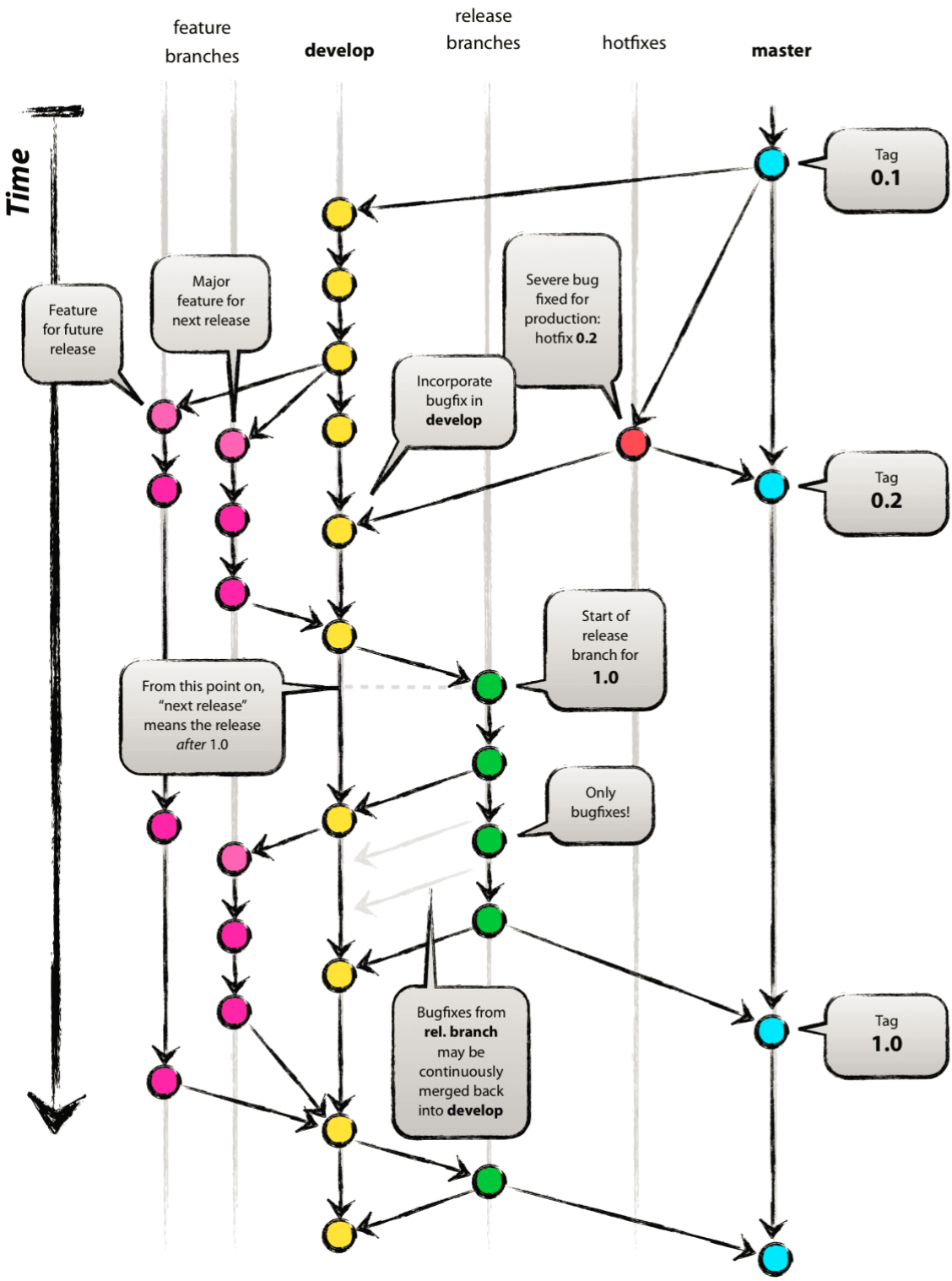
Un modelo de ramificación Git exitoso

En este post os presento el modelo de desarrollo que he introducido para algunos de mis proyectos (tanto en el trabajo como en el privado) hace aproximadamente un año y que ha resultado ser muy exitoso. He estado pensando en escribir sobre esto por un tiempo, pero nunca he encontrado el tiempo para hacerlo a fondo, hasta ahora. No voy a hablar de ninguno de los detalles de los proyectos, simplemente sobre la estrategia de ramificación y la gestión de lanzamientos.



