



# git-flow

Jes s Amieiro Becerra

Version 0.1.0, 2016-09-22

#  ndice

1. Presentaci�n	�1
1.1. �Qu� es Git?	�1
1.2. �Por qu� Git?	�1
1.3. �Qu� es git-flow?	�1
1.4. Errores	�1
1.5. Contacto	�2
2. Flujos de trabajo	�3
2.1. git-flow	�3
2.1.1. Las ramas principales	�4
2.1.2. Ramas de soporte	�5
Ramas feature (topic)	�6
Ramas release	�8
Ramas hotfix	�10
3. Instalaci�n	�12
3.1. Instalaci�n en Linux	�12
3.1.1. Ubuntu	�12
3.1.2. CentOS	�14
4. Comandos de git-flow	�16
4.1. git flow init	�16
4.1.1. Proyecto nuevo	�16
4.1.2. Proyecto existente	�18
4.2. git flow feature	�21
4.2.1. git flow feature list	�21
4.2.2. git flow feature start	�21
4.2.3. git flow feature publish	�22
4.2.4. git flow feature pull	�24
4.2.5. git flow feature finish	�25
4.3. git flow release	�28
4.3.1. git flow release list	�28
4.3.2. git flow release start	�29
4.3.3. git flow release publish	�30
4.3.4. git flow release finish	�31
4.4. git flow hotfix	�34
4.4.1. Subcomandos	�34
4.4.2. git flow hotfix list	�34
4.4.3. git flow hotfix start	�34

4.4.4. git flow hotfix publish. . . . .	35
4.4.5. git flow hotfix finish . . . . .	36

# 1. Presentaci3n

Este es un libro sobre git-flow, un conjunto de extensiones para Git, que proporcionan una abstracci3n de alto nivel sobre Git.

## 1.1. ¿Qu3 es Git?

Como seguramente sabr3s, Git es un sistema de control de versiones distribuido desarrollado por Linux Torvalds en el a3o 2005 y que se ha hecho tremendamente popular gracias a servicios como [GitHub](#), [Bitbucket](#) o [GitLab](#) y a su amplia aceptaci3n en proyectos importantes como el [Kernel](#) de Linux, [Android](#), [Ruby on Rails](#), [Eclipse](#), [GNOME](#), [KDE](#), [Qt](#), [Perl](#) o [PostgreSQL](#) o por empresas como [Google](#), [Facebook](#), [Microsoft](#), [Twitter](#), [LinkedIn](#) o [Netflix](#).

## 1.2. ¿Por qu3 Git?

Si eres programador, desarrollador web, administrador de sistemas, dise3ador, 3 es muy probable que en alg3n momento de tu trabajo te encuentres con un proyecto en el que tengas que colaborar con otras personas usando Git. Puede que trabajes solo pero que te interese tener un seguimiento y control de tu trabajo. En estos dos casos y en muchos m3s un conocimiento m3s o menos profundo de Git te permitir3 ser mucho m3s productivo en tu trabajo y, sobre todo, evitar muchos de los problemas con los que se encuentra a menudo la gente que no trabaja con un sistema de control de versiones.

Si tu 3mbito de trabajo es t3cnico y a3en no usas Git, cuando lleves unos meses us3ndolo te preguntar3s c3mo es posible que no lo hubieras empezado a usar antes.

## 1.3. ¿Qu3 es git-flow?

git-flow es un conjunto de extensiones para Git que proporcionan una abstracci3n de alto nivel. Est3n basadas en un post de [Vincent Driessen](#), un ingeniero de software holand3s, que describe el modelo de desarrollo basado en Git que lleva usando en varios proyectos. Posteriormente desarroll3 un software, publicado en GitHub, en que implementaba la funcionalidad descrita en el post.

## 1.4. Errores

Aunque he tratado de evitar cualquier tipo de error, puede que encuentres alguno. En ese caso te agradecer3a que me lo notificaras a trav3s de cualquiera de los m3todos de contacto indicados en el cap3tulo [Contacto](#).

## 1.5. Contacto

Puedes ponerte en contacto conmigo a través de cualquiera de los métodos de contacto indicados en la dirección web <http://www.jesusamieiro.com/contactaconmigo/>

## 2. Flujos de trabajo

### 2.1. git-flow

Como ya he comentado, git-flow es un conjunto de extensiones para Git que proporcionan una abstracción de alto nivel. Están basadas en un post de [Vincent Driessen](#), un ingeniero de software holandés, que describe el modelo de desarrollo basado en Git que lleva usando en varios proyectos. El post se titula [A successful Git branching model](#), ya da a entender cuál es el ámbito de especialización: un correcto flujo en la ramas de trabajo.



*Imagen 1. Modelo de ramas de git-flow.*

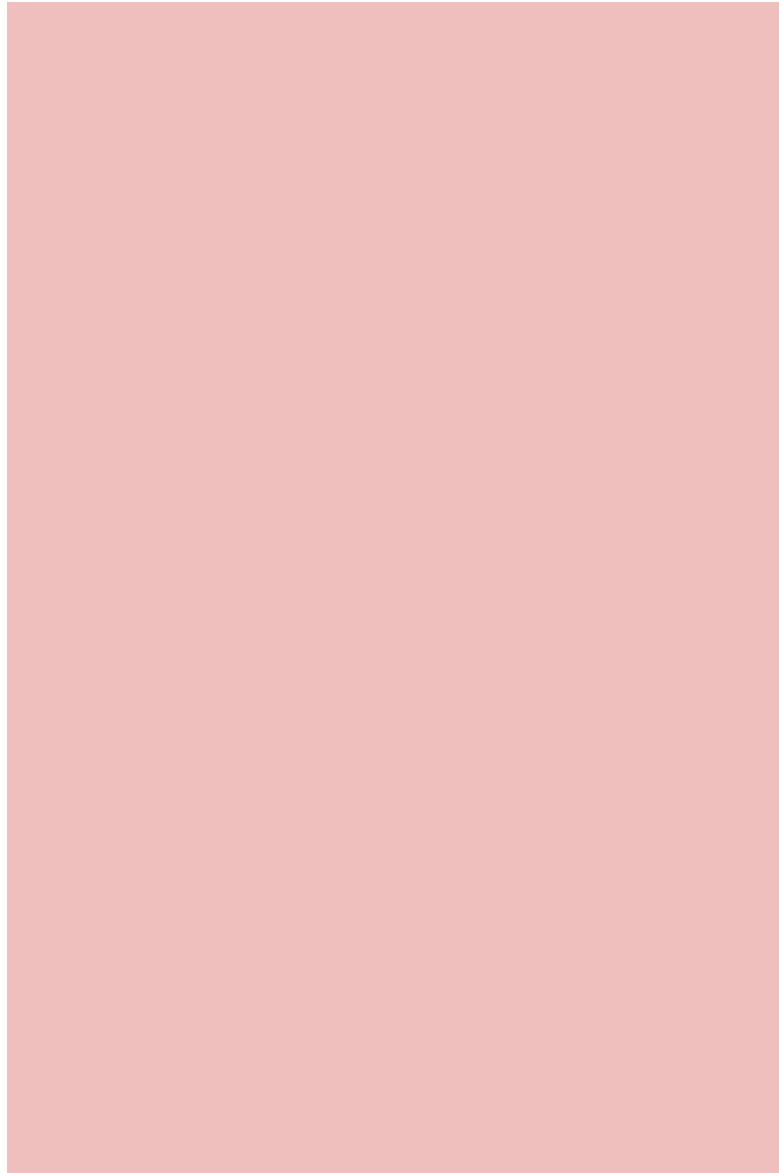
En la ilustración "[Modelo de ramas de git-flow](#)," mostramos el modelo fundamental de ramas en el que se basa git-flow, que iremos desgranando en los siguientes apartados.

