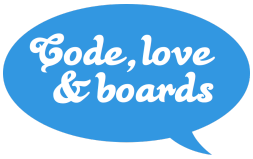

Automatiza tu manera de usar git con git-flow

posted Apr 6, 2013 on code git

Git es probablemente el sistema de control de versiones más popular hoy en día debido a su gran flexibilidad a la hora de trabajar con diferentes ramas de nuestros proyectos, entre otras muchas otras cosas. Esta flexibilidad a la hora de crear diferentes ramas nos permite poder usar el flujo de trabajo que queramos a la hora de funcionar con git diariamente, lo que por otro lado también puede ser un arma de doble filo ya que si no tienes experiencia usándolo o ejecutamos algún comando en el orden que no es o con algún parámetro no deseado, podemos llevarnos algún susto.

Flujo de trabajo acertado

Vincent Driessen diseñó un modelo de ramas exitoso para sus proyectos, el cual decidió compartir con todo el mundo. No voy a entrar en el detalle de este modelo, pero en resumidas cuentas se basa en los siguientes puntos:

- Tenemos dos ramas principales, **master** y **develop**, y varias de soporte para estas que podrán ser ramas de **feature**, **release** y **hotfix**.
- La rama **master** es la que contiene la última versión de nuestro proyecto y usaremos para desplegar en **producción**.
- La rama **develop** es la que contiene el último estado del desarrollo del mismo, es decir, hasta el último commit que hayamos hecho.
- Las ramas de **feature** se usarán para desarrollar nuevas funcionalidades, se crearán a partir de la rama **develop** y al terminar con esta funcionalidad nueva, se tiene que fusionar otra vez con **develop**.
- Las ramas **release** se usarán para lanzar una nueva versión de nuestro proyecto. Se usarán solo para los últimos retoques antes de liberar la nueva versión, como cambiar el número de esta, y crearán a partir de la rama **develop** y se fusionaran tanto con **master** (para poder ser desplegadas en producción) como con **develop**.
- Las ramas "hotfix" se usarán para esos cambios rápidos que queremos realizar como arreglar un bug sencillo en producción mientras que al mismo tiempo estamos desarrollando una nueva funcionalidad y queremos desplegar este arreglo lo antes posible. Se crean a partir de la rama **master** y se fusionan con **master** y **develop**.

Con un poco que lo pensemos, tiene mucho sentido esta manera de trabajar, pero a la hora de la práctica puede parecer un poco complicado el saber que comandos o que flujo seguir para que todo funcione correctamente. Pero no os preocupéis porque Vincent ha creado una herramienta genial que nos ayuda a hacer todo esto de manera casi transparente para nosotros.

git-flow al rescate

Vincent define git-flow como una colección de extensiones de Git que nos facilita un conjunto de operaciones de alto nivel para trabajar con el flujo que antes hemos visto.

Para instalarlo en vuestro Mac OS o Ubuntu tenemos que lanzar la siguiente instrucción desde la consola:

```
# Para Mac OS X
$ brew install git-flow

# Para Ubuntu
$ apt-get install git-flow
```

Una vez instalado y desde el directorio raiz de nuestro proyecto, tenemos que inicializarlo:

```
$  git flow init
```

Esto nos hará una serie de preguntas sobre como queremos nombrar a las diferentes ramas y el prefijo de nuestras versiones. Si queremos usar las de por defecto, pulsamos intro en todas las opciones o podemos usar esta instrucción para inicializarlo:

```
$  git flow init -d
```

Ahora ya tenemos creada nuestra rama **develop** y estamos listos para empezar a trabajar.

Trabajando con features

Para empezar a desarrollar una nueva funcionalidad tenemos que crearla usando:

```
$ git flow feature start nombre_de_funcionalidad
```

Automáticamente te crea la rama `feature/nombre`*`de`*`funcionalidad` y te cambia a ella. Ahora después de varios commits decides que ya has terminado con esa funcionalidad:

```
$ git flow feature finish nombre_de_funcionalidad
```

Mirando la consola vemos que nos fusiona esta rama con **develop**, nos borra esta rama y nos cambia a **develop** automáticamente. Si en algún momento queremos ver que ramas de **features** tenemos actualmente, podemos hacerlo con:

```
$ git flow feature
```

Si quisiéramos retomar una de esas funcionalidades que estamos realizando, por ejemplo después de solucionar un **hotfix**, lo podemos hacer con:

```
$ git flow feature checkout nombre_de_funcionalidad
```

Trabajando con releases

Después de haber desarrollado unas cuantas funcionalidades y haberlas commiteado, queremos liberar una nueva versión de nuestro proyecto. Para crear esta nueva **release** tenemos que usar:

```
$ git flow release start version_1
```

Hacemos los cambios necesarios para la nueva versión y decidimos que está todo preparado:

```
$ git flow release finish version_1
```

Esto buscará cambios en nuestro repositorio remoto, fusionará esta versión con nuestras ramas **master** y **develop**, borrando la **release** y dejándonos todo listo para desplegar desde la rama **master**.

