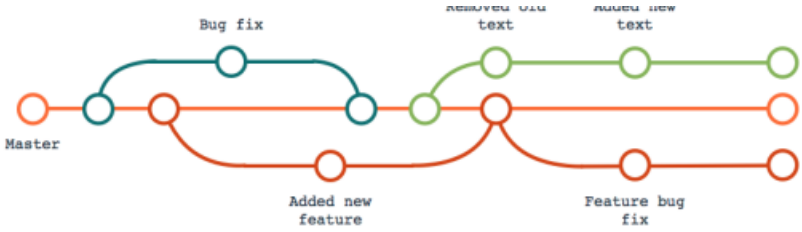
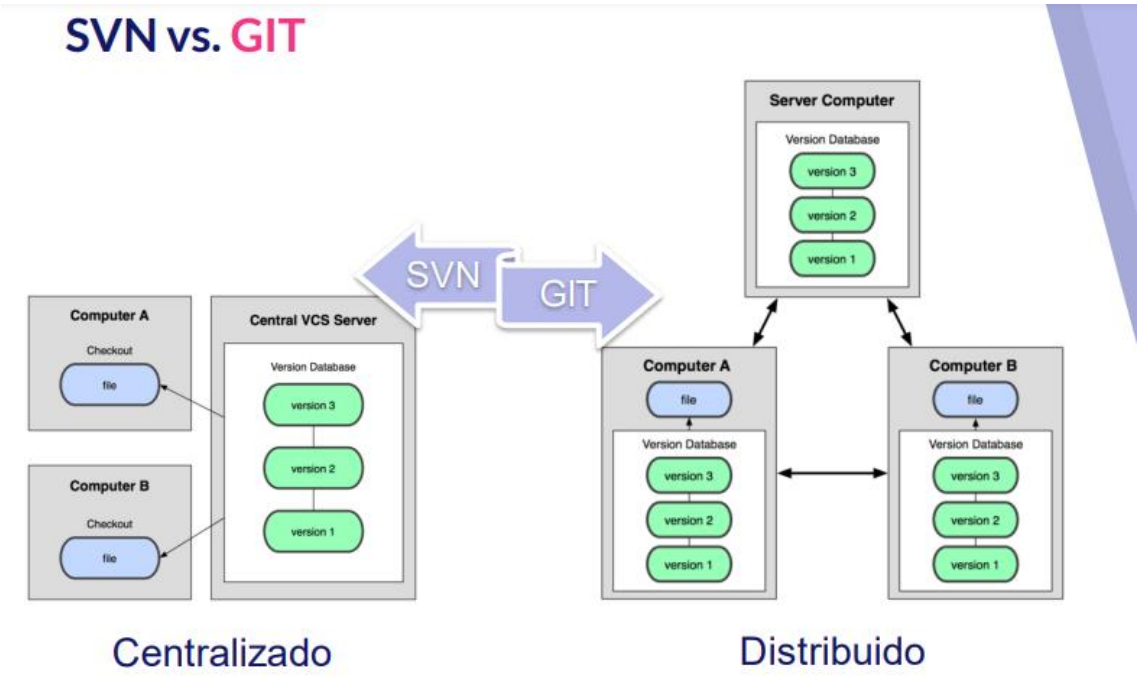
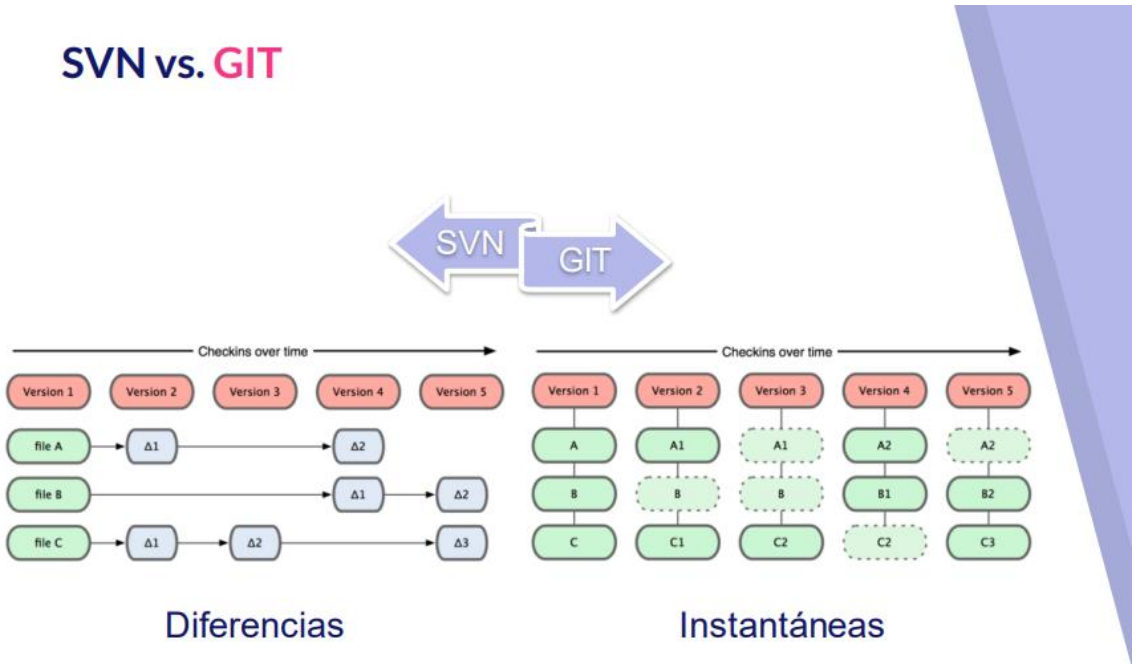
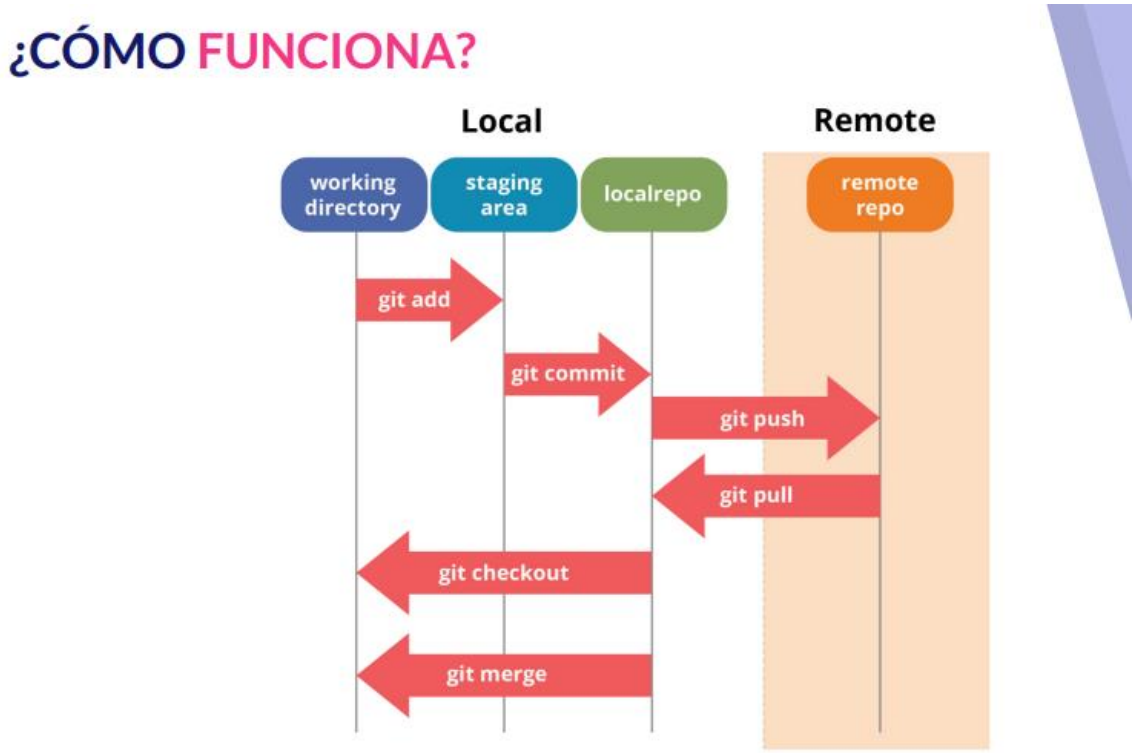
Que es GIT..

