Workflow

WorkFlows

- Hay varias esquemas de trabajo a la hora de implantar Git en un entorno de desarrollo profesional en el que, equipos de personas, colaboran en un proyecto de cierta complejidad.

- Como por ejemplo:
  - WorkFlow Centralizado
  - Feature Branch Workflow
  - GitFlow
  - Otros …

WorkFlows. WorKFlow Centralizado

- La transición de un sistema de control centralizado a un distribuido puede ser muy duro. Pasar de SVN a Git por ejemplo.
  - Ventajas:
    - Cada desarrollador tiene su copia privada del repositorio.
    - Aísla al desarrollador de los cambios que se producen en el repositorio central (hasta que sea conveniente), se centra en su desarrollo, y puede hacer los commit que considere necesarios.
    - Tiene acceso a herramientas de ramificación y fusión que son un mecanismo para integrar código y para compartir código entre repositorios.

WorkFlows. WorKFlow Centralizado. Como funciona

- Utiliza un repositorio central como punto de acceso único para todos los cambios del proyecto. Tiene una única la rama master y todos los cambios se fusiona con ella.

- Los desarrolladores comienzan clonando el repositorio central. En sus propias copias locales del proyecto, editar archivos y enviar los cambios como lo harían con SVN; Sin embargo, estas nuevas confirmaciones se almacenan localmente y están completamente aisladas del repositorio central. Esto permite a los desarrolladores diferir la sincronización con el repositorio central hasta que estén lista para subir.

- Para publicar los cambios en el proyecto oficial, los desarrolladores hacen un ”push" de su rama principal local en el repositorio central. Esto es el equivalente a svn commit, excepto que se añade todos los commits locales que no estén en la rama master del repositorio central.

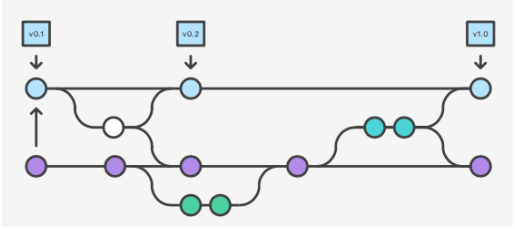
WorkFlows. Feature Branch WorKFlow

Teniendo un repositorio central, el siguiente paso, es crear ramas para cada una de las nuevas características. Esto es una manera fácil de fomentar la colaboración.

- Cada nueva funcionalidad tiene su rama dedicada.
- Esta encapsulación hace que sea fácil para múltiples desarrolladores trabajar en una característica particular sin perturbar el código base principal (rama master).
- También significa que la rama master nunca va a contener código roto, lo cual es una gran ventaja para los entornos de integración continua.

WorkFlows. Feature Branch WorKFlow. Como funciona

- Utilizan un repositorio central y la rama master sigue representando la historia oficial del proyecto. Para cada nueva funcionalidad los desarrolladores se crea una rama independiente. Las ramas debe de tener nombres descriptivos.
- Git no hace ninguna distinción entre la rama master y las ramas de características, por lo que los desarrolladores pueden trabajar con estas ramas como lo hacían con la rama master local del workflow centralizado.
- Las ramas de características deben ser “push” al repositorio central. Esto hace que sea posible compartir una característica con otros desarrolladores sin tocar la rama master. Master es la única rama "especial", el almacenamiento de varias ramas de características en el repositorio central no plantea ningún problema.



Gitflow Workflow

Qué es GitFlow

El flujo de trabajo **GitFlow** define un modelo de ramificación estricta diseñado en torno a las release de un proyecto software.

Proporciona un marco sólido para la gestión de proyectos de envergadura.

Qué es GitFlow

No añade ningún concepto nuevo o comandos de los ya vistos.

Asigna funciones muy específicas para diferentes ramas y define cómo y cuándo deben interactuar.

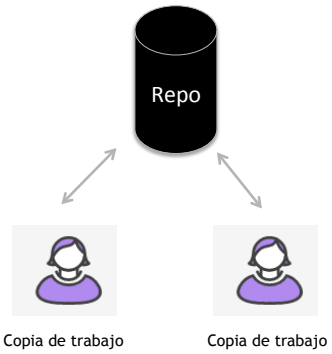
Ramas:

- De características [features].
- Para la preparación[release].
- El mantenimiento o incidencias [hotfix].
- De Soporte [support].
- Versión oficial del proyecto [master].
- De desarrollo [develop].

Tag

- Los commit's en la rama master deben de incluir un tag.

Cómo trabaja GitFlow



Se dispone de un repositorio central del proyecto.