Se centra en torno a Git como la herramienta para el control de versiones de todo nuestro código fuente. (Por cierto, si usted está interesado en Git, nuestra empresa GitPrime proporciona un impresionante análisis de datos en tiempo real sobre el rendimiento de la ingeniería de software.)

¿Por qué git?

Para una discusión exhaustiva sobre los pros y los contras de Git en comparación con los sistemas centralizados de control de código fuente, consulte la web . Hay un montón de guerras de llamas pasando allí. Como desarrollador, prefiero Git sobre todas las demás herramientas de

hoy. Git realmente cambió la forma en que los desarrolladores piensan en la fusión y ramificación. Desde el mundo clásico de CVS / Subversion de donde vengo, la fusión / ramificación siempre ha sido considerada un poco aterradora ("ten cuidado con los conflictos de fusión, te muerden!") Y algo que solo haces de vez en cuando.

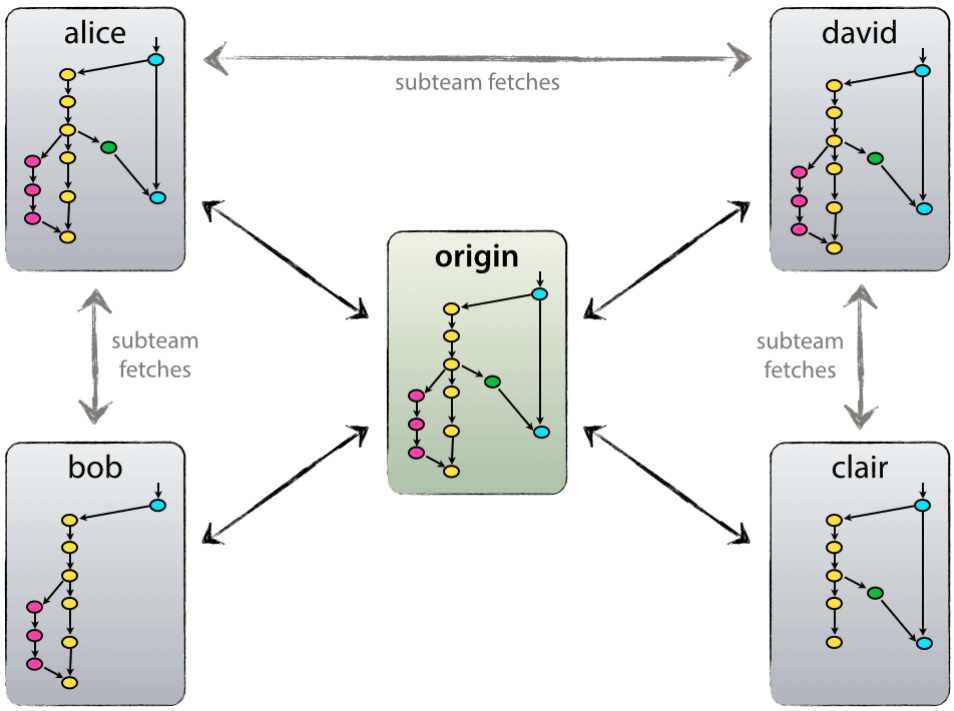
Pero con Git, estas acciones son extremadamente baratas y simples, y se consideran una de las partes fundamentales de su flujo de trabajo *diario* , en realidad. Por ejemplo, en los libros de CVS / Subversion , la ramificación y la fusión se discute por primera vez en los capítulos posteriores (para usuarios avanzados), mientras que en cada libro Git , ya está cubierto en el capítulo 3 (aspectos básicos).

Como consecuencia de su simplicidad y naturaleza repetitiva, la ramificación y la fusión ya no son algo de lo que temer. Se supone que las herramientas de control de versiones ayudan a ramificar / fusionar más que cualquier otra cosa.

Basta con las herramientas, vamos a la cabeza en el modelo de desarrollo. El modelo que voy a presentar aquí no es más que un conjunto de procedimientos que cada miembro del equipo tiene que seguir para llegar a un proceso de desarrollo de software administrado.