Git es un sistema de control de versiones, que nació para dar soporte al core Linux, cuya principal característica es que es capaz de mantener una gran cantidad de código fuente distribuido y gestionado por equipos de envergadura, de forma rápida y sencilla.









GIT FLOW

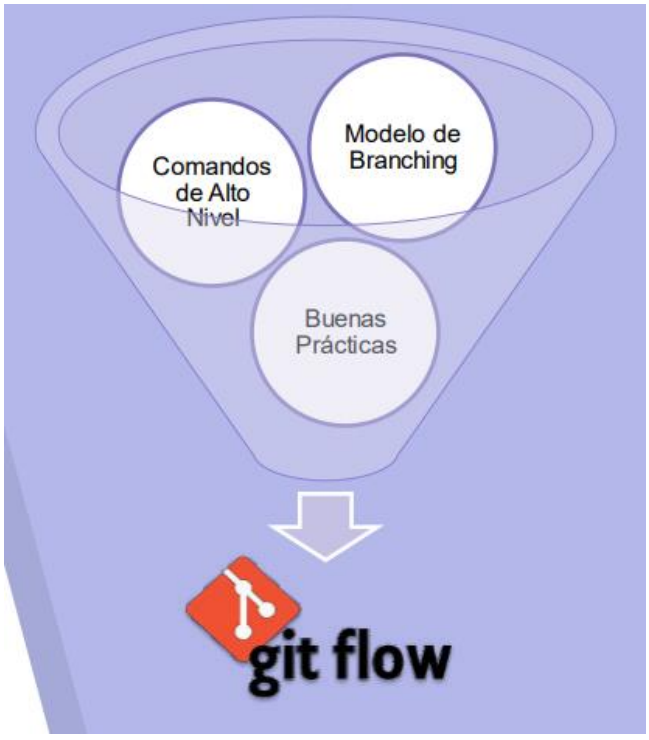
GitFlow es un modelo de branching, que propone una estrategia de ramificación y generación de versiones de productos usando un repositorio Git.

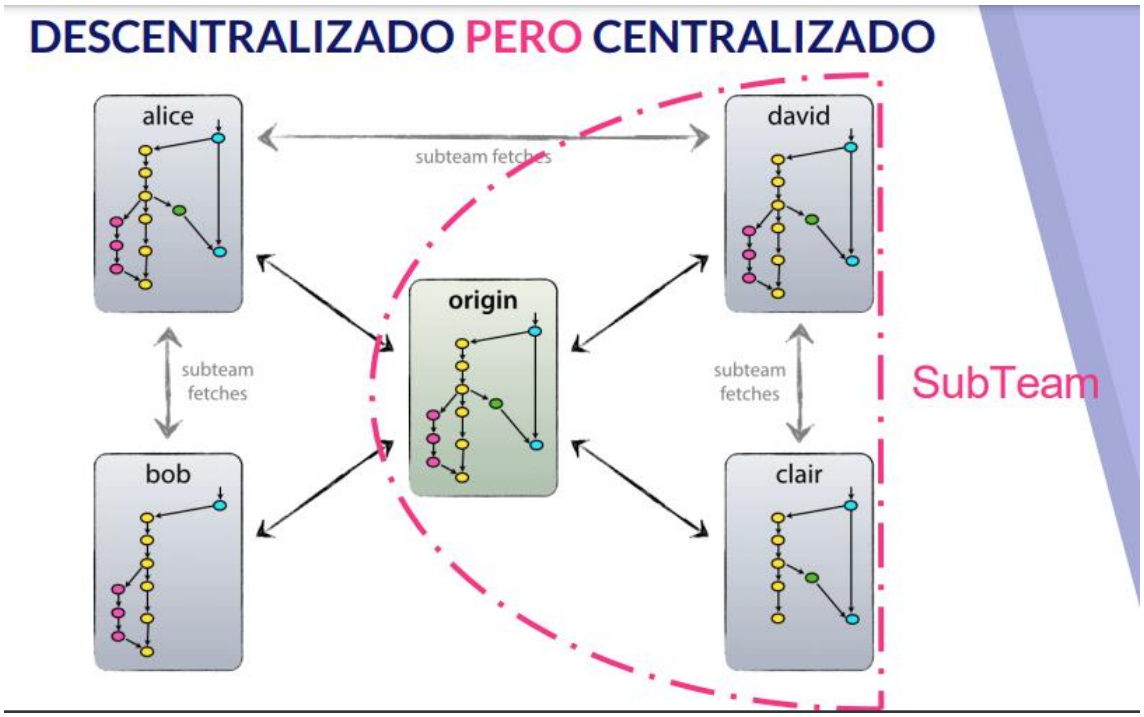
GitFlow es un modelo de branching que propone una estrategia de ramificación y de generación de versiones de productos usando un repositorio GIT. Ayuda a los equipos a asegurar la calidad del código y la eficiencia en la integración en equipos de más de un miembro.

Descentralizado pero Centralizado

Como dijo Vincent Driessen, GIT ha revolucionado la forma de hacer software. Antes con SVN o CVS los merges daban un poco de miedo, ya que los conflictos se generaban en el único repositorio que tienen, el remoto, y directamente sobre el trunk. Eso hace los conflictos extensibles a todos los miembros del equipo y además se generan directamente sobre la rama principal. Digamos que no favorece mucho el branching y en trabajo colaborativo, sin embargo, con git es extremadamente fácil, al contar con el repositorio local.

Por eso decimos que con git-flow partimos de un repositorio centralizado, pero que a la vez es descentralizado. Esto se debe a que todos tenemos nuestro repositorio local, como hemos comentado, pero a su vez apuntamos a un repositorio remoto llamado "origin". Toda la comunicación entre los desarrolladores tiene que pasar por origin y no de un desarrollador a otro, lo que refleja todos los cambios que hace cada uno y les permite integrar código en sus repositorios locales y detectar conflictos de forma temprana.





