Git Flow para gestionar las ramas en Git

> Git Flow es una metodología definida para equipos de desarrollo que detalla la forma en la que gestionar las ramas de nuestro repositorio de Git. Git es una herramienta SCM (Source Code Management) que se utiliza para la gestión de versiones de archivos, lo que permite mantener un histórico de los cambios que se van realizando a los archivos.

Está basado en el modelo de trabajo con ramas de git definido por Vincent Driesse, y se ha convertido en una forma de trabajar muy popular y extendida entre todas las personas que trabajan con git.

Tabla de contenidos

1. Git Flow: Ramas principales (master y develop)

El modelo de Git Flow define dos ramas de trabajo principales:

1.1. Master Branch (Rama principal)

Esta es la rama principal. En esta rama está el código fuente que está listo para ponerse en producción.

1.2. Develop Branch (Rama de desarrollo)

Esta es la rama que se utiliza para los últimos cambios en el desarrollo entregado y que se llevarán próximamente a la rama máster (master branch). Estas ramas existen continuamente. Por lo tanto, se crean desde el inicio del proyecto y no se eliminan mientras que el proyecto esté vivo.

Cuando el código fuente de la rama de desarrollo (develop branch) se encuentra en un punto suficientemente estable y está lista su entrega para su puesta en producción, estos cambios se fusionan con la rama máster (master branch) y se etiquetan con un número de versión.

Por lo tanto, cada vez que fusionamos los cambios con la rama máster (master branch), estamos creando una nueva versión que está lista para ser puesta en producción.

2. Git Flow: Resto de ramas (feature, release y hotfix)

Además de las dos ramas principales, git flow utiliza una serie de ramas para ayudar al desarrollo en paralelo entre todos los miembros del equipo de trabajo.

A diferencia de las dos ramas principales (master y develop), estas ramas tienen un ciclo de vida muy corto. Se crean cuando se necesitan y se destruyen cuando se termina con ellas.

No son tres ramas como tales, sino tres tipos de ramas utilizadas para diferentes usos

2.1 Rama de funcionalidad (feature branch)
2.2 Rama de lanzamiento (release branch)
2.3 Rama de revisión (hotfix branch)

Cada uno de estos tipos de rama sirven para un propósito en particular y git flow define una serie de reglas que hay que respetar al trabajar con ellas. Desde la forma de crearlas hasta su fusión con alguna de las ramas principales.

2.1. Rama de Funcionalidad (feature branch)

Git flow define el uso de esta rama para el desarrollo de nuevas funcionalidades. Cuando se comienza a desarrollar una nueva funcionalidad no tiene por que saberse de antemano cual va a ser la revisión de la que van a terminar formando parte.

Lo mas importante de esta rama es que solo existe mientras se está desarrollando la funcionalidad. Además, esta rama solo existirá dentro de los repositorios locales de git de desarrollo.

2.1.1. Crear una rama de funcionalidad (feature branch)

Esta rama se crea a partir de la rama de desarrollo (develop branch):

git checkout -b feature/nombre-de-funcionalidad

Ejemplo:

```
git checkout -b feature/new-user-registration
```

Mediante este comando creamos la rama feature/nombre-de-funcionalidad a partir de la rama de desarrollo (develop branch). Además, se nos queda la rama feature/nombre-de-funcionalidad como rama activa de nuestro repositorio git.

2.1.2. Fusionar una rama de funcionalidad (merge feature branch)

Una vez terminado el desarrollo de la funcionalidad, tenemos que fusionar ésta rama de funcionalidad (feature branch) con la rama de desarrollo (develop branch)

Primero, nos cambiamos a la rama de desarrollo (develop branch) ya que es la rama sobre la que vamos a fusionar nuestra funcionalidad:

git checkout develop

En segundo lugar, fusionamos la rama de la funcionalidad (feature branch) con la rama en la que estamos, rama de desarrollo (develop branch):

git merge --no-ff feature/nombre_de_funcionalidad

Ejemplo:

```
git merge --no-ff feature/new-user-registration
```

Mediante la opción --no-ff se desactiva el fast-forward en la fusion de la rama. De esta manera, evitamos perder información sobre la existencia de esta rama.

Como ya hemos terminado con el desarrollo de la funcionalidad, eliminamos la rama de la funcionalidad (feature branch):

git branch -d feature/nombre-de-funcionalidad

Ejemplo:

```
git branch -d feature/new-user-registration
```

Por último, sincronizamos estos cambios con el repositorio remoto de git:

git push origin develop

2.2. Rama de Lanzamiento (release branch)

Según git flow, estas ramas se utilizan para fusionar la rama de desarrollo (develop branch) con la rama máster (master branch). En esta rama se pueden hacer los ajustes necesarios para estabilizar el código que se va a fusionar con la rama máster (master branch). También se puede crear esta rama antes de que alguien del equipo de desarrollo cree nuevas ramas de funcionalidad que no van a ser incluidas en esta versión, sin necesidad de que esta rama contenga cambios en el código.

