Scott Chacon

GitHub Flow

31 de agosto de 2011

Problemas con git-flow

Viajo por todo el lugar enseñando Git a la gente y casi todas las clases y talleres que he hecho recientemente me han preguntado qué pienso sobre git-flow . Siempre respondo que creo que es genial: se ha tomado un sistema (Git) que tiene un millón de flujos de trabajo posibles y se documentó un flujo de trabajo flexible y bien probado que funciona para muchos desarrolladores de una manera bastante directa. Se ha convertido en una especie de estándar para que los desarrolladores puedan moverse entre proyectos o empresas y estar familiarizados con este flujo de trabajo estandarizado.

Sin embargo, tiene sus problemas. He escuchado varias opiniones de personas como que no les gusta que las nuevas ramas de características se inicien en `develop`lugar de hacerlo `master`, o la forma en que maneja las revisiones, pero son bastante menores.

Uno de los problemas más importantes para mí es que es más complicado de lo que creo que la mayoría de los desarrolladores y equipos de desarrollo realmente requieren. Es lo suficientemente complicado como para que se haya desarrollado un script de ayuda grande para ayudar a reforzar el flujo. Aunque esto es genial, el problema es que no se puede aplicar en una GUI de Git, solo en la línea de comandos, por lo que las únicas personas que tienen que aprender el flujo de trabajo complejo realmente bien, porque tienen que hacer todos los pasos manualmente, son las mismas personas que no se sienten cómodas con el sistema lo suficiente como para usarlo desde la línea de comandos. Esto puede ser un gran problema.

Ambos problemas se pueden resolver fácilmente con solo tener un proceso mucho más simplificado. En GitHub, no usamos git-flow. Utilizamos, y siempre hemos utilizado, un flujo de trabajo de Git mucho más simple.

Su simplicidad le da una serie de ventajas. Una es que es fácil de entender para las personas, lo que significa que pueden retomarla rápidamente y raramente o nunca la estropean o tienen que deshacer los pasos que hicieron mal. Otra es que no necesitamos una secuencia de comandos envoltorio para ayudar a hacer cumplir o seguir, por lo que el uso de GUI y tal no es un problema.

GitHub Flow

Entonces, ¿por qué no usamos git-flow en GitHub? Bueno, el problema principal es que desplegamos todo el tiempo. El proceso de git-flow está diseñado en gran medida en torno a la "liberación". Realmente no tenemos "lanzamientos" porque desplegamos a la producción todos los días, a menudo varias veces al día. Podemos hacerlo a través de nuestro robot de sala de chat, que es el mismo lugar donde se muestran los resultados de mi CI. Tratamos de hacer que el proceso de prueba y envío sea lo más simple posible para que cada empleado se sienta cómodo haciéndolo.

Hay una serie de ventajas para implementar tan regularmente. Si despliega cada pocas horas, es casi imposible introducir grandes cantidades de errores grandes. Se pueden introducir pequeños problemas, pero luego se pueden solucionar y volver a implementar rápidamente. Normalmente, tendría que hacer una 'revisión' o algo fuera del proceso normal, pero es simplemente parte de nuestro proceso normal: no hay diferencia en el flujo de GitHub entre una revisión y una función muy pequeña.

Otra ventaja de implementar todo el tiempo es la capacidad de abordar rápidamente problemas de todo tipo. Podemos responder a los problemas de seguridad que se nos señalan o implementar solicitudes de funciones pequeñas pero interesantes de forma increíblemente rápida, pero podemos usar el mismo proceso para abordar esos cambios que para el desarrollo de funciones normales o incluso grandes. Es todo el mismo proceso y todo es muy simple.

Como lo hacemos

Entonces, ¿qué es GitHub Flow?

- Cualquier cosa en la `master`rama es desplegable
- Para trabajar en algo nuevo, crear una rama descriptiva llamado fuera de `master`(es decir: `new-oauth2-scopes`)
- Comprométase con esa sucursal localmente y periódicamente envíe su trabajo a la misma sucursal con nombre en el servidor
- Cuando necesite comentarios o ayuda, o si cree que la sucursal está lista para fusionarse, abra una solicitud de extracción
- Después de que otra persona haya revisado y firmado la función, puede fusionarla en principal
- Una vez que se fusiona y se empuja a 'maestro', puede y *debe* implementar de inmediato

Ese es todo el flujo. Es muy simple, muy efectivo y funciona para equipos bastante grandes: GitHub tiene ahora 35 empleados, tal vez 15-20 trabajan en el mismo proyecto (github.com) al mismo tiempo. Creo que la mayoría de los equipos de desarrollo (grupos que trabajan con el mismo código lógico al mismo tiempo y que podrían generar conflictos) tienen un tamaño similar o menor. Especialmente aquellos que son lo suficientemente progresivos como para realizar implementaciones rápidas y consistentes.