### 2.1.1. Las ramas principales

El modelo de desarrollo está basado en 2 ramas continuas (nunca desaparecen):

¥ master

¥ develop



*Imagen 2. Modelo de ramas principales.*

La rama origin/master es la principal, donde se encuentra el código desplegado en producción.

La rama origin/develop es donde se encuentra el código en desarrollo, que va a ser distribuido en futuras release.



## 2.1.2. Ramas de soporte

Además de las ramas master y develop, que como he comentado son permanentes, el modelo usa una serie de ramas de soporte para ayudar al desarrollo paralelo entre miembros del equipo, seguimiento de las nuevas características desarrolladas, preparación de las releases a producción y gestión de la resolución de errores. Al contrario que las dos ramas principales, este tipo de ramas tienen un tiempo de vida limitado.

Los tipos de ramas usados son:

- ¥ Feature

- ¥ Release

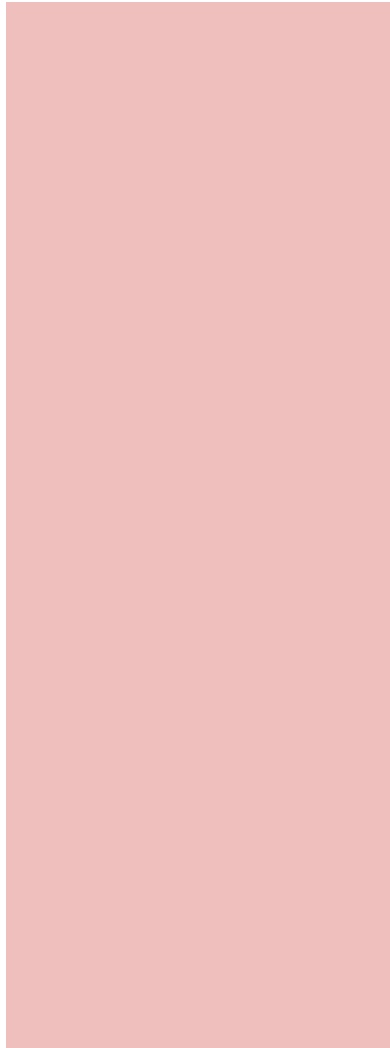
- ¥ Hotfix

Cada uno de estos tipos de ramas tiene un propósito determinado y un conjunto de reglas de uso: desde que ramas se pueden generar y en qué ramas se pueden fusionar. No son ramas especiales; simplemente se usan de una forma determinada siguiendo una serie de reglas, que muestro a continuación.

## Ramas feature (topic)

Este tipo de ramas se usan para el desarrollo de una nueva característica para una release futura.

- ¥ Se crean a partir de la rama *develop*.
- ¥ Se fusionan con la rama *develop*.
- ¥ Convención de nombre: cualquier nombre, excepto *master*, *develop*, *release-\** o *hotfix-\**.



*Imagen 3. Modelo de ramas feature.*

Cuando se inicia el desarrollo de una nueva funcionalidad se crea una rama de este tipo. Esta rama existirá mientras la nueva funcionalidad se encuentra en desarrollo. Cuando se finalice el trabajo, esta rama será fusionada en la rama *develop* (y eliminada) o descartada, si el resultado del trabajo no es el deseado.

Este tipo de ramas existen normalmente en el repositorio local del desarrollador, no en el repositorio remoto (v.gr. origin).

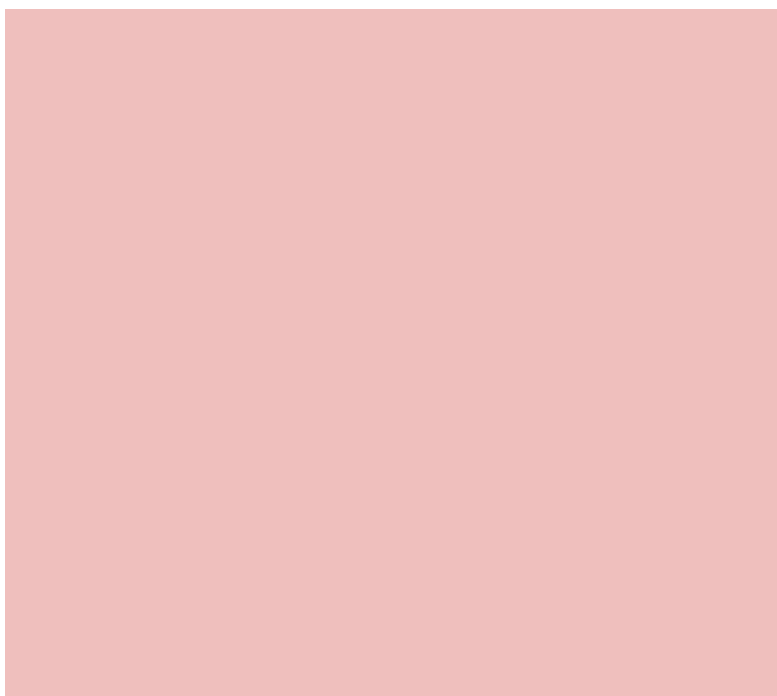
## Trabajo con las ramas feature

La nueva rama se crea a partir de la rama *develop*.

```
$ git checkout -b nueva_funcionalidad develop
```

A continuaci3n, el trabajo es incorporado en la rama *nueva\_funcionalidad* mediante los correspondientes commits. Una vez finalizado el trabajo, me cambio a la rama *develop*, que es la que va a recibir la nueva funcionalidad, borro la rama y la subo al repositorio remoto.

```
$ git checkout develop
$ git merge --no-ff nueva_funcionalidad
$ git branch -d nueva_funcionalidad
$ git push origin develop
```



*Imagen 4. Fusión con y sin commit propio.*

El parámetro "*--no-ff*" en la fusión de las dos ramas fuerza la creaci3n de un nuevo commit aunque la fusión se pudiera haber hecho mediante *fast-forward* (si se hubiera llevado a cabo de forma recursiva este nuevo commit se crea siempre). Este commit permite en un futuro localizar donde se ha incorporado una determinada funcionalidad (un conjunto de commits) y poder revertirla

## Ramas release

Este tipo de ramas se usan para la preparaci3n de una release a producci3n.

