

# Git:

# Desarrollo colaborativo

Módulo 5

# Flujos de trabajo

## Flujos de trabajo

A medida que avancemos en el uso de Git y le demos mayor presencia en cada uno de nuestros proyectos, surgirá la necesidad de tener flujos de trabajo ya sea que estemos trabajando solos o en equipo.

En este último caso, **el flujo de trabajo es útil para unificar conceptos y poder mitigar posibles errores de integración** que ocurrirían si cada desarrollador realiza cambios a su manera en nuestro código.

Presentamos, a continuación, algunos posibles casos comunes de flujos de trabajo.



## Long-Running Branch

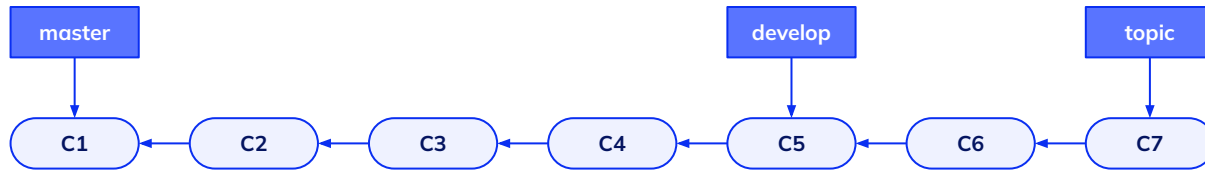
Debido a que Git utiliza un **three-way merge**, generalmente es fácil hacerlo de un *branch* a otro varias veces durante un período prolongado.

Esto significa que **podemos tener varios *branches* que siempre estén abiertos y que utilizamos para diferentes etapas de su ciclo de desarrollo**. Podemos fusionar las ramas regularmente unas con otras.

Más allá de las recomendaciones, el desarrollador puede elegir la organización de las ramas que crea más conveniente según el caso al que se esté enfrentando. En determinados escenarios puede resultar útil conservar el proyecto en una sola rama.

Quizás, por otra parte, resulte útil dividir las ramas en dos partes: código en desarrollo (*dev*) y código de producción (*prod*). O también puede dividir sus ramas en función de la duración de las mismas: estables o de corto plazo.

En realidad, estamos hablando de **punteros que avanzan en la línea de commits**. Los *branches* estables están más abajo en la línea de tu historial de *commits*, y los *branches* más avanzados están más arriba en el historial.



## Topic branch

Los ***branches temáticos*** son útiles en proyectos de cualquier tamaño. Un *branch* temático **es una rama de corta duración que se crea y usa para una característica particular o trabajo relacionado.**

Esto es algo que probablemente nunca hayas hecho antes con un VCS, porque generalmente es demasiado costoso crear y fusionar. **Pero en Git es común crear, trabajar, fusionar y eliminar *branches* varias veces al día.**

Esta técnica te permite cambiar de contexto rápida y completamente, ya que tu trabajo está separado en silos donde todos los cambios en esa rama tienen que ver con ese tema, es más fácil ver lo que sucedió durante la revisión del código.

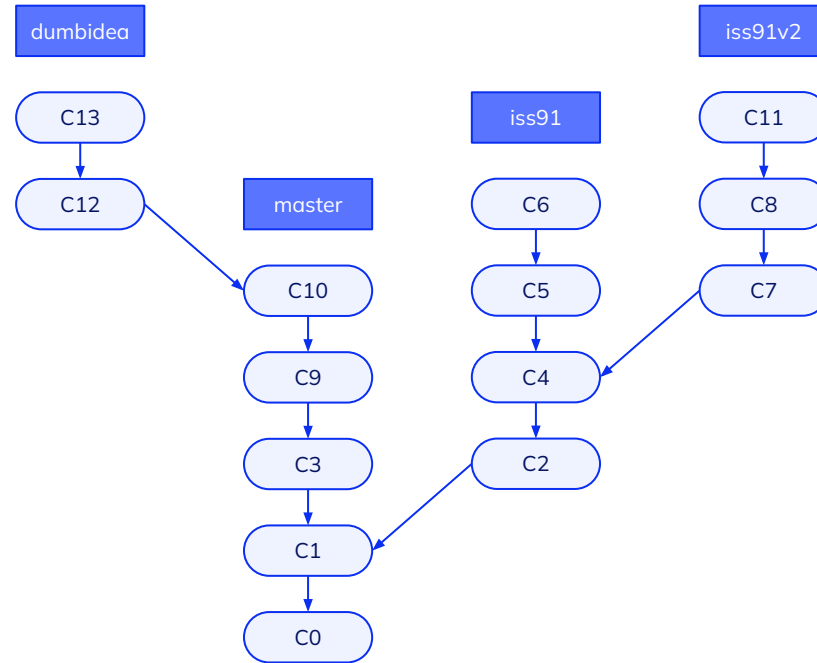
Puedes guardar los cambios allí durante minutos, días o meses y combinarlos cuando estén listos, independientemente del orden en que se crearon o trabajaron.