Entonces, veamos cada uno de estos pasos sucesivamente.

 # 1 - cualquier elemento en la rama principal es desplegable

Esta es básicamente la única *regla* difícil del sistema. Solo hay una rama que tiene un significado específico y consistente y la hemos nombrado `master`. Para nosotros, esto significa que se ha implementado o, en el peor de los casos, se implementará en cuestión de horas. Es increíblemente raro que esto se rebobine (la rama se mueve a un compromiso anterior para revertir el trabajo): si hay un problema, las confirmaciones se revertirán o se introducirán nuevas confirmaciones que solucionen el problema, pero la rama en sí casi nunca retrotraído.

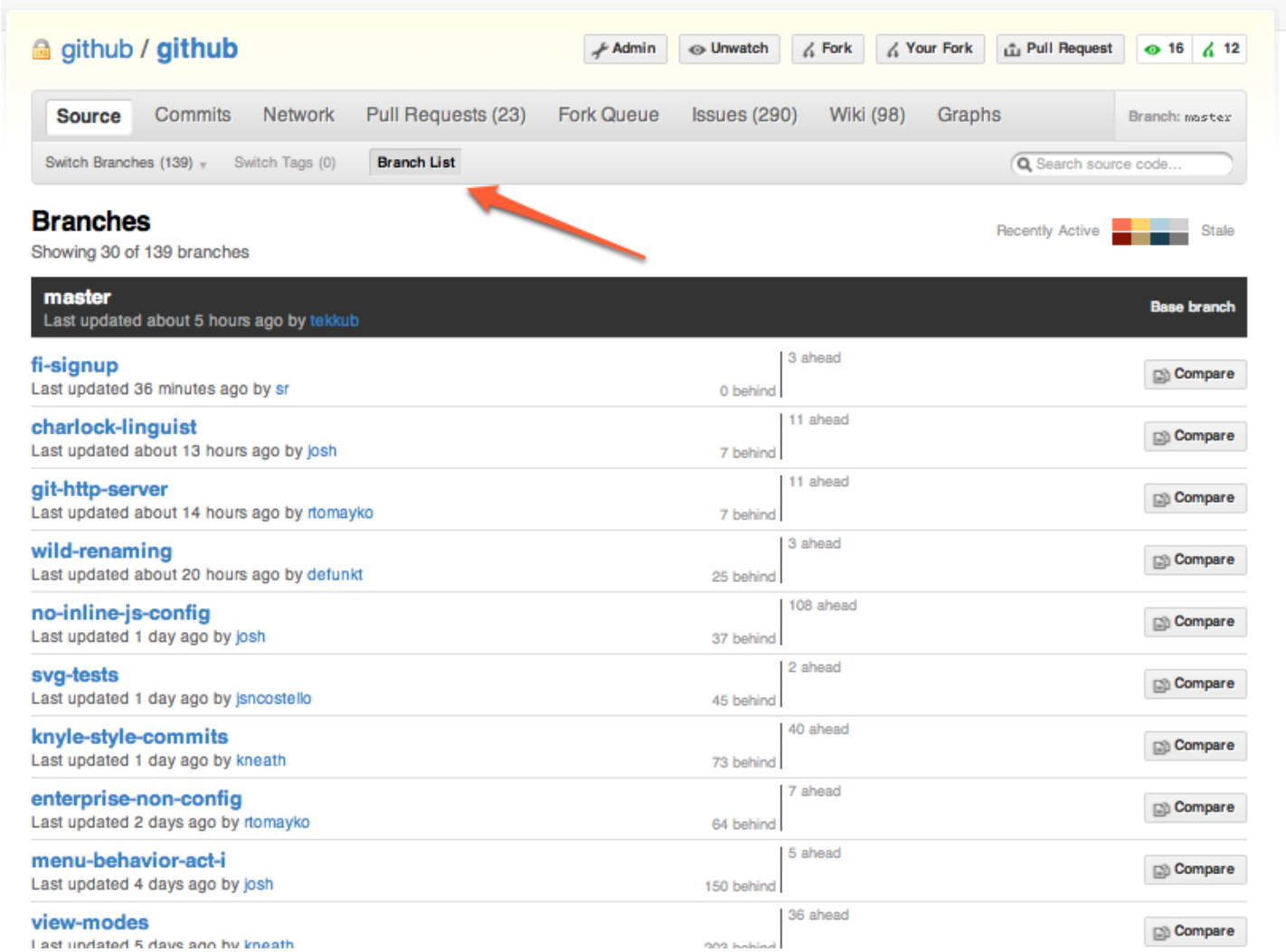
La `master`rama es estable y siempre es seguro desplegar desde allí o crear nuevas ramas a partir de ella. Si empujas algo para dominar que no se prueba o rompe la compilación, rompes el contrato social del equipo de desarrollo y normalmente te sientes bastante mal al respecto. Cada rama que pulsamos tiene pruebas que se ejecutan y se informan en la sala de chat, por lo que si no las ha ejecutado localmente, simplemente puede presionar una rama de tema (incluso una rama con una sola confirmación) en el servidor y esperar a Jenkins para decirte si pasa todo

