



git



# Estrategias de versionamiento con Git

---

# Control de versiones básica

---

## EL PROCESO DE TU TESIS



Tesis



Tesis final



Tesis final  
este sí



Tesis final  
este sí sí sí



Tesis final 2



Tesis final final



Tesis final listo



Tesis final por fin



Tesis final por  
fin eso espero



Tesis final por  
fin the end



Tesis finalisimo



Tesis ultimo



Tesis ultimo  
ahora si



Tesis ultimo de  
los ultimos



Tesis ultimo  
final ok

# ¿Qué es Git? ¿Para qué sirve?

---

Repositorio de código



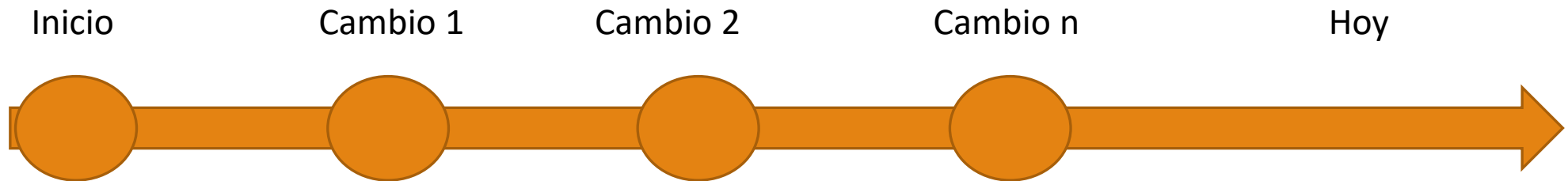
Trabajo en equipo



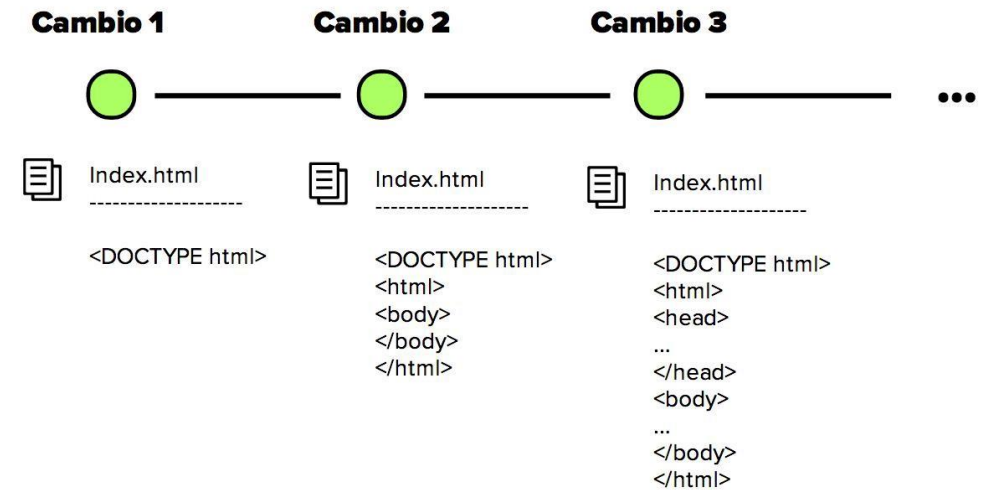
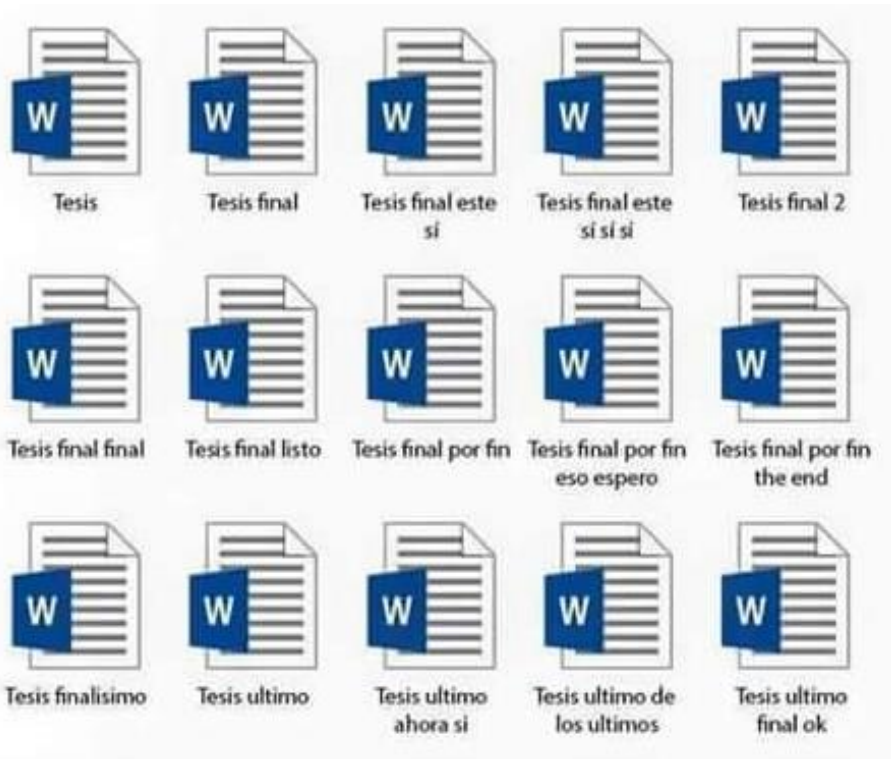
# ¿Qué es Git? ¿Para qué sirve?