Flujo de trabajo de Git Flow

GitFlow define dos tipos de ramas, las principales y las auxiliares. Las principales diferencias entre estos dos grupos de ellas son:

Ramas Principales (master y develop):

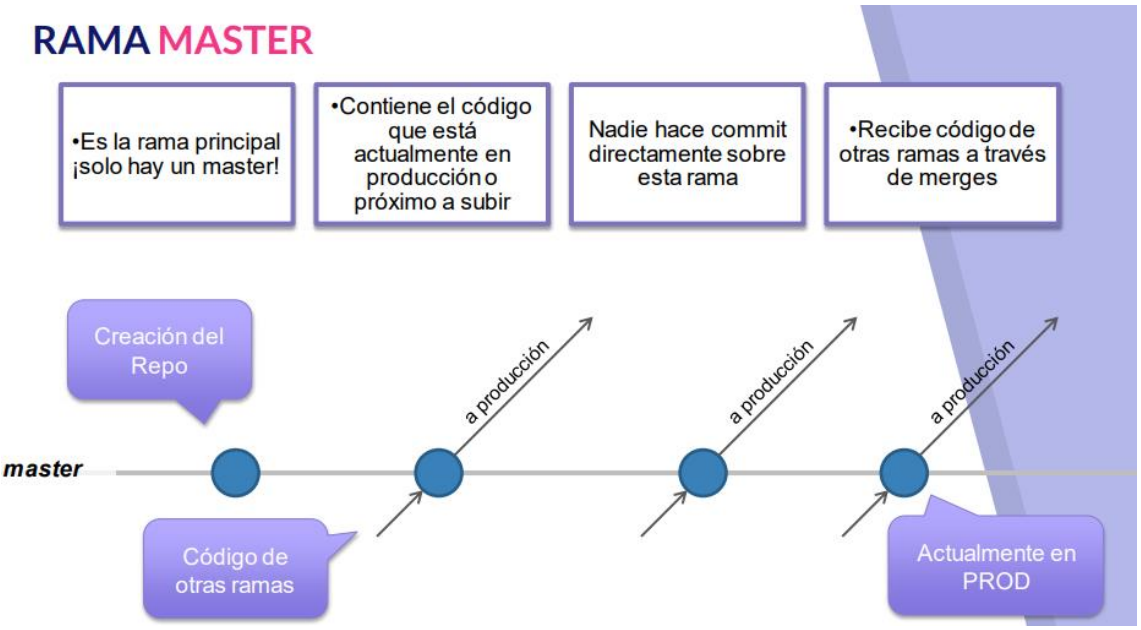
- No se pueden instanciar
- Su nombre no tiene complementos, se llaman master y develop “a secas”
- No se puede commitear código directamente sobre ella, para no perder el control de los cambios, solo recibe merges de otras ramas
- Ramas Auxiliares (feature, release y hotfix):
- Se pueden instanciar tantas veces como características, versiones y defectos haya. No hay límite.
- Su nombre consta de una raíz, que corresponde al tipo de rama, más la característica implementada o versión. Este complemento la hace inconfundible
- El objetivo es trabajar directamente sobre ella, subiendo código código a la rama. Luego se mergea con las ramas principales.

Rama master

La rama master es la rama principal del repositorio y contiene el código que está actualmente en producción o próximo a subir en una situación estable.

No se hace commit de código de ella, salvo circunstancias muy especiales y por alguien con altos conocimientos en el producto.

Por regla general, recibe merges de ramas release o hotfix. En esta rama están etiquetadas todas las versiones del producto.



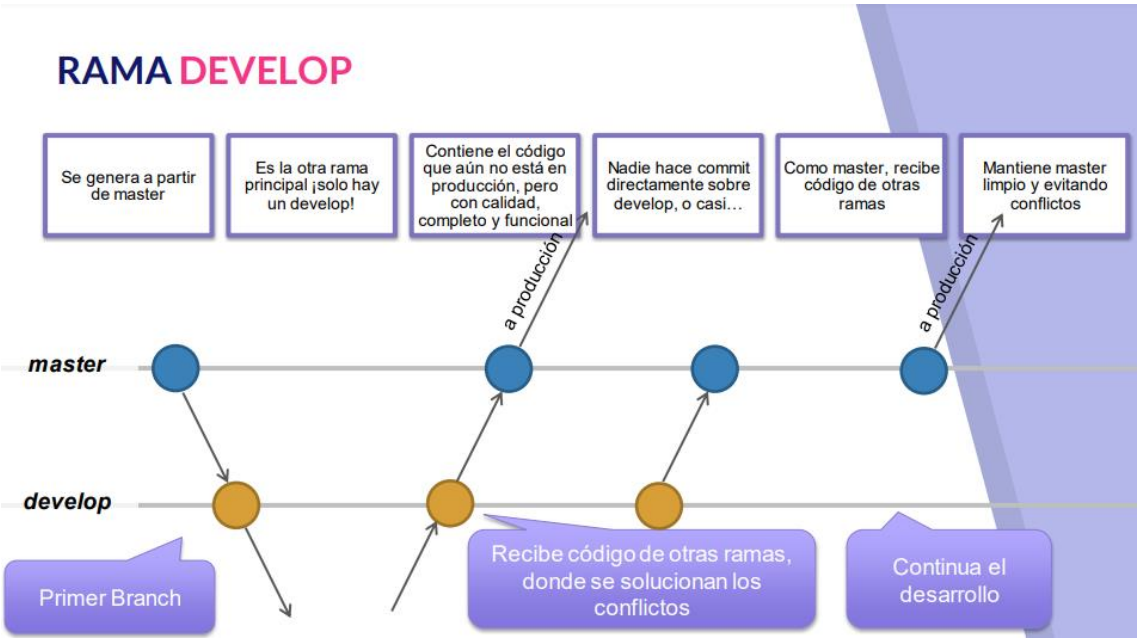
Rama de Develop

La rama develop es la rama principal del repositorio que contiene, además del contenido de master, características desarrolladas que aún no están en producción.

En esta rama se está desarrollando código de forma continua, eso sí, todo el contenido que recibe a través de merges de ramas features principalmente, está testeado y verificado, por lo que es código estable.

No se hace commit de código de ella, salvo circunstancias muy especiales, al igual que en la rama master.

El principal objetivo de esta rama es evitar conflictos es master, por lo que el código se depura en esta rama primero.

