colaborador hace para incorporar cambios en un repositorio. Por lo general, se utiliza cuando alguien ha realizado cambios en su propio "fork" (copia) del repositorio original y desea que esos cambios se fusionen en el repositorio principal.

Características clave:

Los pull requests incluyen una descripción de los cambios realizados y pueden mostrar las diferencias (deltas) entre las ramas.

Los revisores pueden comentar, revisar y aprobar los cambios antes de que se fusionen.

Las pull requests facilitan la revisión de código y la colaboración entre miembros del equipo.

Una vez aprobadas, las modificaciones se fusionan en la rama principal del repositorio.

## Wiki:

¿Qué es?: Un wiki es una sección en la que se puede crear y mantener documentación relacionada con un proyecto de software en GitHub. Los wikis son útiles para proporcionar información adicional, instrucciones de uso, guías de contribución y más.

Características clave:

Los wikis son una forma de documentar el proyecto de manera colaborativa y accesible.

Los miembros del equipo o colaboradores pueden contribuir y editar la documentación del wiki.

Se pueden crear páginas y enlazarlas para organizar la información de manera efectiva.

## Fork (Bifurcación):

¿Qué es?: Un fork es una copia de un repositorio de GitHub que se crea bajo la cuenta de otro usuario. Los forks se utilizan comúnmente cuando alguien desea contribuir a un proyecto de código abierto o realizar cambios en un proyecto existente sin tener permisos de escritura en el repositorio original.


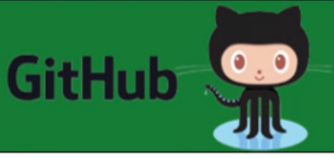
Características clave:

Los forks permiten a los colaboradores realizar cambios en su propia copia del proyecto sin afectar el repositorio original.

Después de realizar cambios en un fork, el colaborador puede crear una pull request para proponer la fusión de esos cambios en el repositorio original.

Los propietarios del repositorio original pueden revisar y aprobar pull requests de forks para incorporar contribuciones externas.

# Diferencias entre Git & GitHub

	
1. Es un software.	1. Es un servicio.
2. Se instala localmente en el sistema.	2. Está alojado en la web.
3. Es una herramienta de línea de comandos.	3. Proporciona una interfaz gráfica.
4. Es una herramienta para gestionar diferentes versiones de ediciones realizadas en archivos en un repositorio de Git.	4. Es un espacio para cargar una copia del repositorio Git.
5. Proporciona funcionalidades como Gestión de Código Fuente y Sistema de Control de Versiones.	5. Ofrece funcionalidades de Git como control de versiones (VCS) y gestión de código fuente, además de agregar algunas de sus propias características.

## Características claves de GitHub



## SVN

SVN es una abreviatura de "Subversion", que es un sistema de control de versiones utilizado en el desarrollo de software y la gestión de proyectos. Un sistema de control de versiones es una herramienta que permite a los desarrolladores rastrear y gestionar los cambios en el código fuente y otros archivos a lo largo del tiempo. SVN es una de las opciones disponibles para realizar esta tarea.

Las características principales de SVN incluyen:

**Control de versiones:** SVN permite a los desarrolladores mantener un registro de todas las versiones anteriores de un archivo o proyecto, lo que facilita la recuperación de versiones anteriores o la comparación entre ellas.

**Colaboración:** Permite que varios desarrolladores trabajen en el mismo proyecto al mismo tiempo, lo que facilita la colaboración en equipo.

**Historial de cambios:** Mantiene un registro detallado de quién hizo qué cambios y cuándo, lo que es útil para el seguimiento y la auditoría.

**Ramificación y fusión:** Permite crear ramas separadas del proyecto para trabajar en nuevas características o correcciones de errores sin afectar la rama principal. Luego, estas ramas pueden fusionarse nuevamente en la rama principal cuando estén listas.