---

Máquina del tiempo



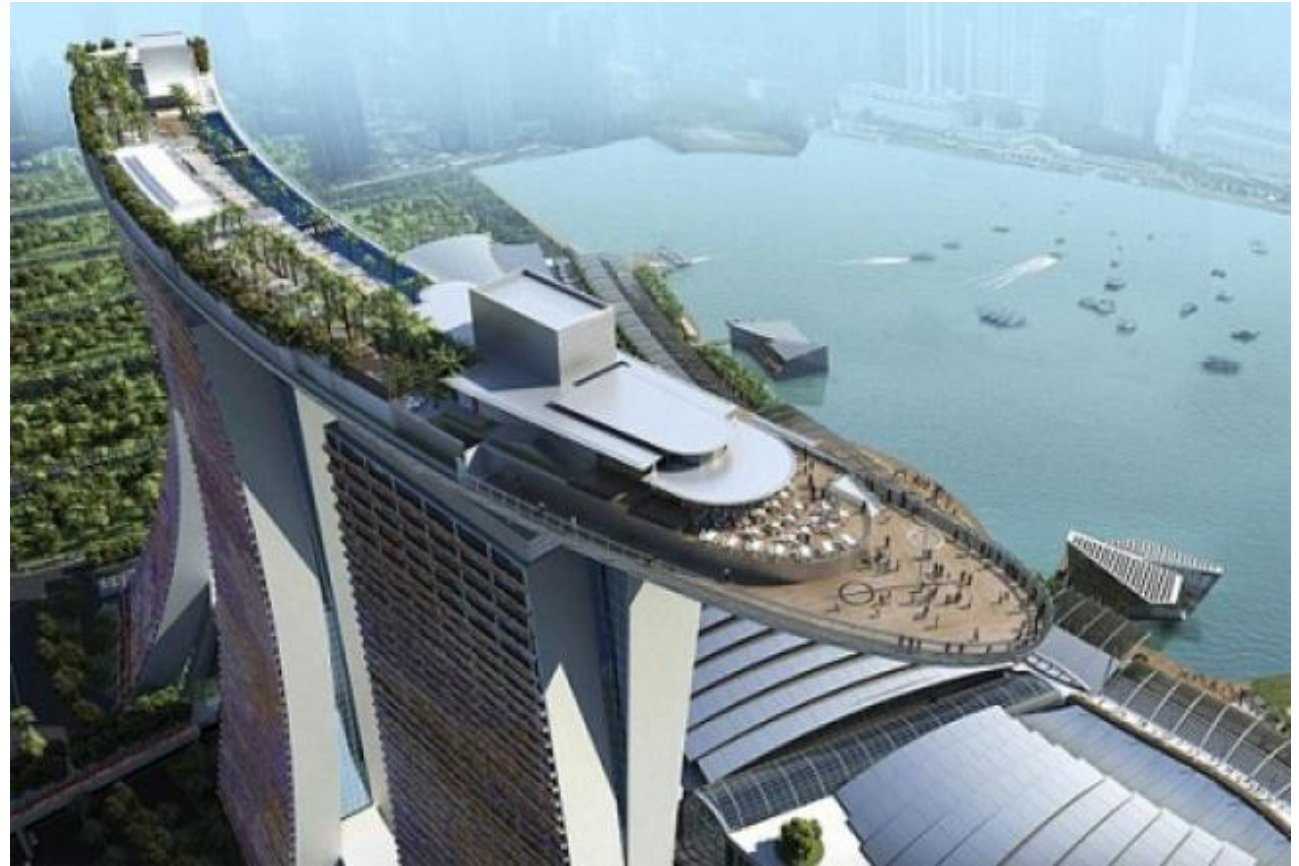
# ¿Qué es Git? ¿Para qué sirve?