Considere un ejemplo de:

1. Hacer algo de trabajo (en *master*).
2. Ramificarse por un problema (*iss91*).
3. Trabajar un poco.
4. Ramificarse de la segunda rama para intentar otra forma de manejar lo mismo (*iss91v2*).
5. Volver a su *master branch* y trabajar allí por un tiempo, y luego ramificarse allí para hacer un trabajo que no estás seguro es una buena idea (*dumbidea branch*).

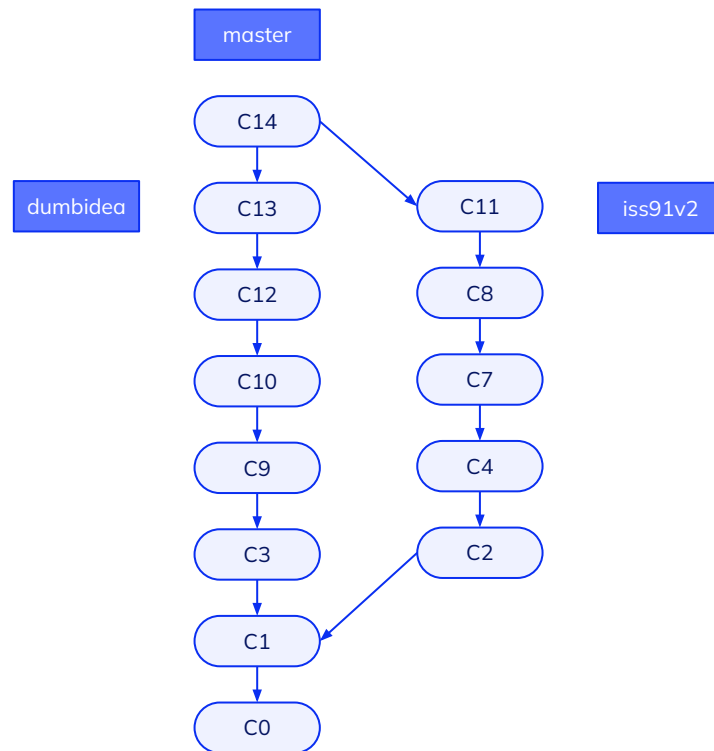


El **historial de commits** se verá más o menos así:





Ahora, supongamos que decides que te gusta más la segunda solución a tu problema (*iss91v2*); y le mostraste el *branch* “*navbar*” a tus compañeros de trabajo, y resulta ser genial. Puedes borrar la rama *iss91* original (perdiendo commits C5 y C6) y hacer merge de las otras dos.



## Administrador de integración

Debido a que Git permite tener múltiples repositorios remotos, es posible tener un flujo de trabajo donde cada desarrollador tenga **acceso de escritura a su propio repositorio público y acceso de lectura a todos los demás**. Este escenario, frecuentemente, incluye un repositorio canónico que representa el proyecto "oficial".

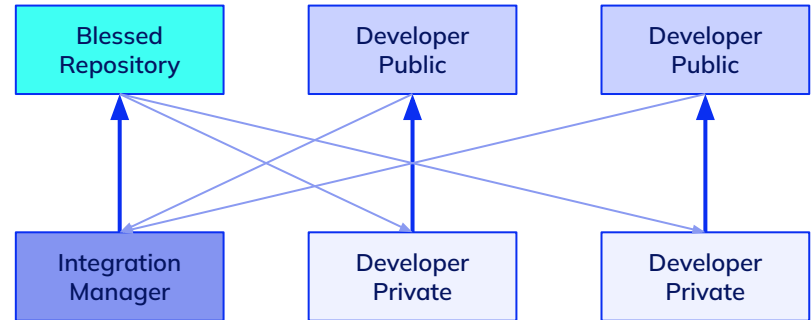
Para contribuir a ese proyecto, se debe crear un clon propio y público del proyecto e impulsar los cambios.

