UTN
Facultad Regional Córdoba

TUP

Tecnicatura Universitaria en Programación

# LABORATORIO DE COMPUTACIÓN III

GitHub usando GitFlow

Anexo

2° Año – 3° Cuatrimestre

# Índice

## Uso de GitHub usando metodología GitFlow

En lugar de una única rama **main**, este tipo de flujo de trabajo utiliza dos ramas para registrar el historial del proyecto. La rama **main** o principal almacena el historial de despliegues productivos y la rama **develop** o de desarrollo sirve como rama de integración para las funciones. Asimismo, conviene etiquetar todas las confirmaciones de la rama **main** con un número de versión (ver Semantic Versioning 2.0.0)

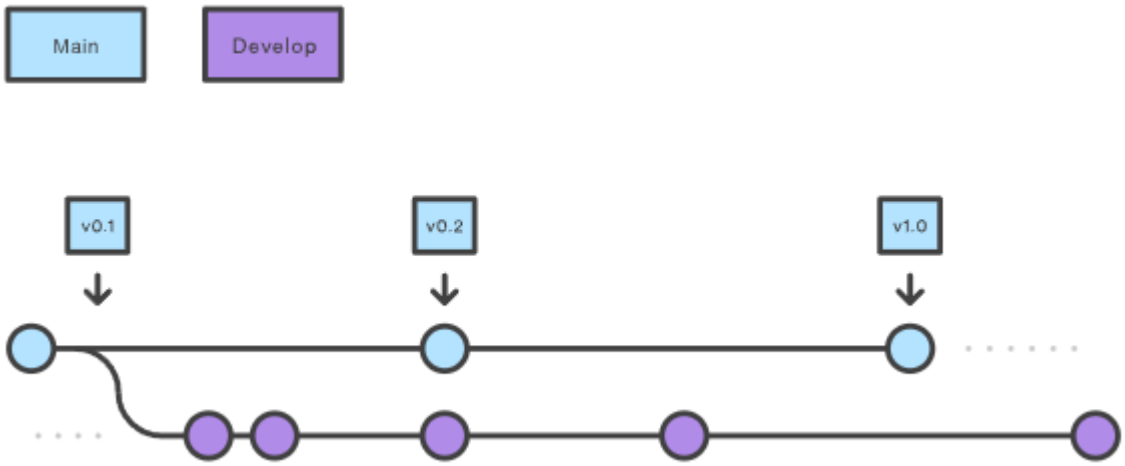


Imagen 1: Elaboración propia

***Nota**: en el presente documento voy a utilizar las palabras **rama** o **branch** y **master** o **main** de forma indistinta ya que no altera la comprensión del mismo.*

La rama **develop** es aquella rama donde convergen las ramas auxiliares y la rama **main** es nuestra rama principal o de producción.

GitFlow define por defecto los siguientes prefijos para las ramas auxiliares, los cuales ayudan a identificar y tener control en el repositorio:

- **feature/** prefijo utilizado cuando generamos un branch a partir de **develop** que luego va a ser mergeado en **develop**
- **release/** prefijo utilizado cuando generamos un branch a partir de **develop** que luego va a ser mergeado en **main**
- **hotfix/** prefijo utilizado cuando generamos un branch a partir de **main** que luego va a ser mergeado en **main**
- **bugfix/** prefijo utilizado cuando generamos un branch a partir de **develop** que luego va a ser mergeado en **develop**

Hay algunos prefijos más que se utilizan, pero estos son los principales.

## Trabajo Practico Integrador (TPI)

En el TPI vamos a trabajar con esta metodología y van a encontrar las ramas principales ya creadas. Para subir cambios a **develop** o **master** se requiere generar un **PR** (pull request) que son los que habilitan a realizar el merge a estas ramas (recuerden que no se permite realizar merge directos sobre **develop** o **master**).

Por este motivo como desarrolladores debemos generar nuestra propia rama de desarrollo (auxiliary branch) desde la cual vamos a realizar todos nuestros cambios. Esta rama o branch se debe crear a partir de **develop.**

A modo de práctica vamos a ver paso a paso cómo realizar nuestros desarrollos sobre el TPI iniciando desde el branch **develop** para luego finalizar en el **main** (nuestro branch productivo).

