Git

**Manual GitFlow**

Versión 1.0



# Tabla de contenido

**Escribir el título del capítulo (nivel 1)1**

Escribir el título del capítulo (nivel 2)2

Escribir el título del capítulo (nivel 3)3

**Escribir el título del capítulo (nivel 1)4**

Escribir el título del capítulo (nivel 2)5

Escribir el título del capítulo (nivel 3)6

PBX: 746-0281

www.gtd.com.co

info@gtd.com.co

Cl. 23 B Bis 81 A 44

Bogotá, Colombia

Contenido

[Tabla de contenido 1](#_Toc66354275)

[1. Objetivo 4](#_Toc66354276)

[2. Responsables 4](#_Toc66354277)

[3. Gitflow 4](#_Toc66354278)

[3.1 Ramas principales 5](#_Toc66354279)

[3.1.1 Rama master 5](#_Toc66354280)

[3.1.2 Rama develop 5](#_Toc66354281)

[3.2 Ramas auxiliares 5](#_Toc66354282)

[3.2.1 Ramas Feature 6](#_Toc66354283)

[3.2.1.1 Crear una rama feature 6](#_Toc66354284)

[3.2.1.2 Finalizar una rama feature 6](#_Toc66354285)

[3.2.2 Rama release 7](#_Toc66354286)

[3.2.2.1 Crear rama release 7](#_Toc66354287)

[3.2.2.2 Finalizar una rama release 8](#_Toc66354288)

[3.2.3 Ramas hotfix 8](#_Toc66354289)

[3.2.3.1 Crear una rama hotfix 9](#_Toc66354290)

[3.2.3.2 Finalizar una rama hotfix 9](#_Toc66354291)

[4. Herramienta git flow 10](#_Toc66354292)

[4.1 Cómo utilizar la herramienta 10](#_Toc66354293)

[Inicialización de metodología git flow en el repositorio 10](#_Toc66354294)

[4.1.1 Gestionar features 10](#_Toc66354295)

[4.1.2 Gestionar releases 11](#_Toc66354296)

[4.1.3 Gestionar hotfixes 11](#_Toc66354297)

[4.2 Resumen herramienta git flow 11](#_Toc66354298)

[4.3 Ventajas 11](#_Toc66354299)

[4.4 Comandos usuales 12](#_Toc66354300)

[4.4.1 git clone 12](#_Toc66354301)

[4.4.2 git pull 12](#_Toc66354302)

[4.4.3 git status 12](#_Toc66354303)

[4.4.4 git add 12](#_Toc66354304)

[4.4.5 git branch 13](#_Toc66354305)

[4.4.6 git commit 13](#_Toc66354306)

[4.4.7 git push 13](#_Toc66354307)

[4.4.8 git checkout 13](#_Toc66354308)

[4.4.9 git stash 14](#_Toc66354309)

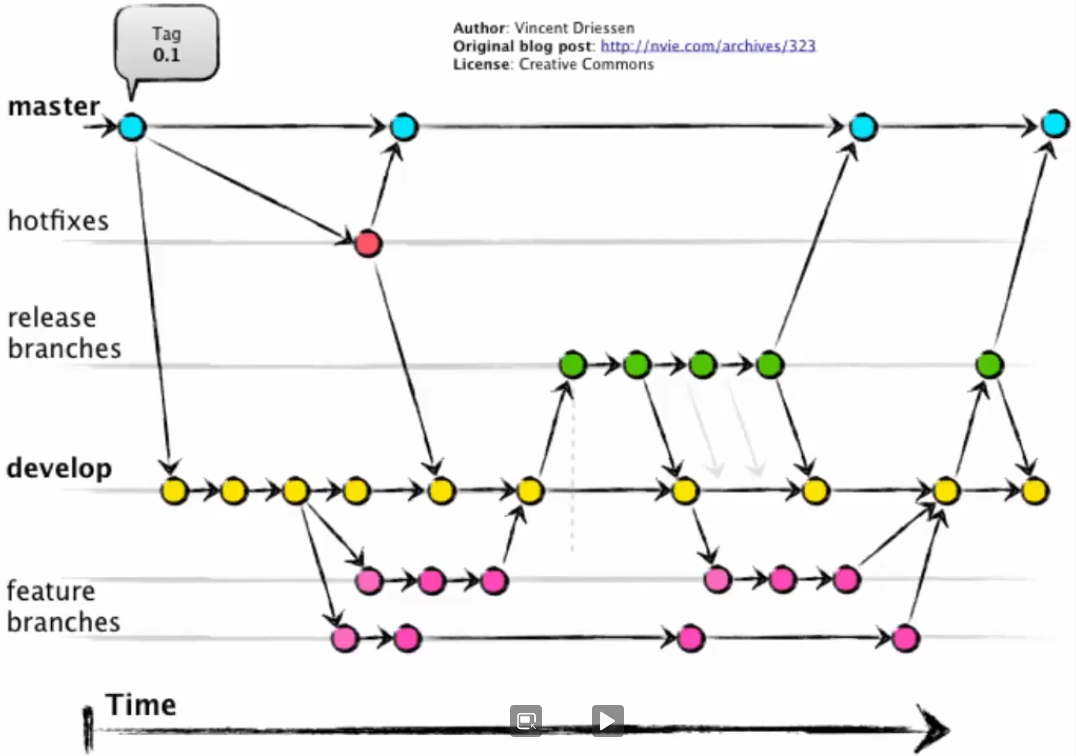
# Objetivo

El objetivo principal de este documento es establecer los lineamientos para trabajar con el control de versiones git y el flujo de trabajo Gitflow.

