**GitFlow**

**¿Qué es GitFlow?**

**Es un flujo de trabajo basado en Git que brinda mayor control y organización en el proceso de integración continua.**

Básicamente es un modelo de branching o de ramificación, una metodología, pero que ha ido evolucionando, por detrás usan los comandos tradicionales de Git.

Actualmente se ha convertido también en una herramienta soportada por comandos y que sirve de gran ayuda en la gestión de repositorios de tipo Git, y que incluso aporta buenas prácticas que ayudan a gestionar todo de forma correcta y de forma eficiente.

**Origen de GitFlow**

La primera persona que habló sobre esto fue Vincent Driessen, un conocido gurú de JavaScript y de Node, que también tiene mucho conocimiento sobre Git.

Un día publicó un post en su blog proponiendo un modelo de branching exitoso que funcionaría en los equipos personales, y fue una revolución.

Incluso hay quién ha ido un paso más allá y lo está evolucionando dentro de organización, ya que como existen muchos tipos de organizaciones que trabajan de forma diferente y con diferentes casuísticas, muchos usuarios lo han ido evolucionando buscando sacarle el máximo partido, eliminando cosas que no son necesarias de este modelo.

**Todo en GitFlow son ramas**

Las que se destacan son Master y Develop. Todo lo que hagamos como desarrolladores terminará, tarde o temprano, en alguna de estas ramas:

* **Develop**: la rama que contiene las features a incluir en una próxima salida a producción (pasando por test previamente). Todos los desarrollos que hagamos; nuevos módulos, refactorings, entre otros, se guardarán en Develop, a la espera de salir a producción.
* **Master**: la rama que contiene el código que está en producción. A esta rama, sólo debe llegar código que está en; o está listo para estar en producción.

**Elementos básicos para trabajar con GitFlow**

Son tres conceptos importantes que aprender: feature, hotfix y release.

**Feature :** Una feature no es otra cosa que una rama de Git con una copia exacta del contenido de Develop

Para poder realizar cambios sobre lo que está en Develop, lo que hacemos es generar una copia exacta de su contenido y lo metemos en una nueva rama que llamamos feature.

El propósito de las features es desarrollar nuevas funcionalidades del proyecto, en un ambiente controlado, de forma tal de que, una vez que estén listos para agregarse a una futura versión, se agreguen a Develop.

Tiene dos ventajas principales:

1. No trabajamos sobre Develop

¿por qué no podemos trabajar directamente sobre Develop?

Consideremos un escenario en el que un grupo de personas trabaja colaborativamente sobre un proyecto. El encargado del proyecto te asigna el desarrollo de un nuevo módulo y tú le dedicas varias jornadas de trabajo. Justo unos días antes de terminar el módulo, el encargado del proyecto decide que, finalmente, ese módulo no será necesario.

Si hubiésemos trabajado directamente sobre Develop, borrar esos cambios probablemente generaría muchos conflictos y rompería el código de algún otro miembro del equipo que se haya descargado los cambios. Además, seguramente tendríamos que eliminar o comentar código manualmente, lo que empeoraría aún más la situación.

Por el contrario, si trabajamos en una rama aislada o feature, simplemente podemos descartar esa rama, sin afectar a Develop. O bien, podríamos dejarla en stand by por si en un futuro próximo vuelve a retomarse la idea de integrar el módulo en cuestión.

1. Logramos estabilidad

Al usar una rama separada, los desarrollos están más controlados y pasan por una serie de revisiones más minuciosas antes de mergearse con Develop. Imagina que si todos trabajasen sobre la misma rama, podrían generarse conflictos.

Con las features, nos aseguramos que los desarrollos están controlados y no vamos a afectar el trabajo de otro miembro del equipo. Sumado a esto, si trabajamos bien, las features deberían atravesar una serie de pruebas exhaustivas antes de mergearse a Develop.

Los nombres de todas las features comienzan con la palabra feature seguido de una barra invertida y la referencia que nosotros querramos usar para identificarla. Ejemplos de nombres de features podrían ser:

* feature/ADD\_FACEBOOK\_AUTHENTICATION
* feature/change\_user\_database\_field

**Hotfix:** Un hotfix es una rama con una copia exacta del contenido de Master. Imagina que la versión que está ejecutándose en producción tiene algún bug que debe ser corregido en forma inmediata. Esa versión es idéntica a lo que está en la rama Master, por lo que estaría bien poder trabajar sobre el código de dicha rama.

Para poder realizar cambios sobre lo que está en Master y corregir el bug en producción, lo que hacemos es generar una copia exacta de su contenido y lo metemos en una nueva rama que llamamos hotfix. De esta forma, evitamos trabajar directamente sobre el código que está en Master, y volvemos a trabajar en un “ambiente” controlado.

Cuando estemos seguros de haber corregido el bug, podemos cerrar el hotfix para que se agregue a Master.

¿basta con solo agregarlo a Master?

Pensemos en el siguiente caso: alguien del equipo encuentra un bug en la versión que se está ejecutando en producción, que hace que la aplicación deje de funcionar aleatoriamente. Si sólo corrigiéramos el bug en Master, todos los nuevos desarrollos que se hagan en Develop estarían haciéndose sobre una versión que contiene ese bug. Por tal motivo, **siempre que cerremos un hotfix, el merge se debe hacer con**Master**y con**Develop.

Al igual que ocurre con las features, los nombres de todos los hotfix comienzan con la palabra hotfix seguido de una barra invertida. Ejemplos de nombres de hotfix podrían ser:

* hotfix/1.10.1
* hotfix/missing\_param\_in\_login\_function
* hotfix/AddDefaultValueToUserDocumentType

**Release:** una release es una rama con una copia exacta del contenido de Develop, que está lista (o casi) para ser subida a Producción.

Supongamos que hemos estado trabajando por unas semanas con nuevos desarrollos. Todos los miembros del equipo trabajamos en sendas features, y luego fusionamos esos cambios en Develop. Llega el momento en que el encargado del proyecto (o cualquiera del equipo) decide que es hora de mandar todos esos nuevos cambios a Producción. Para ello, lo que hacemos es abrir una release.

Una release es el pasaje de Develop a Master.

Alguien del equipo genera la release, que no es otra cosa que una copia de Develop, y la publica en un entorno de pruebas. La idea es someterla a unas últimas revisiones antes de mandarla a producción.

Pero supongamos que entre esas revisiones, se encuentra un bug menor que surge de los nuevos cambios, y que es mejor que no llegue a Producción. Entonces, lo que hacemos es solucionar el bug directamente en la release y volvemos a probar. Cuando estamos seguros de que ya está ok para subirse a Producción, fusionamos la release con Develop y Master.

* Con Master porque es de donde sale nuestro código a producción
* Con Develop porque queremos que los próximos desarrollos se correspondan con lo que, a partir de ahora, va a estar en Master.

Recuerda que ese bug que solucionamos directamente en la release todavía está presente en la rama Develop, por lo que al fusionar la release estamos seguros de que el bug también se va a corregir en dicha rama.

Finalmente, sólo resta publicar la nueva versión en Producción a partir del contenido de Master.