George Stocker
Descubrir cómo construir un mejor software, juntos.

Por favor, deja de recomendar Git Flow.

Git-flow es una metodología de ramificación y fusión popularizada por esta entrada del blog, titulada "A Successful Git branching model".

En los últimos diez años, innumerables equipos han sido engañados por el titular y me atrevo a decir que *se les ha mentido*.

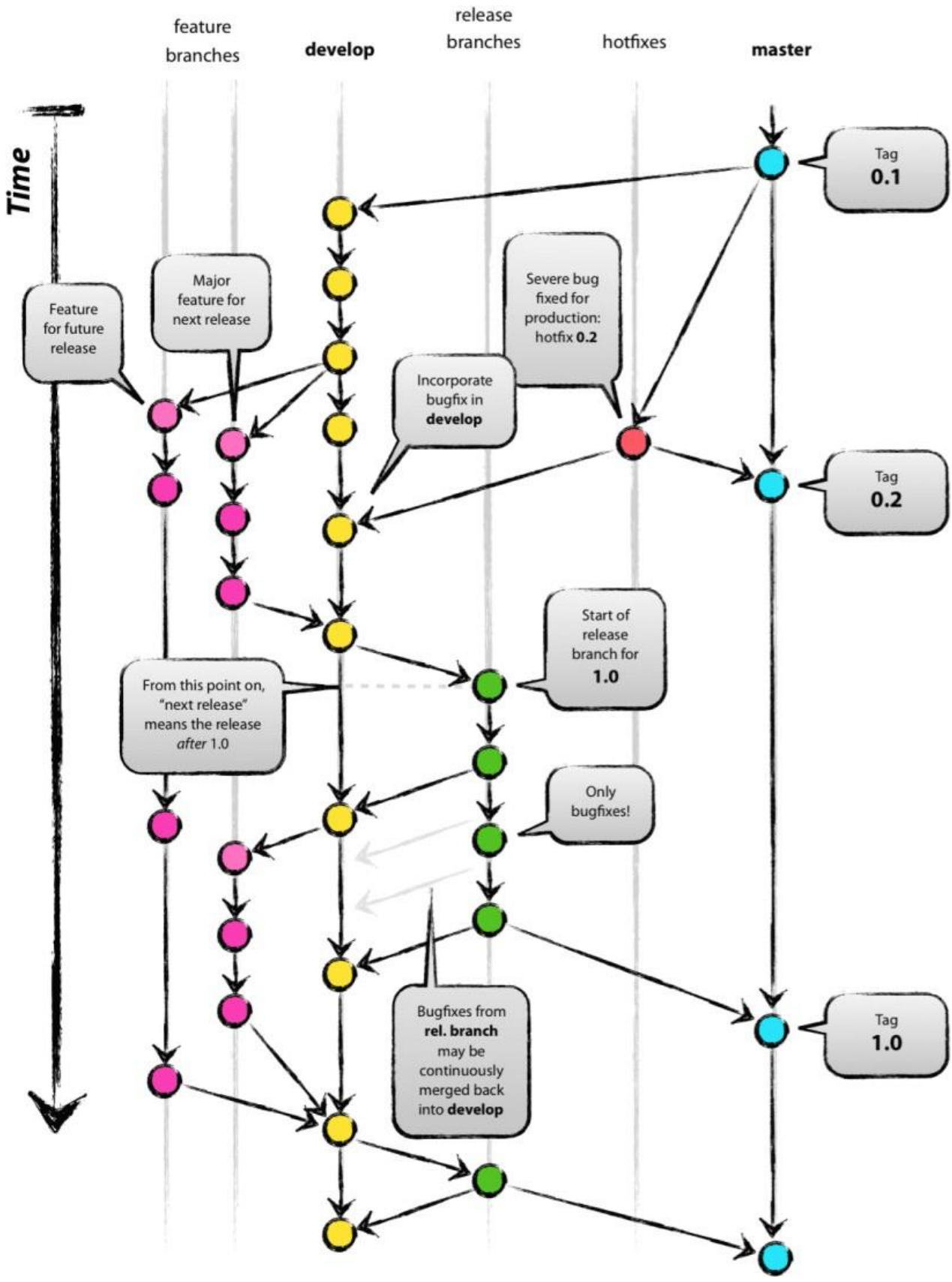
Si lees la entrada del blog, el autor afirma que lo introdujo con éxito en sus proyectos, pero *no habla* a propósito *de los detalles del proyecto que lo hicieron exitoso*.

Y para el resto de nosotros, este es el error nº 1 de confiar en la entrada del blog. Voy a reivindicar como una obviedad que no todas las estrategias funcionan en todas las situaciones, con todas las personas, en todos los contextos, y aplico esa misma lógica a este modelo de ramificación.

El final, ¿verdad? Pues no del todo. Me doy cuenta de que a algunos de vosotros no os convence esta línea de razonamiento, así que vamos a profundizar en por qué el modelo de bifurcación de gitflow debería *morir en un incendio*.

GitFlow es complicado a primera vista

Incluso antes de pensar en los microservicios, o en la entrega continua, gitflow es complicado. Echa un vistazo a esta imagen y dime que es inmediatamente intuitivo:



(fuente: https://nvie. com/posts/a-successful-git-branching-model/)

Así que aquí tienes ramas de características, ramas de liberación, master, develop, una rama hotix, y etiquetas git. Todas estas cosas tienen que ser rastreadas, entendidas y contabilizadas en su proceso de construcción y liberación.

Además, hay que saber en *todo momento* qué rama es cada una. El modelo mental que necesitas retener para que esto sea útil conlleva una alta carga cognitiva. He estado usando git durante 10 años, y ni siquiera estoy seguro de estar en el punto en el que podría mantener mentalmente lo que está pasando aquí.