# Responsables

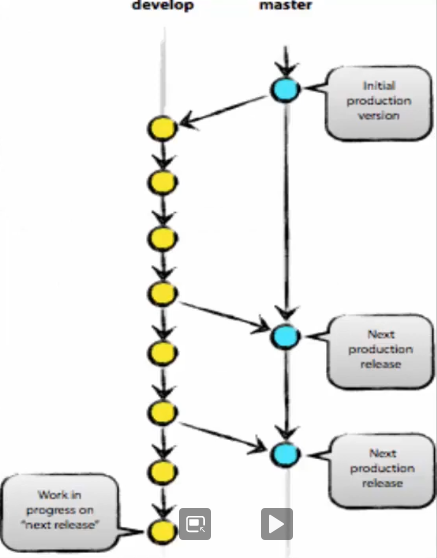
* + Luis Carlos Infante Palacios – Desarrollador de Software
  + Julián Andrés Gómez – Líder Técnico

# Gitflow

Gitflow es un modelo de administración de ramas que nos permite clasificar de forma ordenada las diferentes características y etapas de implementación de software dentro del gestor de versiones git.

Se clasifica en dos tipos de ramas que son ramas principales y ramas auxiliares.

## Ramas principales

En Gitflow se manejan 2 ramas principales o permanentes que son ***master*** y ***develop***:

### Rama master

Esta rama contiene las versiones estables del proyecto. En esta rama cualquier *commit* realizado debe estar preparado para incluir a *producción*. Esto se debe a que cada vez que se incorpora código a esta rama es porque se tiene una nueva versión terminada del proyecto.

### Rama develop

Esta rama es creada a partir de la rama ***master*** y contiene las características estables de la siguiente versión planificada del proyecto.

## Ramas auxiliares

Las ramas auxiliares o temporales solo están activas mientras cumplen su función para luego ser fusionadas a una rama principal. Las ramas auxiliares son:

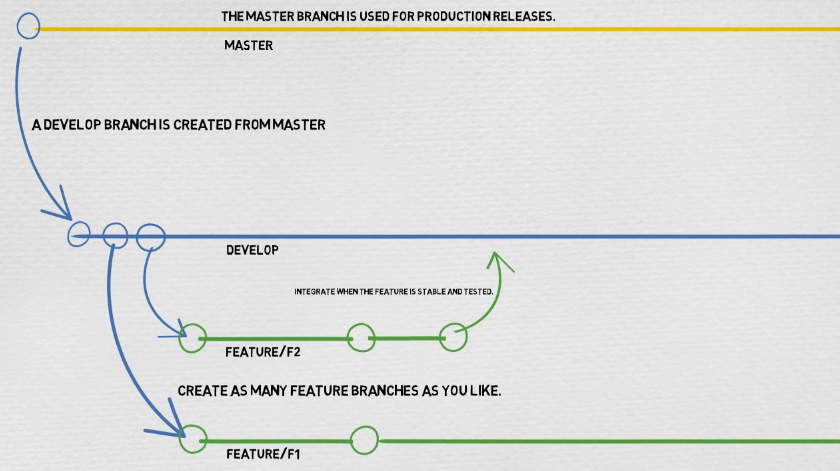
* feature
* release
* hotfix

### Ramas Feature

Las ramas Feature o Topic se originan de la rama ***develop*** y se incorporan siempre a la rama ***develop***.

Se pueden crear tantas ramas features como se requieran y cada una de estas ramas almacenan código de desarrollo con nuevas características. Típicamente existen solamente en los repositorios locales de los desarrolladores y no en el repositorio origen. Una vez terminado y probado su desarrollo, se incorporarán nuevamente a la rama ***develop***, que contendrá la última versión de código en desarrollo.

*Convención de nombres:* puede ser cualquiera que no sea **master**, **develop**, **hotfix-\*** o **release-\***.



Cualquier cambio encontrado en la rama ***develop*** debe ser fusionado (merged) en las ramas ***feature*.**

### Crear una rama feature

#### $ git checkout -b feature/myfeature develop

Switched to a **new** branch "feature/myfeature"

### Finalizar una rama feature

$ git checkout develop

Switched to branch 'develop'

$ git merge --no-ff feature/myfeature

Updating ea1b82a..05e9557

(Summary **of** changes)

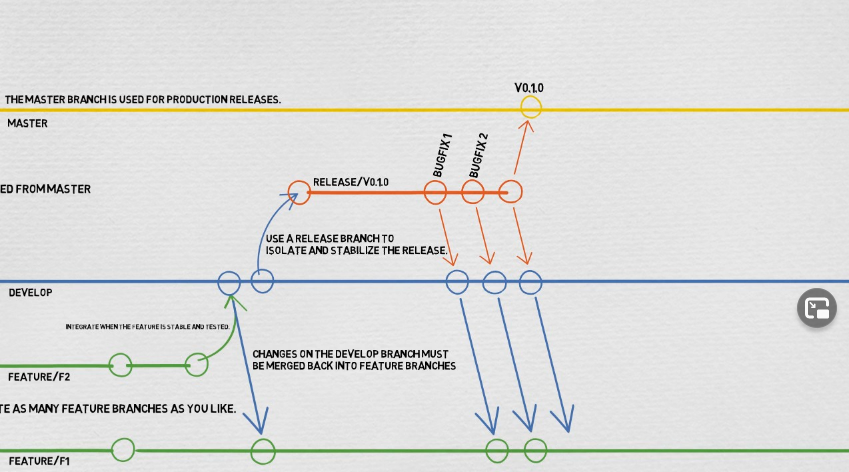
$ git branch -d feature/myfeature

Deleted branch feature/myfeature (was 05e9557).