Cada uno de los desarrolladores tiene su Repositorio local del proyecto.

Los desarrolladores utilizan pull-push para actualizar el repositorio central (desde o hacia)

Diferencia con otras formas de trabajar: **LA ESTRUCTURA DE RAMAS DEL PROYECTO.**

Cómo trabaja GitFlow

En lugar de utilizar una única rama principal del proyecto: **master**

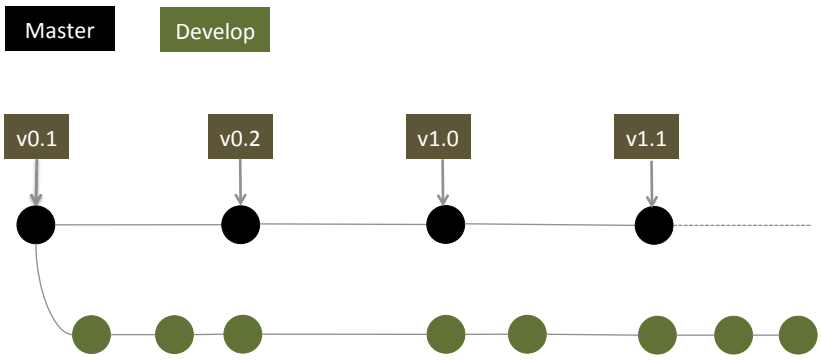
GitFlow utiliza dos ramas principales:

La rama **master** → Contiene la release oficial.

La rama **develop** → Rama de integración de la feature.

Además todos los commits en la rama master son etiquetados [tag] con el número de versión.

Cómo trabaja GitFlow



El resto de este flujo de trabajo gira en torno a la distinción entre estas dos ramas.

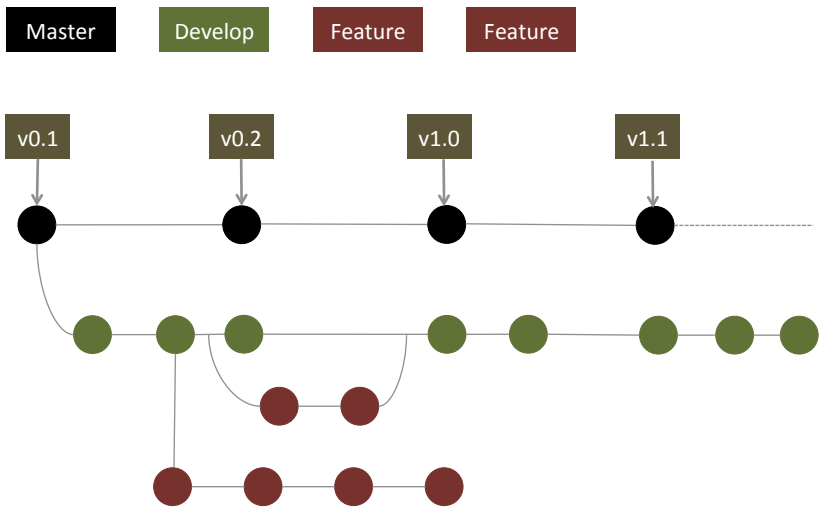
Cómo trabaja GitFlow. Ramas de feature

Cada nueva funcionalidad o característica debe residir en **su propia rama**. La rama puede ser **pushed** al repositorio central para hacer una copia de seguridad o para compartirla con otros desarrolladores.

Pero, en lugar de utilizar **master** como rama principal, utilizan la rama **develop** como rama principal para el desarrollo.

Cuando se termina una nueva características, se fusiona con la rama **develop**, nunca con la rama **master.**

Cómo trabaja GitFlow. Ramas de feature



Cómo trabaja GitFlow. Ramas de release

Una vez se han desarrollado un número de características suficiente o se ha alcanzado la fecha de una nueva release.

Hay que bifurcar (crear) una rama en **develop**. Con la creación de esta rama se inicia el siguiente ciclo de una nueva release, por lo que no se puede añadir ya nuevas características.

Excepciones:

- Resolución de Bug.
- Generación de documentación.
- Y otras tareas solo orientadas a la nueva release.

Cómo trabaja GitFlow. Ramas de release

Una vez que la rama, específica para la nueva release, está lista se deberá fusionar, ahora si con la rama **master** y se crea un **tag** que contiene el nuevo número de versión.