- ¥ Se crean a partir de la rama *develop*.
- ¥ Se fusionan con la rama *develop* y *master*.
- ¥ Convenci3n de nombre: *release-\**

Este tipo de ramas est3n pensadas para llevar a cabo peque3os ajustes antes de su publicaci3n, para la correcci3n de peque3os bugs y para la preparaci3n de los elementos propios de una release (n3meros de versi3n, etiquetas, fechas de publicaci3n,É). Al llevar a cabo estas tareas en la rama de *release* estoy liberando la rama *develop* para poder recibir las nuevas funcionalidades que se publicar3n en una release futura y que, por lo tanto, no afectar3n a la release que estoy preparando en esta rama.

### Trabajo con las ramas release

Como ya he comentado, las ramas de tipo *release* se crean a partir de la rama *develop* en el momento en el que las funcionalidades que se encuentran en la rama *develop* coinciden con las que quiero publicar en esa release.

Voy a suponer que la versi3n estable que se encuentra en producci3n es la 2.5.1 y que he decidido publicar la versi3n 2.6 en la siguiente release (siguiendo, por ejemplo, la convenci3n de [versionamiento sem3ntico](#)).

```
$ git checkout -b release-2.6 develop
```

A continuaci3n ejecuto los scripts o herramientas necesarias, dependiendo de mi proceso de publicaci3n. Esta rama puede existir durante un tiempo, mientras las distintas personas encargadas de las pruebas, tests unitarios,É trabajan en el testeo de la versi3n, corrigiendo los posibles errores, pudiendo llevar a cabo los commits necesarios.

Cuando la release est3 lista para ser publicada, llevo esta rama de *release* a la rama *master* y etiqueto este commit para poder conocer posteriormente qu3 commit ha sido enviado a producci3n con esa versi3n concreta.

```
$ git checkout master
$ git merge --no-ff release-2.6
$ git tag -a 2.6
```

Los cambios que se han producido en la rama de *release* también tienen que ser llevados a la rama de *develop*

```
$ git checkout develop  
$ git merge --no-ff release-2.6
```

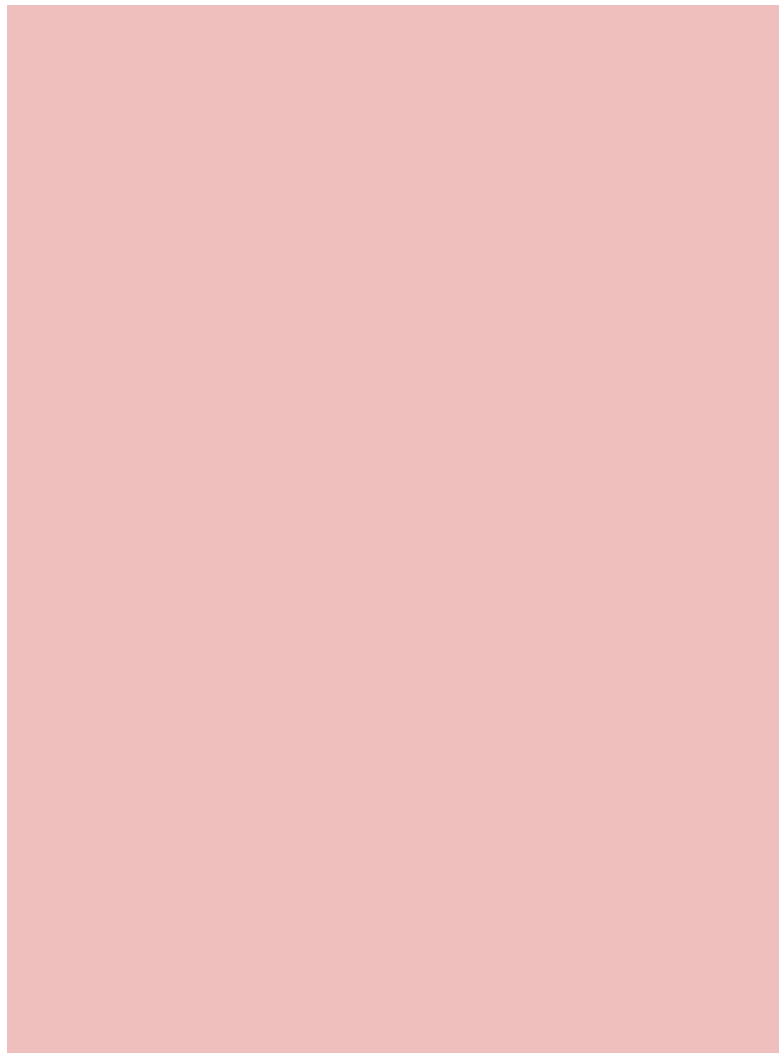
Tras llevar los cambios a las ramas *master* y *develop*, ahora puedo borrar la rama de *release*

```
$ git branch -d release-2.6
```

## Ramas hotfix

Este tipo de ramas se usan para la resolución de errores en producción.

- ¥ Se crean a partir de la rama master.
- ¥ Se fusionan con la rama develop y master.
- ¥ Convención de nombre: *hotfix-\**



*Imagen 5. Ramas principales y de hotfix.*

Este tipo de ramas son muy similares a las ramas de *release*, ya que estoy preparando un nuevo envío a producción, aunque no planificado, al contrario que las ramas de *release*.

Estas ramas se crean al aparecer un error en producción que necesita ser resuelto lo antes posible, produciendo una nueva versión en la que se incrementa la versión del software.

Lo que busco con esta rama es que la persona encargada de resolver el error pueda trabajar de forma paralela a las personas que se encuentran desarrollando nuevas funcionalidades para futuras *release* planificadas.

## Trabajo con las ramas hotfix

Dado que las ramas *hotfix* resuelven problemas en producción, estas se crean desde la rama *master*.

Voy a suponer que he encontrado en la release anterior, etiquetada con el nombre "2.6" un error. Lo primero que hago es crear una nueva rama de *hotfix* a partir de la rama *master*.

```
$ git checkout -b hotfix-2.6.1 master
```

Luego, en esta nueva rama, llevo a cabo los cambios necesarios para corregir el error y preparo la siguiente release (scripts,É).

A continuación ejecuto los scripts o herramientas necesarias, dependiendo de mi proceso de publicación. Esta rama puede existir durante un tiempo, mientras se resuelve el problema y se verifica que los errores de este bug han desaparecido.

Cuando la release está lista para ser publicada, llevo esta rama de *bugfix* a la rama *master* y etiqueto este commit para poder conocer posteriormente qué commit ha sido enviado a producción con esa versión concreta.

```
$ git checkout master
$ git merge --no-ff bugfix-2.6.1
$ git tag -a 2.6.1
```

Los cambios que se han producido en la rama de *bugfix* también tienen que ser llevados a la rama de *develop*

```
$ git checkout develop
$ git merge --no-ff bugfix-2.6.1
```

Tras llevar los cambios a las ramas *master* y *develop*, ahora puedo borrar la rama de *bugfix*

```
$ git branch -d bugfix-2.6.1
```

## 3. Instalaci3n

A continuaci3n voy a explicar c3mo instalar git-flow en Linux, tanto en Debian como en Fedora (y derivados).