$ git push origin develop

### Rama release

Como las ramas feature, las ramas release también tienen que surgir de la rama ***develop***.Contienen el código de la versión que se va a liberar próximamente Es un paso previo y preparatorio para la versión definitiva de *producción*. En ella se incluye todo el código de ***develop*** necesario para el lanzamiento. Puede que contenga algún error pequeño que se debe de arreglar en este momento para no incluirlo en *producción*. Una vez finalizada la rama, esta se debe incluir tanto en la rama ***develop*** como en la rama ***master***.

*Convención de nombres:* deben de seguir la siguiente convención: **release-\***, sustituyendo el \* por el número de versión (1.1, 2.3, 4.7, etc)

### Crear rama release

$ git checkout -b release-1.2 develop

Switched to a **new** branch "release-1.2"

(Hacer las modificaciones necesarias)

$ git commit -a -m "Release version 1.2 of the project"

[release-1.2 74d9424] Release version 1.2 **of** the project

1 files changed, 1 insertions(+), 1 deletions(-)

### Finalizar una rama release

Primero debemos actualizar la rama ***master***.

$ git checkout master

Switched to branch 'master'

$ git merge --no-ff release-1.2

Merge made by recursive.

(Summary **of** changes)

$ git tag -a 1.2

A continuación, debemos guardar esos cambios en la rama ***develop***.

$ git checkout develop

Switched to branch 'develop'

$ git merge --no-ff release-1.2

Merge made by recursive.

(Summary **of** changes)

Una vez integrada la rama tanto en ***master*** como en ***develop*** eliminaremos la rama en el repositorio local.

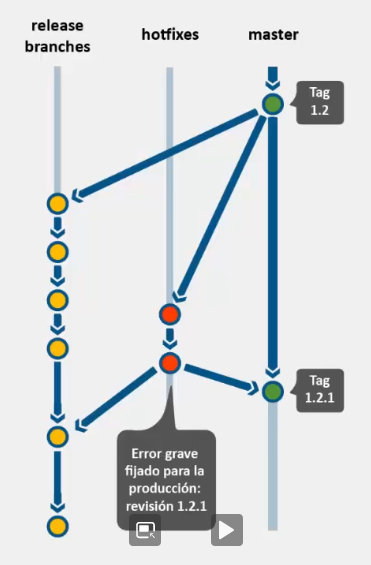
$ git branch -d release-1.2

Deleted branch release-1.2 (was ff452fe).

### Ramas hotfix

Estas ramas se originan a partir de rama ***master***. Contienen una versión de producción con un error que se desea arreglar urgentemente. Una vez arreglado el error, se incluye el contenido de esta rama en las ramas ***master*** y ***develop*** para subsanar el error. Además, hay que marcar la versión arreglada de *producción* con un *tag* en la rama ***master***.

*Convención de nombres:* deben de seguir la siguiente convención: ***hotfix****-\**, sustituyendo el \* por el número de la revisión (1.1.5, 2.3.1, 4.7.9, etc)



### Crear una rama hotfix

$ git checkout -b hotfix-1.2.1 master

Switched to a **new** branch "hotfix-1.2.1"

(Hacer las modificaciones necesarias)

$ git commit -a -m "Bumped version number to 1.2.1"

[hotfix-1.2.1 41e61bb] Bumped version number to 1.2.1

1 files changed, 1 insertions(+), 1 deletions(-)

### Finalizar una rama hotfix

Primero debemos actualizar la rama ***master*** y etiquetarla.

$ git checkout master

Switched to branch 'master'

$ git merge --no-ff hotfix-1.2.1

Merge made by recursive.

(Summary **of** changes)

$ git tag -a 1.2.1

A continuación, debemos incluir el ***hotfix*** en ***develop*** también.

$ git checkout develop

Switched to branch 'develop'

$ git merge --no-ff hotfix-1.2.1

Merge made by recursive.

(Summary **of** changes)

Una vez integrada la rama tanto en ***master*** como en ***develop*** eliminaremos la rama en el repositorio local.

$ git branch -d hotfix-1.2.1

Deleted branch hotfix-1.2.1 (was abbe5d6).

# Herramienta git flow

No es necesario ejecutar manualmente cada uno de los comandos expuestos anteriormente para administrar las ramas. Tenemos a nuestra disposición una herramienta de línea de comandos que nos ayudará en este proceso, ya que se encarga de realizar todos los pasos intermedios necesarios para gestionar las ramas.

## Cómo utilizar la herramienta

### Inicialización de metodología git flow en el repositorio

Para comenzar a utilizar la metodología Git flow, debemos iniciarla dentro de un repositorio git existente. Para ello ejecutaremos el comando:

$ git flow init

Con ello nos iniciará un proceso guiado con preguntas relacionadas con las convenciones de nombres para las ramas. Para utilizar la convención que hemos descrito arriba, debemos dejar los valores por defecto, sin introducir nada.

## Gestionar features

Crear una nueva ***feature*** (en el ejemplo *my-feature*):

$ git flow feature start my-feature

Finalizar la rama ***feature***:

$ git flow feature finish my-feature

Publicar la rama ***feature*** en el repositorio remoto:

$ git flow feature publish my-feature

Obtener una rama ***feature*** del repositorio remoto:

$ git flow feature pull origin my-feature

Seguir los cambios de la ***feature***:

$ git flow feature track my-feature

## Gestionar releases

Comenzar una ***release*** (en el ejemplo *release-1.2*):

$ git flow release start release-1.2

Concluir una ***release***:

$ git flow release finish release-1.2

Publicar la ***release*** en el repositorio remoto:

$ git flow release publish release-1.2

Debemos, también, publicar los *tags* en el repositorio remoto:

$ git push --tags

Seguir los cambios de la ***release***:

$ git flow release track release-1.2

## Gestionar hotfixes

Crear un ***hotfix*** (en este ejemplo *hotfix-1.2.1*):