**Etiquetado:** Puedes etiquetar versiones específicas del proyecto para marcar hitos importantes en su desarrollo.

**Seguridad:** SVN ofrece opciones de autenticación y control de acceso para proteger el código fuente y los datos del proyecto.

**Integración con otras herramientas:** Puede integrarse con una variedad de herramientas de desarrollo, como IDEs (entornos de desarrollo integrados) y sistemas de seguimiento de problemas.

## TortoiseSVN

Es una aplicación de software de código abierto que proporciona una interfaz gráfica de usuario (GUI) para el sistema de control de versiones SVN (Subversion). Está diseñada para funcionar en sistemas operativos Windows y facilita la gestión y el control de proyectos de software utilizando SVN sin tener que depender completamente de comandos de línea de texto.

Algunas de las características y funcionalidades clave de TortoiseSVN incluyen:

**Integración con el Explorador de Windows:** Una de las características más destacadas de TortoiseSVN es su integración directa con el Explorador de Windows. Esto significa que puedes realizar operaciones de control de versiones directamente desde el menú contextual (clic derecho) en los archivos y carpetas en tu sistema de archivos.

**Iconos de estado de archivos y carpetas:** TortoiseSVN utiliza iconos superpuestos en los archivos y carpetas para indicar su estado en el repositorio SVN. Esto facilita la identificación de cambios pendientes, archivos modificados, archivos no versionados, etc.

**Visor de registro de cambios:** Permite ver el historial de cambios de un archivo o carpeta, incluyendo quién realizó los cambios, cuándo se hicieron y qué se modificó.

**Comprobación y confirmación de cambios:** TortoiseSVN permite comprobar archivos para ver las diferencias antes de confirmar los cambios en el repositorio SVN.

**Gestión de repositorios:** Puedes crear, clonar, importar y administrar repositorios SVN directamente desde la interfaz de TortoiseSVN.

**Soporte para fusión (merge) y resolución de conflictos:** Facilita la fusión de cambios desde una rama a otra y ofrece herramientas para resolver conflictos de manera visual.

**Etiquetado y ramificación:** Permite crear etiquetas y ramas de manera sencilla para marcar versiones importantes o trabajar en paralelo en el desarrollo.

**Soporte para propiedades y propiedades avanzadas:** Puedes configurar propiedades de SVN directamente a través de la interfaz de TortoiseSVN.

## GitFlow

GitFlow es un modelo de flujo de trabajo basado en Git que se utiliza comúnmente en proyectos de desarrollo de software para gestionar y organizar las ramas de desarrollo, las versiones y las colaboraciones entre equipos. Fue propuesto por Vincent Driessen y se ha convertido en un enfoque popular para la gestión de ramas en Git. El flujo de trabajo de GitFlow se basa en la idea de tener dos ramas principales: **master** y **develop**, junto con varias ramas de características y versiones intermedias. Aquí está un resumen de las principales ramas y cómo se utilizan en el modelo de GitFlow:

1. **master**: Esta rama representa la versión estable y de producción del software. Los commits en **master** deberían reflejar el estado actual de la aplicación que está lista para ser lanzada. Los commits en **master** generalmente se etiquetan con números de versión.
2. **develop**: La rama **develop** es la rama principal de desarrollo. Es donde se integran todas las características y cambios antes de ser probados en la rama **master**. Esta rama puede contener código en desarrollo y no necesariamente estable.
3. **feature branches (ramas de características)**: Cada nueva característica o mejora se desarrolla en su propia rama separada, generalmente basada en **develop**. Estas ramas se crean y se fusionan de vuelta en **develop** una vez que la característica está completa.
4. **release branches (ramas de versión)**: Cuando se está preparando una nueva versión del software, se crea una rama de versión a partir de **develop**. En esta rama, se realizan las pruebas finales, correcciones de errores y ajustes para la versión. Una vez que se considera estable, se fusiona tanto en **develop** como en **master**, y se etiqueta con la versión.
5. **hotfix branches (ramas de corrección)**: Si se descubren errores críticos en la rama **master**, se crea una rama de corrección a partir de **master**. Se corrige el error en esta rama y luego se fusiona tanto en **master** como en **develop**. Esto garantiza que los errores críticos se solucionen tanto en la versión actual como en el desarrollo futuro.