Gitflow viola la regla de las ramas "de corta duración

En git, el número de conflictos de fusión con personas que se comprometen a una rama aumentará con el número de personas que trabajan en esa rama. Con git-flow, ese número se incrementa aún más, porque hay otras tres ramas (de diferente duración) que se fusionan con develop: Ramas de características, ramas de liberación, y hot- ixes. Así que ahora el potencial de conflictos de fusión no es lineal, va a triplicar potencialmente las oportunidades de conflictos de fusión.

No, gracias.

Aunque dudo en decir que "preocuparse por los conflictos de fusión" es una razón válida para no seguir una estrategia de ramificación como gitflow; la cantidad de complejidad potencial que se introduce cuando todas estas ramas se juntan es demasiado para pasarla por alto.
Esto sería ineficaz si se trata de una organización con una baja tasa de commit, pero para cualquier organización o startup de movimiento rápido apreciable, este no será el caso.

Gitflow abandona el rebasamiento

Reconozco que el rebasing es un tema complejo; pero es importante para esta conversación. Si persigues gitflow, vas a tener que renunciar a rebasing.
Recuerda que el rebasing elimina el merge commit - el punto en el que puedes ver dos ramas uniéndose. Y con la complejidad visual de gitflow, vas a necesitar hacer un seguimiento visual de las ramas, y eso significa que no hay rebasing si quieres deshacer un problema.

Gitflow hace improbable la entrega continua

La entrega continua es una práctica en la que el equipo libera directamente en producción con cada "check-in" (en realidad, una fusión con el maestro), de forma automatizada.
Mira el desastre que es gitflow y explícame cómo vas a ser capaz de entregar continuamente *eso*?

Todo el modelo de bifurcación se basa en un ciclo de liberación predecible y a largo plazo; no en la liberación de nuevo código cada pocos minutos u horas. Hay demasiada sobrecarga para eso; por no mencionar que una de las prácticas centrales de la CD es avanzar con ixes; y Gitflow trata los hotixes como una entidad separada que debe ser cuidadosamente preservada y controlada y separada del resto del trabajo.

Es imposible trabajar con Gitflow en múltiples repositorios

Con la llegada de los microservicios, ha habido un mayor impulso hacia la idea de los microrrepositorios también (cue commenter shouting "they're orthogonal to each other"), donde los equipos individuales tienen el control de sus repositorios y flujos de trabajo, y donde pueden controlar quién comprueba en sus repositorios y cómo funcionan sus flujos de trabajo.

¿Alguna vez has *intentado* un modelo de bifurcación complejo como gitflow con múltiples equipos, y has esperado que todos estén en la misma página? No puede suceder. Pronto, el sistema se convierte en un manifiesto de las diferentes revisiones de los diferentes repositorios, y las únicas personas que saben dónde está todo son las personas que están machacando el YAML para actualizar los manifiestos. "Qué hay en producción" se convierte en una pregunta existencial, si no tienes cuidado.

También es imposible trabajar con Gitflow en un monorepo

Así que si los microrrepos están fuera debido a la dificultad en la coordinación de los lanzamientos, ¿por qué no sólo un gran flujo de trabajo de ramificación que todos los equipos de microservicios tienen que cumplir para los lanzamientos?

Esto funciona durante unos 3,2 segundos, o el tiempo que tarda un equipo en decir "Esto tiene que salir ahora", cuando los otros equipos no están listos para que sus cosas sean liberadas. Si los equipos son independientes y los microservicios deben ser desplegables de forma independiente, no puedes atar tu flujo de trabajo al modelo de ramificación centralizado que creaste en tu mono-repo.

¿Quién debe (y no debe) utilizar Gitflow?

Si tu organización tiene un ciclo de publicación mensual o trimestral y es un equipo que trabaja en varias publicaciones en paralelo, Gitflow *puede* ser una buena opción para ti. Si tu equipo es una empresa emergente, o un sitio web o una aplicación web orientada a Internet, donde puedes tener varias versiones en un día; gitflow no es bueno para ti. Si tu equipo es pequeño (menos de 10 personas), gitflow pone demasiada ceremonia y sobrecarga en tu trabajo.

Por otro lado, si tus equipos están formados por más de 20 personas que trabajan en lanzamientos paralelos, gitflow introduce la suficiente ceremonia para garantizar que no se estropeen las cosas.

Ok, entonces mi equipo no debería usar gitflow. ¿Qué deberíamos usar?

No puedo responder a eso. No todos los modelos de ramificación funcionan para todos los equipos, en todos los contextos y en todas las culturas. Si practica la DC, querrá algo que agilice su proceso tanto como sea posible. Algunas personas apuestan por el desarrollo basado en troncos y las banderas de características. Sin embargo, a mí me dan mucho miedo desde el punto de vista de las pruebas.

El punto crucial que planteo *es hacer preguntas* a tu equipo: ¿Qué problemas nos ayudará a resolver este modelo de ramificación? ¿Qué problemas creará? ¿Qué tipo de desarrollo fomentará este modelo? ¿Queremos fomentar ese comportamiento? Cualquier modelo de bifurcación que elijas tiene como objetivo final hacer que los humanos trabajen juntos más fácilmente para producir software, y por lo tanto el modelo de bifurcación tiene que tener en cuenta las necesidades de los humanos particulares que lo utilizan, no algo que alguien

escribió en Internet y afirmó que era "exitoso".