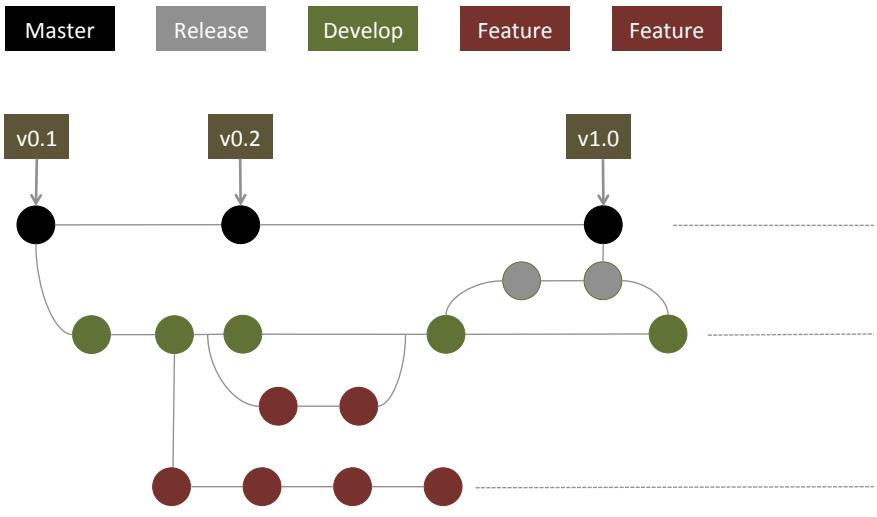
También esta rama de release debe ser fusionada con la rama **develop**, que puede haber progresado, desde el inicio del ciclo de la release.

Cómo trabaja GitFlow. Ramas de release

Este esquema de trabajo permite que un equipo se dedique a pulir la versión actual para publicar la release, mientras equipos de desarrollo siguen trabajando nuevas características de la próxima versión.

También crea fases bien definidas de desarrollo (por ejemplo, es fácil decir, "esta semana nos estamos preparando para la versión 4.0" y utilizarlas para verlo en la estructura del repositorio).

Cómo trabaja GitFlow. Ramas de release



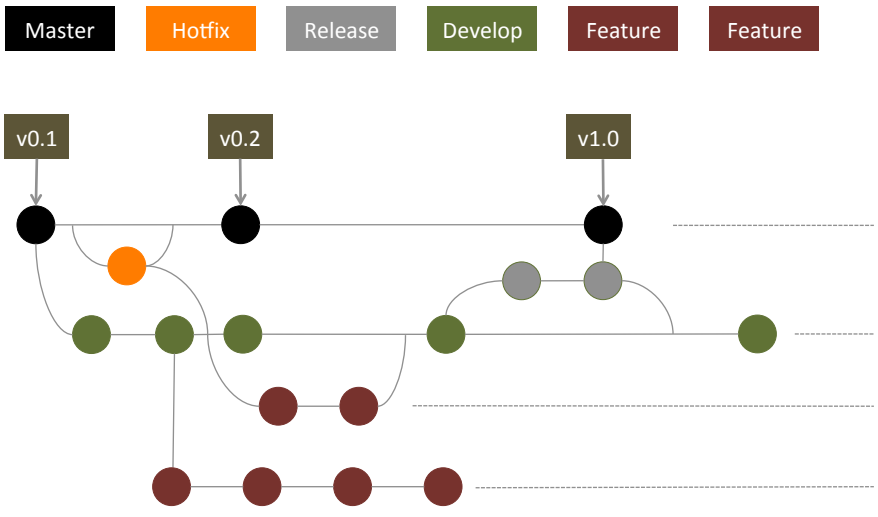
Cómo trabaja GitFlow. Ramas de mantenimiento

Las ramas de mantenimiento "revisión" (**HotFix**) se utilizan para reparar rápidamente incidencias en producción que hay que liberar urgentemente. Este tipo de ramas son las únicas que bifurcan directamente de **master**.

Tan pronto como se haya solucionado la incidencia, debe fusionarse tanto en **master** como **develop** (o la rama versión actual). La rama **master** debe ser etiquetada con un **tag** que contenga el número de la versión actualizada.