Como el modelo de Gitflow está pensado para proyectos donde trabajan más de 2 personas el ejemplo plantea que el desarrollo se divide entre dos programadores: ***Persona1*** y ***Persona2***:

### Step 1

La ***Persona1*** genera su branch de trabajo **feature** partiendo de **develop**

```
git checkout develop
git checkout -b feature/buildFunctionalityA
```

La ***Persona2*** también genera su branch de trabajo **feature** partiendo de **develop**

```
git checkout develop
git checkout -b feature/buildFunctionalityB
```

El primer comando es para posicionarnos en el branch **develop** desde donde queremos generar nuestra rama y el segundo comando es para generar el branch **feature** en sí.

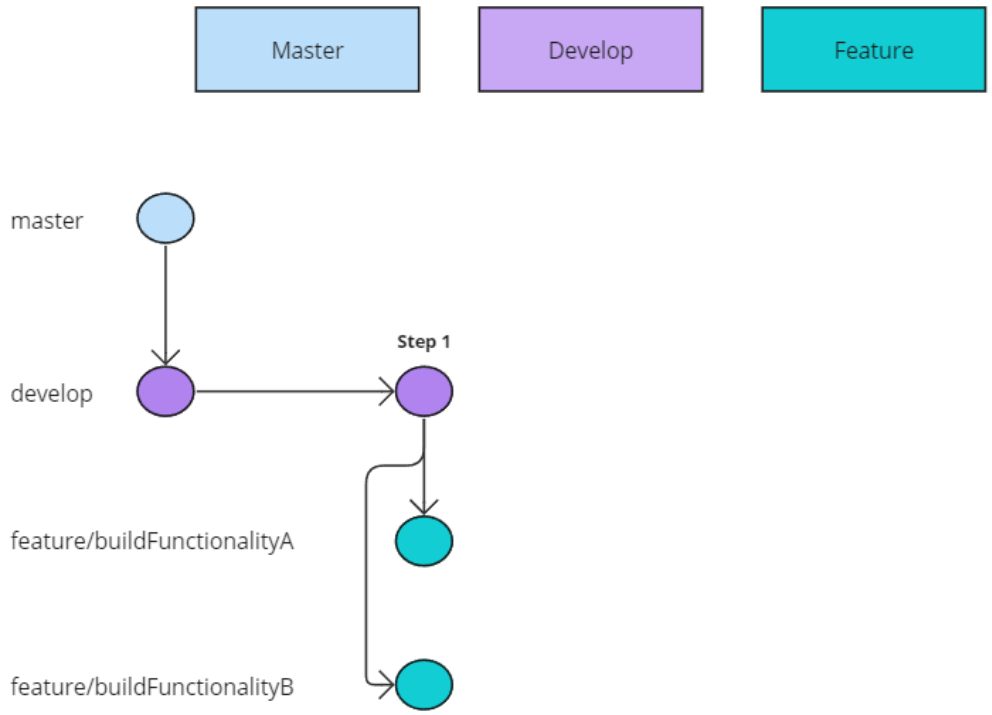


Imagen 2: Elaboración propia

### Step 2

La ***Persona1*** luego de realizar sus cambios termina su primer parte del trabajo y requiere realizar un merge a **develop** a través de un PR

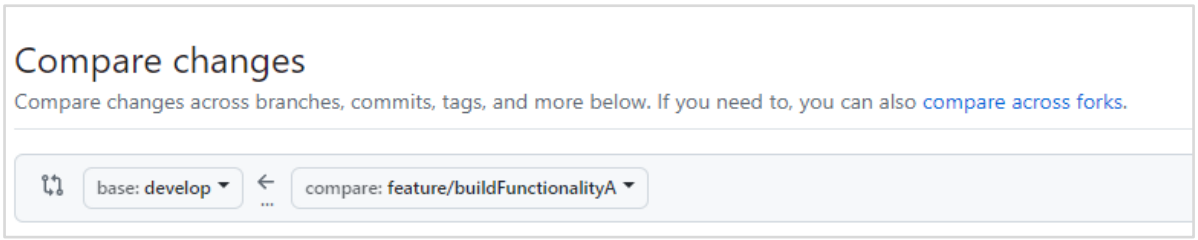


Imagen 3: Elaboración propia

Recordemos que no podemos ejecutar el comando `git merge feature/buildFuncionalityA` porque no tenemos permisos para realizar los cambios de forma directa sobre **develop** o **main**.