Para ello, estas ramas se crean a partir de la rama de desarrollo (develop branch). Esta rama contendrá todas las funcionalidades de aquellas ramas de desarrollo (feature branch) que se hayan fusionado con la rama de desarrollo (develop branch).

2.2.1. Crear una rama de lanzamiento (release branch)

Las ramas de lanzamiento (release branch) se crean a partir de la rama de desarrollo (develop branch). El nombre de la rama de lanzamiento contendrá el número de versión.

git checkout -b release-num.num[.num] develop

Ejemplo:

```
git chechout -b release-1.2.3
```

```
git checkout -b release-1.3
```

El número de versión que utilicemos deberá ser al menos inmediatamente superior de la última versión que se haya fusionado con la rama máster (master branch).

2.2.2. Fusionar una rama de lanzamiento (merge release version)

Cuando la rama de lanzamiento ya esta lista para convertirse en una autentica versión de lanzamiento, hay que fusionar esta rama en la rama master (master branch) y con la rama de desarrollo (develop branch).

Lo primero que hay que hacer es cambiarse a la rama máster (master branch) para comenzar las fusiones:

git checkout master

En segundo lugar, fusionamos la rama de lanzamiento (release branch) con la rama en la que estamos, rama máster (master branch):

git merge --no-ff release-num.num[.num]

Ejemplo:

```
git merge --no-ff release-1.2.3
git merge --no-ff release-1.3
```

Adicionalmente, deberíamos crear una etiqueta (tag) como referencia a la nueva versión creada:

git tag -a 'num.num[.num]'

Ejemplo:

```
git tag -a '1.2.3'
git tag -a '1.3'
```

Para que los cambios que hayamos realizado sobre esta rama también se mantengan en la rama de desarrollo (develop branch), tenemos que fusionar los cambios con esa rama.

Nos cambiamos a la rama de desarrollo (develop branch):

git checkout develop

Fusionamos la rama de lanzamiento (release branch) con la rama en la que estamos, rama de desarrollo -(develop branch):

git merge --no-ff release-num.num[.num]

Ejemplo:

```
git merge --no-ff release-1.2.3
git merge --no-ff release-1.3
```

Finalmente, solo hay que eliminar la rama de lanzamiento (release branch):

git branch -d release-num.num[.num]

Ejemplo:

```
git branch -d release-1.2.3
git branch -d release-1.3
```

2.3. Rama de Revisión (hotfix branch)

Las ramas de revisión (hotfix branches) se crean debido a la necesidad de actuar inmediatamente frente a algún error detectado en una versión que está en producción. Cuando se detecta un error crítico en una versión de producción que debe ser resuelto inmediatamente, git flow nos dice que tenemos que crear una rama de revisión (hotfix branch) a partir de la etiqueta correspondiente de la rama máster (master branch).

2.3.1. Crear una rama de revisión (hotfix branch)

Como hemos dicho anteriormente, las ramas de revisión (hotfix branches) se crean a partir de la rama máster (master branch):

git checkout -b hotfix-num.num[.num]

Ejemplo:

```
git checkout -b hotfix-1.2.1
```

2.3.2. Fusionar una rama de revisión (merge hotfix branch)

Cuando terminamos de solucionar el problema detectado, la rama de revisión (hotfix branch) hay que fusionarla tanto con la rama máster (master branch) como con la rama de desarrollo (develop branch).

En primer lugar, fusionamos con la rama máster (master branch):

git checkout master git merge --no-ff hotfix-num.num[.num]

Ejemplo:

```
git merge --no-ff hotfix-1.2.1
```

Al igual que con la rama de lanzamiento (release branch), cuando fusionamos una rama de revisión (hotfix branch) con la rama máster (master branch) hay que crear una etiqueta (tag):

git tag -a num.num[.num]

Ejemplo:

```
git tag -a 1.2.1
```

A continuación, hay que fusionar con la rama de desarrollo (develop branch):

git checkout develop

git merge --no-ff hotfix-num.num[.num]

Ejemplo:

```
git merge --no-ff hotfix-1.2.1
```

Cuando en el momento de realizar la fusión de la rama de revisión (hotfix branch) exista una rama de lanzamiento (release branch) creada, en vez de fusionar la rama de revisión (hotfix branch) con la rama de desarrollo (develop branch), fusionaremos la rama de revisión (hotfix branch) con la rama de lanzamiento (release branch).

Por último, solo hay que eliminar la rama de revisión (hotfix branch):

git branch -d hotfix-num.num[.num]

Ejemplo:

```
git branch -d hotfix-1.2.17
```

3. Git Flow: Enlaces

Os dejo dos enlaces que os pueden servir para entender mejor cómo funciona git flow:

Daniel Krummer – Git Flow CheatSheet: Una “chuleta” muy útil para tener toda la información resumida para una consulta rápida de git flow (También disponible en diferentes idiomas). Vincent Driessen – Git Branching Model: Un pdf donde ver visualmente la forma correcta de trabajar con git flow.n