El flujo de trabajo de GitFlow proporciona una estructura organizada para el desarrollo de software en equipo y facilita la gestión de versiones y correcciones de errores. Sin embargo, es importante recordar que GitFlow es solo uno de los muchos modelos de flujo de trabajo de Git disponibles, y puedes personalizarlo según las necesidades de tu proyecto.

## Funcionamiento Gitflow





GitFlow es un modelo de flujo de trabajo que se basa en Git y se utiliza para gestionar ramas y colaboración en proyectos de desarrollo de software. Aunque GitFlow en sí mismo no es una herramienta de línea de comandos, se puede implementar utilizando comandos Git estándar. A continuación, te mostraré cómo usar algunos comandos Git comunes en el contexto de GitFlow:

1. **Inicializar un repositorio GitFlow:** Antes de comenzar a trabajar con GitFlow, es necesario inicializarlo en tu repositorio Git existente. Puedes hacerlo utilizando la herramienta `git-flow` o manualmente con estos comandos:  

```
# Instala GitFlow si aún no lo tienes instalado git clone --recursive  
https://github.com/nvie/gitflow.git # Inicializa GitFlow en tu repositorio git  
flow init
```
2. **Crear una nueva rama de función (feature):** Para desarrollar una nueva característica, puedes crear una rama de función usando el siguiente comando:  

```
git flow feature start nombre-de-la-caracteristica
```

Esto creará una nueva rama de función basada en la rama `develop` y te llevará a esa rama.
3. **Terminar una rama de función:** Una vez que hayas completado el desarrollo de una característica, puedes finalizar la rama de función utilizando:  

```
git flow feature finish nombre-de-la-caracteristica
```

Esto fusionará la rama de función en la rama `develop` y eliminará la rama de función.
4. **Iniciar una rama de versión (release):** Cuando estés listo para lanzar una nueva versión, puedes crear una rama de versión con este comando:  

```
git flow release start nombre-de-la-version
```

Esto creará una rama de versión basada en la rama `develop`.
5. **Finalizar una rama de versión (release):** Una vez que hayas realizado las pruebas finales y estés listo para lanzar la versión, puedes finalizar la rama de versión utilizando:  

```
git flow release finish nombre-de-la-version
```

Esto fusionará la rama de versión en `master` y `develop`, etiquetará la versión y eliminará la rama de versión.
6. **Iniciar una rama de hotfix:** Cuando surja un error crítico en producción, puedes crear una rama de hotfix para solucionarlo:  

```
git flow hotfix start nombre-del-hotfix
```

Esto creará una rama de hotfix basada en `master`.
7. **Finalizar una rama de hotfix:** Una vez que hayas corregido el error en la rama de hotfix, puedes finalizarla utilizando:  

```
git flow hotfix finish nombre-del-hotfix
```

Esto fusionará la rama de hotfix en `master` y `develop`, etiquetará la versión y eliminará la rama de hotfix.

Estos son algunos de los comandos GitFlow más comunes que puedes utilizar para gestionar ramas en tu proyecto. Ten en cuenta que debes tener GitFlow instalado y configurado en tu sistema para utilizar estos comandos.