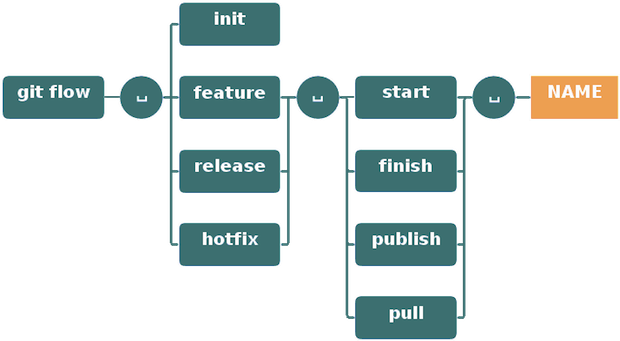
$ git flow hotfix start hotfix-1.2.1

Concluir un ***hotfix***:

$ git flow hotfix finish hotfix-1.2.1

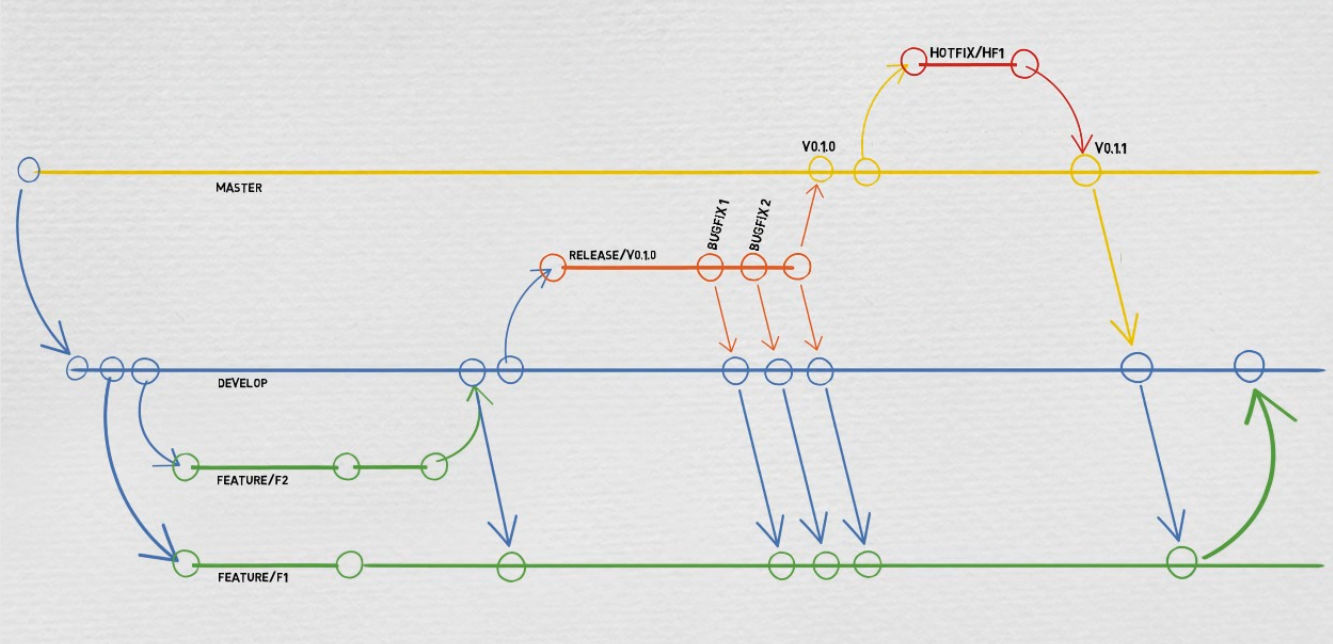
## Resumen herramienta git flow

A continuación, este gráfico tiene un resumen con los comandos y opciones posibles sobre esta herramienta:

Fuente: <https://danielkummer.github.io/git-flow-cheatsheet/index.es_ES.html>

## Ventajas

* Permite trabajar en diferentes ramas de forma ordenada sin necesidad de sobre cargar todo sobre una rama ***master***.
* Evita que se detenga la implementación del proyecto al poder manejar ramas auxiliares.
* Ofrece un canal exclusivo para ***hotfixes*** en *producción*.



## Comandos usuales

### git clone

Clona un repositorio remoto en tu folder local. Una vez clonado el repositorio, estarás trabajando dentro de la rama por defecto.

Sintaxis:

git clone <url\_del\_repositorio> && cd <nombre-repo>

### git pull

Obtiene los cambios de la rama actual en la que se está trabajando.

### git status

Obtiene los cambios realizados en el proyecto local.

### git add

Agrega archivos para su commit. Haz cambios y agrega tus archivos (repite el comando `git add` por cada archivo, o utiliza `git add .` para agregarlos todos)

Sintaxis:

git add <nombre-archivo>

### git branch

Lista las ramas existentes, por defecto mostraran los nombres de las ramas locales.

Para mostrar todas las ramas, incluidas las remotas:

git branch -a

Para mostrar todas las ramas, incluidas las remotas incluyendo su SHA-1 y asunto del commit:

git branch -a -v

Para crear una nueva rama:

git branch <branch\_name>

Para borrar una rama especifica:

git branch -d <branch\_name>

### git commit

El comando git commit captura una instantánea de los cambios preparados en ese momento del proyecto. Las instantáneas confirmadas pueden considerarse como versiones "seguras" de un proyecto: Git no las cambiará nunca a no ser que se lo pidas expresamente. Antes de ejecutar git commit, se utiliza el comando [git add](https://www.atlassian.com/es/git/tutorials/saving-changes) para pasar o "preparar" los cambios en el proyecto que se almacenarán en una confirmación. Estos dos comandos, git commit y git add, son dos de los que se utilizan más frecuentemente.