Descentralizado pero centralizado

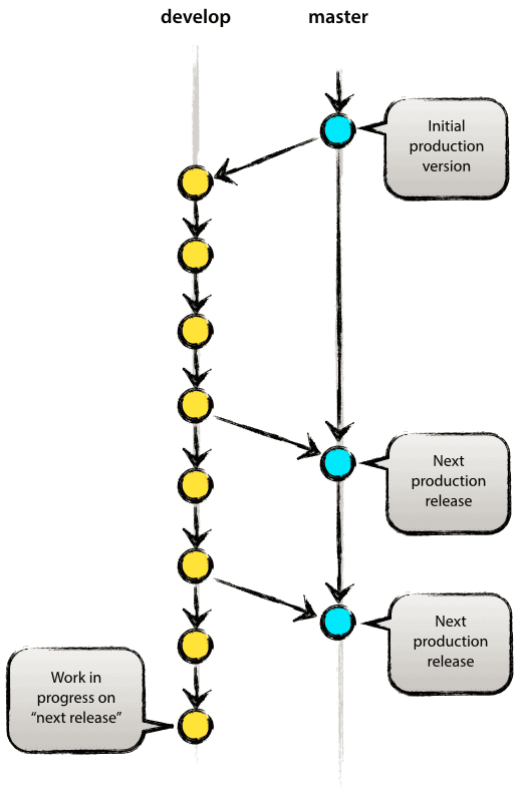
La configuración del repositorio que usamos y que funciona bien con este modelo de ramificación, es que con un repo central de "verdad". Tenga en cuenta que este repo sólo se *considera* el central (ya que Git es un DVCS, no hay tal cosa como un repo central a nivel técnico). Nos referiremos a este repo como `origin`, ya que este nombre es familiar para todos los usuarios de Git.



Cada desarrollador tira y empuja a su origen. Pero además de las relaciones push-pull centralizadas, cada desarrollador también puede extraer cambios de otros pares para formar sub-equipos. Por ejemplo, esto podría ser útil para trabajar junto con dos o más desarrolladores en una gran nueva característica, antes de empujar el trabajo en curso a `origin` prematuramente. En la figura anterior, hay subases de Alice y Bob, Alice y David, y Clair y David.

Técnicamente, esto significa nada más que que Alice ha definido un mando a distancia Git, nombrado `bob`, apuntando al repositorio de Bob, y viceversa.

Las ramas principales



En el núcleo, el modelo de desarrollo está muy inspirado en los modelos existentes. El repo central tiene dos ramas principales con una vida útil infinita:

- `master`
- `develop`

La `master` rama en `origin` debe ser familiar a cada usuario de Git. Paralelamente a la `master` rama, se llama otra rama `develop`.

Consideramos `origin/master` que es la rama principal donde el código fuente de `HEAD` siempre refleja un estado *listo para la producción* .

Consideramos `origin/develop` que es la rama principal donde el código fuente de `HEAD` siempre refleja un estado con los últimos cambios de desarrollo entregados para la próxima versión. Algunos lo llaman la "rama de integración". Aquí es donde se construyen todas las construcciones nocturnas automáticas.

Cuando el código fuente en la `develop` rama alcanza un punto estable y está listo para ser liberado, todos los cambios deben ser fusionados de nuevo de `master` alguna manera y luego etiquetados con un número de liberación. La manera en que se hace esto se discutirá más adelante.

Por lo tanto, cada vez que los cambios se fusionan de nuevo `master`, esto es una nueva versión de producción *por definición* . Tendemos a ser muy estrictos en esto, de modo que teóricamente, podríamos usar un script de gancho Git para construir y desplegar automáticamente nuestro software a nuestros servidores de producción cada vez que hubiese un compromiso `master`.

Apoyo a las ramas

Al lado de las ramas principales `master` y `develop`, nuestro modelo de desarrollo utiliza una variedad de ramas de apoyo para ayudar al desarrollo paralelo entre los miembros del equipo, la facilidad de seguimiento de características, preparar comunicados de producción y para ayudar a fijar rápidamente los problemas de producción en vivo. A diferencia de las ramas principales, estas ramas siempre tienen un tiempo de vida limitado, ya que serán eliminadas eventualmente.