Luego, se envía una solicitud al responsable del proyecto principal para que realice sus cambios. El responsable de mantenimiento puede agregar su repositorio como un control remoto, probar sus cambios en forma local, fusionarlos en su rama y volver a su repositorio.

El proceso funciona como se detalla en el siguiente slide.




1. El responsable del proyecto hace push a su repositorio público.
2. Un contribuidor clona ese repositorio y realiza cambios.
3. Luego hace push a su propia copia pública.
4. Más tarde envía un correo electrónico al responsable de mantenimiento pidiéndole que haga pull de los cambios.
5. El mantenedor agrega el repositorio del contribuyente como remoto y se fusiona localmente.
6. A continuación, empuja los cambios combinados al repositorio principal.



## Dictador y tenientes


**Esta es una variante de un flujo de trabajo de repositorio múltiple.** Generalmente, es utilizada por grandes proyectos con cientos de colaboradores.

Un ejemplo famoso es el kernel de Linux. Varios gerentes de integración están a cargo de ciertas partes del repositorio; se llaman *tenientes*.



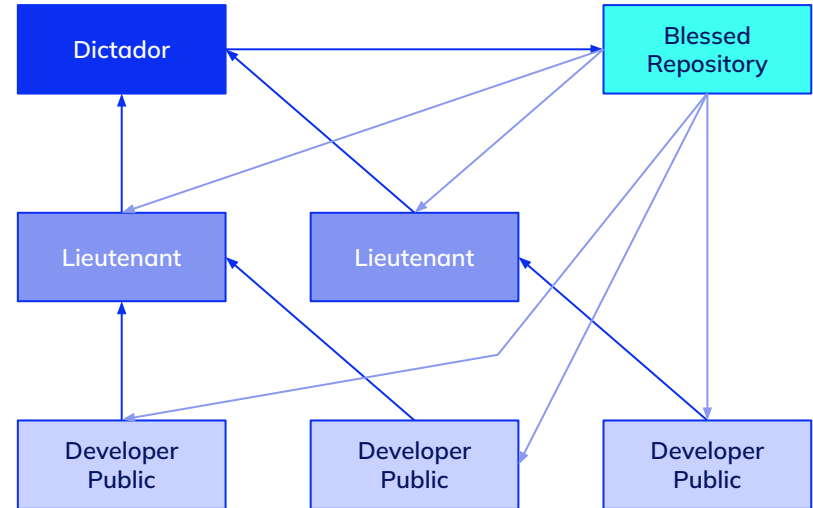
Todos los tenientes tienen un administrador de integración conocido como el *dictador benevolente*.

El dictador benevolente hace push desde su directorio a un repositorio de referencia del que todos los colaboradores deben hacer `pull`.



El proceso funciona así:

1. Los desarrolladores trabajan en su rama temática y hacen **rebase** de su trabajo por encima de **master**. La rama **master** es la del repositorio de referencia al que hace **push** el dictador.
2. Los tenientes fusionan las ramas temáticas de los desarrolladores en su rama **master**.
3. El dictador fusiona las ramas maestras de los tenientes en la rama **master** del dictador.
4. Finalmente, el dictador hace **push** de esa rama **master** al repositorio de referencia para que los otros desarrolladores puedan reajustarse en ella.




## Gitflow

Gitflow Workflow es un diseño de flujo de trabajo de Git que se publicó por primera vez y se hizo popular por Vincent Driessen en [nvie.com](http://nvie.com).

**Gitflow define un modelo de ramificación estricto diseñado en torno al lanzamiento del proyecto.** Esto proporciona un marco robusto para gestionar proyectos más grandes.

**Gitflow es ideal para proyectos que tienen un ciclo de lanzamiento programado.**



Este flujo de trabajo no agrega nuevos conceptos o comandos más allá de lo que se requiere para el flujo de trabajo de la rama de funcionalidades.