Sintaxis:

git commit -a -m “Message”

### git push

El comando git push te permite subir los commits desde tu rama (branch) local en tu repositorio git local al repositorio remoto.

Para poder subir a tu repositorio remoto, te debes asegurar de **hacer commit a todos tus cambios al repositorio local.**

Sintaxis:

git push <nombre\_del\_repositorio> <nombre\_de\_la\_rama>

Existen diferentes opciones que puedes pasar a este comando, puedes aprender más acerca de estos en la documentación de Git o ejecuta:

git push --help

### git checkout

Este comando se puede usar para cambiar de rama y para restaurar un archivo a vierta revisión.

Sintaxis para cambiar de rama:

git checkout <branch\_name>

Sintaxis para crear y cambiar a esta rama:

git checkout -b <branch\_name>

Sintaxis para restaurar un archivo a cierta revisión:

git checkout <file\_name>

Se crearon nuevos comandos que permite usar de manera más clara los dos diferentes usos de git checkout.

Cambiar de rama.

Sintaxis para cambiar de rama:

git switch <branch\_name>

Sintaxis para crear y cambiar a esta rama:

git switch -c <branch\_name>

Restaurar archivos a ciertas revisiones.

Sintaxis para cambiar de rama:

git restore <file\_name>

### git stash

El comando git stash almacena temporalmente (o guarda en un *stash*) los cambios que hayas efectuado en el código en el que estás trabajando para que puedas trabajar en otra cosa y, más tarde, regresar y volver a aplicar los cambios más tarde. Guardar los cambios en stashes resulta práctico si tienes que cambiar rápidamente de contexto y ponerte con otra cosa, pero estás en medio de un cambio en el código y no lo tienes todo listo para confirmar los cambios.

git stash

Para volver a aplicar los cambios de un stash se puede realizar con el comando:

git stash pop

Al hacer *pop* del stash, se eliminan los cambios de este y se vuelven a aplicar en el código en el que estás trabajando.

Otra opción es volver a aplicar los cambios en el código en el que estás trabajando *y* conservarlos en tu stash mediante el comando:

git stash apply

Si añadimos la opción -u (o --include-untracked), se indica a git stash que también guarde en el stash los archivos sin seguimiento:

git stash -u

## Procedimiento para administrar usuarios

* + 1. Enviar un correo a un Grupo Global - DevOps\_Support [<DevOps\_Support@chubb.com](mailto:%3cDevOps_Support@chubb.com)>; Github Enterprise [<github@Chubb.com](mailto:%3cgithub@Chubb.com)> para que creen el usuario con rol de Miembro.



* + 1. Cada usuario con Rol de Miembro, podrá crear Repositorios y adicionar a otros usuarios con el mismo rol para que a su vez tengan los mismos privilegios.
    2. Roles disponibles en GITHUB:
       1. Propietarios: tienen acceso administrativo completo a tu organización. Este rol debe limitarse a dos personas, por lo mucho, en tu organización. Para obtener más información, consulta la sección "[Mantener la continuidad de la propiedad para tu organización](https://docs.github.com/es/github/setting-up-and-managing-organizations-and-teams/maintaining-ownership-continuity-for-your-organization)".
       2. Gerente de Facturación: le permite a una persona administrar las configuraciones de facturación. Para obtener más información, consulta "[Agregar un gerente de facturación a tu organización](https://docs.github.com/es/articles/adding-a-billing-manager-to-your-organization)".
       3. Mantenedor del equipo: rol con un subconjunto de privilegios disponibles para los propietarios de la organización, un usuario con Rol Propietario, puede promover a cualquier miembro de la organización a este rol.
    3. Permisos del Rol Miembro: rol por defecto para todo el resto, definido en la matriz de la [documentación oficial de GITHUB](https://docs.github.com/es/github/setting-up-and-managing-organizations-and-teams/permission-levels-for-an-organization).

Entre los permisos más relevantes están:

* Crear y administrar repositorios.
* Realizar tareas propias dentro de cada repositorio para extraer, subir código, mezclar código, entre otros.
* Realizar revisiones en ramas protegidas.
* Crear equipos.
* Ver todos los miembros y equipos de la organización.

## Administrar una regla de protección de rama

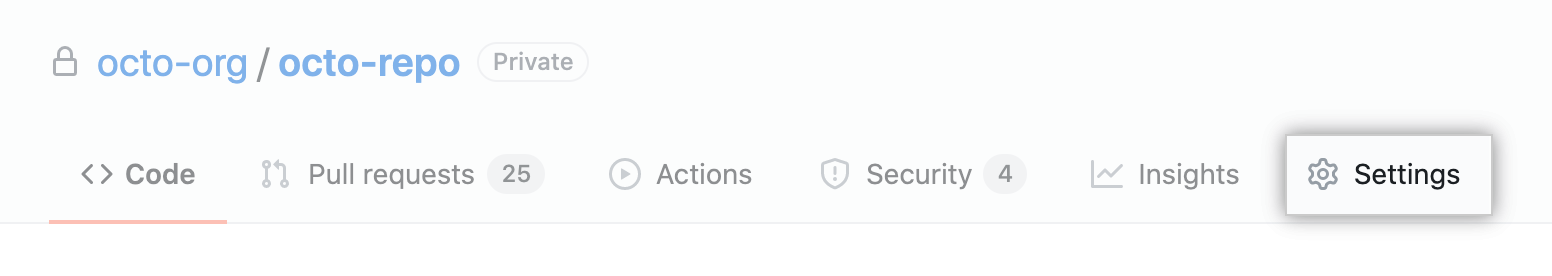
Puede crear una regla de protección de rama para hacer cumplir ciertos flujos de trabajo para una o más ramas, como solicitar una revisión de aprobación o pasar verificaciones de estado para todas las solicitudes de extracción fusionadas en la rama protegida.