El proyecto inicial fue publicado en [GitHub](#) por Vincent Driessen, el ingeniero de software que desarroll3 inicialmente el concepto de [git-flow](#) en su blog. Este proyecto fue introducido como paquete en Debian y en Ubuntu. El 3ltimo commit de este proyecto fue llevado a cabo en el [25 de septiembre de 2012](#), mientras que la 3ltima *release* es del [14 de febrero del 2011](#), por lo que est3 claro que el proyecto no est3 siendo mantenido.

Este proyecto ha sido continuado, tras un fork, por otro desarrollador, tambi3n en [GitHub](#), y actualmente es la versi3n distribuida por Debian y por Ubuntu.

### 3.1. Instalaci3n en Linux

A continuaci3n voy a explicar c3mo se instala git-flow en Linux, tanto en una versi3n Debian como en una Fedora, ya que usan diferentes tipos de paquetes y son las dos variantes m3s utilizadas.

#### 3.1.1. Ubuntu

Empezar3 por una m3quina Ubuntu. De una forma casi id3ntica se puede realizar la instalaci3n en distribuciones que usen el formato de paquete .deb: Debian, Linux Mint,É

Accedo a una consola y verifico que Git se encuentre instalado ejecutando

```
$ git
```

En caso de que no se encuentre Git paso a instalarlo, ejecutando

```
$ sudo apt-get update
```

y posteriormente

```
$ sudo apt-get upgrade
```

para actualizar el listado de paquetes disponibles e instalar las 3ltimas actualizaciones. De esta forma tengo el sistema actualizado. Estos dos pasos son opcionales, aunque recomendables.

Ejecuto



```
$ sudo apt-get install git
```

para instalar Git en nuestra máquina.

Tras finalizar la instalación ejecuto

```
$ git --version

git version 2.1.4
```

para comprobar que se ha instalado correctamente.

A continuación ejecuto

```
$ sudo apt-get install git-flow
```

para instalar el paquete git-flow.

Para comprobar que se ha instalado correctamente ejecuto

```
$ git flow
usage: git flow <subcommand>

Available subcommands are:
  Ê init      Initialize a new git repo with support for the branching
model .
  Ê feature   Manage your feature branches.
  Ê release   Manage your release branches.
  Ê hotfix    Manage your hotfix branches.
  Ê support    Manage your support branches.
  Ê version   Shows version information.
  Ê config    Manage your git-flow configuration.

Try 'git flow <subcommand> help' for details.
```

y obtengo una salida similar a la anterior.

Para comprobar la versión de git-flow instalada ejecuto

```
$ git flow version
1.6.1 (AVH Edition)
```

y compruebo que la versión instalada es el fork comentado (AVH Edition).

Existe un PPA (Personal Package Archives) con la última versión de git-flow disponible para las siguientes versiones de Ubuntu:

- ¥ Precise (12.04)

- ¥ Trusty (14.04)

- ¥ Wily (15.10)

- ¥ Xenial (16.04)

Para llevar a cabo la instalación de software desde un PPA, lo primero que tengo que hacer es indicarle a Ubuntu dónde encontrar el PPA.

```
$ sudo add-apt-repository ppa:pdoes/gitflow-avh
```

A continuación tengo que indicarle a Ubuntu que obtenga el listado actualizado de los repositorios, incluyendo el del PPA que acabo de añadir, ejecutando:

```
$ sudo apt-get update
```

A continuación ya puedo instalar el software, en su versión más actualizada, ejecutando:

```
$ sudo apt-get install git-flow
```

### 3.1.2. CentOS

Ahora voy a explicar cómo se instala Git en una distribución CentOS. De una forma casi idéntica se puede realizar la instalación en distribuciones basadas en Red Hat, como Red Hat Enterprise Linux, Fedora, É

Accedo a una consola y verifico que Git se encuentra instalado, ejecutando

```
$ git
```

En caso de que no se encuentre Git paso a instalarlo, ejecutando

```
$ sudo dnf upgrade
```

para instalar las últimas actualizaciones. De esta forma tengo el sistema actualizado. Este paso es opcional, aunque recomendable.

Ejecuto

```
$ sudo dnf install git-flow
```

para instalar git-flow en nuestra máquina.

Para comprobar que se ha instalado correctamente ejecuto

```
$ git flow
usage: git flow <subcommand>

Available subcommands are:
  Ê init      Initialize a new git repo with support for the branching
model .
  Ê feature   Manage your feature branches.
  Ê release   Manage your release branches.
  Ê hotfix    Manage your hotfix branches.
  Ê support   Manage your support branches.
  Ê version   Shows version information.
  Ê config    Manage your git-flow configuration.

Try 'git flow <subcommand> help' for details.
```

y obtengo una salida similar a la anterior.

Para comprobar la versión de git-flow instalada ejecuto

```
$ git flow version
init: required argument missing.
0.4.2-pre
```

y compruebo que la versión instalada es una release beta (pre) posterior a la última publicación del paquete original: [versión 0.4.1](#).

## 4. Comandos de git-flow

Tras realizar la instalación de git-flow, en este apartado voy a explicar los principales subcomandos que vamos a utilizar.

Los subcomandos disponibles son los siguientes:

```
$ git flow
usage: git flow <subcommand>

Available subcommands are:
  Ê init      Initialize a new git repo with support for the branching
model .
  Ê feature   Manage your feature branches.
  Ê bugfix    Manage your bugfix branches.
  Ê release   Manage your release branches.
  Ê hotfix    Manage your hotfix branches.
  Ê support   Manage your support branches.
  Ê version   Shows version information.
  Ê config    Manage your git-flow configuration.
  Ê log       Show log deviating from base branch.

Try 'git flow <subcommand> help' for details.
```

### 4.1. git flow init

#### 4.1.1. Proyecto nuevo

Este comando inicializa un repositorio Git vacío y establece la configuración de Git que voy a necesitar, creando las ramas usadas en el flujo de trabajo de git-flow.

Para utilizar este comando ejecuto:

```
$ mkdir proyecto
$ cd proyecto
$ git flow init
```

Tras ejecutar este comando, git-flow me pregunta una serie de cuestiones sobre la configuración del repositorio:

¥ Nombre de las ramas.

¥ Prefijo de las etiquetas.

Pulso la tecla *Intro* en todos los casos para aceptar la sugerencia de nomenclatura que me sugiere git-flow.

```

Initialized empty Git repository in /home/user/project/.git/
No branches exist yet. Base branches must be created now.
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-flow configura el repositorio Git usando el comando *"git config"*. git-flow obtiene los valores introducidos al ejecutar este comando y, usando el comando *"git config"*, lo almacena en el archivo *".git/config"*

Si ejecuto

```

$ cat .git/config
[core]
  repositoryformatversion = 0
  filemode = true
  bare = false
  logallrefupdates = true
[gitflow "branch"]
  master = master
  develop = develop
[gitflow "prefix"]
  feature = feature/
  release = release/
  hotfix = hotfix/
  support = support/
  versiontag =

```

veo que git-flow almacena todos esos valores en este archivo.

Si ejecuto

```
$ git branch
* develop
È master
```

veo que se acaban de crear las dos ramas principales y que la activa es la desarrollo: "*develop*".

#### 4.1.2. Proyecto existente

También puedo usar git-flow en un proyecto existente que estuviera usando esta metodología. Para ello, primero clono el proyecto

```
$ git clone https://github.com/jquery/jquery.git
$ cd jquery
```

Si consulto todas las ramas del proyecto, veo que tengo una local *master*, donde se realizan las publicaciones de producción, y varias ramas de seguimiento.