Tener una línea dedicada de desarrollo para la corrección de errores permite a la dirección del equipo solucionar los problemas sin interrumpir el resto del flujo de trabajo o de espera para el siguiente ciclo de liberación. **Se puede pensar en las ramas de mantenimiento como ramas de liberación ad hoc que trabajan directamente con la rama master.**

Cómo trabaja GitFlow. Ramas de mantenimiento



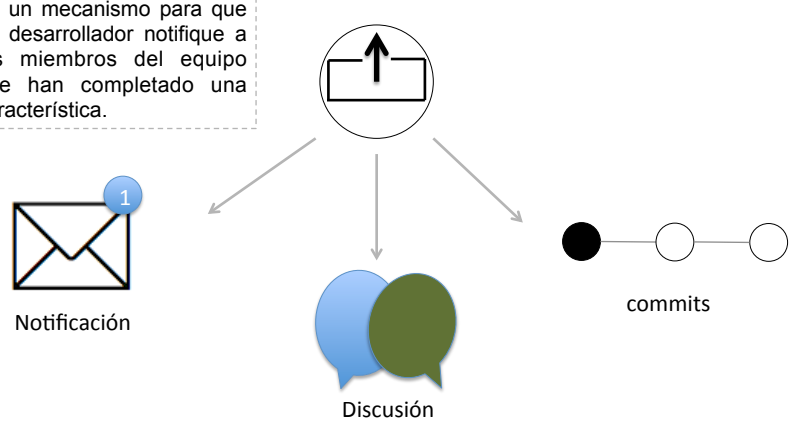
Pull Request

Pull Request es una funcionalidad que hace que sea más fácil, para los desarrolladores, colaborar usando repositorios remotos.

Proporcionan una interfaz web fácil de usar para la discusión de los cambios propuestos antes de su integración en el proyecto oficial.

Pull Request

Es un mecanismo para que un desarrollador notifique a los miembros del equipo que han completado una característica.



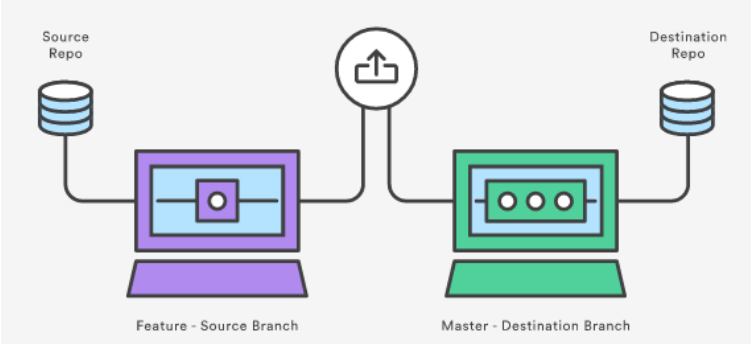
También es un foro dedicado para la discusión de la feature propuesta. Si hay algún problema con los cambios, compañeros de equipo pueden publicar comentarios en la solicitud de "Pull Request" e incluso ajustar la feature y haciendo un push en el repositorio.

Pull Request

Cuando hacemos una Pull Request:

Solicitamos a otro desarrollador (por ejemplo, el responsable del proyecto) que haga un pull de nuestra rama en su repositorio.

Esto significa que es necesario proporcionar 4 piezas de información para Pull Request:



Pull Request

Cuando hacemos una Pull Request:

Muchos de estos valores se ajustarán por defecto. Sin embargo, dependiendo de su flujo de trabajo de colaboración, su equipo puede tener que especificar valores diferentes. El diagrama anterior muestra una solicitud de extracción que pide fusionar una rama de característica en la rama master oficial, pero hay muchas otras maneras de utilizar las solicitudes de extracción.

Pull Request se pueden utilizar con distintos workflow entre otros con GitFlow. Sin embargo, Pull Request requiere o bien dos ramas distintas o dos repositorios distintos.

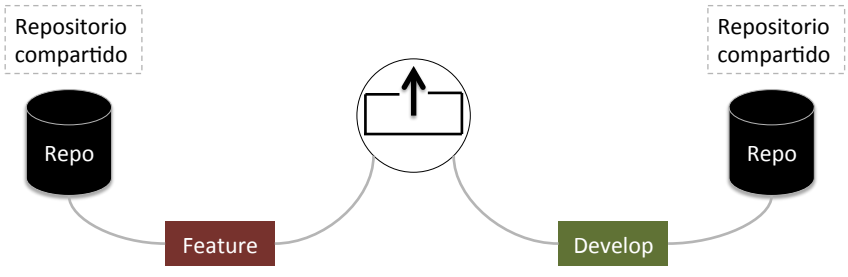
Pull Request

El uso de los Pull Request diferirá según el workflow, pero el proceso general es el siguiente:

1. **Un desarrollador crea una feature en una rama dedicada en su repo local.**
2. **El desarrollador push la rama en un repositorio público.**
3. **El desarrollador realiza un Pull Request a través de Bitbucket|GitHub.**
4. **El resto del equipo revisa el código, lo discute, y lo altera**
5. **El manager del proyecto combina la feature en el repositorio oficial y cierra la solicitud de extracción.**