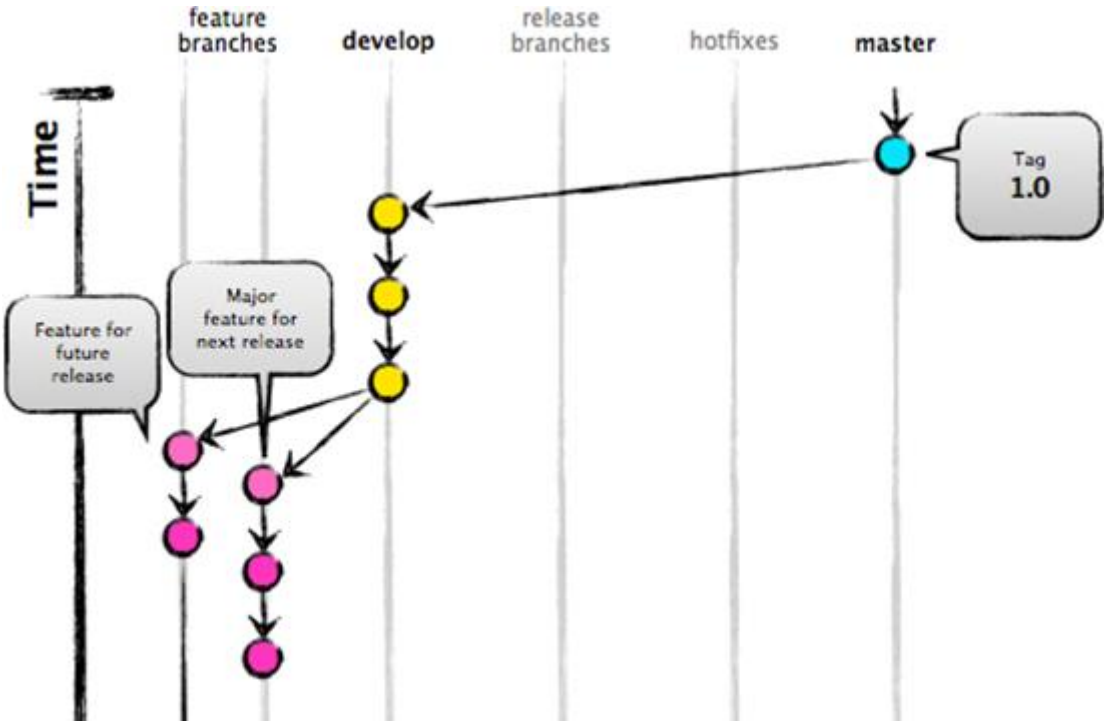


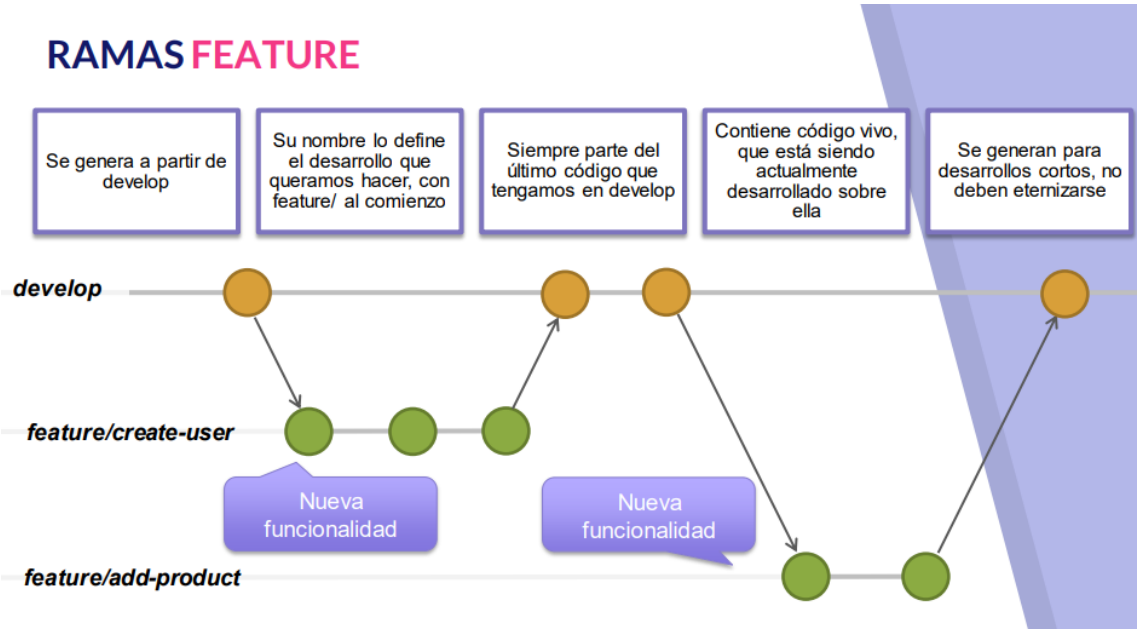
Rama Feature

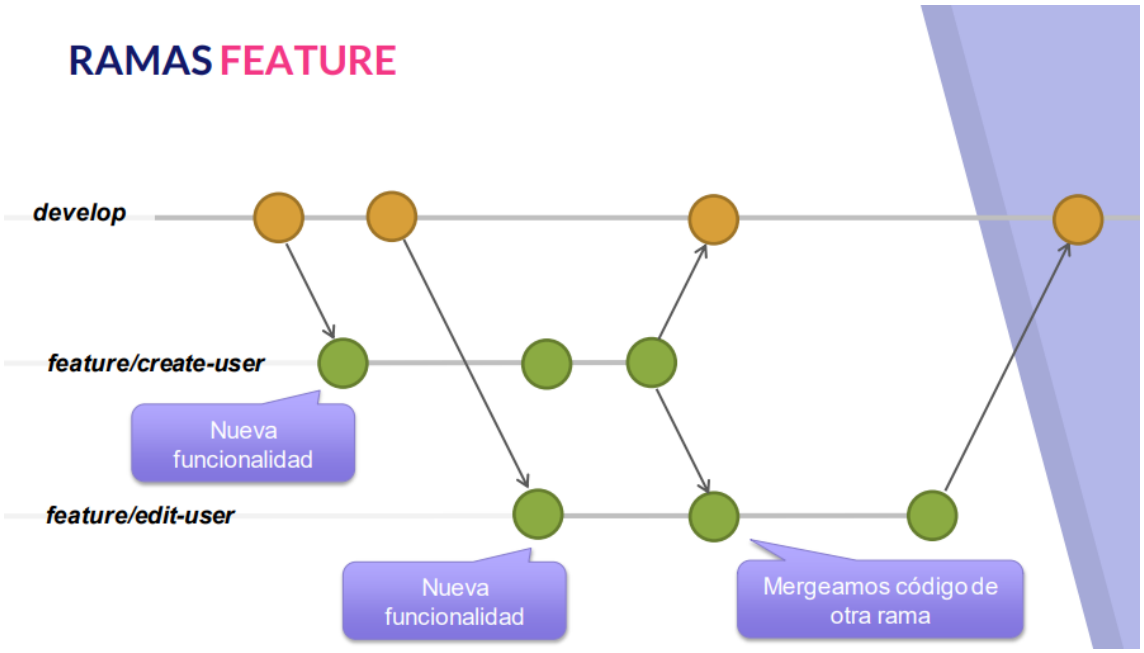
Cada vez que se va a desarrollar una nueva funcionalidad o modificar una existente, se genera una rama de tipo feature, a partir de develop, cuyo nombre está formado por esta palabra y por la característica a implementar, por ejemplo: “feature/nueva-interfaz”.

Sobre ella se sube el código a través de commits del equipo que debe ser probado antes de mergear con develop, por lo que durante este proceso hay una funcionalidad en curso codificándose, pudiendo trabajar varios miembros a la vez en ella.

Una buena práctica es hacerlas lo más atómicas posibles, pudiendo generar más de una para cubrir una característica y mergeando código de unas a otras.