Nota final del autor: Pensé en utilizar el apelativo de "considerado perjudicial" que es tan común en posts como este; pero luego hice una búsqueda en Google y me di cuenta de que alguien más ya escribió Gitflow considerado perjudicial. Ese artículo también merece la pena.

 geostock / 4 de marzo de 2020 / Sin categoría

---

61 thoughts on "¡Por favor, dejen de recomendar Git Flow!"

 **Stephen Nield**

4 de marzo de 2020 a las 10:03 pm

Esta es una lectura interesante. Me interesaría conocer tu propio flujo de trabajo en git, al menos como una alternativa viable.

 **Louie Christie**

5 de marzo de 2020 a las 1:22 am

Ok, entonces mi equipo no debería usar gitflow. ¿Qué deberíamos usar?

 **CR Drost**

5 de marzo de 2020 a las 12:52

¡Usa Threeflow! (El blog "No Machete Juggling" si tienes problemas para buscarlo en Google)

Quiero decir que cada caso de uso va a ser diferente, pero si estás en un lugar pequeño con repos para aplicaciones individuales, Threeflow va a ser una mejora sólida.

La idea básica es que quieres, en este caso, que todo el mundo esté en la misma sala teniendo la misma conversación, así que te comprometes con la idea de que debemos tener un lugar

donde fusionamos el código de desarrollo a diario como nuestra fuente de verdad sobre el sistema. No hay ramas de características de larga duración. Subordinamos todo el resto de nuestro proceso a esta visión. "Pero necesito controlar qué características se despliegan en mis versiones" - ¡No es un problema para su Git! Eso necesita ser manejado en la lógica del propio repo, ya sea con estados de la DB o env vars que empujan esa cuestión hasta el último momento posible o por un check in ile que define estáticamente qué características están habilitadas, que los devs sobrescriben localmente. "Pero si tenemos 7 toggles de características entonces el equipo de QA tiene que probar 2^7 = 128 coniguraciones diferentes, tal vez"-tienes que hacer que hablen con todos mejor y también restringir el flujo de solicitudes de características en tu sistema de desarrollo para que sea impulsado por tu paso más lento, un límite en el número de toggles de características que permitimos: y tienes que limpiar las ramas después de que han sido desplegadas por una semana. "Pero mis desarrolladores registran el código que se rompe": entonces, necesitas conjuntos de pruebas, cambios de características y revisión del código para asegurarte de que no lo hacen.

Subordinas todo lo demás: "Haré lo que sea necesario para asegurarme de que todo el mundo esté en la misma sala teniendo la misma conversación sobre el mismo código base. Esa es la única cosa en la que no voy a ceder, que todos mis desarrolladores comprueben su trabajo diariamente y vean los conflictos de fusión antes de tiempo para que puedan colaborar en lugar de pisarse unos a otros". Y funciona muy bien a pequeña escala, SI se consigue la participación de los desarrolladores.

(Los desarrolladores se inclinan por las funciones para que sean *feas*, así que hay que insistir mucho en el hecho de "mira, necesitamos que hagas esto para no perder la mitad de tu tiempo en reuniones de planificación de lanzamientos y otras mierdas. Sé que es feo, pero aquí están las formas en que vamos a limitar su fealdad y dejar que el código finalmente se convierta en hermoso").

Runa

9 de marzo de 2020 a las 9:25

Sólo he utilizado los conmutadores de funciones un puñado de veces, pero sospecho que es el camino a seguir.

Suponiendo que tengas una política de publicación sensata. Si te encuentras haciendo muchos parches mientras mantienes varias versiones, entonces estarás atascado con un

un flujo de trabajo más enrevesado.

Mi inicialismo menos favorito: "LTS". Imagínate parchando una rama LTS y luego llevando esa `ix` a tu versión actual así como a tu rama principal de desarrollo. O mejor aún: Empiezas a parchear la versión actual y al cabo de un tiempo alguien pregunta "¿qué pasa con los que todavía tienen la versión LTS?". (mi respuesta suele ser: "Bien, la versión actual se considera ahora LTS. Haz que se actualicen ya que hay un montón de otras `ixes` útiles allí también"). Oh, espera, ¿deberíamos fusionar la LTS `ix` en el master también...?

Sospecho que muchos de nosotros podríamos adoptar el Despliegue Continuo con mucho más vigor. Al menos cuando la interfaz de usuario es principalmente web y no hay despliegue local. En ese punto creo que el flujo de trabajo completo de git se vuelve excesivo.

Geshan Manandhar

5 de marzo de 2020 a las 2:23 am

Utiliza gitflow simplificado.

ertre

19 de noviembre de 2020 a las 10:35

Esta es la respuesta. No estás obligado a utilizar todas las ramas que propone GF.

Craig

5 de marzo de 2020 a las 3:25 am

Entonces, ¿tu post es para hacer pedazos la idea de alguien mientras no ofreces nada de valor tú mismo? Me decepciona que este artículo haya llegado a HN

Jef

9 de marzo de 2020 a las 11:07

Literalmente, escribió una razón reflexiva de por qué no recomendaba algo específico y lo más importante de todo, hacer preguntas a su equipo. Los equipos deberían pensar más por sí mismos y confiar menos en las publicaciones mágicas de los blogs. No critiques a alguien si no te vas a tomar el tiempo de entender su punto de vista.

Chris Graham

19 de mayo de 2022 a las 12:07 am

Los desarrolladores (como la gente en general) sufren de mucho pensamiento grupal y pereza intelectual. En lugar de imponer una solución, la mejor respuesta posible es Preguntar al equipo qué es lo que funciona para ellos.
Sin embargo, donde esto falla es cuando se trata de un equipo joven e inexperto (que cree que lo sabe todo) pero que carece de experiencia para no meterse en problemas. Ese es el momento en el que mando las cosas. Para salvarlos de sí mismos.