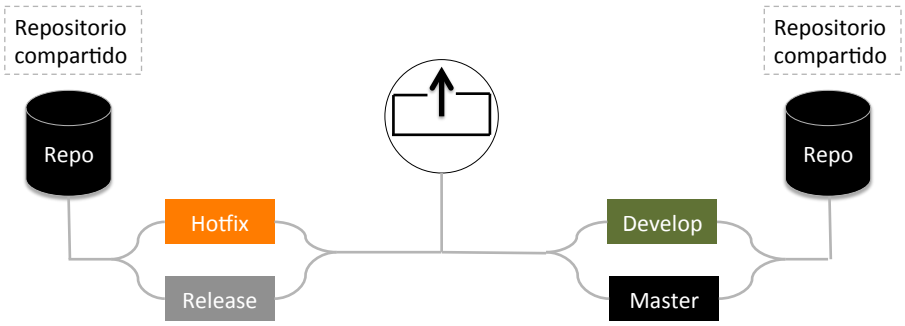
Pull Request con GitFlow

Añadir a GitFlow Pull Request ofrece a los desarrolladores un lugar donde discutir sobre, la liberación de una nueva rama características o de una rama de mantenimiento, mientras se está trabajando en ella.



Pull Request con Gitflow

Añadir a GitFlow Pull Request ofrece a los desarrolladores un lugar donde discutir sobre, la liberación de una nueva rama características o de una rama de mantenimiento, mientras se está trabajando en ella.



Resolver conflictos en GitFlow

El repositorio central [**develop**] representa el proyecto oficial, por lo que su historia debe ser tratada como sagrada e inmutable. Si commits locales de un desarrollador divergen del repositorio central, Git se negará a a hacer “push” de sus cambios ya que esto sobrescribe commits oficiales.

Antes de que el desarrollador publique su feature, tienen que buscar a los commits que han actualizado [**develop]** y hacer un rebase a sus cambios. Esto es como decir: "Quiero añadir mis cambios a lo que los demás ya han hecho." El resultado es una historia perfectamente lineal.

Comando:

```
$ git pull –rebase <remote> <branch>
```

Buscar la copia remota especificada de la rama actual e inmediatamente realiza un rebase con la copia local. Básicamente:

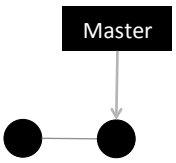
```
$ git fetch <remote> y
$ git rebase origin /<rama-actual>
```

Resolver conflictos en GitFlow

Si, en los cambios locales hay conflicto con commits oficiales, Git hará una pausa en el proceso de rebase y le dará una oportunidad al desarrollador a resolverlos de forma manual.

Git utiliza los mismos comandos git status y git add para generar commits y resolución de conflictos de merge. Esto hace que sea fácil para los desarrolladores gestionar sus propios “merges”. Además, si aparecen problemas, Git hace que sea fácil abortar todo el proceso de rebase e intentarlo de nuevo.

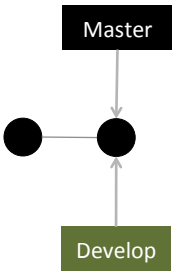
Cómo trabaja GitFlow. Pasos



Uno de los desarrolladores crea una rama develop vacía y luego hace un push al repositorio. Inicializamos un repositorio. Creamos un fichero con los miembros del equipo y hacemos un primer commit y luego:

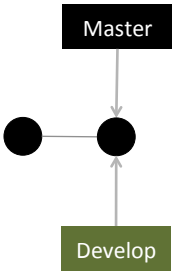
```
$ git branch develop
$ git push --all <url>
```

Cómo trabaja GitFlow. Pasos



La rama develop contendrá la historio completa del proyecto. Mientras que master tendrá una versión abreviada.
Otros desarrolladores deberán clonar el repositorio central y crear una rama develop

Cómo trabaja GitFlow. Pasos

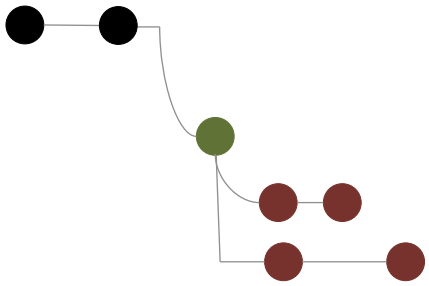



```
$ git clone https://bitbucket.org/ejrbalma/gitflow.git
$ git checkout –b develop origin/develop
```

Ahora todos los desarrolladores tienen una copia local del repositorio del proyecto incluyendo los dos branch (master/develop).