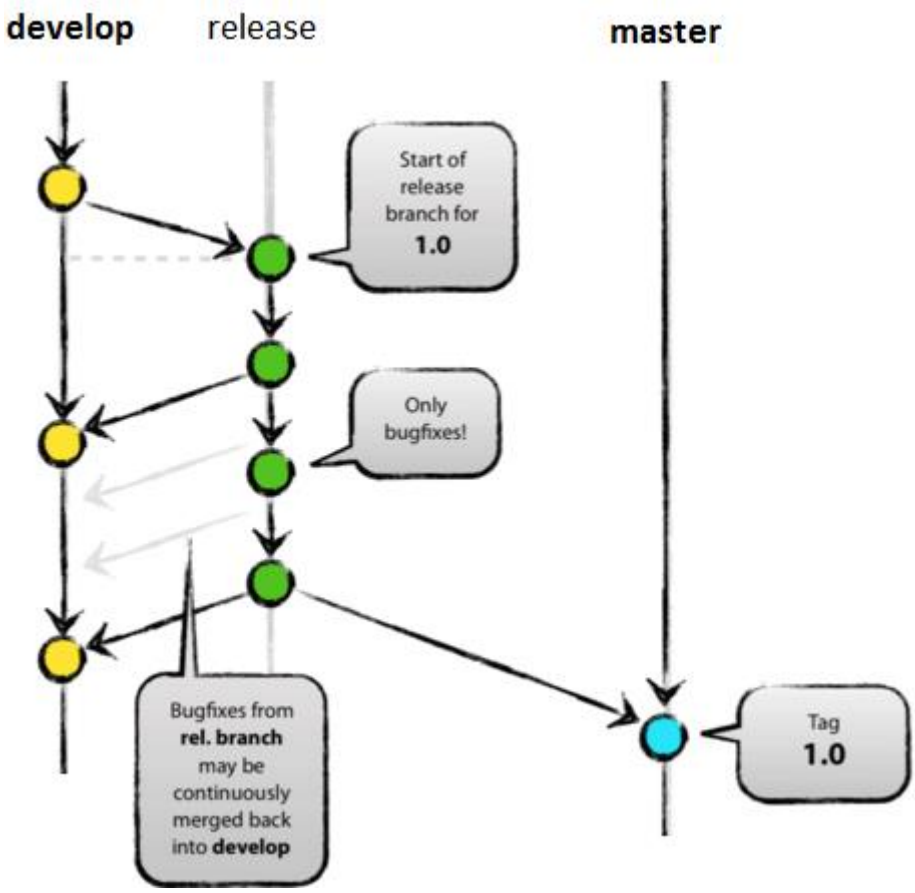


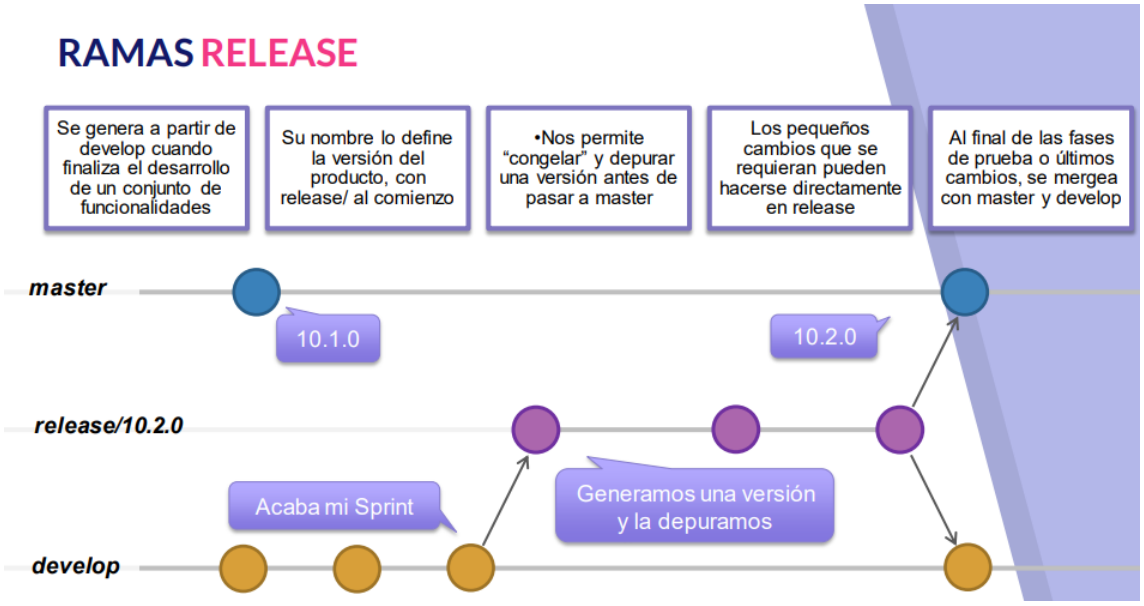
Rama Release

Estas ramas se generan cuando quiere realizarse la tentativa de entrega de una versión, habitualmente cuando se aproxima la finalización de un sprint.

Esto congela los desarrollos, que prosiguen en develop, y agrupa una serie de nuevas características en la nueva rama release, que puede llamarse, por ejemplo: “release/v2.2.0”.

Sobre ella pueden realizarse pequeños cambios, corrección de defectos, establecer versiones de producto… pero no desarrollar nuevas características. Una vez finalizado y testeado, se mergea con master, como nueva versión que es candidata a subir a producción; y con develop, ya que hay defectos corregidos que deben volcarse a la rama de desarrollo.


develop
release
master
Start of release branch for 1.0
Only bugfixes!
Bugfixes from rel. branch may be continuously merged back into develop
Tag 1.0

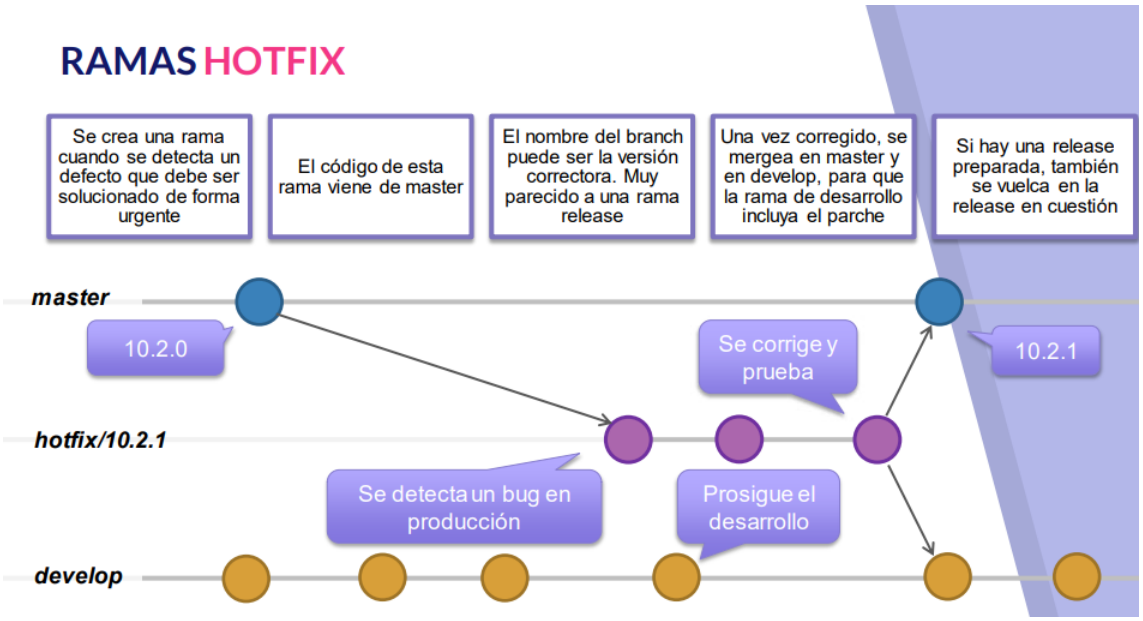

RAMAS RELEASE
Se genera a partir de develop cuando finaliza el desarrollo de un conjunto de funcionalidades
Su nombre lo define la versión del producto, con release/ al comienzo
•Nos permite "congelar" y depurar una versión antes de pasar a master
Los pequeños cambios que se requieran pueden hacerse directamente en release
Al final de las fases de prueba o últimos cambios, se mergea con master y develop
master
10.1.0
10.2.0
release/10.2.0
Acaba mi Sprint
Generamos una versión y la depuramos
develop

Rama Hotfix

Estas ramas se generan habitualmente cuando se detecta un defecto en producción que debe ser corregido con agilidad. Para ello se genera una rama de tipo hotfix, que parte del código existente en master.