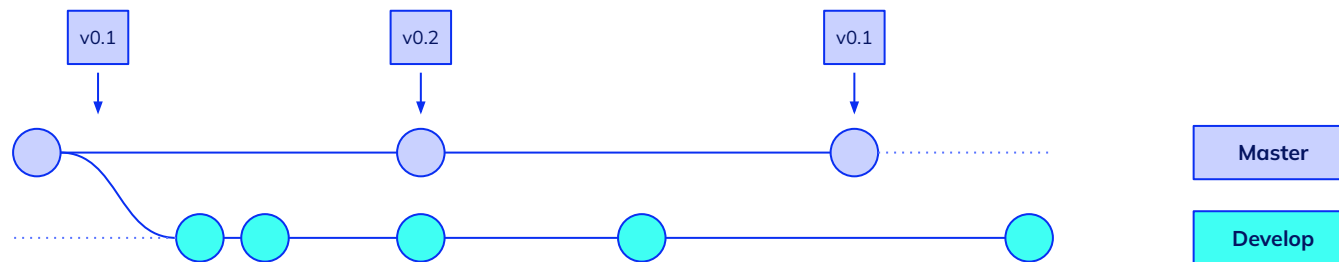
En cambio, asigna roles muy específicos a diferentes *branches* y define cómo y cuándo deben interactuar. Además de los *branches* de características, utiliza *branches* individuales para preparar, mantener y grabar lanzamientos.

Por supuesto, también puede aprovechar todos los beneficios del flujo de trabajo de *branches* de funcionalidades: `pull requests`, experimentos aislados y una colaboración más eficiente.

En lugar de una sola rama maestra, **este flujo de trabajo usa dos ramas para registrar el historial del proyecto.**

**La rama maestra almacena el historial de lanzamiento oficial, y la rama de desarrollo sirve como una rama de integración de características.**

También es conveniente etiquetar todos los `commits` en la rama maestra con un número de versión.



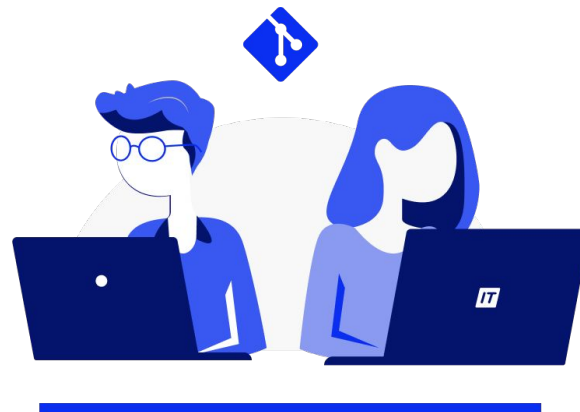
El primer paso es complementar el maestro predeterminado con una rama de desarrollo.

Una manera simple de hacer esto es que un desarrollador cree una rama de desarrollo vacía localmente y la envíe al servidor:

```
> git branch develop  
> git push -u origin develop
```

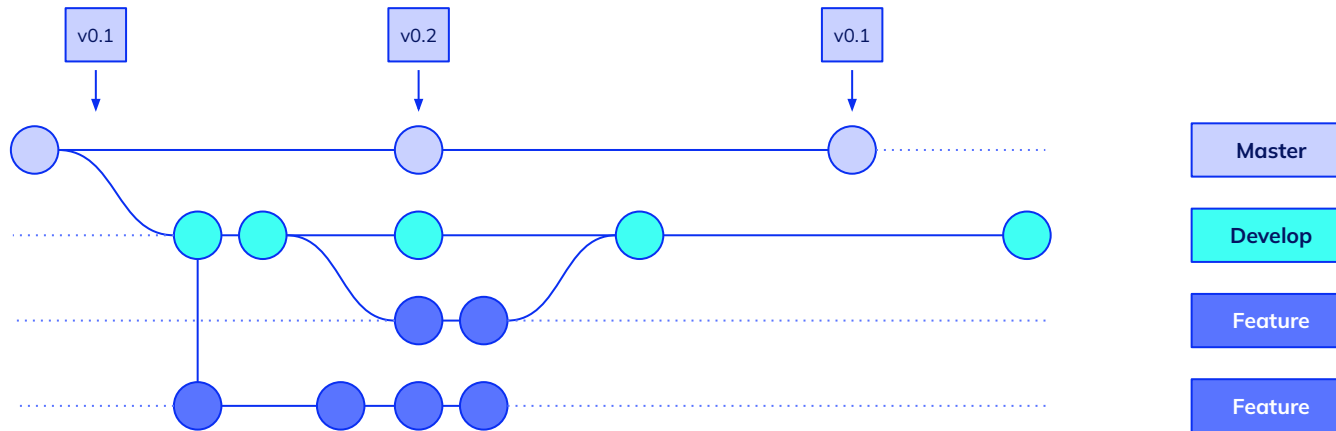
**Cada nueva característica debe residir en su propio *branch***, que se puede enviar al repositorio central para respaldo / colaboración. Pero, en lugar de bifurcarse del maestro, **las ramas de características usan la de desarrollo como su rama principal**.