Las personas con permisos de administrador en un repositorio pueden administrar las reglas de protección de sucursales.

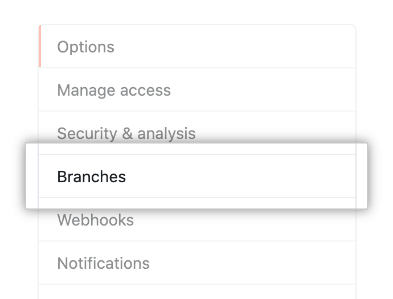
### [Crear una regla de protección de rama](https://docs.github.com/es/github/administering-a-repository/managing-a-branch-protection-rule#creating-a-branch-protection-rule)

Cuando crea una regla de rama, la rama que especifica no tiene que existir todavía en el repositorio..

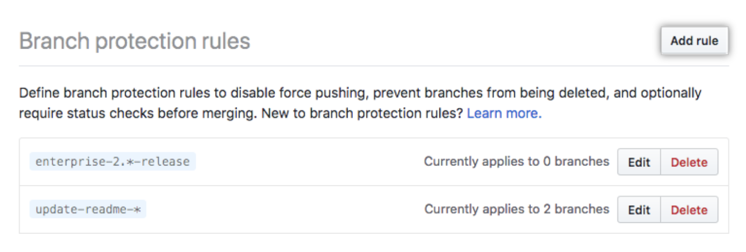
1. En GitHub, visita la página principal del repositorio.
2. Debajo de tu nombre de repositorio, da clic en  **Configuración**.



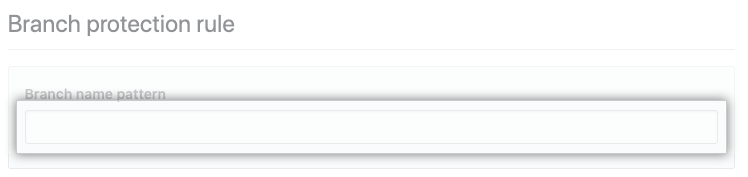
1. En el menú izquierdo, da clic en **Ramas**.



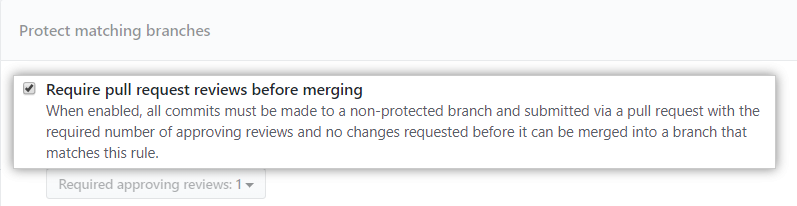
1. Junto a "Reglas de protección de rama", da clic en **Agregar regla**.



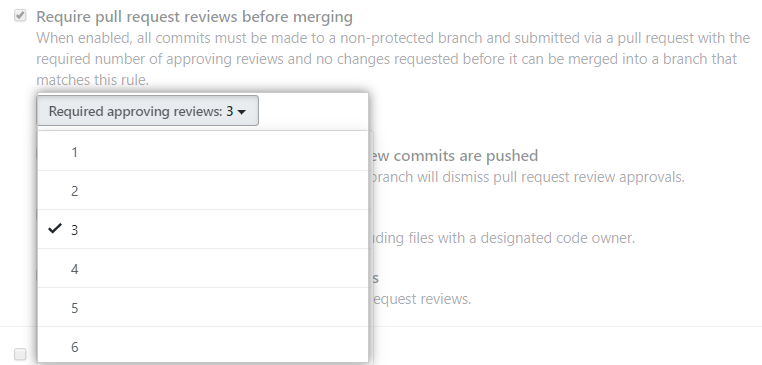
1. Debajo del "Patrón del nombre de la rama", teclea el nombre de la rama o el patrón que quieras proteger.



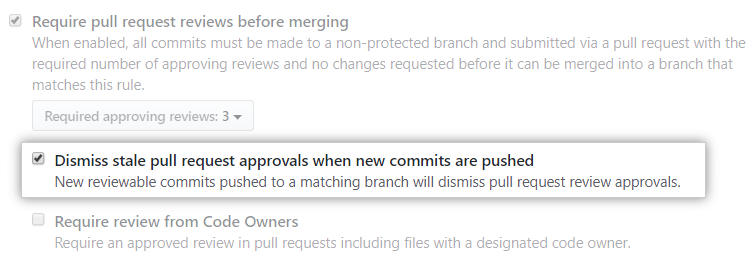
1. Opcionalmente, habilite las revisiones de solicitudes de extracción requeridas.
   * En "Proteger ramas coincidentes", seleccione Requerir revisiones de solicitudes de extracción antes de fusionar



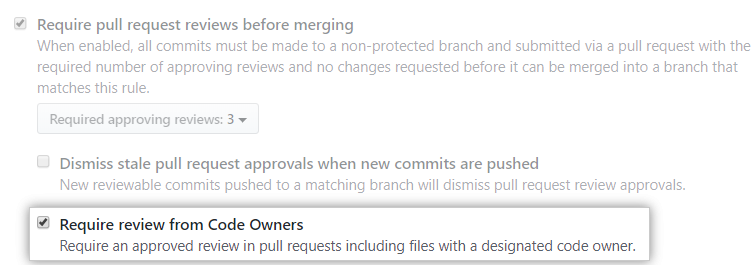
* + Click el menu drop-down **Required approving reviews**, luego seleccione la cantidad de revisiones aprobadas que le gustaría solicitar en la rama.