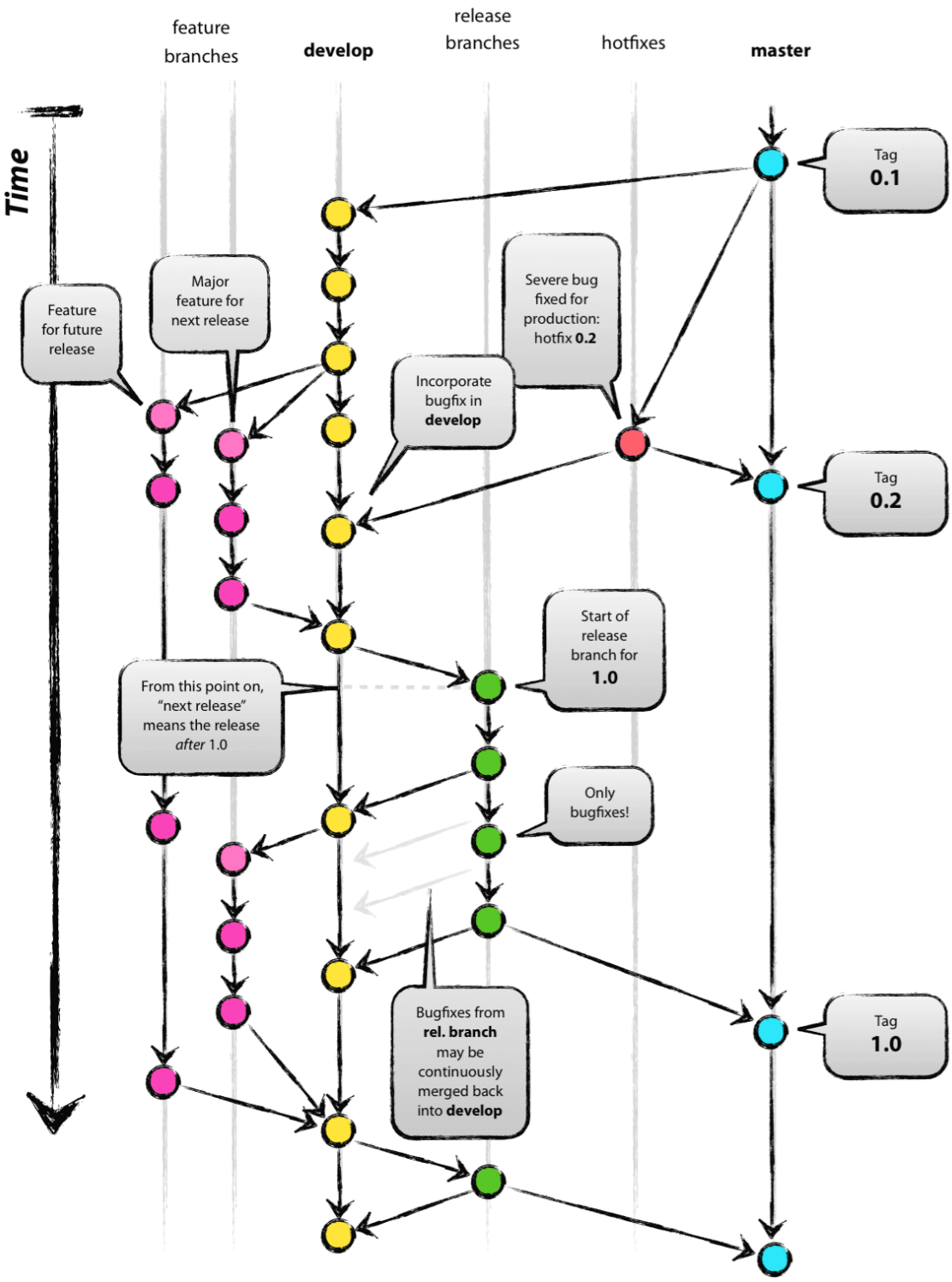
Sobre ella pueden realizarse las correcciones que sean necesarias para arreglar el bug, para luego volver a mergear con master, añadiéndole un nuevo tag de versión.

También es necesario mergear con develop para que el defecto también quede corregido en desarrollo.



Uniendo Todo

Uniéndolo todo, el proceso cobra sentido, y todos los tipos de ramas se complementan los unos a los otros para alcanzar los objetivos que propone GitFlow.


feature branches
develop
release branches
hotfixes
master
Time
Tag 0.1
Feature for future release
Major feature for next release
Severe bug fixed for production: hotfix 0.2
Incorporate bugfix in develop
Tag 0.2
Start of release branch for 1.0
From this point on, "next release" means the release after 1.0
Only bugfixes!
Bugfixes from rel. branch may be continuously merged back into develop
Tag 1.0

Introducción a los Comandos

GitFlow también es una librería que dispone de una serie de comandos git de alto nivel que se simplifica el uso de los comandos originales y lo adaptan al flujo de trabajo propuesto anteriormente.

Aun así, **hay puristas que prefieren utilizar los comandos originales para sentir más el control sobre el repositorio**.

Instalación de paquetes

Gitflow puede instalarse en cualquier equipo Windows, Linux o macOS de la siguiente forma:

macOS
Git instalado y funcional: `$git`

Instalación con Homebrew: `$ brew install git-flow-avh`

Instalación con Macports: `$ port install git-flow-avh`

Instalación correcta: `$ git flow / $ git flow version`

Linux
Git instalado y funcional: `$git`

Instalación de la extensión en Debian / Ubuntu: `$ apt-get install git-flow`

Instalación de la extensión en CentOS / RedHat / Fedora: `$ sudo dnf install gitflow`

Instalación correcta: `$ git flow / $ git flow version`

Windows
Git instalado y funcional: `$git`

Git for Windows incluye GitFlow por defecto

Instalación correcta: `$ git flow / $ git flow version`

Para iniciar un repositorio desde cero solo es necesario escribir los siguientes comandos:

```
$ mkdir gitflow-Project
$ git flow init
$ git branch
```

Esto generará un repositorio inicial basado en gitflow y con las ramas principales ya creadas.

```
$ git flow init
Initialized empty Git repository in ~/project/.git/
No branches exist yet. Base branches must be created
Branch name for production releases: [master]
Branch name for "next release" development: [develop]
How to name your supporting branch prefixes?
Feature branches? [feature/]
Release branches? [release/]
Hotfix branches? [hotfix/]
Support branches? [support/]
Version tag prefix? []
$ git branch
* develop
  master
```

Comandos de la Rama Master y Develop

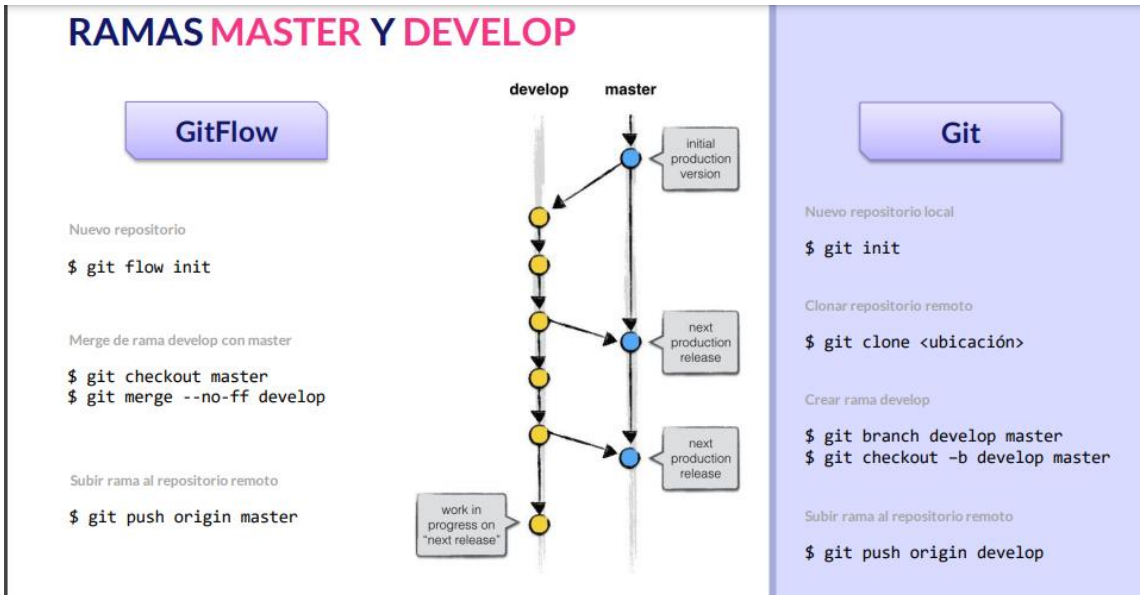
Nuevo repositorio `$ git flow init`

Merge de rama develop con master

```
$ git checkout master
```

```
$ git merge --no-ff develop
```