**Cuando se completa una característica, se fusiona nuevamente en el desarrollo.** Las características nunca deberían interactuar directamente con el maestro.





En forma gráfica:



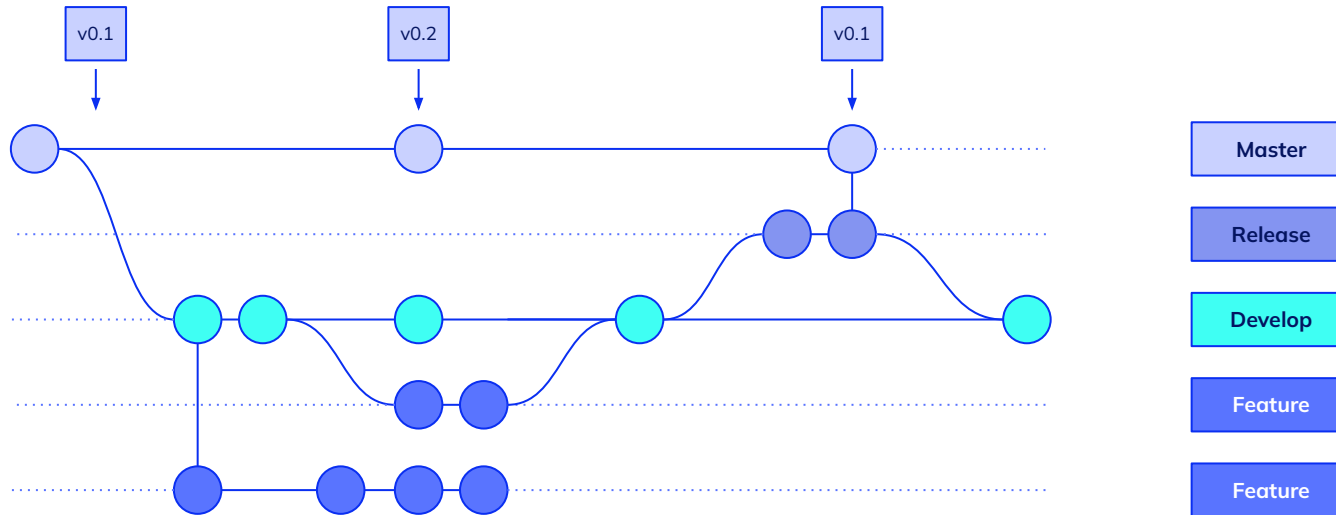
Una vez que el desarrollo ha adquirido suficientes características para un lanzamiento (o se acerca una fecha de lanzamiento predeterminada), **podemos bifurcar una rama de lanzamiento** fuera del desarrollo. La creación de esta rama inicia el siguiente ciclo de lanzamiento, por lo que no se pueden agregar nuevas características después de este punto, solo las correcciones de errores, la generación de documentación y otras tareas orientadas a la versión deben ir en esta rama.

**Cuando está listo para enviar, la rama de lanzamiento se fusiona en *master* y se etiqueta con un número de versión.** Además, debe fusionarse nuevamente en el desarrollo.

**El uso de una rama dedicada para preparar versiones hace posible que un equipo pueda pulir la versión actual mientras que otro continúa trabajando en las características para la próxima versión.** También crea fases de desarrollo bien definidas (por ejemplo, es fácil decir: "Esta semana nos estamos preparando para la versión 4.0" y verlo realmente en la estructura del repositorio).

Hacer ramas de lanzamiento es otra operación de ramificación sencilla. Al igual que las ramas de características, las ramas de lanzamiento se basan en la rama de desarrollo.