Los diferentes tipos de ramas que podemos utilizar son:

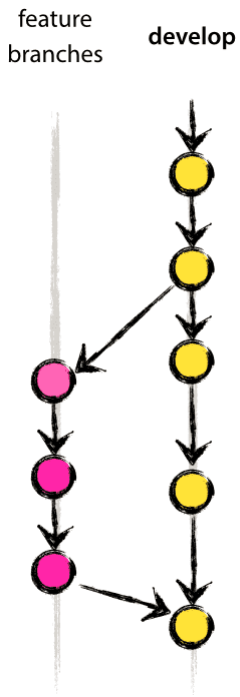
- Rama de funciones
- Liberar las ramas
- Ramas de Hotfix

Cada una de estas ramas tiene un propósito específico y está obligada a reglas estrictas en cuanto a qué ramas pueden ser su rama de origen y cuáles ramas deben ser sus blancos de fusión. Vamos a caminar a través de ellos en un minuto.

De ninguna manera estas ramas son "especiales" desde una perspectiva técnica. Los tipos de sucursal se clasifican por la forma en que los *usamos* . Por supuesto, son ramas viejas de Git.

Rama de funciones

Puede derivarse de:
`develop`
Debe fusionarse de nuevo en:
`develop`
Convención de nomenclatura de sucursal:
nada, excepto `master`, `develop`, `release-*`, o `hotfix-*`



Las ramas de características (o algunas veces llamadas ramas temáticas) se utilizan para desarrollar nuevas características para la próxima o futura versión futura. Al iniciar el desarrollo de una característica, el lanzamiento de destino en el que esta característica se incorporará bien puede ser desconocido en ese momento. La esencia de una rama de la característica es que existe mientras la característica está en desarrollo, pero eventualmente será fusionada nuevamente en `develop` (para agregar definitivamente la nueva característica a la próxima liberación) o desechada (en caso de un experimento decepcionante).

Las ramas de característica suelen existir en repos de desarrollador solo, no en `origin`.

Creación de una rama de entidad

Al iniciar el trabajo en una nueva función, bifurcar de la `develop` sucursal.

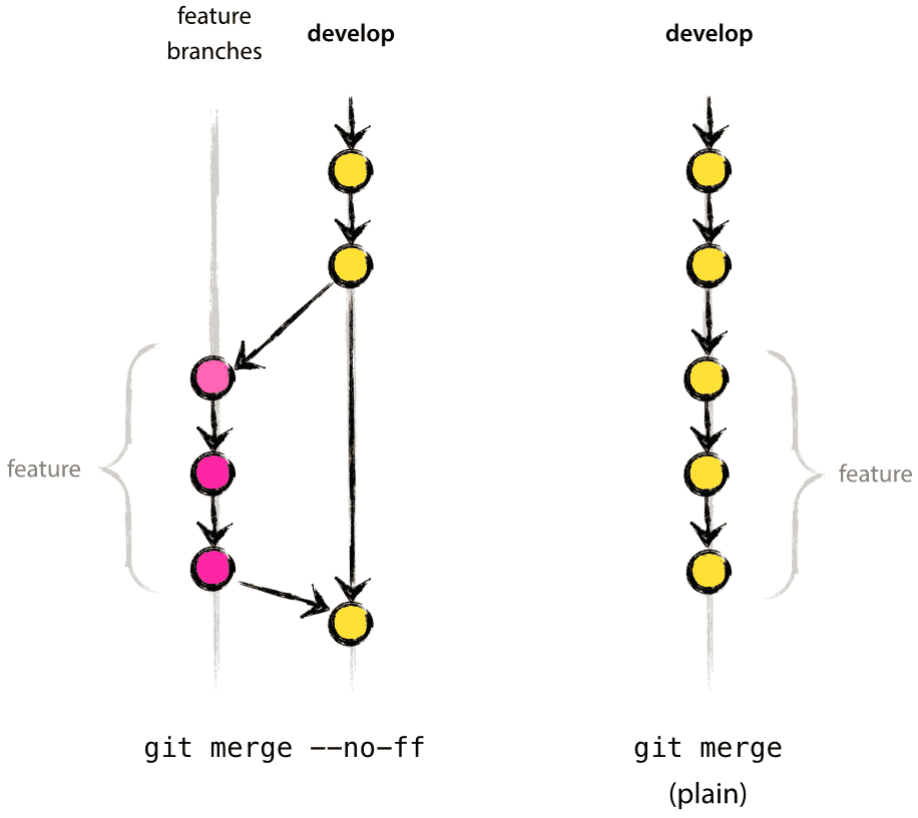
```
$ git checkout -b myfeature develop
 Cambiado a una nueva rama "myfeature"
```