## Hotfix

Un hotfix (corrección rápida) es una rama de Git utilizada para abordar y solucionar problemas críticos o errores en la rama principal de producción (generalmente `master`) de un proyecto de software. Los hotfixes se utilizan cuando se descubre un error que debe solucionarse de manera inmediata, incluso antes de que se realice la próxima versión programada. Aquí tienes los pasos típicos para crear y aplicar un hotfix:

1. **Crear una rama de hotfix:** Desde la rama `master`, crea una nueva rama para tu hotfix. Puedes hacerlo utilizando el siguiente comando:  
bashCopy code  
`git checkout -b hotfix/nombre-del-hotfix`  
Esto creará una nueva rama llamada "hotfix/nombre-del-hotfix" basada en la rama `master`.
2. **Hacer las correcciones:** Realiza las correcciones necesarias en la nueva rama de hotfix. Esto implica solucionar el error crítico o realizar cualquier modificación necesaria en el código.
3. **Comprometer y fusionar el hotfix:** Una vez que hayas hecho las correcciones y probado que funcionan correctamente, compromete los cambios en la rama de hotfix y luego fúndela en la rama `master`. Esto se hace generalmente utilizando los siguientes comandos:  
bashCopy code  
`git checkout master git merge --no-ff hotfix/nombre-del-hotfix`  
La opción `--no-ff` asegura que se cree un commit de fusión incluso si no hay cambios en `master` desde que se creó la rama de hotfix.
4. **Fusionar en la rama de desarrollo:** Si también deseas aplicar los cambios del hotfix a la rama de desarrollo (`develop` en el modelo de GitFlow), puedes hacerlo utilizando el siguiente comando:  
bashCopy code  
`git checkout develop git merge --no-ff hotfix/nombre-del-hotfix`  
Esto asegura que los cambios se reflejen en futuras versiones de tu software.
5. **Eliminar la rama de hotfix:** Después de que el hotfix se haya fusionado con éxito en `master` y, opcionalmente, en `develop`, puedes eliminar la rama de hotfix si ya no es necesaria:  
bashCopy code  
`git branch -d hotfix/nombre-del-hotfix`  
Si se necesitara la corrección en el futuro, aún puedes acceder a ella a través de la historia de Git.

Los hotfixes son una forma efectiva de abordar problemas críticos y errores en una rama de producción sin afectar el desarrollo continuo en las ramas de desarrollo.

## Branching

El "branching" (ramificación en español) es una práctica fundamental en sistemas de control de versiones como Git. Se refiere a la creación de líneas de desarrollo separadas, o "ramas," que divergen del tronco principal de desarrollo (a menudo llamado "master" o "main"). Cada rama representa una línea de desarrollo independiente con su propia historia de cambios y commits.

El uso de ramas en Git y otros sistemas de control de versiones proporciona varios beneficios clave:

1. **Desarrollo paralelo:** Permite a múltiples miembros del equipo trabajar en diferentes características o tareas de manera simultánea sin interferir en el trabajo de los demás. Cada tarea puede tener su propia rama.
2. **Experimentación:** Las ramas brindan un entorno seguro para experimentar y probar nuevas ideas sin afectar la rama principal de desarrollo. Si un experimento no funciona, simplemente puedes descartar la rama.
3. **Gestión de versiones:** Las ramas facilitan la gestión de versiones y la liberación de software. Puedes crear ramas de versión para trabajar en una próxima versión mientras mantienes la rama principal estable.
4. **Corrección de errores:** Las correcciones de errores críticos pueden desarrollarse en una rama separada (conocida como "hotfix") y luego fusionarse rápidamente en la rama principal sin perturbar el flujo de trabajo normal.

5. **Colaboración:** Facilita la colaboración entre equipos o desarrolladores individuales que trabajan en diferentes partes de un proyecto.

El "branching" es una práctica esencial en GitFlow, un modelo de flujo de trabajo basado en Git que organiza las ramas en una estructura específica para gestionar el desarrollo de software de manera eficiente. Sin embargo, Git permite una flexibilidad significativa en la creación y gestión de ramas, lo que te permite adaptar tu flujo de trabajo a las necesidades específicas de tu proyecto. En general, el "branching" es una estrategia poderosa para mantener el control de versiones y gestionar el desarrollo de software de manera efectiva.