Se pasa de un control de versiones básico a un control sistemático



# ¿Es posible trabajar juntos en el mismo proyecto?



Tú decides hasta donde quieres llegar.

# Estrategias de versionamiento

“Cuando utilizamos Git, siempre debemos de tener una estrategia de versionamiento, al no tenerla quizás el resultado no sea el esperado”.



# ¿Cuáles estrategias de versionamiento existen?

---

Producto funcional estable

**Master**



**Develop**



Continuidad de mejora del producto



# ¿Cuáles estrategias de versionamiento existen?

---

Producto funcional estable

**master**



Pruebas unitarias, pruebas funcionales...

**release**



**develop**



Continuidad de mejora del producto

# ¿Cuáles estrategias de versionamiento existen?

---



# ¿Cuáles estrategias de versionamiento existen?

---

## Trunk-based development

Trunk

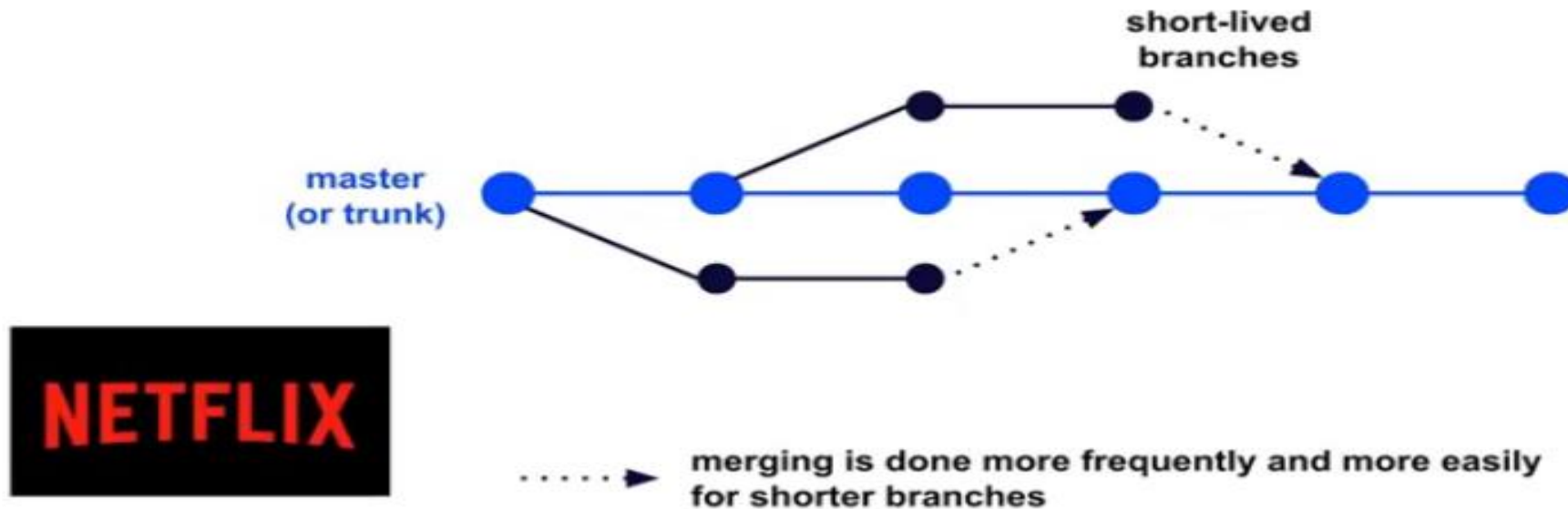


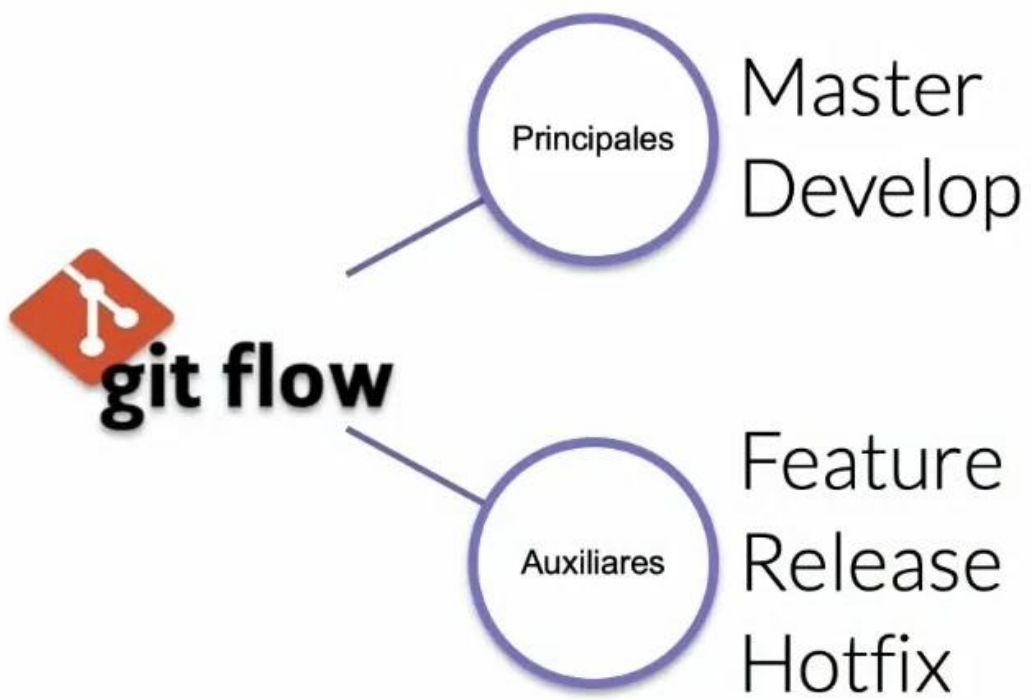
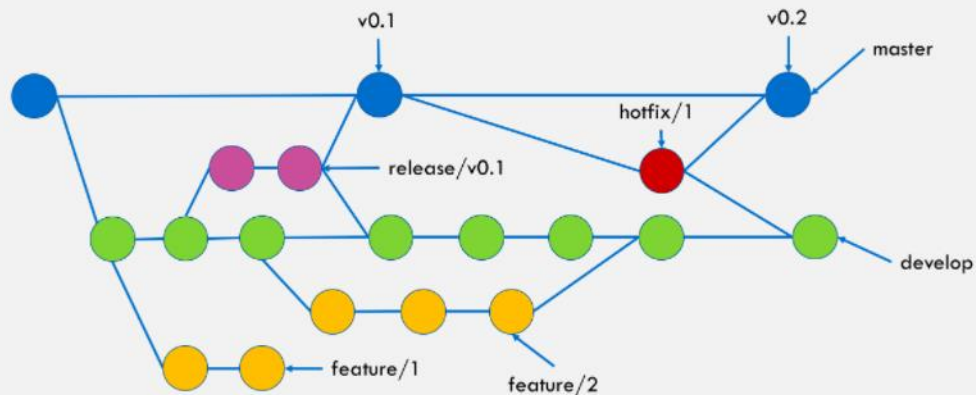
¿Cuándo podría usarse esta estrategia?



# ¿Cuáles estrategias de versionamiento existen?

## Trunk-based development





# Estrategia Gitflow



# Objetivo

---

El objetivo es no instanciar o recibir código de forma directa a través de commit en la rama Master, se debe recibir a través de ramas de tipo Feature, Release y Hotfix, siempre a través de ramas auxiliares.






# Riesgo

Es un riesgo recibir código directamente en la rama Master, porque puede generar defectos en el repositorio en las subidas a producción, que no se contemplen o se prevean, por lo que siempre es mejor integrar código en otras ramas antes de integrar con las ramas Master y Develop.