Incorporación de una característica terminada en el desarrollo

Las funciones finalizadas se pueden fusionar en la `develop` sucursal para agregarlas definitivamente a la próxima versión:

```
$ git checkout develop
 Cambiado a la rama 'develop'
$ git merge --no-ff myfeature
 Actualizando ea1b82a..05e9557
(Resumen de cambios)
$ git branch -d myfeature
 Eliminado branch myfeature (was 05e9557).
$ git push origen se desarrollan
```

El `--no-ff` indicador hace que la combinación siempre cree un nuevo objeto de confirmación, incluso si la combinación se podría realizar con un avance rápido. Esto evita perder información acerca de la existencia histórica de una rama característica y agrupa todos los compromisos que juntos agregaron la función. Comparar:



En este último caso, es imposible ver en el historial de Git cuáles de los objetos de confirmación han implementado una característica en conjunto: tendrías que leer manualmente todos los mensajes de registro. Revertir toda una característica (es decir, un grupo de compromisos), es un verdadero dolor de cabeza en esta última situación, mientras que es fácil de hacer si `--no-ff` se usa la bandera.

Sí, creará un poco más (vacío) objetos de confirmación, pero la ganancia es mucho mayor que el costo.

Liberar las ramas

Puede derivarse de:

`develop`

Debe fusionarse de nuevo en:

`develop` y `master`

Convención de nomenclatura de sucursal:

`release-*`

Las ramas de lanzamiento apoyan la preparación de una nueva versión de producción. Permiten punteo de última hora de i's y cruce de t's. Además, permiten correcciones de errores menores y la preparación de metadatos para una versión (número de versión, fechas de compilación, etc.). Haciendo todo este trabajo en una rama de liberación, la `develop` sucursal se borra para recibir características para la próxima gran versión.

El momento clave para derivar una nueva rama de liberación `develop` es cuando se desarrolla (casi) refleja el estado deseado de la nueva versión. Por lo menos todas las características que están dirigidas para el lanzamiento a ser construido deben ser combinadas en `develop` en este punto en el tiempo. Todas las funciones dirigidas a versiones futuras no pueden: deben esperar hasta que la rama de lanzamiento se bifurque.

Es exactamente al comienzo de una rama de lanzamiento que la próxima versión obtiene asignado un número de versión, no anterior. Hasta ese momento, la `develop` rama refleja los cambios para la "próxima versión", pero no está claro si esa "próxima versión" se convertirá en 0,3 o 1,0, hasta que la rama de lanzamiento se inicie. Esa decisión se toma en el inicio de la rama de liberación y se lleva a cabo por las reglas del proyecto sobre el número de versión de choque.

Creación de una rama de liberación

Las ramas de liberación se crean desde la `develop` rama. Por ejemplo, digamos que la versión 1.1.5 es la versión actual de la producción y tenemos un gran lanzamiento. El estado de `develop` está listo para el "próximo lanzamiento" y hemos decidido que se convertirá en la versión 1.2 (en lugar de 1.1.6 o 2.0). Así que nos ramificamos y damos a la rama de lanzamiento un nombre que refleje el nuevo número de versión:

```
$ git checkout -b release-1.2 develop
 Conectado a una nueva rama "release-1.2"
$ ./bump-version.sh 1.2
 Archivos modificados con éxito, la versión cambió a 1.2.
$ git commit -a -m "Número de versión
descargado
a 1,2" [release-1.2 74d9424] Número de versión descargado a 1,2 1 archivos modificados, 1 :
```

Después de crear una nueva sucursal y cambiar a ella, nos topamos con el número de versión. Aquí, `bump-version.sh` es un script de ficción de shell que cambia algunos archivos en la copia de trabajo para reflejar la nueva versión. (Esto puede ser un cambio manual, el punto es que *algunos* archivos cambian).