Gráfico:



**Las ramas de mantenimiento, revisión o "hotfix" se utilizan para parchear rápidamente las versiones de producción.** Las ramificaciones de revisión son muy parecidas a las ramificaciones de lanzamiento y ramificaciones de características, excepto que se basan en master en lugar de desarrollo.

**Esta es la única rama que debe bifurcarse directamente del maestro.** Tan pronto como se complete la corrección, debe fusionarse tanto en el maestro como en desarrollo (o en la rama de la versión actual), y el maestro debe etiquetarse con un número de versión actualizado.

Tener una línea de desarrollo dedicada para la corrección de errores le permite a su equipo abordar problemas sin interrumpir el resto del flujo de trabajo o esperar el próximo ciclo de lanzamiento. Puede pensar en las ramas de mantenimiento como ramas de lanzamiento ad hoc que funcionan directamente con el maestro.

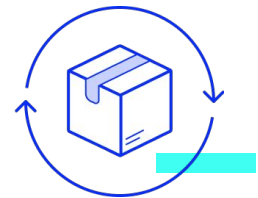
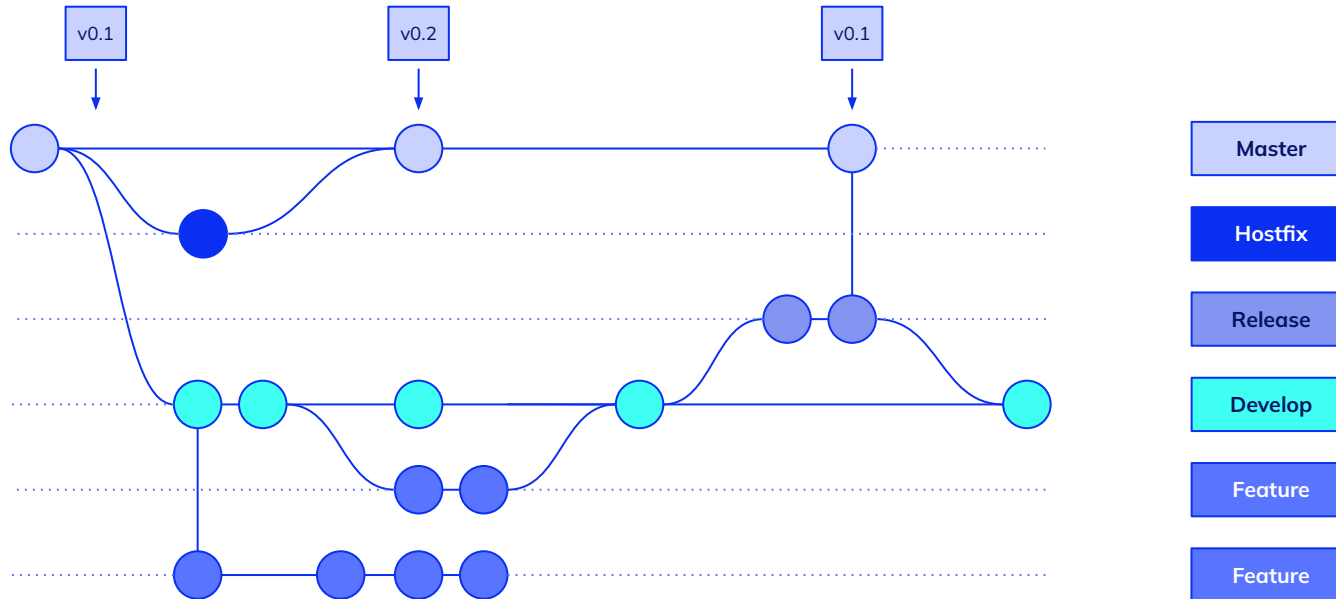


Gráfico:



Se debe tener en cuenta que si bien podemos realizar todo este trabajo de manera manual, es decir, ponernos de acuerdo con nuestro equipo de trabajo para que todos adopten el mismo tipo de flujo de trabajo, **podrían presentarse errores hasta que el sistema de bifurcación esté bien pulido y practicado por todos.**



Para resolver este inconveniente **se pueden encontrar varios clientes gráficos de Git que ya incorporan el sistema de Gitflow** para comenzar o continuar con proyectos existentes o bien existen determinadas extensiones para la línea de comandos que realizan el mismo tipo de trabajo por nosotros sin que corramos ningún problema a la hora de realizar cada tipo de *branch*.



**¡Sigamos  
trabajando!**