CONTROL DE VERSIONES AVANZADO

Necesidad de ramas

Las ramas con las cuales vamos a trabajar son las siguientes:

- main / master
- hotfix
- release
- develop
- features

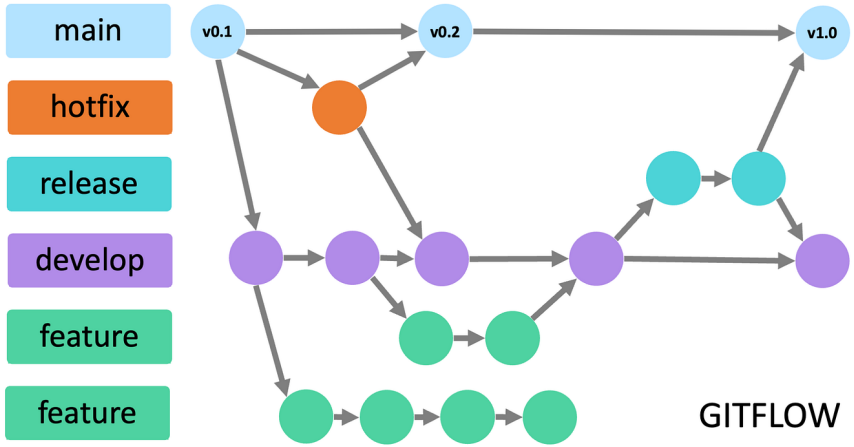
Sigue una estrategia muy similar a GitFlow, la cual permite el trabajo en equipo en un proyecto. Su funcionamiento es muy sencillo, cada miembro tiene asignada una feature que hace referencia a un issue, una vez terminada la funcionalidad se une a develop, posteriormente develop y release se unen, al igual que main y release. Esto permite crear un histórico lineal.

> **Note**
>
> Las ramas de funcionalidad una vez terminadas deben ser eliminadas.

> **Note**
>
> Las ramas de funcionalidad deben nombrarse como:
> `nombrePersona/numIssue`
>
> Un ejemplo sería: pedro/4
> Quiere decir que en el issue 4, en esa funcionalidad está trabajando Pedro.

Cada miembro todos los días antes de empezar a trabajar debería obtener los cambios del remoto y ver donde se encuentra `develop` y así ver si es necesario obtener los cambios más recientes de los compañeros, por si hubieran conflictos con su código ir solucionando los poco a poco y no todos de golpe una vez terminada la feature cuando se va a unir a develop. Es decir debería hacer un `git fetch && git merge`.

Representado de forma gráfica un histórico sencillo podría verse de la siguiente forma.



Definición de las ramas

- develop: Lo que se ha desarrollando y el equipo de desarrollo dice que ya funciona. Cuando se quiera se pasa a `release`.
- release: Lo que se esta probando en un entorno similar al de producción. Cuando está probado, se copia a `master`.
- master: Lo que ya se puede instalar en el producción.
- rama de funcionalidad *(feature)*: Cada vez que alguien quiere hacer algo , crea una rama desde `develop` y cuando acaba la fusiona en `develop`.

> **Tip**
>
> En la rama de funcionalidad se pueden hacer todos los commits que queramos ya que luego se van a unificar en uno solo. Es lo que llamamos "microcommits". Es así ya que son commits *parciales* que se hacen para acabar la tarea pero finalmente en la rama `develop` queremos que sean un solo commit.
>
> Ese único commit debe tener el formato anteriormente explicado (Control de versiones (Git) > Mensajes de commit):
> `type(#issue): titulos`

Pasos para hacer una modificación

- Crea la rama `feature` desde `develop`.

```
git branch pedro/4
git switch pedro/4
```

- Realizar la tarea con tantos commits como sea necesario.

```
touch file1.txt
git commit -a -m "cambio 1"

touch file2.txt
git commit -a -m "cambio 2"

touch file3.txt
git commit -a -m "cambio 3"
```