```
$ git branch -a -v
* master                fcb8a1b Core: remove precautionary variable
`readyFiring`
È remotes/origin/1.12-stable e09907c Build: Update grunt-contrib-uglify
È remotes/origin/2.2-stable b14ce54 Build: Update grunt-contrib-uglify
È remotes/origin/HEAD      -> origin/master
È remotes/origin/killphp    90347a7 Tests: first step in using a node-
based test server: static files
È remotes/origin/master     fcb8a1b Core: remove precautionary variable
`readyFiring`
```

Si ejecuto el comando *git flow init* vuelvo a indicar cuáles son las ramas y los prefijos que usaré para este proyecto. En este caso uso los valores por defecto sugeridos por *git-flow*.

```
$ git flow init
```

Which branch should be used for bringing forth production releases?

```
Ê - master
```

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? []

Si vuelvo a comprobar las ramas del proyecto veo que aparece una nueva rama, la de desarrollo "*develop*", apuntando al mismo commit que la rama "*master*", ya que se ha creado a partir del último commit de la rama "*master*".

```
$ git branch -a -v
```

```
* develop                                fcb8a1b Core: remove precautionary variable
```

```
`readyFiring`
```

```
Ê master                                fcb8a1b Core: remove precautionary variable
```

```
`readyFiring`
```

```
Ê remotes/origin/1.12-stable e09907c Build: Update grunt-contrib-uglify
```

```
Ê remotes/origin/2.2-stable b14ce54 Build: Update grunt-contrib-uglify
```

```
Ê remotes/origin/HEAD -> origin/master
```

```
Ê remotes/origin/killphp 90347a7 Tests: first step in using a node-based test server: static files
```

```
Ê remotes/origin/master fcb8a1b Core: remove precautionary variable
```

```
`readyFiring`
```

En el archivo de configuración de Git puedo ver las entradas de git-flow

```
$ cat .git/config
[core]
Ê      repositoryformatversion = 0
Ê      filemode = true
Ê      bare = false
Ê      logallrefupdates = true
[remote "origin"]
Ê      url = https://github.com/jquery/jquery.git
Ê      fetch = +refs/heads/*:refs/remotes/origin/*
[branch "master"]
Ê      remote = origin
Ê      merge = refs/heads/master
[gitflow "branch"]
Ê      master = master
Ê      develop = develop
[gitflow "prefix"]
Ê      feature = feature/
Ê      release = release/
Ê      hotfix = hotfix/
Ê      support = support/
Ê      versiontag =
```



## 4.2. git flow feature

A continuación voy a crear la primera *feature* en el proyecto vacío. Para simplificar, voy a crear una rama en la que introduciré el archivo de licencia del proyecto, "*license.txt*", para posteriormente llevar esta *feature* a la rama *develop* y finalizar la *feature*. Este comando presenta una serie de subcomandos que iré explicando.

Los subcomandos de "*git flow feature*" son los siguientes:

```
$ git flow feature help
usage: git flow feature [list]
  or: git flow feature start
  or: git flow feature finish
  or: git flow feature publish
  or: git flow feature track
  or: git flow feature diff
  or: git flow feature rebase
  or: git flow feature checkout
  or: git flow feature pull
  or: git flow feature delete

  Manage your feature branches.

  For more specific help type the command followed by --help
```

### 4.2.1. git flow feature list

Para visualizar las ramas de tipo *feature* usaré este comando.

```
$ git flow feature list

No feature branches exist.

You can start a new feature branch:

  git flow feature start <name> [<base>]
```

### 4.2.2. git flow feature start

Este comando inicializa una nueva *feature*

```
$ git flow feature start LicenciaAdd
Switched to a new branch 'feature/LicenciaAdd'

Summary of actions:
- A new branch 'feature/LicenciaAdd' was created, based on 'develop'
- You are now on branch 'feature/LicenciaAdd'

Now, start committing on your feature. When done, use:

$ git flow feature finish LicenciaAdd
```

proyecto puedan trabajar de forma colaborativa desde el inicio del proyecto:

```
$ git push -u origin master
Counting objects: 2, done.
Writing objects: 100% (2/2), 158 bytes | 0 bytes/s, done.
Total 2 (delta 0), reused 0 (delta 0)
To https://ami ei ro@bi tbucket. org/ami ei ro/gi t_fl ow. gi t
Ê* [new branch]      master -> master
Branch master set up to track remote branch master from origin.

$ git push -u origin develop
Total 0 (delta 0), reused 0 (delta 0)
remote:
remote: Create pull request for develop:
remote:   https://bi tbucket. org/ami ei ro/gi t_fl ow/pul l -
requests/new?source=develop&t=1
remote:
To https://ami ei ro@bi tbucket. org/ami ei ro/gi t_fl ow. gi t
Ê* [new branch]      develop -> develop
Branch develop set up to track remote branch develop from origin.
```

A continuaci3n ejecuto el comando

```
$ git flow feature publish Licenci aAdd
Counting objects: 4, done.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 316 bytes | 0 bytes/s, done.
Total 3 (delta 0), reused 0 (delta 0)
remote:
remote: Create pull request for feature/Licenci aAdd:
remote:   https://bi tbucket. org/ami ei ro/gi t_fl ow/pul l -
requests/new?source=feature/Licenci aAdd&t=1
remote:
To https://ami ei ro@bi tbucket. org/ami ei ro/gi t_fl ow. gi t
Ê* [new branch]      feature/Licenci aAdd -> feature/Licenci aAdd

Already on 'feature/Licenci aAdd'
Your branch is up-to-date with 'origin/feature/Licenci aAdd' .

Summary of actions:
- A new remote branch 'feature/Licenci aAdd' was created
- The local branch 'feature/Licenci aAdd' was configured to track the
remote branch
- You are now on branch 'feature/Licenci aAdd'
```

#### 4.2.4. git flow feature pull

Cada compa ero de mi equipo que quiera obtener los commits de la rama tendr  que usar este comando.

Primero tendr  que clonar el proyecto remoto

```
$ git clone https://JesusAmi ei ro@bi tbucket. org/ami ei ro/gi t_flow. gi t
remote: Counting objects: 5, done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 5 (delta 0), reused 0 (delta 0)
Unpacking objects: 100% (5/5), done.
Checking connecti vi ty... done.
```

Si consulto el estado de las ramas

```
$ git branch -a -v
* master                                a98e02a Ini ti al commi t
Ê remotes/ori gi n/HEAD                  -> ori gi n/master
Ê remotes/ori gi n/devel op              a98e02a Ini ti al commi t
Ê remotes/ori gi n/feature/I i cenci aAdd e2d68d0 A ado el archi vo de I i cenci a
Ê remotes/ori gi n/master                a98e02a Ini ti al commi t
```

A continuaci n ejecuto la inicializaci n de git-flow con este usuario