Cómo trabaja GitFlow. Pasos

Master Hotfix Release Develop Feature

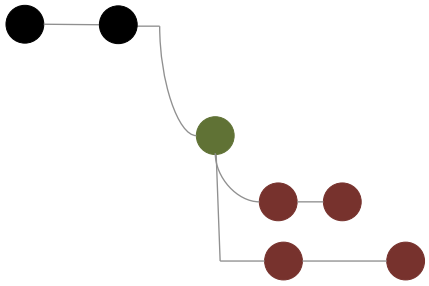


Dos desarrolladores comienzan dos características. Supongamos que cada uno de ellos trabaja en una nueva funcionalidad independientes. Por lo que cada uno de ellos debe crear su rama, pero en lugar de bifurcar de master deben bifurcar de develop. Cada uno debe de ejecutar:

```
$ git checkout –b feature[n] develop
```

Cómo trabaja GitFlow. Pasos

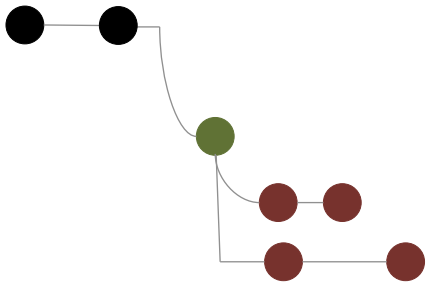
Master Hotfix Release Develop Feature



Uno de los desarrolladores termina antes. Y como el equipo utilizan el “pull request”. Pregunta al resto del equipo para saber si hay conflicto. Si no hay problema. Puede hacer un merge sobre develop local y hacer un push al repositorio central.

Cómo trabaja GitFlow. Pasos

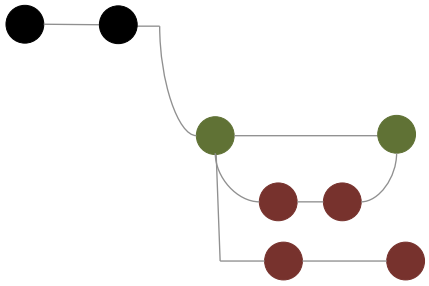
Master Hotfix Release Develop Feature



```
$ git pull orgin develop # Con este comando actualiza develop
$ git checkout develop
$ git merge feature[N]
```

Cómo trabaja GitFlow. Pasos

Master Hotfix Release Develop Feature

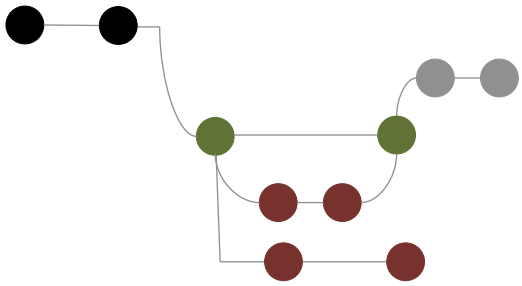


```
$ git push origin develop
$ git branch –d feature[n]
```

Y el desarrollador que ha terminado comienza a preparar una release.

Cómo trabaja GitFlow. Pasos

Master Hotfix Release Develop Feature



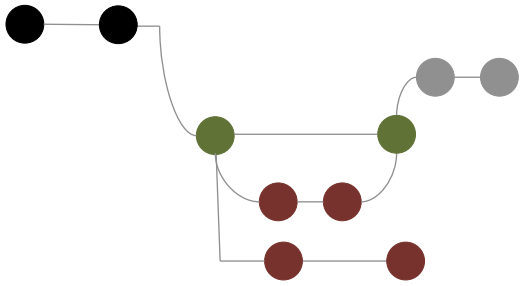
En la preparación de la nueva release. Se debe de crear un nuevo branch que bifurca de develop

```
$ git checkout –b release-0.1 develop
```

Esta rama sirve para pulir la release.(documentar/test/…)

Cómo trabaja GitFlow. Pasos

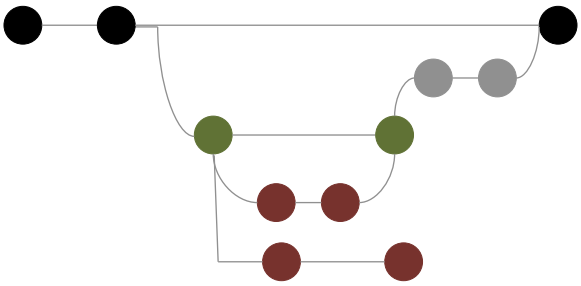
Master Hotfix Release Develop Feature



Tan pronto el desarrollador crea esta rama y hace un push al repositorio central. La release se congela. Cualquier otra funcionalidad que no esté en develop será postergada al siguiente ciclo de release.