Trabajando con hotfixes

Si estamos trabajando en una funcionalidad y tenemos que cambiar algo que no funciona bien en nuestro entorno de producción, creamos una rama de **hotfix**:

```
$ git flow hotfix start error_en_produccion
```

Esta rama será creada a partir de nuestra **master**, ya que es lo último que hemos desplegado. Al terminar de arreglar el problema tenemos que:

```
$ git flow hotfix finish error_en_produccion
```

Esto fusionará los cambios tanto con **develop** como con **master**, teniendo la solución al problema listo para ser desplegado en producción.

Como veis **git-flow** automatiza muy bien la manera de trabajar con **git**, aplicando un flujo de trabajo más que probado y usado, adaptándose perfectamente a la mayoría de los proyectos. Espero que os ayude y facilite las cosas tal y como lo me lo ha hecho a mi.

Love & Boards!

7 Comments **Code, Love and Boards** **Javier Martín Alon…**

♡ **Recommend** 2 **Share** **Sort by Best**

Join the discussion…

Gustavo Herrera • a month ago

Buen artículo, saludos.

^ v • Reply • Share ›

Ricardo García Vega Mod → Gustavo Herrera • a month ago

Muchas gracias!

^ v • Reply • Share ›

Eduardo • 2 years ago

Me ha gustado mucho el artículo, pero me surge una duda.

Si trabajamos con equipos de más de un desarrollador y repositorios remotos, ¿como hacemos las copias de seguridad de nuestras nuevas ramas de funcionalidad? Me explico:

Si tu estás desarrollando una nueva funcionalidad que tardas por ejemplo 3 días en realizarla, si quieres tener una copia de seguridad de los commits que vas realizando durante estos 3 días, ¿es una buena práctica crear esta rama también en el repositorio remoto o tener un repositorio remoto personal y subir esta rama a él?

Un saludo.

^ v • Reply • Share ›

Ricardo García Vega Mod → Eduardo • 2 years ago

Hola **@Eduardo**, me alegra que te guste :)
Lo que solemos hacer es aunque tu estés trabajando en una rama tuya, digamos `/feature/new-task`, hacer también push al repositorio remoto para como tu bien dices tener una copia de seguridad de los cambios. Una vez que hemos terminado, solemos crear un pull request, para que los compañeros revisen los cambios y se haga merge de tu feature en la rama principal.
Un saludo!

1 ^ v • Reply • Share ›

Edgar De La Cruz • 2 years ago

buena explicacion! gracias!

^ v • Reply • Share ›

Alejandro Steinmetz • 4 years ago

Muy buena explicación. Ahora tengo una consulta, ¿cómo o en qué momento se haría la integración con un repositorio remoto tipo github?

^ v • Reply • Share ›

Ricardo García Vega Mod → Alejandro Steinmetz • 4 years ago

¡Gracias! La verdad es que lo puedes hacer en cualquier momento. Yo personalmente suelo hacer push a la rama develop de mi repositorio remoto después de terminar una feature por ejemplo, y a la rama master al preparar una release.

Reply • Share ›

ALSO ON CODE, LOVE AND BOARDS

Integration tests fun with Phoenix and React

2 comments • 2 years ago•

Avatar **Eamon Penland** — got phantom to work by setting the screen size https://github.com/angular/...

Looking forward to 2016 - Code, Love & Boards

3 comments • 2 years ago•

Avatar **Ricardo García Vega** — Hi Yoni, thanks for your comment :)I'm planning to start writing some posts about it as soon as posible... have you tried

Rails and Flux with Marty.js - Code, Love & Boards

10 comments • 2 years ago•

Avatar **Derek Yau** — Really interesting post Ricardo - I've been getting into React and trying to find a suitable way to integrate it with FLUX and the

Rails and Flux with Marty.js IV: Implementing Flux

4 comments • 2 years ago•

Avatar **Ricardo García Vega** — I guess so, there's at least one post more left deploying it into a production server.

PREVIOUS POST

Debugueando JavaScript desde tu Sublime Text

posted Mar 19, 2013 on code, javascript, sublime text

NEXT POST

Cuatro gemas de Ruby que te ayudarán con tus modelos y base de datos.

posted May 12, 2013 on ruby, rails, gems, data base

Archives RSS