Kira

5 de marzo de 2020 a las 6:56 am

Para ser honesto, se empieza con:

"no habla de los detalles del proyecto que lo hicieron exitoso"
Bueno, Gitflow tiene unos 10 años, así que hay muchos buenos proyectos que ya han trabajado con él.

"GitFlow es complicado en su cara"
En realidad no, tal vez sea redundante pero no es complicado.

"También es imposible trabajar con Gitflow en un monorepo"
No, quizá no sea la mejor solución para un desarrollo ágil y rápido, pero por lo demás funciona muy bien.
Incluso si tiene un ciclo de publicación quincenal está trabajando ine.
Me encantaría compartir un proyecto de nuestro equipo, pero como es de suponer no se me permite

Ok, entonces mi equipo no debería usar gitflow. ¿Qué
deberíamos usar? ¿No puedes responder a eso?
Bueno, hay tantos otros flujos de trabajo que podría dar algunos ejemplos

Pero sí, has mostrado algún ejemplo en el que puedes tener problemas pero decir que no es bueno es un poco injusto.
Así que creo que no estás completamente equivocado, pero también tengo que decir que no entiendes realmente el concepto detrás de gitflow ya que no es realmente complicado si entiendes git.
También se discute muy a menudo en diferentes foros y demás.

https://stackoverflow. com/questions/18188492/cuáles son los pros y los contras de git-flow-vs-github-flow

---

Leszek

6 de mayo de 2021 a las 4:32 am

exactamente.

Ian

5 de marzo de 2020 a las 10:27

Francamente, Git debería morir en un incendio.

No es que lo que hace Git sea malo. Pero la idea de que no necesitaba una interfaz visual estructurada en forma de árbol que se pareciera a ile explorer, SourceSafe o team foundation server es simplemente arrogancia de programador en su peor momento. Esas interfaces se produjeron por una sólida razón de factores humanos y no fueron caprichosas ideas de última hora. Daban al usuario un modelo mental, mostraban la información del servidor local y remoto de forma precisa y permitían realizar acciones puntuales sobre una ile, carpeta o proyecto que podías VER claramente. Las opciones de la línea de comandos estaban disponibles y las usaba en los archivos por lotes con frecuencia, pero los archivos por lotes eran útiles porque podía VER lo que estaba haciendo, simultáneamente, tanto localmente como en el servidor.

La visibilidad y la acción local es lo que hizo que los antiguos sistemas de control de versiones

inmediatamente útil sin apenas formación. Git, SourceTree GitHub desktop, etc. No tanto. Git parece *diseñado* para ocultar información, posiblemente con la intención de evitar errores. Por supuesto, si haces que sea difícil cometer errores fácilmente, también puedes hacer que sea extremadamente difícil hacer algo.

No es que Git no se pueda aprender. Yo lo uso a diario. Es sólo que era una alternativa mal pensada. Es *técnicamente* genial. Es la parte de los factores humanos la que es un desastre.

Y, francamente, si no entiendes el sistema nervioso humano y estás haciendo un producto orientado al ser humano, eres un pésimo ingeniero porque no entiendes o ignoras la mitad del sistema que estás construyendo.

DarkSwordsman

5 de marzo de 2020 a las 13:45

Soy desarrollador desde hace sólo 2 o 3 años y no tengo ningún problema para visualizar las ramas de git y lo que no en la terminal. La única interfaz de usuario que utilizo es el sitio web de Github, y eso es estrictamente para cuestiones y PRs. Rara vez miro los gráficos de las ramas reales o algo así.

erwer

19 de noviembre de 2020 a las 10:40

He sido un desarrollador de ~ 25 años en grandes proyectos y empresas como IBM, y el sangrado de borde en Manhattan. Yo ind Git a ser overengineered y mucho de eso es forzado en el desarrollador de usuario. Si eres un nuevo desarrollador y has vivido en el vacío puede que no hayas experimentado sistemas que no tengan ni de lejos ese nivel de falta de intuición pero que sigan teniendo la mayor parte de la potencia. Esta es la razón por la que un gran punto de venta de los clientes de Git es oscurecer eso, lo que sólo pueden hacer hasta cierto punto.

---

5 de marzo de 2020 a las 10:29 pm

"No es que Git no se pueda aprender. Yo lo uso a diario. Es sólo que era una alternativa mal pensada. Es *técnicamente* genial. Es la parte de los factores humanos la que es un desastre".

Estoy de acuerdo con esto y añado que en este aspecto Git es típico para lo que viene del mundo Linux.

 **Fabricio Bertoncello Scariot**

28 de septiembre de 2020 a las 20:58

> Dijo todo lo que quería decir, todo se está convirtiendo en Linux, estamos perdiendo el tiempo decorando y tecleando comandos en lugar de dedicar este tiempo a las mejoras del negocio.

 **Thomas Wheeler**

5 de marzo de 2020 a las 10:42 pm

"Francamente, Git debería morir en un incendio. "

Me gustaría tomar tus palabras de vuelta y dispararlas. La interfaz de línea de comandos de Git es criticada con razón, pero git es el único sistema de versionado que he utilizado que lo hace bien[1]. Y punto. Su DAG, y especialmente, el concepto de que cada confirmación representa todo el árbol de trabajo en un momento dado, es simple y correcto.

100 veces más, prefiero usar una herramienta que lo hace bien y tiene una interfaz de usuario completamente de mierda, que una herramienta que tiene una hermosa interfaz de usuario pero no hace las cosas bien.

[1] He oído hablar muy bien de Mercurial y de un par de herramientas más, pero conozco git y este hilo es sobre git.