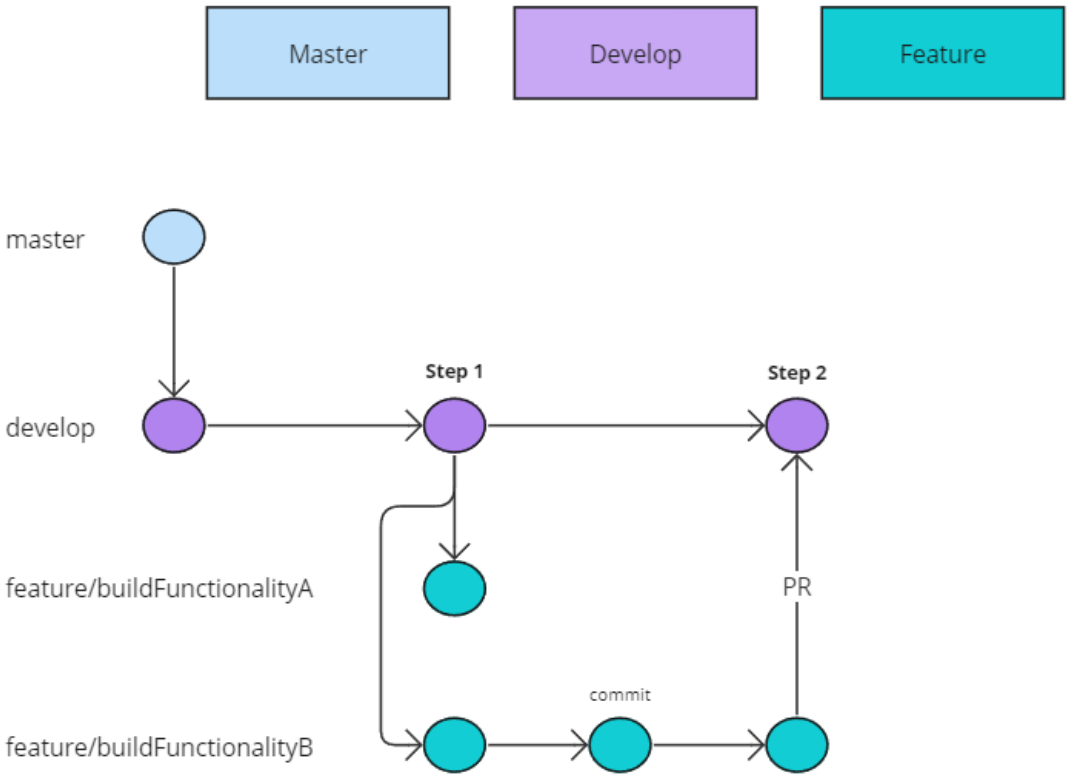


Imagen 4: Elaboración propia

En el gráfico agrego un `commit` a modo de ejemplo para recordar que es importante ir realizándolos con frecuencia durante nuestros desarrollos.

### Step 3

La ***Persona2*** necesita mergear los cambios que la ***Persona1*** subió **develop** para poder finalizar su desarrollo

```
git checkout feature/buildFunctionalityB
git merge feature/buildFunctionalityA
```

El comando `checkout` inicial no es necesario de ejecutar siempre que estemos seguros de estar parados en el repositorio de la ***Persona2*** (donde necesitamos traer los cambios de la ***Persona1***)