Esta nueva sucursal puede existir allí por un tiempo, hasta que la liberación se pueda desplegar definitivamente. Durante ese tiempo, las correcciones de errores se pueden aplicar en esta rama (en lugar de en la `develop` rama). Agregar estrictamente nuevas características aquí está estrictamente prohibido. Deben ser fusionados en `develop`, y por lo tanto, esperar a la próxima gran liberación.

Finalización de una rama de liberación

Cuando el estado de la rama de liberación está listo para convertirse en una liberación real, algunas acciones deben realizarse. En primer lugar, la rama de lanzamiento se fusiona en `master` (ya que cada compromiso `master` es una nueva versión *por definición* , recuerde). A continuación, ese commit on `master` debe ser etiquetado para una fácil referencia futura a esta versión histórica. Por último, los cambios realizados en la rama de lanzamiento deben fusionarse de nuevo `develop`, de modo que las futuras versiones también contengan estas correcciones de errores.

Los dos primeros pasos en Git:

```
$ git checkout master
 Conmutado a la rama 'master'
$ git merge --no-ff release-1.2
 Fusionar por recursivo.
(Resumen de cambios)
$ git tag -a 1.2
```

El lanzamiento ahora se hace, y etiquetado para la referencia futura.

> **Editar:** Es posible que desee utilizar los indicadores `-s` o `-u <key>` para firmar su etiqueta de forma criptográfica.

Sin `develop` embargo, para mantener los cambios realizados en la rama de lanzamiento, debemos fusionarlos . En Git:

```
$ git checkout develop
 Cambiado a la rama 'develop'
$ git merge --no-ff release-1.2
 Fusionar por recursivo.
(Resumen de Cambios)
```

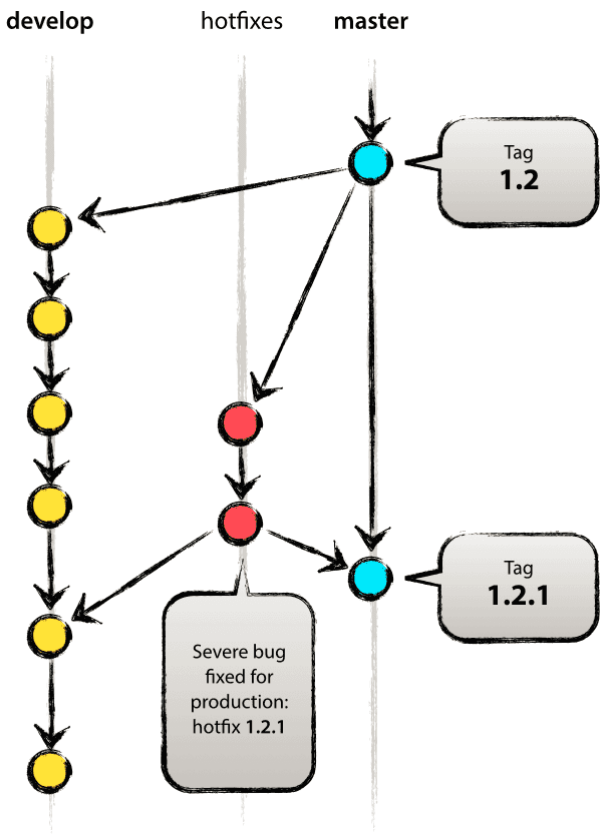
Este paso puede conducir a un conflicto de combinación (probablemente incluso, ya que hemos cambiado el número de versión). Si es así, arreglarlo y comprometerse.

Ahora estamos realmente hechos y la rama de lanzamiento puede ser eliminada, ya que no lo necesitamos más:

```
$ git branch -d release-1.2
 Liberación de rama eliminada-1.2 (fue ff452fe).
```

Ramas de Hotfix

Puede derivarse de:
    `master`
Debe fusionarse de nuevo en:
    `develop` y `master`
Convención de nomenclatura de sucursal:
    `hotfix-*`



Las ramas de Hotfix son muy parecidas a las ramas de liberación, ya que también están destinadas a prepararse para una nueva versión de producción, aunque no planificada. Surgen de la necesidad de actuar inmediatamente sobre un estado no deseado de una versión de producción en vivo. Cuando un error crítico en una versión de producción debe resolverse inmediatamente, una rama de hotfix puede ser derivada de la etiqueta correspondiente en la rama maestra que marca la versión de producción.