Fabricio Bertoncello Scariot

28 de septiembre de 2020 a las 21:01

Para los que conocían SourceSafe, CVS y SVN, no, Git no tiene ningún sentido. No vi ningún beneficio, y hasta hoy no he conocido a nadie que utilice la gran diferencia que supone el commit local

---

ertre

19 de noviembre de 2020 a las 10:42

Es posible tener ambas cosas. El desarrollo de software no empezó hace 15 años y fue sólo ine.

Juan Blanco

14 de abril de 2021 a las 10:39

Claro, el desarrollo de software existía hace más de 15 años antes de git, pero déjame decirte cómo era:

Solía usar Subversion. Hacer ramas era una pesadilla de pesadillas. Nunca lo hice por mi cuenta, contratamos a alguien para que lo gestionara. Le pagábamos a alguien 50 mil al año para que hiciera eso.

Perforce... oh, Perforce. Para evitar conflictos, se bloquean los archivos en Perforce. Si quiere hacer un cambio en un archivo de código fuente, espere que nadie lo haga primero. Literalmente, resolvieron los conflictos al no permitir que NINGUNA DE LAS DOS PERSONAS trabajara en el mismo íle al mismo tiempo.

¿SourceSafe? Teníamos un EQUIPO de ingenieros de SourceSafe para gestionar esa mierda.

Entonces sí, ¿Git es bastante bueno? ¿Puede ser complejo? A veces, tal vez. Pero está resolviendo un problema MUY COMPLEJO. Da las gracias.

John

5 de marzo de 2020 a las 11:34 pm

git -all -decorate -oneline -graph Alias
que a 'git adog'.

Git se diseñó para personas que pensaban de una determinada manera, que se adherían a una determinada metodología y que tenían un cierto nivel de habilidad técnica. No todos los humanos son iguales. No estaba mal pensado, simplemente no era lo que tú querías. Los diseñadores de git no son los que sufren de arrogancia de programador en su peor momento.

Eduardo Brites

6 de marzo de 2020 a las 7:56 am

Me encanta Git y estoy de acuerdo contigo, yo uso Git en Visual Studio 2019 y no recuerdo la última vez que necesité usar la línea de comandos.

Bloodgain

18 de septiembre de 2020 a las 21:00 horas

Tienes críticas justas a la interfaz de línea de comandos de git, pero se supone que git nunca fue la interfaz directa para el usuario general. Por eso se desarrollaron GUIs para él. Estas GUIs *deberían* ser desarrolladas por separado, y la filosofía de diseño de git es apoyar las herramientas externas haciendo que todo esté disponible, incluyendo la fontanería y el propio árbol del repositorio.

Dicho esto, para los usuarios avanzados (es decir, yo), los comandos de porcelana de la línea de comandos son excelentes. A veces tengo que referenciar una página de documentación (`git help log`) para recordar las opciones - como todo lo demás en el mundo de la shell - pero puedo inducir lo que quiera con un comando o tres. Sí, la letanía de opciones disponibles para comandos como `git log` es abrumadora, pero las opciones son en su mayoría intuitivas, y este es exactamente el caso de uso para el que están los alias. La salida por defecto

de `git log` es sólo moderadamente útil para los propósitos más comunes, que son

es cierto para muchos otros comandos de git, pero también es la salida más obvia: "muéstrame el registro de confirmaciones".

Xedni

10 de octubre de 2020 a las 4:53 am

Parafraseando a David MacIver:
"Optimizar tu notación para que no confunda a la gente en los primeros 10 minutos de verla, pero para que dificulte la legibilidad siempre, es un error muy grave". Viniendo de svn/tfs/visit, sí. Git es desalentador. Admitiré que me llevó un buen año sentir que no era una amenaza activa para la consistencia de cualquier base de código en la que trabajara. Pero tomarse en serio la comprensión de Git es esencial para
ser un programador porque a) está realmente *realmente* bien diseñado y b) la gente lo conoce. Si todo lo que buscas es una Gui, hay muchas (Git viene con una Gui minimalista; otro ejemplo es GitKraken). Pero la verdadera razón por la que parece estar mal diseñado es (afirmo) un error del usuario. Git es dramáticamente diferente de la mayoría de los otros sistemas de control de fuentes. Sí, ambos hacen ruido, pero las similitudes no van mucho más allá de eso.
Así que "ixar" git para que se parezca más a svn es como pedirle a tu concesionario que vuelva a poner la piscina en tu coche. Los coches no tienen piscinas, y sólo acabarás con un coche que se maneja como una mierda, con una piscina en la que nadie se baña.

Dejando de lado toda la cera poética, si todavía estás temblando con Git, te recomiendo este vídeo:
https://youtu.be/1ffBJ4sVUb4
Puede que no salgas del otro lado preparado para fusionar 10 ramas de código diferentes mientras haces girar una pelota de baloncesto sobre tu nariz, pero *entenderás* las decisiones de diseño que se tomaron, e identificarás los lugares en los que puedes haber estado metiendo clavijas cuadradas en agujeros redondos.

ewtwe

19 de noviembre de 2020 a las 10:38

Estoy de acuerdo, y la terrible y poco intuitiva nomenclatura de ciertas funciones es engañosa a veces. Todavía me aterra a veces cuando realizo ciertas funciones porque

varias veces en el pasado ha explotado.
Todo el sistema es excesivamente complicado en comparación con algo lógico como el viejo sourcesafe (que de hecho tiene sus propios problemas)

Utilizo Git a diario, y Gitkracken es una interfaz de usuario fantástica, pero sigo pensando que la tecnología tiene problemas importantes.

 **Bryan Finster**

5 de marzo de 2020 a las 10:36

Hay muchas cosas buenas aquí. Publicando este enlace internamente.