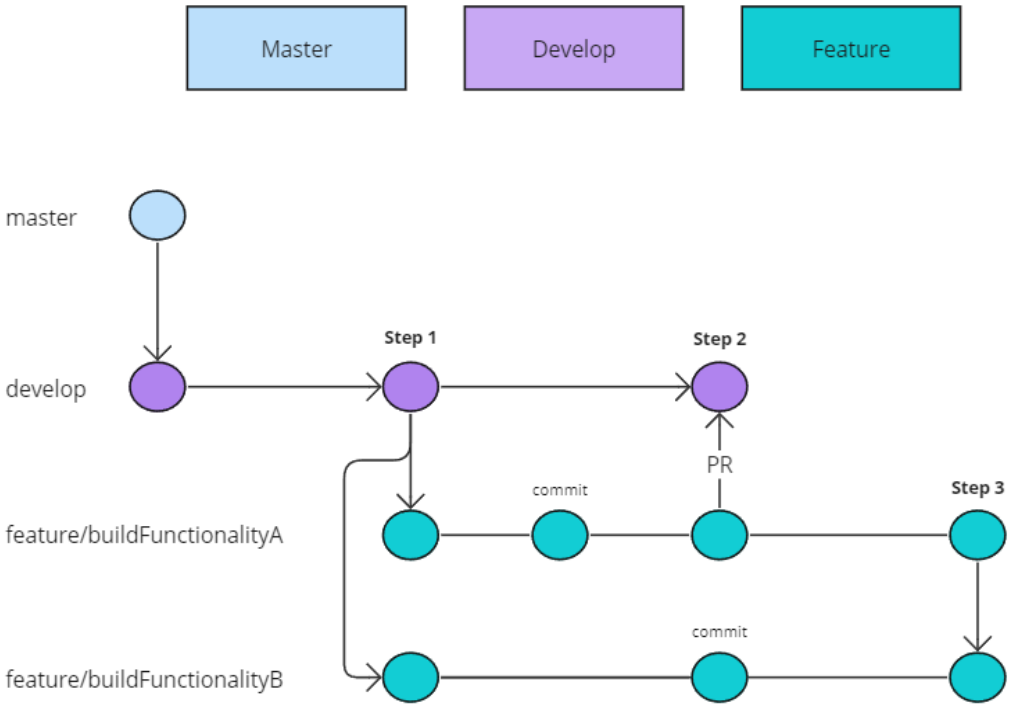


Imagen 5: Elaboración propia

### Step 4

La ***Persona2*** luego de realizar sus cambios también termina su primer trabajo y requiere realizar un merge a **develop** a través de un PR.

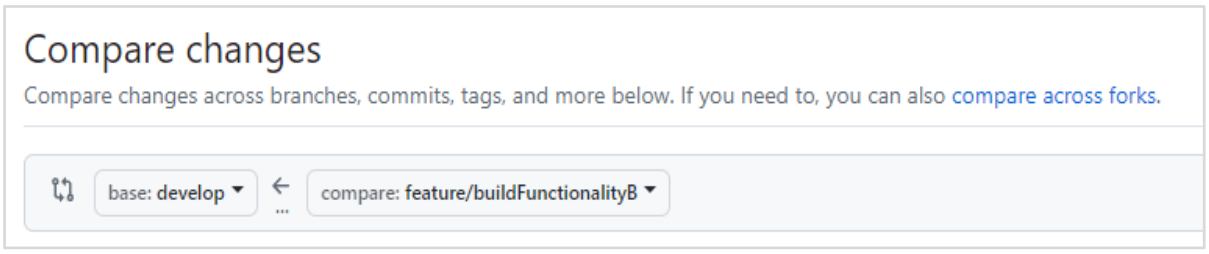


Imagen 6: Elaboración propia

Cuando se aprueba el merge los cambios de **buildFunctionalityB** quedan en **develop**