```
$ git flow i ni t

Which branch shoul d be used for bringi ng forth producti on rel eases?
Ê - master
Branch name for producti on rel eases: [master]
Branch name for "next rel ease" devel opment: [devel op]

How to name your supporti ng branch prefi xes?
Feature branches? [feature/]
Rel ease branches? [rel ease/]
Hotfi x branches? [hotfi x/]
Support branches? [support/]
Versi on tag prefi x? []
```

Y posteriormente el comando

```
$ git flow feature track licenciaAdd
Branch feature/licenciaAdd set up to track remote branch
feature/licenciaAdd from origin.
Switched to a new branch 'feature/licenciaAdd'
```

Summary of actions:

- A new remote tracking branch 'feature/licenciaAdd' was created
- You are now on branch 'feature/licenciaAdd'

para crear la rama local *"feature/licenciaAdd"* que va a seguir a esa misma rama en el *"origin"*. A continuaci3n obtengo la informaci3n de la rama remota con el siguiente comando:

```
$ git flow feature pull origin licenciaAdd
Pulled origin's changes into feature/licenciaAdd.
```

Si consulto las ramas veo que aparece esta rama de *"feature"* y que es la activa:

```
$ git branch -a
Ê devel op
* feature/licenciaAdd
Ê master
Ê remotes/origin/HEAD -> origin/master
Ê remotes/origin/devel op
Ê remotes/origin/feature/licenciaAdd
Ê remotes/origin/master
```

#### 4.2.5. git flow feature finish

Tras acabar el desarrollo de la *feature*, la cierro, usando el comando:

```

$ git flow feature finish licenciaAdd
Switched to branch 'develop'
Your branch is up-to-date with 'origin/develop'.
Updating a98e02a..e2d68d0
Fast-forward
 Ê licencia.txt | 2 ++
 Ê 1 file changed, 2 insertions(+)
 Ê create mode 100644 licencia.txt

 Ê To https://ami ei ro@bi tbucket.org/ami ei ro/git_flow.git
 Ê - [deleted]          feature/licenciaAdd
 Deleted branch feature/licenciaAdd (was e2d68d0).

Summary of actions:
- The feature branch 'feature/licenciaAdd' was merged into 'develop'
- Feature branch 'feature/licenciaAdd' has been locally deleted; it has
  been remotely deleted from 'origin'
- You are now on branch 'develop'

```

Este comando realiza las siguientes operaciones:

- ¥ La rama *'feature/licenciaAdd'* es fusionada en la rama *'develop'*
- ¥ La rama local *'feature/licenciaAdd'* se elimina.
- ¥ La rama remota *'feature/licenciaAdd'* se elimina del *'origin'*.
- ¥ La rama activa pasa a ser *'develop'*.

Si compruebo el estado de las ramas

```

$ git branch -a -v
* develop                e2d68d0 [ahead 1] A-ado el archi vo de l i cenci a
 Ê master                a98e02a I ni ti al commi t
 Ê remotes/origi n/devel op a98e02a I ni ti al commi t
 Ê remotes/origi n/master  a98e02a I ni ti al commi t

```

Veo que la rama *'develop'* no ha sido actualizada en el *'origin'*. La actualizo con el comando

```
$ git push
Counting objects: 4, done.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 316 bytes | 0 bytes/s, done.
Total 3 (delta 0), reused 0 (delta 0)
remote:
remote: Create pull request for develop:
remote:   https://bitbucket.org/ami ei ro/git_flow/pull -
requests/new?source=develop&t=1
remote:
To https://ami ei ro@bitbucket.org/ami ei ro/git_flow.git
Ê a98e02a..e2d68d0 develop -> develop
```

Si vuelvo a comprobar el estado de las ramas, veo que ahora se encuentran sincronizadas

```
$ git branch -a -v
* develop                e2d68d0 A-ado el archi vo de l i cenci a
Ê master                 a98e02a I n i t i a l  c o m m i t
Ê remotes/origi n/develop e2d68d0 A-ado el archi vo de l i cenci a
Ê remotes/origi n/master  a98e02a I n i t i a l  c o m m i t
```

En el resto de usuarios, para borrar las ramas que han desaparecido del servidor, ejecuto el siguiente script:

```
$ git checkout master; git pull origin master; git fetch --all -p; git
branch -vv | grep ": gone]" | awk '{ print $1 }' | xargs -n 1 git branch
-D
```

### 4.3. git flow release

Una vez que tengo desarrolladas unas cuantas características del código, llega el momento de realizar un despliegue a producción. Antes de enviar a producción el código es posible que quiera que realizar determinados tests de integración, usabilidad, que el departamento de control lo revise, y puede que mis compañeros o yo tengamos que realizar determinados cambios antes de publicar esta versión.

En este proceso lo que quiero es separar el control de calidad de esta versión de las nuevas funcionalidades en publicaciones futuras, por lo que lo ideal es separar el trabajo en ramas. Esto lo hago introduciendo ramas de *"release"*, desde el punto de vista de que es código que está en "Release Candidate" para ser publicado, no código que ha sido publicado. Para lograr este flujo de trabajo uso el comando *"git flow release"* y sus subcomandos. Lo ideal es tener una única rama de *"release"*, para evitar conflictos entre este tipo de ramas.

Los subcomandos de *"git flow release"* son los siguientes:

```
$ git flow release help
usage: git flow release [list]
    or: git flow release start
    or: git flow release finish
    or: git flow release publish
    or: git flow release track
    or: git flow release delete

    Manage your release branches.

    For more specific help type the command followed by --help
```

#### 4.3.1. git flow release list

Para visualizar las ramas de tipo *release* usar este comando.

```
$ git flow release list
No release branches exist.

You can start a new release branch:

    git flow release start <name> [<base>]
```



### 4.3.2. git flow release start

Este comando inicializa una nueva rama de *"release"*.

```
$ git flow release start 0.1.0
Switched to a new branch 'release/0.1.0'

Summary of actions:
- A new branch 'release/0.1.0' was created, based on 'develop'
- You are now on branch 'release/0.1.0'

Follow-up actions:
- Bump the version number now!
- Start committing last-minute fixes in preparing your release
- When done, run:

$ git flow release finish '0.1.0'
```

Este comando crea la nueva rama *"release/0.1.0"* a partir de la rama *"develop"* y la establece como activa.

Si compruebo las ramas existentes:

```
$ git branch -a -v
* develop                e2d68d0 A-ado el archivo de licencia
* master                 a98e02a Initial commit
* release/0.1.0          e2d68d0 A-ado el archivo de licencia
* remotes/origin/develop e2d68d0 A-ado el archivo de licencia
* remotes/origin/master  a98e02a Initial commit
```

aparece esta nueva rama como activa.

El nombre que se le pasa como parámetro (0.1.0 en este caso) será la etiqueta asignada cuando finalice el desarrollo de esta *"release"*, de tal forma que siempre podrá volver a una etiqueta determinada para resolver un error (hotfix) o para realizar un ciclo de desarrollo de soporte.

Tengo que actualizar la versión de la aplicación, lo que voy a simular mediante un archivo *"version.txt"* en el que incluiré el número de la versión:

```
$ echo "0.1.0" >> version.txt
```

A continuación realizo el commit, momento en el que la *"release"* queda lista para que el

departamento de QA, por ejemplo, realice las últimas comprobaciones, previas a la finalización de esta *"release"*.