Cómo trabaja GitFlow. Pasos

Master Hotfix Release Develop Feature

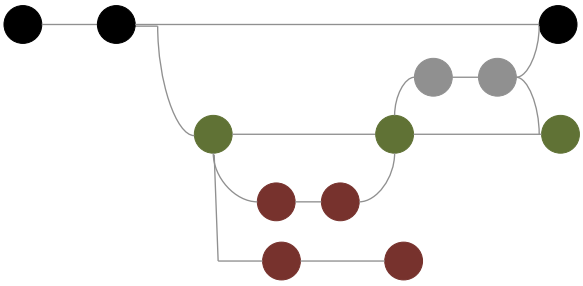


Una vez que la release está lista los pasos son: hacer un merge en master

```
$ git checkout master
$ git merge release-0.1
$ git push origin master
```

Cómo trabaja GitFlow. Pasos

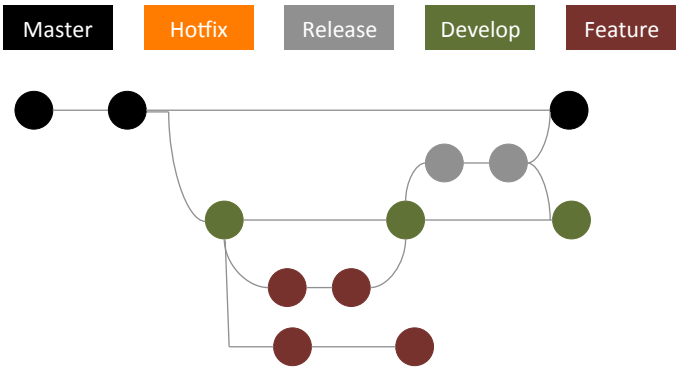
Master Hotfix Release Develop Feature



Una vez que la release está lista los pasos son: hacer un merge en develop

```
$ git checkout develop
$ git merge release-0.1
$ git push origin develop
$ git branch –d release-0.1 # borramos la rama.
```

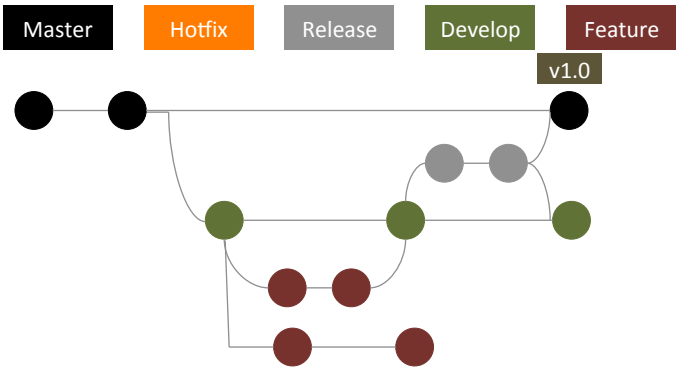
Cómo trabaja GitFlow. Pasos



Siempre que se fusiona algo con master se debe de añadir un tag al commit del merge para tener una referencia.

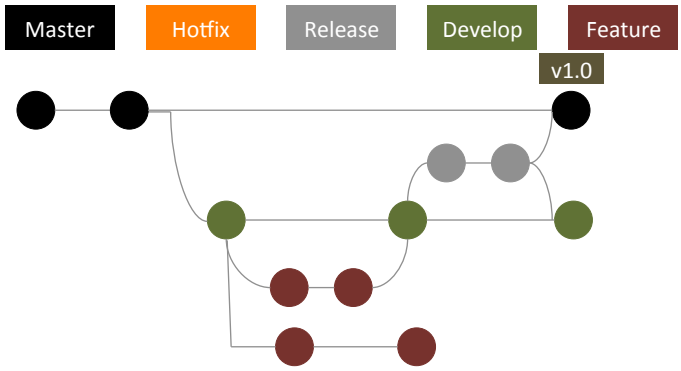
```
$ git tag –a 0.1 –m "Initial public release" master
$ git push –tags origin master
```