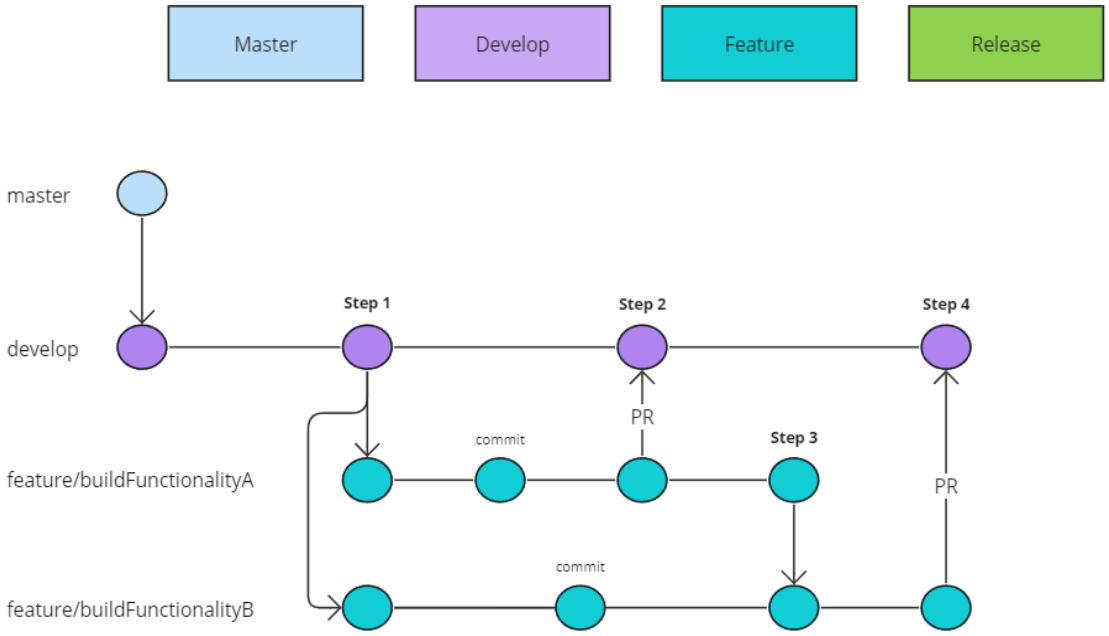


Imagen 7: Elaboración propia

### Step 5

Luego de las pruebas realizadas en conjunto ambos coinciden en que los desarrollos están listos para ser desplegados en producción.

Para subir los cambios a **master** se genera un nuevo branch **release** partiendo de **develop**

```
git checkout develop
git checkout -b release/v1.0.0
```

Notar que el nombre del branch además de contener el prefijo **release** también tiene un número de versión. Esto no es obligatorio, pero sí necesario para mantener un orden y una nomenclatura simple que indique (sin conocer el código) si lo que se está desplegando son funcionalidades nuevas, mejoras o corrección de bugs (ver Semantic Versioning 2.0.0)

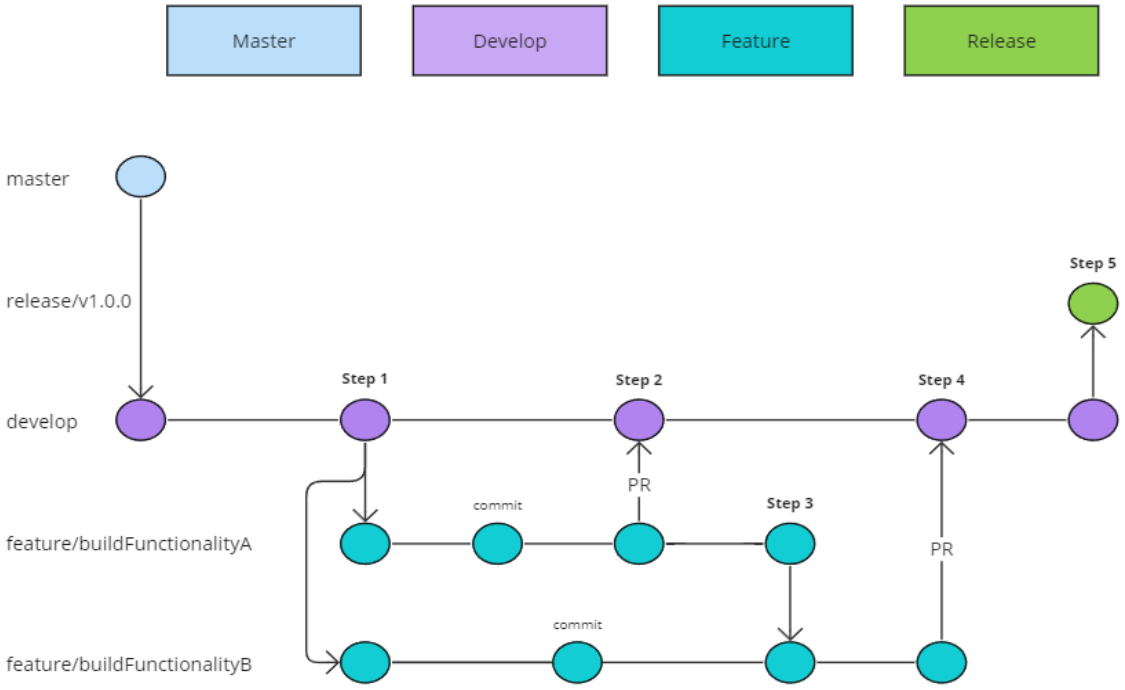


Imagen 8: Elaboración propia

### Step 6

Una vez generado el branch se solicita el merge a **master** a través de un PR.