Subir rama al repositorio remoto `$ git push origin master`



Commandos de Rama Feature

Comandos de Ramas Feature

Comandos disponibles a nivel de feature `$ git flow feature help`

Visualizar **lista** de ramas de tipo feature `$ git flow feature list`

Inicia una **nueva rama** de tipo feature a partir de develop `$ git flow feature start <nombre>`

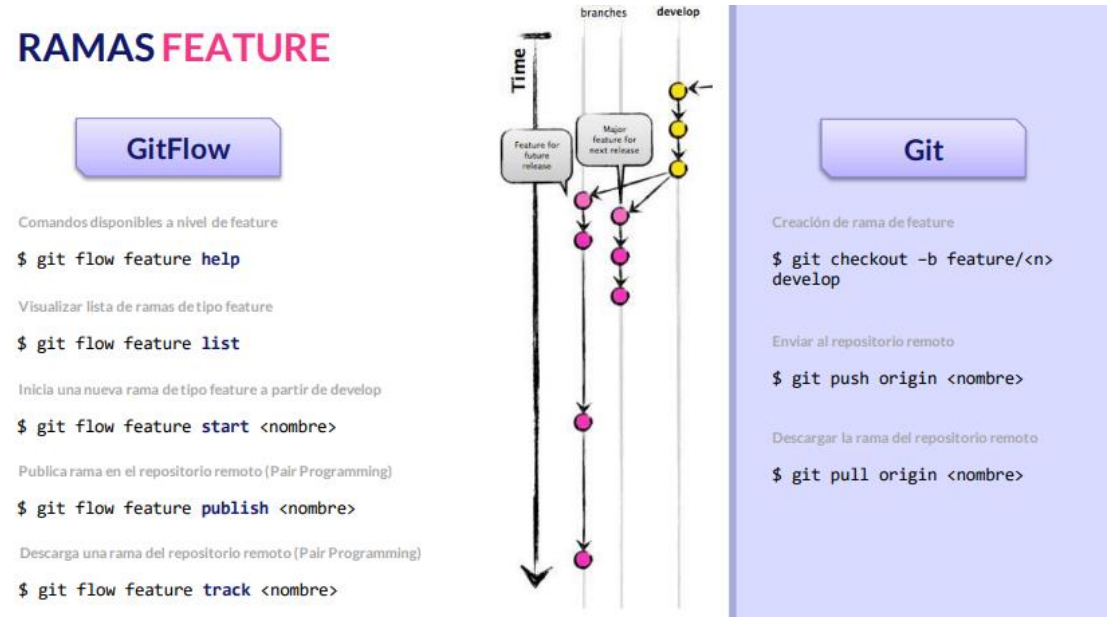
Publica rama en el repositorio remoto (Pair Programming) `$ git flow feature publish <nombre>`

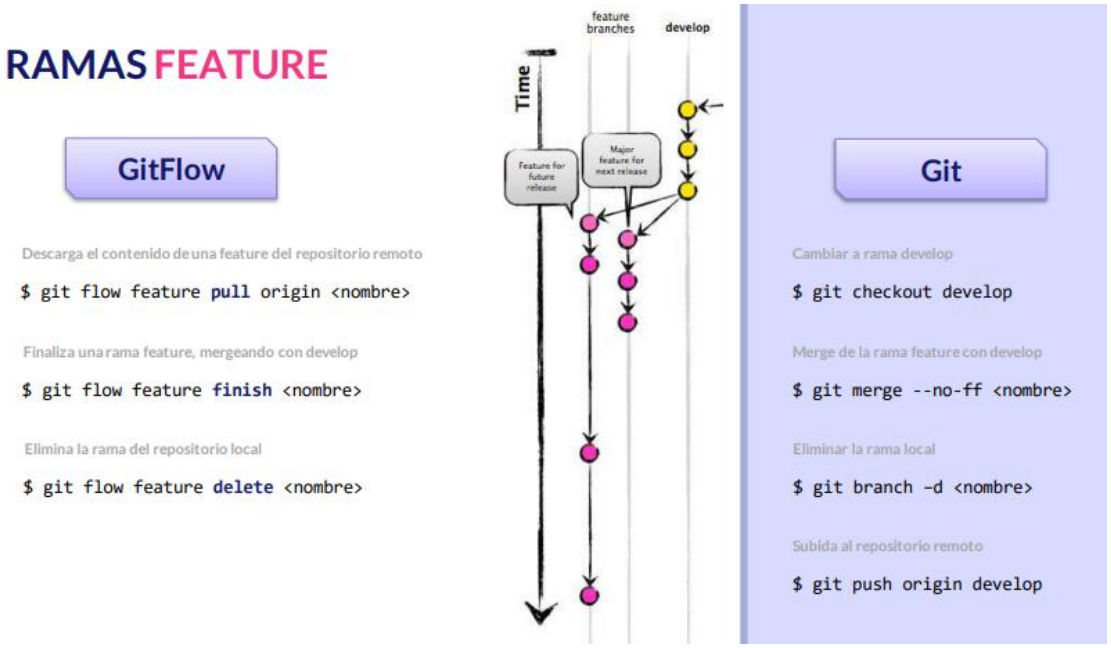
Descarga una rama del repositorio remoto (Pair Programming) `$ git flow feature track <nombre>`

Descarga el contenido de una feature del repositorio remoto `$ git flow feature pull origin <nombre>`

Finaliza una rama feature, mergeando con develop `$ git flow feature finish <nombre>`

Elimina la rama del repositorio local `$ git flow feature delete <nombre>`





Rama Release

Comandos disponibles a nivel de release `$ git flow release help`

Visualizar **lista** de ramas de tipo release `$ git flow release list`

Inicia una **nueva rama** de tipo release a partir de develop `$ git flow release start <versión>`

Publica rama en el repositorio remoto (Pair Programming) `$ git flow release publish <versión>`

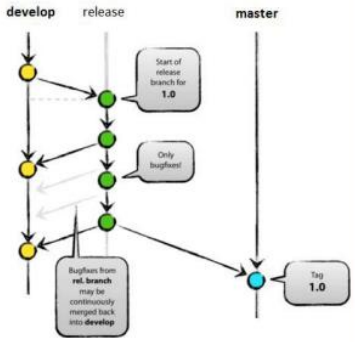
Descarga una rama del repositorio remoto (Pair Programming) `$ git flow release track <versión>`

Finaliza una rama release, mergeando con develop y master `$ git flow release finish <versión>`

Elimina la rama del repositorio local `$ git flow release delete <versión>`

RAMAS RELEASE

GitFlow



Comandos disponibles a nivel de release

```
$ git flow release help
```

Visualizar lista de ramas de tipo release

```
$ git flow release list
```

Inicia una nueva rama de tipo release a partir de develop

```
$ git flow release start <versión>
```