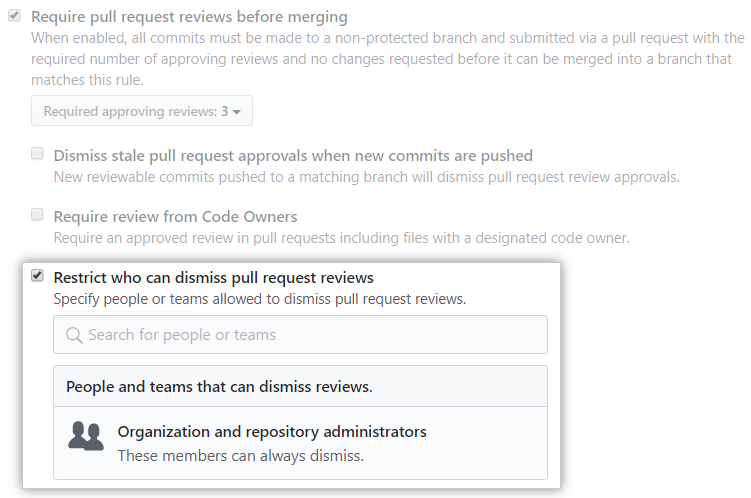
* + Opcionalmente, para descartar una revisión de aprobación de solicitud de extracción cuando se envía una confirmación de modificación de código a la rama, seleccione Descartar aprobaciones de solicitud de extracción obsoletas cuando se envían nuevas confirmaciones.



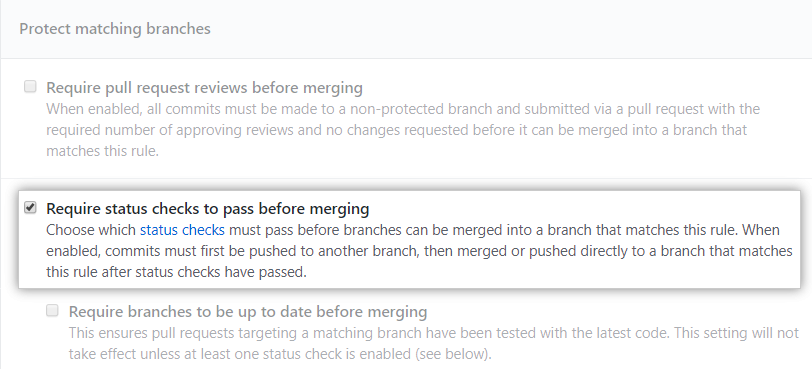
* + Opcionalmente, para solicitar la revisión de un propietario del código cuando la solicitud de extracción afecta el código que tiene un propietario designado, seleccione Solicitar revisión de los propietarios del código. Para obtener más información, consulte "[Acerca de los code owners](https://docs.github.com/es/github/creating-cloning-and-archiving-repositories/about-code-owners)".



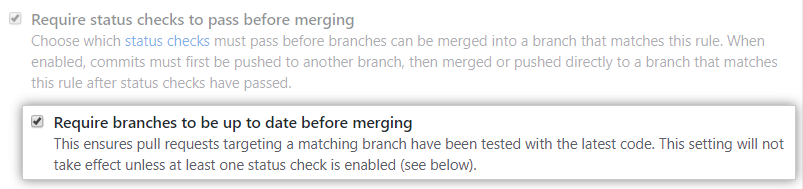
* + Opcionalmente, si el repositorio es parte de una organización, seleccione Restringir quién puede descartar las revisiones de solicitudes de extracción. Luego, busque y seleccione las personas o equipos que pueden descartar las revisiones de solicitudes de extracción. Para obtener más información, consulte "[Descartar una revisión de solicitud de extracción](https://docs.github.com/es/github/collaborating-with-issues-and-pull-requests/dismissing-a-pull-request-review)".



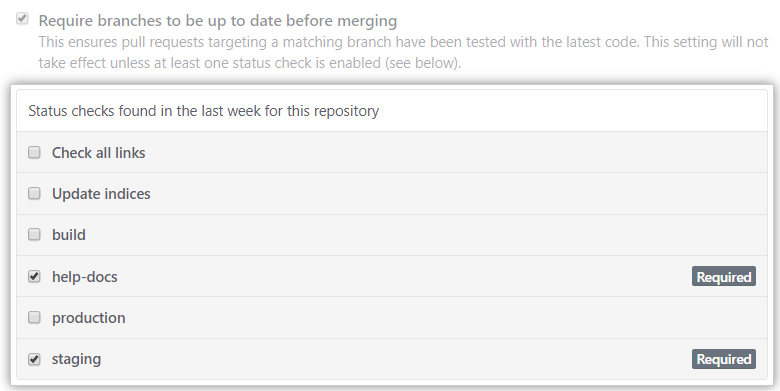
1. Opcionalmente, habilite las verificaciones de estado requeridas.
   * Seleccione Requerir que se aprueben las verificaciones de estado antes de fusionarse.



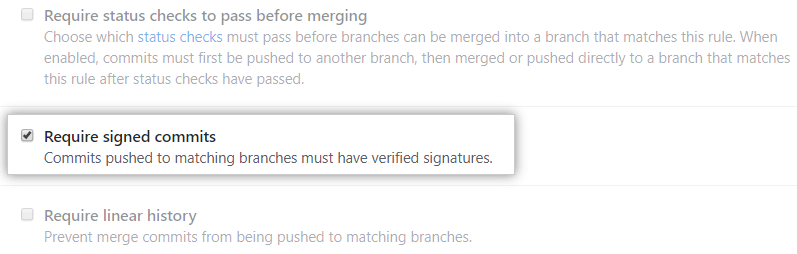
* + Optionally, to ensure that pull requests are tested with the latest code on the protected branch, select **Require branches to be up to date before merging**.