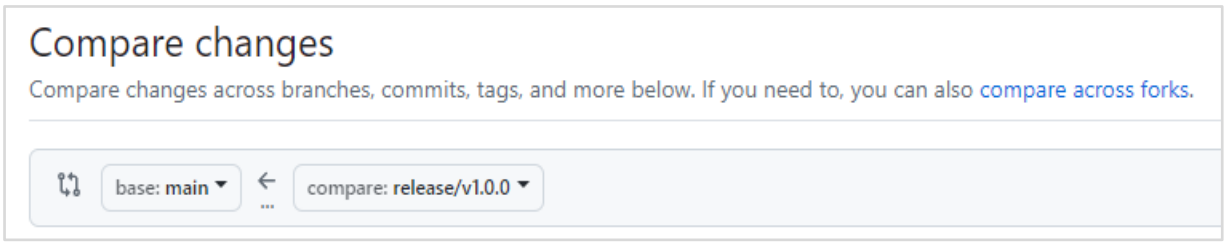


Imagen 9: Elaboración propia

Cuando se aprueba el merge los cambios de **develop** quedan en **main**

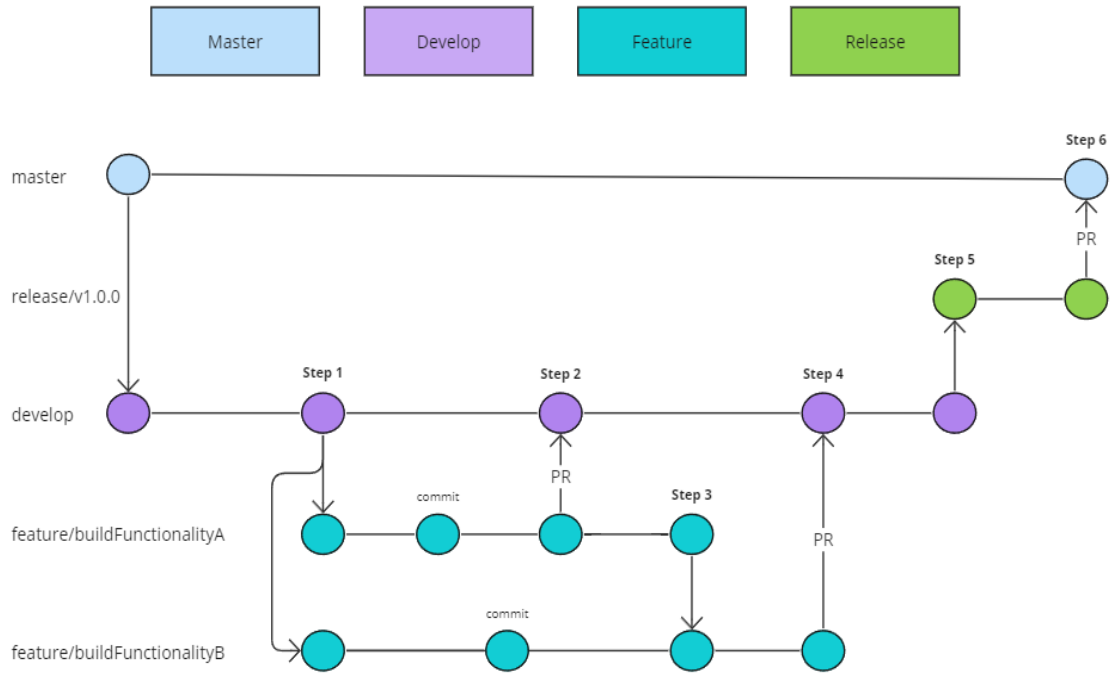


Imagen 10: Elaboración propia

## Step 7

Una vez realizado el merge se va a generar un nuevo branch **backport** desde **main** a **develop** para asegurar que ambas ramas se mantengan sincronizadas.