- Mergear la rama `develop` en `feature` ya que así se resuelven los problemas de mergeo en la rama `feature`

```
git switch pedro/4
git merge develop # Resolver conflictos si hay
```

- Ir a la rama `develop` y mergear la rama `feature` con `--squash`

```
git switch develop
git merge --squash pedro/4
git commit -m "feat(#12):Login validation"
```

- Borrar la rama `feature`

```
git branch -d pedro/4 # local
git push origin --delete pedro/4 # remoto
```

- Subir la rama `develop`

```
git push develop
```

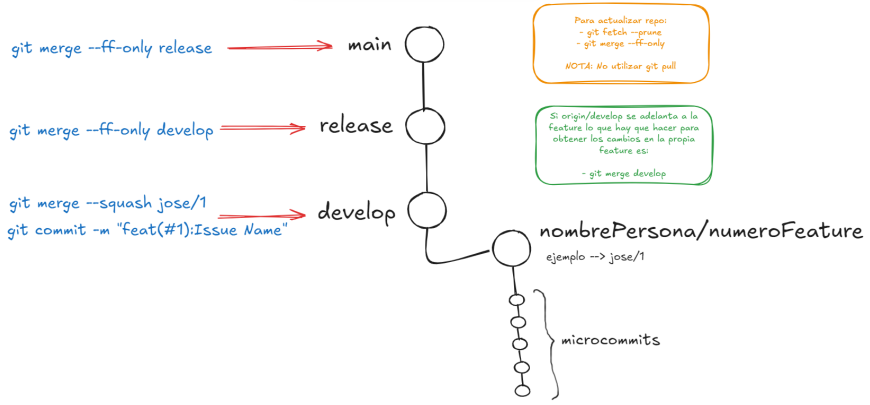
- Mergear la rama `develop` en `release` y subir `release`

```
git switch release
git merge --ff-only develop
```

- Mergear la rama `release` en `master` y subir `master`

```
git switch main
git merge --ff-only release
```

Estrategia de trabajo Git



Note

```
git merge --squash pedro/4
```

Se utiliza para juntar todos los microcommits en un solo commit que habrá que indicar. Se usa para pasar los microcommits de una rama feature a develop

Note

```
git merge --ff-only develop
```

La opción `--ff-only` es usada para que si la unión no es *Fast-Forward* no se llegue a hacer y falle indicándolo.

La unión puede ser Fast-Forward que indica que no hay problemas o en el caso de haber conflictos serían ramas divergentes.

Cuando usar MERGE y REBASE y de qué tipo utilizarlo

COMANDO	CARACTERÍSTICAS	CUANDO USARLO
`git rebase rama`	Pone los commits en la nueva rama siempre después de los que ya hay, es decir que los reordena	Para incluir los cambios de origin/develop en develop. Pasos: `git fetch --prune` `git switch develop` `git rebase origin/develop`
`git merge --squash rama`	Junta todos los "microcommits" de la rama en uno solo en la rama destino. Es necesario acto seguido el commit	*De una rama feature a develop.*
`git merge --ff-only rama`	Nunca crea nuevos commits en la rama destino pero falla si es necesario crear un nuevo commit, por lo que `--ff-only` ayuda a detectar si hay algún problema	*De develop a Release.* De release a master o para actualizar cualquier rama desde GitHub cuando no hay commits nuevos en local
`git merge rama`	Es el merge normal que puede crear nuevos commits	*De una rama develop a feature.* Para resolver los conflictos desde nuestra rama feature ya que nos da igual si hay nuevos commits

Mini-Script de automatización

- Bajarse lo último de las 3 ramas.

```
git fetch --prune && git switch develop && git merge --ff-only origin/develop &
```

- Hacer un merge de las 3 ramas

```
git switch release && git merge --ff-only develop && git switch master && git m
```

- Subir las 3 ramas

```
git switch master && git push && git switch release && git push && git switch d
```