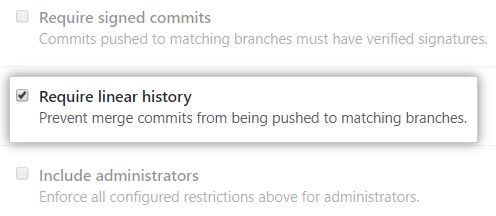
* + From the list of available status checks, select the checks you want to require.



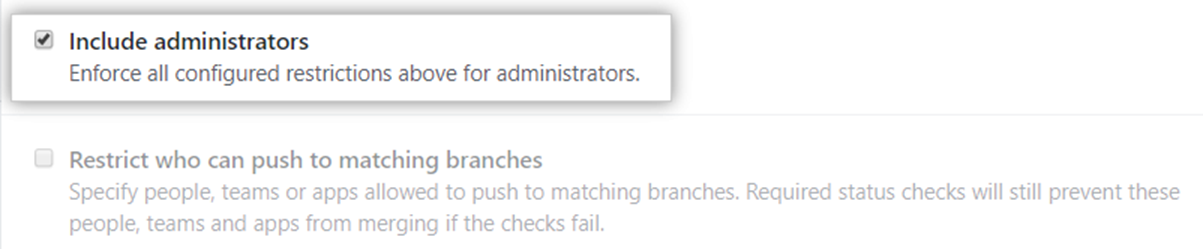
1. Optionally, select **Require signed commits**.



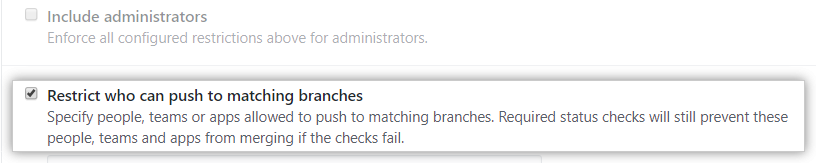
1. Optionally, select **Require linear history**.



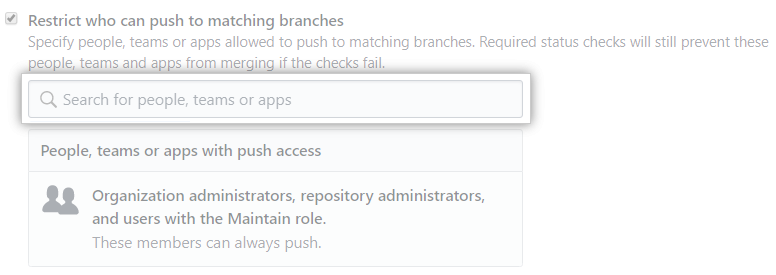
1. Optionally, select **Include administrators**.



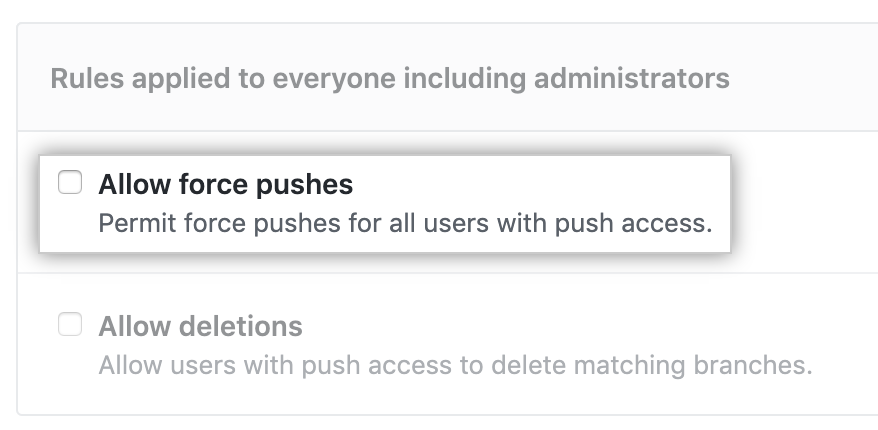
1. Optionally, if your repository is owned by an organization using GitHub Team or GitHub Enterprise Cloud, enable branch restrictions.
   * Select **Restrict who can push to matching branches**.



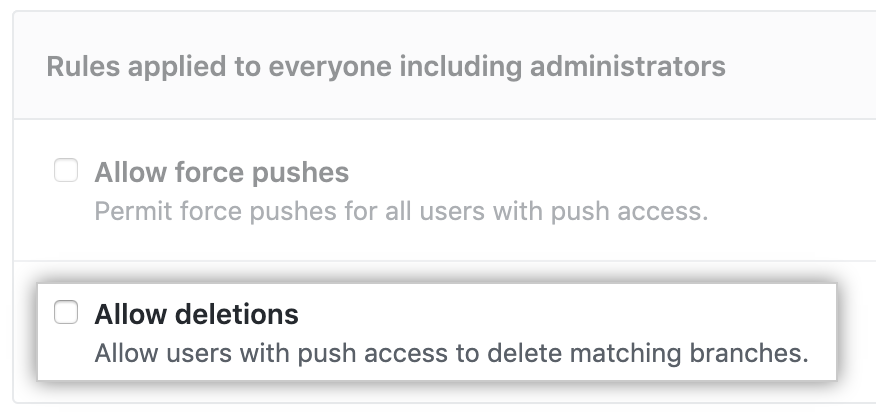
* + Search for and select the people, teams, or apps who will have permission to push to the protected branch.



1. Optionally, under "Rules applied to everyone including administrators", select **Allow force pushes**.



1. Optionally, select **Allow deletions**.



1. Click **Create**.