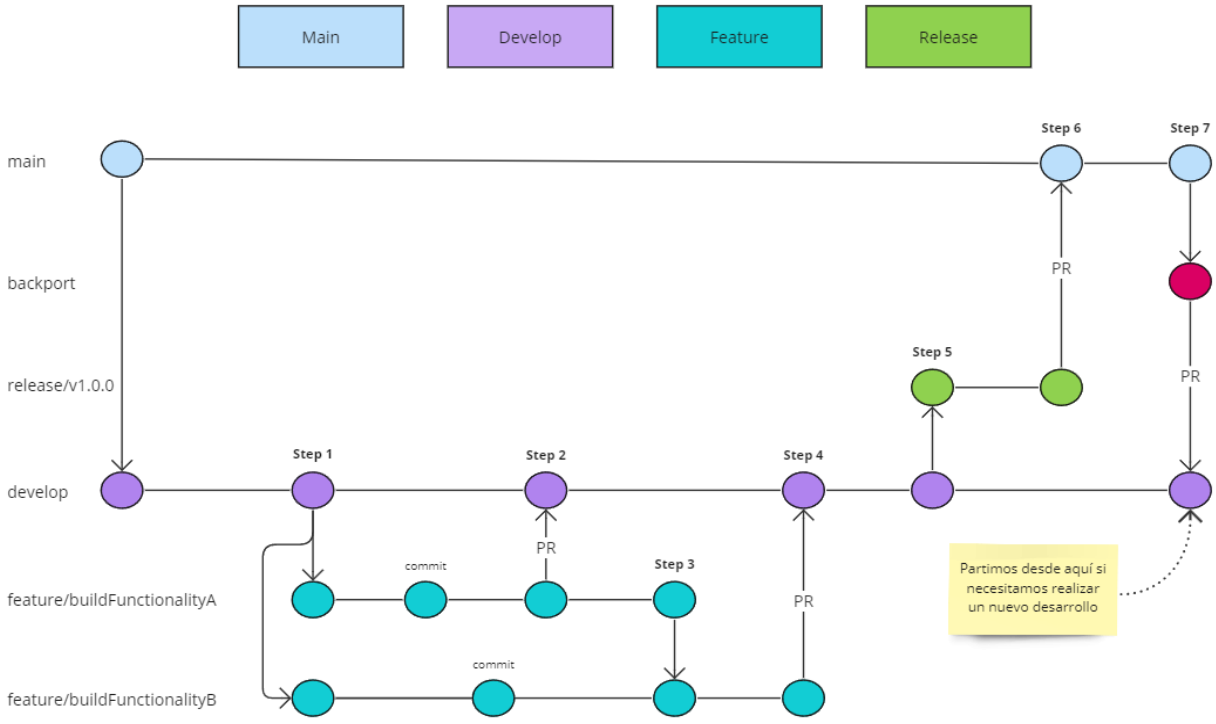


Imagen 11: Elaboración propia

Estos simples pasos se repiten indefinidamente (con algunas variaciones) durante todo el ciclo de vida de desarrollo de nuestro repositorio.

## Conclusiones

Se recomienda usar Gitflow cuando:

- El equipo de trabajo está conformado por más de dos (2) personas.
- Se emplean metodologías ágiles.
- El proyecto tiene cambios frecuentes y se requiere actualizar el ambiente de producción garantizando continuidad en la operación.
- El proyecto tiene un nivel de complejidad considerable.
- Se desea tener un proceso de soporte a errores efectivo con actualizaciones rápidas.

Resumen del flujo general de Gitflow:

1. Se crea una rama **develop** a partir de **main**.
2. Se crea una rama **release** a partir de la rama **develop**.
3. Se crean ramas **feature** a partir de la rama **develop**.
4. Cuando se termina una rama **feature**, se fusiona (merge) en la rama **develop**.
5. Cuando la rama **release** está lista, se fusiona (merge) en la rama **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 (merge) tanto en **develop** como en **main**.