```
$ git add version.txt
$ git commit -m "Añado el archivo de versión al proyecto"
```

### 4.3.3. git flow release publish

Para compartir esta rama con los compañeros de control de calidad tengo que usar este comando, para publicar la rama de *release* en el repositorio remoto.

```
$ git flow release publish 0.1.0
Counting objects: 5, done.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 301 bytes | 0 bytes/s, done.
Total 3 (delta 0), reused 0 (delta 0)
remote:
remote: Create pull request for release/0.1.0:
remote:   https://bitbucket.org/ami ei ro/git_flow/pull-
requests/new?source=release/0.1.0&t=1
remote:
To https://ami ei ro@bitbucket.org/ami ei ro/git_flow.git
Ê* [new branch]      release/0.1.0 -> release/0.1.0

Already on 'release/0.1.0'
Your branch is up-to-date with 'origin/release/0.1.0'.

Summary of actions:
- A new remote branch 'release/0.1.0' was created
- The local branch 'release/0.1.0' was configured to track the remote
  branch
- You are now on branch 'release/0.1.0'
```

Voy a suponer que el departamento de QA me notifica un error, y que tengo que añadir en el archivo de licencia la URL desde donde se puede descargar la licencia.

Añado una línea al archivo *"licencia.txt"*:

```
$ echo "Puedes consultar la licencia en la URL
https://opensource.org/licenses/MIT" >> licencia.txt
```

Realizo el commit y lo publico, para poder realizar la revisión previa a la publicación por parte del

personal de QA

```
$ git commit -am "A-ado la URL de consulta en el archivo de licencia"
$ git flow release publish 0.1.0
Counting objects: 7, done.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 431 bytes | 0 bytes/s, done.
Total 3 (delta 0), reused 0 (delta 0)
remote:
remote: Create pull request for release/0.1.0:
remote:   https://bitbucket.org/ami ei ro/git_flow/pull -
requests/new?source=release/0.1.0&t=1
remote:
To https://ami ei ro@bitbucket.org/ami ei ro/git_flow.git
   5850bc8..77fc293  release/0.1.0 -> release/0.1.0
Already on 'release/0.1.0'
Your branch is up-to-date with 'origin/release/0.1.0'.
```

Summary of actions:

- A new remote branch 'release/0.1.0' was created
- The local branch 'release/0.1.0' was configured to track the remote branch
- You are now on branch 'release/0.1.0'

#### 4.3.4. git flow release finish

Una vez que el trabajo de pulido de la nueva versión está finalizado, puedo finalizar la "release" mediante este comando:

```
$ git flow release finish --help
usage: git flow release finish [-h] [-F] [-s] [-u] [-m | -f] [-p] [-k] [-n] [-b] [-S] <version>
```

Ê Finish a release branch

```
Ê -h, --help          Show this help
Ê --showcommands     Show git commands while executing them
Ê -F, --[no]fetch     Fetch from origin before performing finish
Ê -s, --sign          Sign the release tag cryptographically
Ê -u, --signingkey    Use the given GPG-key for the digital signature
                      (implies -s)
Ê -m, --message       Use the given tag message
Ê -f, --[no]messagefile ...
Ê                      Use the contents of the given file as a tag
                      message
Ê -p, --[no]push      Push to origin after performing finish
Ê -k, --[no]keep      Keep branch after performing finish
Ê --[no]keepremote    Keep the remote branch
Ê --[no]keeplocal     Keep the local branch
Ê -D, --[no]force_delete
Ê                      Force delete release branch after finish
Ê -n, --[no]tag       Don't tag this release
Ê -b, --[no]nobackmerge
Ê                      Don't back-merge master, or tag if applicable,
in develop
Ê -S, --[no]squash    Squash release during merge
```

En el ejemplo uso el comando:

```
$ git flow release finish -F -p 0.1.0
```

```
Branches 'master' and 'origin/master' have diverged.
And local branch 'master' is ahead of 'origin/master'.
Switched to branch 'develop'
Your branch is up-to-date with 'origin/develop'.
Merge made by the 'recursive' strategy.
  Elicencia.txt | 1 +
  Eversion.txt   | 1 +
  E2 files changed, 2 insertions(+)
  Ecreate mode 100644 version.txt
To https://ami ei ro@bi tbucket.org/ami ei ro/git_flow.git
  E- [deleted]          release/0.1.0
Deleted branch release/0.1.0 (was 77fc293).
```

Summary of actions:

- Release branch 'release/0.1.0' has been merged into 'master'
- The release was tagged '0.1.0'
- Release tag '0.1.0' has been back-merged into 'develop'
- Release branch 'release/0.1.0' has been locally deleted; it has been remotely deleted from 'origin'
- You are now on branch 'develop'

Este comando realiza las siguientes operaciones:

- ¥ Fusiona la rama *'release/0.1.0'* en *'master'*.
- ¥ Etiqueta el commit con *'0.1.0'*.
- ¥ La etiqueta de publicaci–n *'0.1.0'* ha sido fusionada en la rama *'develop'*.
- ¥ La rama local *'release/0.1.0'* ha sido borrada.
- ¥ La rama remota *'release/0.1.0'* ha sido borrada del *'origin'*.
- ¥ La rama activa es *'develop'*.

Si consulto las ramas existentes veo que no hay rastro de la rama de *'release'* entre las ramas existentes y que la rama activa es *'develop'*.

```
$ git branch -a -v
* develop                a325e19 Merge tag '0.1.0' into develop
  E master                f18cd9c Merge branch 'release/0.1.0'
  E remotes/origin/develop a325e19 Merge tag '0.1.0' into develop
  E remotes/origin/master  f18cd9c Merge branch 'release/0.1.0'
```

## 4.4. git flow hotfix

Una vez que he publicado la versión *0.1.0* aparece un error. En este caso es donde aparecen las ramas de *"hotfix"*, precisamente para poder resolver el error concreto de forma aislada

### 4.4.1. Subcomandos

Los subcomandos disponibles son los siguientes:

```
$ git flow hotfix help
usage: git flow hotfix [list]
  or: git flow hotfix start
  or: git flow hotfix finish
  or: git flow hotfix publish
  or: git flow hotfix delete

  Manage your hotfix branches.

  For more specific help type the command followed by --help
```

### 4.4.2. git flow hotfix list

Para comprobar las ramas de tipo *"hotfix"* existente ejecuto:

```
$ git flow hotfix list
No hotfix branches exist.