 **Piotr Mionskowski**

5 de marzo de 2020 a las 12:09

A mí también nunca me ha gustado git flow. Para nosotros, suponiendo un equipo pequeño, lo siguiente funciona bien https://brightinventions. pl/blog/how-do-we-use-git/

 **Carl**

5 de marzo de 2020 a las 12:14

El flujo de Git es lo suficientemente bueno para empezar y realmente te llevará muy lejos. Es una receta de cómo usar ramas de manera efectiva, porque son baratas.
Ahora hay muchos otros modelos de ramificación que son buenos para muchos escenarios diferentes.
Quejarse de un modelo de ramificación sin ni siquiera recomendar cuando los otros son adecuados es una mala lectura.

 **Varilla**

5 de marzo de 2020 a las 12:14

Recomiendo Three-Flow como alternativa: https://www.

nomachetejuggling. com/2017/04/09/a-different-branching-

estrategia/

 **Tristan Zimmerman**

5 de marzo de 2020 a las 12:40

No puedo estar más de acuerdo y tuve la misma reacción cuando vi ese diagrama hace muchos años.

Para cualquiera que se pregunte qué hacer en lugar de Gitflow, yo sugeriría lo siguiente: Empezar de forma sencilla.

— No te comprometas con el Maestro (obviamente).
— Todo el trabajo se realiza en una rama separada que se fusionará con la maestra.
— Mantenga sus relaciones públicas pequeñas para que sean fáciles de entender y de revisar.
— Mantener a Master en un estado de despliegue constante.

Para manejar las complicaciones de la adición de grandes características, buscar maneras de ocultarlas en prod para que puedan entrar en Master tan pronto como sea posible. Las banderas de características son fantásticas para esto, pero incluso la construcción de una ruta temporal para que pueda ver los cambios, pero no exponerlos a su usuario funcionará simplemente.

Además, escribir pruebas para el trabajo también hace que las fusiones pequeñas y frecuentes sean menos peligrosas. No son perfectas, pero si todo tiene una prueba alrededor, es menos probable que se rompan cosas, incluso si se fusionan una docena de PRs al día.

De todos modos, me encanta el post. 👍

 **Bill**

13 de julio de 2020 a las 8:38

Esto es esencialmente gitflow, el autor ha confundido de alguna manera la ramificación de características con las ramas que viven para siempre.

No estoy del todo seguro de por qué cree que gitflow no funciona con CI/CD ya que cuando se utiliza gitflow sólo se libera desde una rama de liberación, que es

tomada de la rama de desarrollo, que es un compuesto de todas las ramas de características (de corta duración).

las ramas feature/hotix/bugix son de corta duración, nada en gitflow dice que estas ramas deban vivir más tiempo del necesario

Puede que gitflow no sea lo más adecuado para todo el mundo y para todos los proyectos, pero desde luego no plantea ningún problema para la integración o la entrega continuas. Significa que un equipo no necesita implementar la ocultación de características en absoluto, lo que generalmente implica código adicional que es superfluo para la solución real.

Ryan Cline

5 de marzo de 2020 a las 15:58

Para continuar donde el autor lo dejó:

En resumen, el desarrollo basado en troncos es la solución que está buscando. https://trunkbaseddevelopment.com/

Trunk-based Development lo tiene todo, y ese sitio web debería considerarse de lectura obligatoria para cualquiera que implemente o actualice su flujo de trabajo de desarrollo personal o profesional.

Si ese sitio no te convence, entonces lee State of Devops 2019 https://services. google. com/fh/iles/misc/state-of-devOps-2019.pdf

Hasta luego GitFlow, lo que teníamos... no era bueno.

¡Salud, por su futuro mejor desarrollo! Bienvenido.

Bill

13 de julio de 2020 a las 8:40

Realmente no estoy convencido de que trunkbaseddevelopment se diferencie de

Eric Rini

5 de marzo de 2020 a las 16:43

Sí. Git flow es un buen ejemplo de gente que sabe lo que tiene que hacer y se olvida de por qué lo hace.

Ben

5 de marzo de 2020 a las 17:00

Me hago eco de los otros comentaristas: ¿puedes dar un ejemplo de un proyecto en el que hayas trabajado, y qué modelo de ramificación de Git tuvo éxito y por qué? De lo contrario, este post sólo sirve para desanimar y confundir a las personas que, como yo, han utilizado Gitflow en el pasado.

---

d potter

5 de marzo de 2020 a las 18:24

¿Eh? Hay muchos menos conflictos de fusión cuando se utiliza el flujo de git correctamente. En el diagrama de arriba faltan algunas fusiones fantasma desde desarrollo hasta las ramas de características (cada vez que hay un push de otra persona a desarrollo, específicamente). Pero esa es una de las principales ventajas: tener una estrategia de ramificación altamente jerárquica y rígidamente adherida elimina los conflictos de fusión que no están directamente relacionados con el código que el desarrollador que está haciendo el pull creó.

Podría ir punto por punto sobre por qué parece que nunca has trabajado con gente que sabía hacer esto correctamente y por qué eso está coloreando tu opinión, pero acabo de conseguir un 70 años de edad en este modelo de TFS el año pasado y lo tengo amando, así que no es demasiado tarde para ti.

Pero para que lo sepas: "Esto tiene que salir ya" es como se sabe que hay que usar una rama hotix que funciona sólo contra el código de producción/master. "Lo que está en producción" es siempre fácil - es lo que está en master. La entrega continua es una casilla de verificación para el desarrollo, y se agrega un combo /

desplegable que se actualiza una vez por versión para QA para apuntar a la nueva rama de liberación. Y si no puedes coordinar los lanzamientos y

código compartido con GitFlow, no lo vas a tener más fácil rodando tu propia estructura y haciendo un seguimiento de que sea diferente cada versión...