# Ramas Auxiliares

---

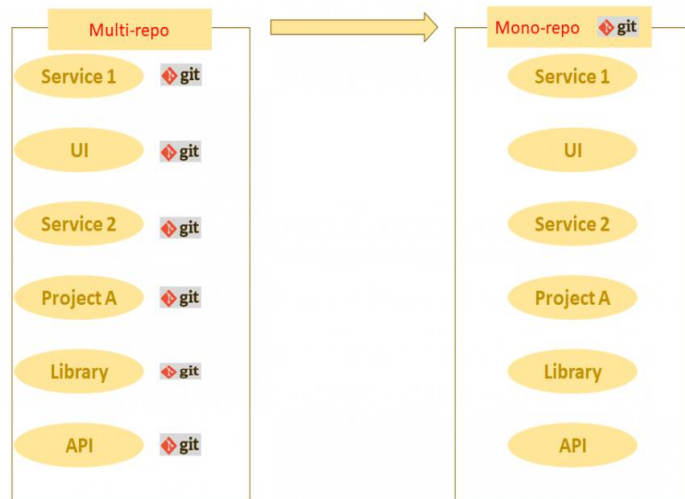
**Feature**, para nuevas características, nuevos requisitos o nuevas historias de usuario.



**Release**, para estandarizar o cortar una serie de código que ha estado desarrollándose en la rama Develop.



**Hotfix**, habitualmente se utiliza para depurar el código de producción, por haberse detectado un defecto crítico en producción que deba resolverse de inmediato, al que se le va a hacer una Release puntual para corregirlo.



# Estrategias de organización de repositorios

Mono-Repo  
Vs  
Multi-Repo



# Ventajas Mono Repo

---

- Un solo lugar para almacenar todo el código del proyecto, y todos los miembros del equipo pueden acceder a él.
- Fácil de reutilizar y compartir código, colaborar con el equipo.
- Fácil de comprender el impacto de su cambio en todo el proyecto.
- La mejor opción para la refactorización de código y grandes cambios en el código.
- Los miembros del equipo pueden obtener una vista general de todo el proyecto.
- Dependencias fáciles de administrar

# Desventajas Mono Repo

---

- El rendimiento, si el proyecto crece y se agregan más archivos las operaciones de extracción y otras pueden volverse lentas, por otro lado, las búsquedas de archivos pueden demorar más.
- El acceso es total, ya que al tener una asignación de permisos granular al código es más complicado. Además, al realizar procesos *continuos de integration* y *continuos deployment* requeriría de configuraciones adicionales.
- Además, si se tiene varios contratistas para el proyecto, darle acceso a todo el código base puede no ser tan seguro.

# Ventajas Multi Repo

---

Cada servicio y biblioteca tiene su propio control de versiones.

Las comprobaciones y extracciones de código son pequeñas e independientes, por lo que no hay problemas de rendimiento incluso si el tamaño del proyecto crece.

Los equipos pueden trabajar de forma independiente y no necesitan tener acceso a todo el código base.

Desarrollo y flexibilidad más rápidos.

Cada servicio puede lanzarse por separado y tener su propio ciclo de implementación, lo que facilita la implementación de CI y CD.

Mejor control de acceso: no es necesario que todos los equipos tengan acceso completo a todas las bibliotecas, pero pueden obtener acceso de lectura si lo necesitan





Las dependencias y bibliotecas utilizadas en los servicios y proyectos deben sincronizarse periódicamente para obtener la última versión.



Fomenta una cultura aislada en algún momento, lo que genera código duplicado y equipos individuales que intentan resolver el mismo problema.



Cada equipo puede seguir un conjunto diferente de mejores prácticas para su código, lo que genera dificultades para seguir las mejores prácticas comunes.