La esencia es que el trabajo de los miembros del equipo (en la `develop` rama) puede continuar, mientras que otra persona está preparando una solución de producción rápida.

Creación de la rama de hotfix

Las ramas de Hotfix se crean desde la `master` rama. Por ejemplo, digamos que la versión 1.2 es la versión de producción actual que se ejecuta en directo y que causa problemas debido a un error grave. Pero los cambios `develop` aún son inestables. Podemos entonces ramificar una rama de hotfix y comenzar a solucionar el problema:

```
$ git checkout -b hotfix-1.2.1 maestro
 Cambiado a una nueva rama "hotfix-1.2.1"
$ ./bump-version.sh 1.2.1
 Archivos modificados con éxito, la versión cambió a 1.2.1.
$ git commit -a -m "Cambió el número de versión a 1.2.1"
[hotfix-1.2.1 41e61bb] Cambió el número de versión a 1.2.1
1 archivos cambiados, 1 inserciones (+), 1 eliminaciones (-)
```

No te olvides de golpear el número de la versión después de la ramificación de!

A continuación, corrija el error y confirme la corrección en uno o varios commit separados.

```
$ git commit -m "Problema de producción grave fijo"
[hotfix-1.2.1 abbe5d6] Problema de producción grave fijo
5 archivos modificados, 32 inserciones (+), 17 supresiones (-)
```

Cómo finalizar una rama de hotfix

Cuando se termina, el bugfix necesita ser fusionado de nuevo en `master`, pero también necesita ser fusionado de nuevo en `develop`, con el fin de salvaguardar que el bugfix se incluye en la próxima versión también. Esto es completamente similar a cómo se terminan las ramas de liberación.

En primer lugar, actualizar `master` y etiquetar la versión.

```
$ git checkout master
 Conmutado a la rama 'master'
$ git merge --no-ff hotfix-1.2.1
 Fusionar por recursivo.
(Resumen de cambios)
$ git tag -a 1.2.1
```

Editar: Es posible que desee utilizar los indicadores `-s` o `-u <key>` para firmar su etiqueta de forma criptográfica.

A continuación, incluya el bugfix en `develop`:

```
$ git checkout develop
 Cambiado a la rama 'develop'
$ git merge --no-ff hotfix-1.2.1
 Fusión hecha por recursivo.
(Resumen de Cambios)
```

La única excepción a la regla aquí es que, **cuando existe una rama de release actualmente, los cambios de revisión deben combinarse en esa rama de release, en lugar de** `develop` . Volver a fusionar el bugfix en la rama de lanzamiento eventualmente resultará en la fusión de bugfix en `develop` demasiado, cuando la rama de liberación ha terminado. (Si el trabajo en `develop` inmediatamente requiere esta corrección de errores y no puede esperar a que la rama de lanzamiento esté terminada, puede fusionar el bugfix de forma segura también `develop` ahora.)

Finalmente, elimine la sucursal temporal:

```
$ git branch -d hotfix-1.2.1
 Suprimido rama hotfix-1.2.1 (era abbe5d6).
```

Resumen

Si bien no hay nada realmente sorprendente en este modelo de ramificación, la figura de "gran imagen" que comenzó con este post ha resultado ser tremendamente útil en nuestros proyectos. Forma un elegante modelo mental que es fácil de comprender y permite a los miembros del equipo desarrollar una comprensión compartida de los procesos de ramificación y liberación.

Aquí se proporciona una versión PDF de alta calidad de la figura. Adelante y colgarlo en la pared para referencia rápida en cualquier momento.

Actualización: Y para cualquiera que lo solicitó: aquí está el gitflow-model.src.key de la imagen del diagrama principal (Apple Keynote).

Git-ramificación-modelo.pdf

Si quieres ponerte en contacto, estoy @nvie en Twitter.

Vincent Driessen is an independent Python software engineer and consultant from the Netherlands. You may hire him.