Buena suerte ahí fuera.

Bill

13 de julio de 2020 a las 8:42

Estoy 100% de acuerdo

El uso de gitflow es un cambio de mentalidad, pero ciertamente no impide el CI o el CD.

No puedo entender por qué el autor asume que las ramas de características no son de corta duración cuando se utiliza gitflow

---

Zeno Lee

5 de marzo de 2020 a las 11:09 pm

El desarrollo basado en el tronco es una gran alternativa sencilla

---

Brent DeMark

7 de marzo de 2020 a las 19:52

Esto es lo que hemos estado usando durante años. Y funciona bien. Yo diría que empieces por aquí, y si necesitas algo más, entonces busca estrategias de ramificación más involucradas como git flow. Yo advertiría en contra de saltar a algo tan complicado a la derecha de la puerta para los nuevos proyectos.

Bill

13 de julio de 2020 a las 8:43

En realidad no es diferente a gitflow, así que no es realmente una alternativa

Scott Davey

6 de marzo de 2020 a las 3:46 am

Llevo más de 5 años utilizando con éxito Git Flow en un entorno de entrega continua con múltiples repositorios y microservicios, y he comprobado que es realmente bueno para nuestro equipo.

Lo que Git Flow hizo por nosotros es estandarizar un flujo de desarrollo, que eliminó las diferencias entre nuestros expertos en Git y los demás, y dio a todos un modelo mental compartido para mejorar el trabajo en equipo.

A pesar de las experiencias de esta entrada de blog, en nuestro equipo realmente redujo los conflictos de fusión, redujo las ramas de larga duración, y eliminó las ramas perdidas porque también eliminó las fusiones difíciles de git-fu de las ramas temáticas junto con un desprendimiento de bicicletas.

Nos dio una mayor claridad a nuestro flujo de trabajo.
Incluso hemos vinculado nuestros procesos a los conceptos de flujo de git, y esto significó que todos teníamos una comprensión compartida en todo el equipo, incluidos los desarrolladores, los operadores e incluso los gerentes, que sabían, por ejemplo, qué tipo de trabajo podría ser un hotixed frente a la necesidad de una liberación completa.

¿Podrían llegar todos estos beneficios de otra manera? Bueno, ciertamente. Pero para nosotros hubo una clara transición antes/después en la que todos nuestros problemas anteriores con Git desaparecieron después de introducir Git Flow. No es perfecto, pero resuelve más problemas de los que crea para nosotros, y lo hemos hecho funcionar en un entorno de entrega continua, de microservicios y de cumplimiento de las palabras de moda.

Paul Hammant

6 de marzo de 2020 a las 4:19 am

Oye George, lo que quieres decir es desarrollo basado en el tronco, no desarrollo sin tronco. Se ha escrito y hablado de ello desde hace 20 años 🙂

Salem Korayem

6 de marzo de 2020 a las 4:46 am

El flujo Git es una metodología escalable que usted. El del diagrama es el caso extremo.

En mi empresa, usamos master, staging (como la rama de desarrollo en el diagrama) y feature branch todo desde staging. Eso es todo. No hay rama hotix o rama de liberación.

Vinculamos todos los PRs con Jira a través de un script que actualiza en Jira un campo personalizado que especifica si esta incidencia ha sido fusionada o no y si se ha fusionado, en qué rama: staging o master o ambas. De esta forma podemos hacer fácilmente un JQL y saber en qué punto se encuentra cada incidencia.

Si hoy hago una rama de características desde el staging y cuando termino de trabajar, el banco de staging recibe actualizaciones, vuelvo a basar la rama de características en el staging.

El hecho de que no sugieras una alternativa podría implicar que no has probado nada más que realmente funcione antes de escribir este post.

Bob Barbell

6 de marzo de 2020 a las 8:20

Este nivel de publicación está en torno a esto:

¡Los coches son malos! Porque sí:
— demandas de gas
— debes tener permiso de conducir y estudiar las leyes para conducir un coche
— sólo se mueven cuando se arranca el motor y se pisan los pedales
— requiere una revisión periódica
— hay mayores riesgos de accidente de tráfico
— debe decidir qué color debe tener su coche

Cualquier enfoque tiene sus propias ventajas e inconvenientes. Y se trata de saber utilizar adecuadamente las herramientas accesibles y no de proclamar el "stop

recomendando esta herramienta". Gitflow es la herramienta perfecta para realizar cambios en el código y es definitivamente mejor que muchos enfoques personalizados que la gente suele inventar.
Las desventajas reales de Gitflow afectan sobre todo a los procesos de gestión y planificación de proyectos y no al desarrollo de software en sí.

Capitán Obvio

6 de marzo de 2020 a las 9:07

Así que para resumir: no hay sólo un enfoque git-flow y usted debe elegir de los equipos necesitan. También suena como un problema para manejar más de una rama

---

Anushervon Saidmuradov

6 de marzo de 2020 a las 17:13

Esto es algo que funciona para nosotros:

— Rama maestra única donde van todos los cambios. No se hacen commits aquí directamente (se puede hacer una excepción con los hotixes)
— Las ramas de características se crean a partir del HEAD maestro y deben ser de corta duración.
— Las ramas de entorno (dev, stage, prod), a diferencia de las ramas de características, son permanentes y siempre se basan en la última HEAD de master. Los cambios en estas ramas activan el canal de CI/CD para desplegar en esos entornos automáticamente. Utilizamos clusters Kubernetes separados alojados en GCP/Azure/AWS como entornos de despliegue. No se deben hacer commits directamente en estas ramas, todos los commits van al master a través de las ramas de características

---

Daniel Marbach

7 de marzo de 2020 a las 4:04 am

Tuvimos GitFlow durante muchos años y el mayor problema que teníamos es que la gente continuamente se olvidaba de devolver los hotix a la rama de desarrollo.