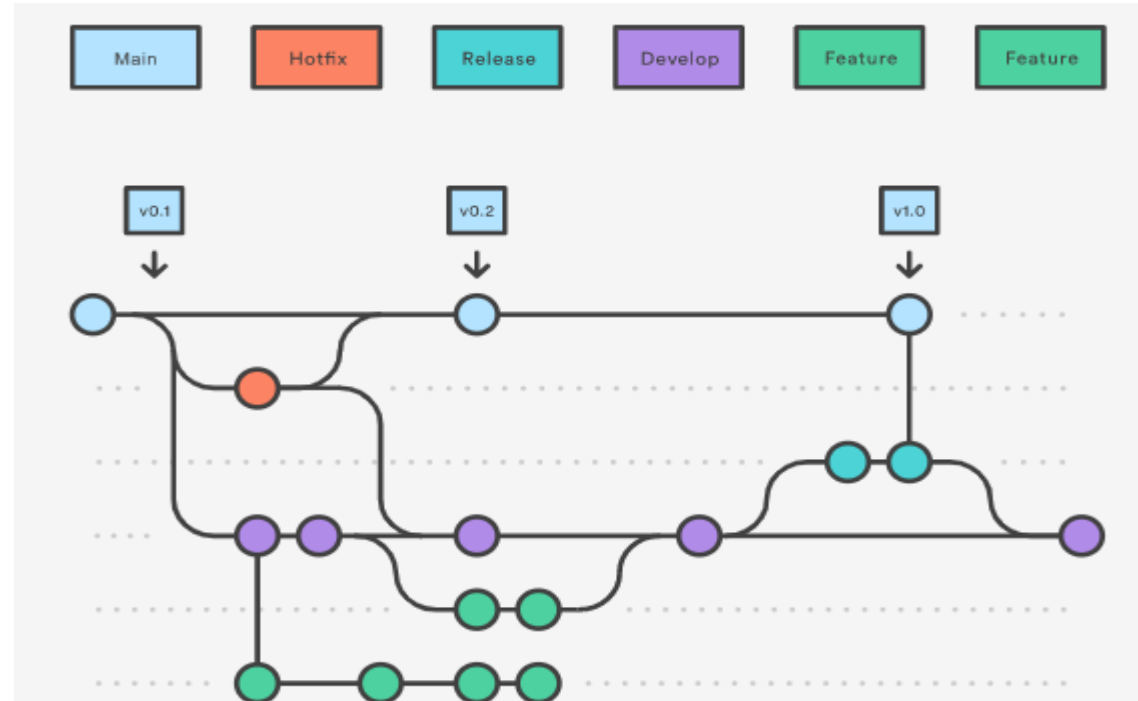
## Desventajas Multi Repo

Mono-repositorio	Repositorio múltiple
<p>Todo el código de todos los proyectos de una organización reside en un repositorio central</p>	<p>Cada servicio y proyecto tiene un repositorio separado</p>
<p>Los equipos pueden colaborar y trabajar juntos; pueden ver los cambios de los demás</p>	<p>Los equipos pueden trabajar de forma autónoma; los cambios individuales no afectan los cambios de otros equipos o proyectos</p>
<p>Cada persona tiene acceso a toda la estructura del proyecto.</p>	<p>Los administradores pueden limitar el control de acceso al proyecto o servicio al que el desarrollador necesita acceder</p>
<p>Pueden ocurrir problemas de ampliación si el tamaño del proyecto sigue creciendo</p>	<p>Buen rendimiento, debido al código limitado y a las unidades de servicio más pequeñas.</p>
<p>Difícil de implementar Despliegue Continuo (CD) e Integración Continua (CI)</p>	<p>Los desarrolladores pueden lograr fácilmente CD y CI porque pueden crear servicios de forma independiente</p>
<p>Los desarrolladores pueden compartir fácilmente bibliotecas, API y otro código común a medida que se actualizan en el repositorio central.</p>	<p>Cualquier cambio en las bibliotecas y otro código común debe sincronizarse periódicamente para evitar problemas más adelante.</p>

# Flujo general de GitfLow

El flujo general de Gitflow es el siguiente:

1. Se crea una rama `develop` a partir de `main`.
2. Se crea una rama `release` a partir de la `develop`.
3. Se crean ramas `feature` a partir de la `develop`.
4. Cuando se termina una rama `feature`, se fusiona en la rama `develop`.
5. Cuando la rama `release` está lista, se fusiona en las ramas `develop` y `main`.
6. Si se detecta un problema en `main`, se crea una rama `hotfix` a partir de `main`.
7. Una vez terminada la rama `hotfix`, esta se fusiona tanto en `develop` como en `main`.





Sígueme

---

# Fuentes

---

- <https://www.atlassian.com/es/git/tutorials/comparing-workflows/gitflow-workflow>
- <https://git-scm.com/book/en/v2>
- <https://geekflare.com/es/code-repository-strategies/>