Publica rama en el repositorio remoto (Pair Programming)

```
$ git flow release publish <versión>
```

Descarga una rama del repositorio remoto (Pair Programming)

```
$ git flow release track <versión>
```

Git

Creación de rama de release

```
$ git checkout -b release/<v>
develop
```

Enviar al repositorio remoto

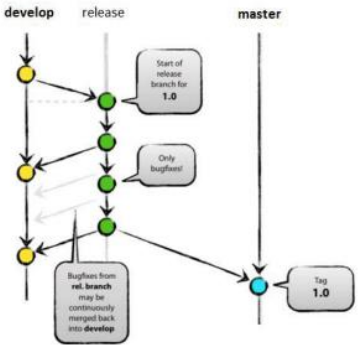
```
$ git push origin <versión>
```

Descargar la rama del repositorio remoto

```
$ git pull origin <versión>
```

RAMAS RELEASE

GitFlow



Finaliza una rama release, mergeando con develop y master

```
$ git flow release finish <versión>
```

Elimina la rama del repositorio local

```
$ git flow release delete <versión>
```

Git

Mergea la rama release con master

```
$ git checkout master
$ git merge --no-ff release /<v>
$ git tag -a <versión>
$ git push origin master
```

Mergea la rama release con develop

```
$ git checkout develop
$ git merge --no-ff release /<v>
$ git push origin develop
```

Eliminar la rama local

```
$ git branch -d <versión>
```

Comando Hotfix

Comandos disponibles a nivel de hotfix `$ git flow hotfix help`

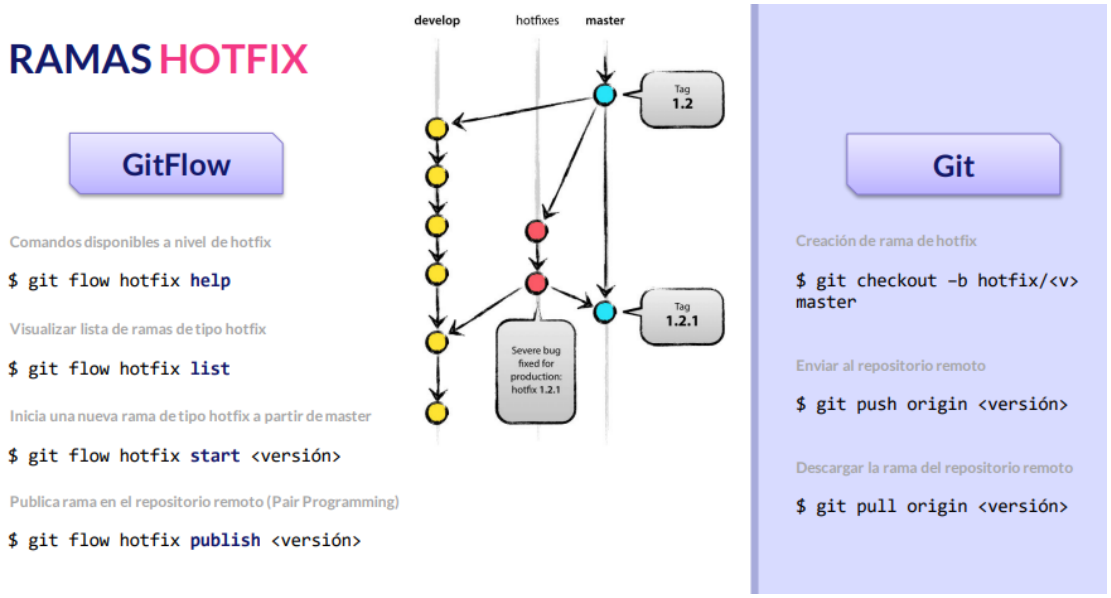
Visualizar **lista** de ramas de tipo hotfix `$ git flow hotfix list`

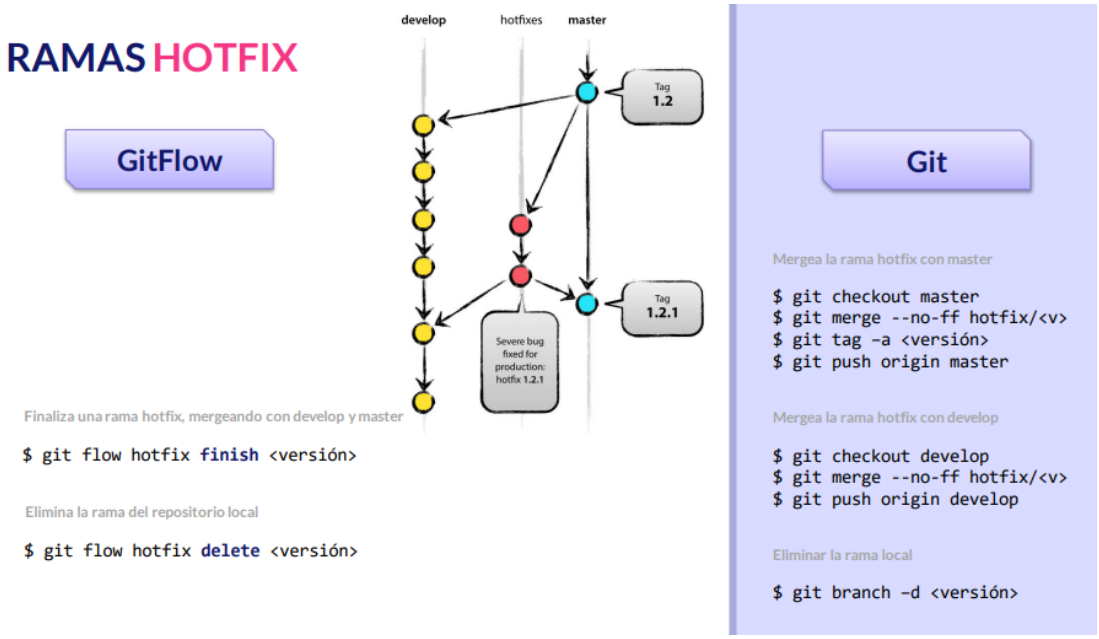
Inicia una **nueva rama** de tipo hotfix a partir de master `$ git flow hotfix start <versión>`

Publica rama en el repositorio remoto (Pair Programming) `$ git flow hotfix publish <versión>`

Finaliza una rama hotfix, mergeando con develop y master `$ git flow hotfix finish <versión>`

Elimina la rama del repositorio local `$ git flow hotfix delete <versión>`





Que son las Revisiones cruzadas.

La revisión por pares (o Pair Review) la **revisión de tu código fuente** por **el resto de miembros de tu equipo** con el objetivo de **encontrar defectos** en fases tempranas y **proponer mejoras**, además de **controlar la aplicación de reglas** de codificación.

Esta práctica, cuando ya llega a un estado de madurez óptimo, supone un importante ahorro de costes derivados en defectos futuros, mejora el feedback en el producto, reduce la frustración del equipo y ayuda a mantener unos estándares y reglas de codificación vivos.

Proceso

El proceso es muy simple, **cuando un desarrollador se encuentra próximo a mergear su código con otra rama, genera una "Pull Request"** o petición de revisión, por la cual otros miembros del equipo dan su aprobación o rechazo al cambio introducido.

Una vez que se cumpla la regla de revisión establecida por el equipo en consenso, se puede hacer el merge. Estas reglas pueden ser del tipo, revisar por un número

de miembros, un senior tiene que dar aceptación... y son establecidas por el propio equipo.