Ahora con release flow este problema ha desaparecido y todavía podemos mantener y hacer hotix de múltiples versiones soportadas, lo cual es una preocupación clave para nosotros.

https://docs. microsoft. com/en-us/azure/devops/learn/devops-at-microsoft/release-flow

---

Iurii

10 de marzo de 2020 a las 21:12

1. ¿Qué es la producción? Es sencillo: el maestro 🙂
2. ¿Por qué se mezclan microservicios y GIT? ¿Por qué el equipo de microservicios debe depender de otro? Ese es el punto de que los equipos son totalmente independientes, uno puede tener Git, otro Subversion o X(poner nombre). Diferentes versiones, etc.
3. Proponer en lugar de culpar.

Stefano

14 de marzo de 2020 a las 7:25 am

Dejé de leer en el párrafo sobre el rebase. Amigo, si "reconoces que el rebase es un tema complejo", vuelve a las aulas y aprende GIT desde cero. El rebase es -si alguna vez lo entendiste- para nada complejo. Yo uso GitFlow a diario y soy capaz de hacer rebase.

Bill

13 de julio de 2020 a las 8:47

rebasar en lugar de fusionar desde dev es una buena práctica IMO, y gitflow realmente no impide esto en absoluto

---

Alex

30 de abril de 2021 a las 11:29

Esta fue también mi impresión.

El autor tiene las mismas quejas que los desarrolladores junior recién introducidos en git y CI/CD como concepto. Después de manejar diariamente gitflow durante un par de semanas y ver la

problemas que previene/negocia nunca se vuelve atrás.

LincWong

19 de marzo de 2020 a las 21:38

creo que gitlab-flow es la respuesta.

Gesiel Kloeppel

20 de marzo de 2020 a las 16:16

No entiendo cómo a tanta gente le gusta GitFlow. ¿Son esas personas las que gestionan los lanzamientos y los hotixes? Lo dudo. Si crees que te gusta y no tienes ramas de release y hotix, probablemente no haces GitFlow.

A mi entender, GitFlow es valioso sólo si necesitas dar soporte a más de una versión de tu software. En otros casos, un maestro de producción es suficiente.

Alexey Lebedev

5 de junio de 2020 a las 11:25

Querido George,
¡Muchas gracias por este post! Me duele ver a la gente usando gitflow. Personalmente creo que es un modelo muy ineficaz e incómodo.
Recuerdo cómo elegíamos el modelo de ramificación. Rechazamos contiamente gitflow (y fue la mejor decisión que tomamos), aunque se hizo muy popular.
Se nos ocurrió el siguiente modelo de ramificación bastante simple:
-tenemos la rama master, que corresponde a prod
-si queremos desplegar algunos cambios en prod, preparamos o rama, que se fusionará en master
-las ramas de las tareas desplegadas se fusionan en la
rama - se fusiona en el maestro
-Antes de fusionar cualquier rama, se debe rehacer, los conflictos se resuelven y todas las fusiones son siempre de avance rápido (sin merge-commits)

Teníamos más de 20 personas haciendo commit en el mismo repositorio y nuestro historial de git estaba limpio y claro sin un solo merge commit. Llevamos años utilizando este modelo y ha demostrado ser muy bueno.
Me siento mal cuando miro los repositorios de la gente con todos estos commits de fusión... ¿Cómo saben siquiera, lo que está pasando allí?
Sí, debes saber cómo usar rebase. Pero si eres un desarrollador profesional debes ser bueno con git

---

Bill

13 de julio de 2020 a las 8:49

¿En qué se diferencia esto de usar gitflow?

Pingback: Weekly CW011-2020 - [¡ÚNICA INSTANCIA DEV!] - Hallazgos y cositas

---

ograterol

8 de mayo de 2021 a las 7:28 pm

Interesante... Creo que gitflow es una alternativa, pero hay otras que ofrecen más capacidades. Me gusta gitlab o github flow.

Pingback: Así nos ramificamos - laredoute.io

Rineez Ahmed

18 de julio de 2021 a las 11:09 pm

¡Acabo de leer esto y me siento tan ClickBaited! x-(

Si bien hay algunos puntos válidos sobre no seguir ciegamente algún flujo de trabajo sólo porque algunas personas lo llaman defacto o estándar (eso es cierto para todas las prácticas,

no sólo el flujo de trabajo) la esencia del artículo no coincide con el título del artículo. Este post en sí está recomendando Gitflow para ciertos casos. ¡Así que definitivamente CLICKBAIT!

Vali Spam

17 de septiembre de 2021 a las 4:58 am

Yo llamo a esto un artículo de cebo de clic.

Tiene un título pegadizo "¡Por favor, deja de recomendar Git Flow!". El título hace que suene como un punto de vista final sobre gitflow. Luego pasa a argumentar cómo no se debe usar gitflow si se tiene una arquitectura de microservicios que se despliega con frecuencia.

La mayoría de los desarrolladores trabajan en aplicaciones de línea de negocio en las que se utiliza un ciclo conservador de desarrollo-cuadro-lanzamiento-día. La mayoría de los desarrolladores de mi zona trabajan en este tipo de sistemas.

Damien

17 de noviembre de 2021 a las 18:05

Realmente depende de cómo lo uses. Hemos estado usando gitflow durante 6 años y nunca hemos tenido ningún problema con él. Yo sigo confiando en él.