Cómo trabaja GitFlow. Pasos



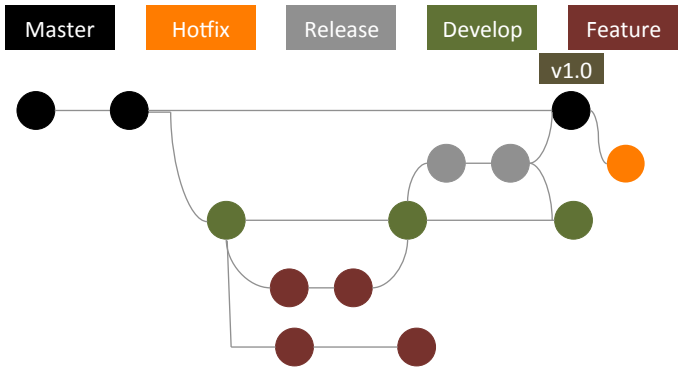
Un usuario del aplicativo informa de un bug en producción.
Uno de los desarrolladores debe abrir una nueva rama en master para resolver el bug. Resuelve la incidencia con el menor número de commits y fusiona directamente en master.

Cómo trabaja GitFlow. Pasos




```
$ git checkout –b issue-#001 master
```

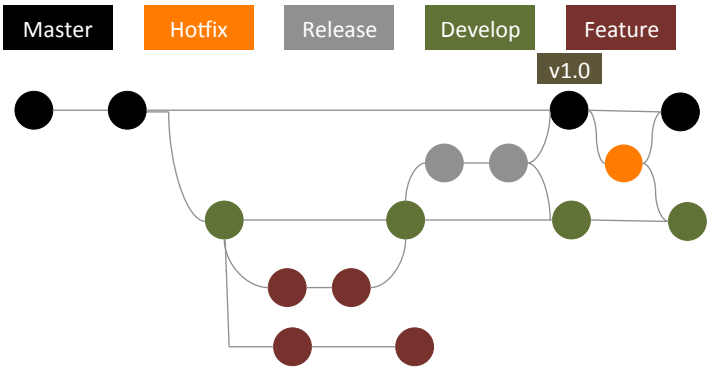
Cómo trabaja GitFlow. Pasos



Se resuelve el bug con el menor número posible de commit. Nos posicionamos en master y fusionamos y push al repo-central.

```
$ git checkout master
$ git merge issue-#001
$ git push origin master
```

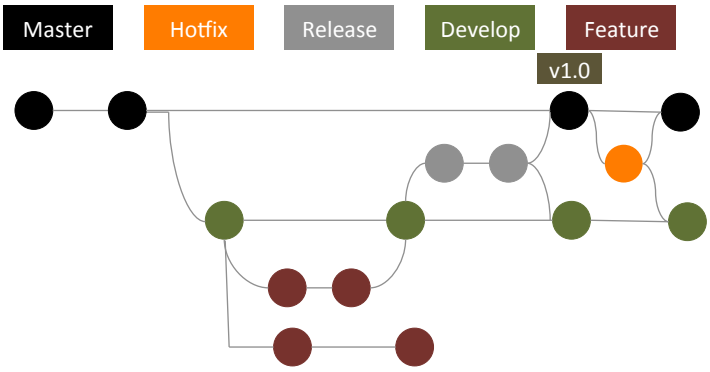
Cómo trabaja GitFlow. Pasos



Se ha de actualizar la rama de develop para incluir el hotfix, para ello:

```
$ git checkout develop
$ git merge issue-#001
$ git push origin develop
$ git branch –d issue-#001
```

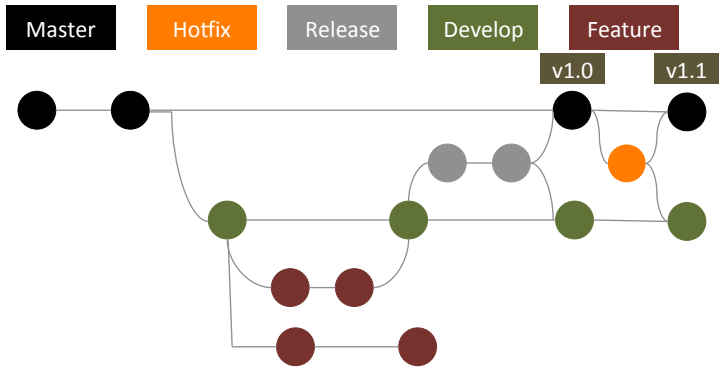
Cómo trabaja GitFlow. Pasos



Añadimos un tag a la rama de producción para que refleje el cambio de versión.

```
$ git checkout master
$ git tag v1.1
$ git push –tags origin master
```

Cómo trabaja GitFlow. Pasos



Se inicia el proceso otra vez:

- Nuevas características.
- Nuevos release.
- Nuevos bug.