You can start a new hotfix branch:

  git flow hotfix start <version> [<base>]
```

### 4.4.3. git flow hotfix start

Para iniciar una rama de *"bugfix"* a partir de un commit concreto de producción ejecuto:

```
git flow hotfix start <version> [rama_base]
```

```
$ git flow hotfix start 0.1.1
Switched to a new branch 'hotfix/0.1.1'
```

Summary of actions:

- A new branch 'hotfix/0.1.1' was created, based on 'master'
- You are now on branch 'hotfix/0.1.1'

Follow-up actions:

- Bump the version number now!
- Start committing your hot fixes
- When done, run:

```
Ê git flow hotfix finish '0.1.1'
```

Simulo la resolución de un bug, añadiendo un archivo .gitignore

```
$ echo "0.1.1" > version.txt
$ echo "vendor/" >> .gitignore
$ git add .gitignore
$ git commit -m "Añado el archivo .gitignore"
```

#### 4.4.4. git flow hotfix publish

Para compartir esta información con el departamento de QA y que pueda revisar la resolución de este fallo antes de enviarlo a producción, utilizo este comando

```
$ git flow hotfix publish <version>
```

```
$ git flow hotfix publish 0.1.1
Counting objects: 6, done.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (4/4), 347 bytes | 0 bytes/s, done.
Total 4 (delta 0), reused 0 (delta 0)
remote:
remote: Create pull request for hotfix/0.1.1:
remote:   https://bitbucket.org/ami ei ro/git_flow/pull -
requests/new?source=hotfix/0.1.1&t=1
remote:
To https://ami ei ro@bitbucket.org/ami ei ro/git_flow.git
Ê* [new branch]      hotfix/0.1.1 -> hotfix/0.1.1

Already on 'hotfix/0.1.1'
Your branch is up-to-date with 'origin/hotfix/0.1.1'.
```

Summary of actions:

- A new remote branch 'hotfix/0.1.1' was created
- The local branch 'hotfix/0.1.1' was configured to track the remote branch
- You are now on branch 'hotfix/0.1.1'

#### 4.4.5. git flow hotfix finish

Cuando tengo la aceptación de la corrección por parte del departamento de QA, lo siguiente que hago es finalizar la rama de *"hotfix"*:



```
$ git flow hotfix finish --help
usage: git flow hotfix finish [-h] [-F] [-s] [-u] [-m | -f] [-p] [-k] [-n] [-b] <version>

    Finish hotfix branch <version>

    -h, --help                Show this help
    --showcommands            Show git commands while executing them
    -F, --[no]fetch           Fetch from origin before performing finish
    -s, --[no]sign            Sign the release tag cryptographically
    -u, --[no]signingkey      Use the given GPG-key for the digital signature
                              (implies -s)
    -m, --[no]message         Use the given tag message
    -f, --[no]messagefile ... Use the contents of the given file as tag
                              message
    -p, --[no]push            Push to origin after performing finish
    -k, --[no]keep            Keep branch after performing finish
    --[no]keepremote          Keep the remote branch
    --[no]keeplocal           Keep the local branch
    -D, --[no]force_delete    Force delete hotfix branch after finish
    -n, --[no]notag           Don't tag this hotfix
    -b, --[no]nobackmerge     Don't back-merge master, or tag if applicable,
                              in develop
```

El estado previo de las ramas a la ejecución de este comando es el siguiente:

```
$ git branch -a -v
* develop                a325e19 Merge tag '0.1.0' into develop
* hotfix/0.1.1           a025ade Añado el archivo .gitignore
* master                 f18cd9c Merge branch 'release/0.1.0'
remotes/origin/develop    a325e19 Merge tag '0.1.0' into develop
remotes/origin/hotfix/0.1.1 a025ade Añado el archivo .gitignore
remotes/origin/master     f18cd9c Merge branch 'release/0.1.0'
```

Ejecuto el comando de finalización

```
$ git flow hotfix finish -F -p 0.1.1

Switched to branch 'master'
Your branch is up-to-date with 'origin/master'.
```

```

Merge made by the 'recursive' strategy.
Ê.gitignore | 1 +
Êversion.txt | 2 +-
Ê2 files changed, 2 insertions(+), 1 deletion(-)
Êcreate mode 100644 .gitignore
Switched to branch 'develop'
Your branch is up-to-date with 'origin/develop'.
Merge made by the 'recursive' strategy.
Ê.gitignore | 1 +
Êversion.txt | 2 +-
Ê2 files changed, 2 insertions(+), 1 deletion(-)
Êcreate mode 100644 .gitignore

To https://ami ei ro@bi tbucket.org/ami ei ro/git_flow.git
Ê- [deleted]          hotfix/0.1.1
Deleted branch hotfix/0.1.1 (was a025ade).

Counting objects: 8, done.
Compressing objects: 100% (4/4), done.
Writing objects: 100% (6/6), 628 bytes | 0 bytes/s, done.
Total 6 (delta 1), reused 0 (delta 0)
remote:
remote: Create pull request for develop:
remote:   https://bi tbucket.org/ami ei ro/git_flow/pull -
requests/new?source=develop&t=1
remote:
To https://ami ei ro@bi tbucket.org/ami ei ro/git_flow.git
Ê a325e19..4a3ddb4 develop -> develop

Total 0 (delta 0), reused 0 (delta 0)
To https://ami ei ro@bi tbucket.org/ami ei ro/git_flow.git
Ê f18cd9c..90794fa master -> master

Counting objects: 2, done.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (2/2), 285 bytes | 0 bytes/s, done.
Total 2 (delta 0), reused 0 (delta 0)
To https://ami ei ro@bi tbucket.org/ami ei ro/git_flow.git
Ê* [new tag]          0.1.0 -> 0.1.0
Ê* [new tag]          0.1.1 -> 0.1.1

Summary of actions:
- Latest objects have been fetched from 'origin'
- Hotfix branch 'hotfix/0.1.1' has been merged into 'master'
- The hotfix was tagged '0.1.1'
- Hotfix tag '0.1.1' has been back-merged into 'develop'

```

- Hotfix branch 'hotfix/0.1.1' has been locally deleted; it has been remotely deleted from 'origin'
- 'develop', 'master' and tags have been pushed to 'origin'
- You are now on branch 'develop'

El estado de las ramas tras ejecutar este comando es el siguiente:

```
$ git branch -a -v
* develop                4a3ddb4 Merge tag '0.1.1' into develop
È master                90794fa Merge branch 'hotfix/0.1.1'
È remotes/origin/develop 4a3ddb4 Merge tag '0.1.1' into develop
È remotes/origin/master  90794fa Merge branch 'hotfix/0.1.1'
```

Compruebo el flujo de commits para ver el camino seguido en todo este ejemplo:

```
$ git log --oneline --all --decorate --graph
*    4a3ddb4 (HEAD, origin/develop, develop) Merge tag '0.1.1' into develop
|\
| *    90794fa (tag: 0.1.1, origin/master, master) Merge branch
'hotfix/0.1.1'
| |\
| | * a025ade A-ado el archivo .gitignore
| | /
* |    a325e19 Merge tag '0.1.0' into develop
|\ \
| | /
| *    f18cd9c (tag: 0.1.0) Merge branch 'release/0.1.0'
| |\
| | * 77fc293 A-ado la URL de consulta en el archivo de licencia
| | * 5850bc8 A-ado el archivo de versión al proyecto
| | /
| /|
* |    e2d68d0 A-ado el archivo de licencia
| /
* a98e02a Initial commit
```