Podría tener una `deployed`sucursal que se actualice solo cuando implemente, pero no hacemos eso. Simplemente exponemos el SHA implementado actualmente a través de la aplicación web en sí y `curl`si es necesario hacer una comparación.

 # 2 - crea ramas descriptivas fuera del maestro

Cuando desee comenzar a trabajar en cualquier cosa, cree una rama denominada descriptivamente fuera de la `master`rama estable . Algunos ejemplos en la base de código de GitHub en este momento serían `user-content-cache-key`, `submodules-init-task`o `redis2-transition`. Esto tiene varias ventajas: una es que cuando busca, puede ver los temas en los que todos los demás han estado trabajando. Otra es que si abandonas una sucursal por un tiempo y vuelves a ella más tarde, es bastante fácil recordar de qué se trataba.

Esto es bueno porque cuando vamos a la página de lista de sucursales de GitHub, podemos ver fácilmente en qué ramas se han trabajado recientemente y aproximadamente cuánto trabajo tienen en ellas.



Es casi como una lista de características próximas con el estado bruto actual. Esta página es increíble si no la está usando; solo muestra las ramas que tienen un trabajo exclusivo en relación con su rama seleccionada actualmente y las ordena de modo que las que más recientemente trabajaron estén en la parte superior. Si tengo mucha curiosidad, puedo hacer clic en el botón "Comparar" para ver cuál es la lista unificada real de diferencias y confirmaciones que es exclusiva de esa rama.

Por lo tanto, al momento de escribir esto, tenemos 44 sucursales en nuestro repositorio con trabajos no compartidos en ellas, pero también puedo ver que solo alrededor de 9 o 10 de ellas se han visto presionadas en la última semana.

 # 3 - empujar a las ramas nombradas constantemente

Otra gran diferencia con respecto a git-flow es que presionamos constantemente las sucursales con nombre en el servidor. Como lo único de lo que realmente tenemos que preocuparnos es `master`desde el punto de vista de la implementación, presionar al servidor no ensucia a nadie ni confunde cosas, todo lo que no `master`es simplemente algo en lo que se está trabajando.

También se asegura de que nuestro trabajo esté siempre respaldado en caso de pérdida de la computadora portátil o falla del disco duro. Más importante aún, pone a todos en comunicación constante. Una simple 'búsqueda de git' básicamente le dará una lista de TODO de cada uno de los trabajos actualmente.

```
$ git fetch
remote: Counting objects: 3032, done.
remote: Compressing objects: 100% (947/947), done.
remote: Total 2672 (delta 1993), reused 2328 (delta 1689)
Receiving objects: 100% (2672/2672), 16.45 MiB | 1.04 MiB/s, done.
Resolving deltas: 100% (1993/1993), completed with 213 local objects.
From github.com:github/github
 * [new branch]      charlock-linguist -> origin/charlock-linguist
 * [new branch]      enterprise-non-config -> origin/enterprise-non-config
 * [new branch]      fi-signup  -> origin/fi-signup
   2647a42..4d6d2c2  git-http-server -> origin/git-http-server
 * [new branch]      knyle-style-commits -> origin/knyle-style-commits
   157d2b0..d33e00d  master     -> origin/master
 * [new branch]      menu-behavior-act-i -> origin/menu-behavior-act-i
   ea1c5e2..dfd315a  no-inline-js-config -> origin/no-inline-js-config
 * [new branch]      svg-tests  -> origin/svg-tests
```

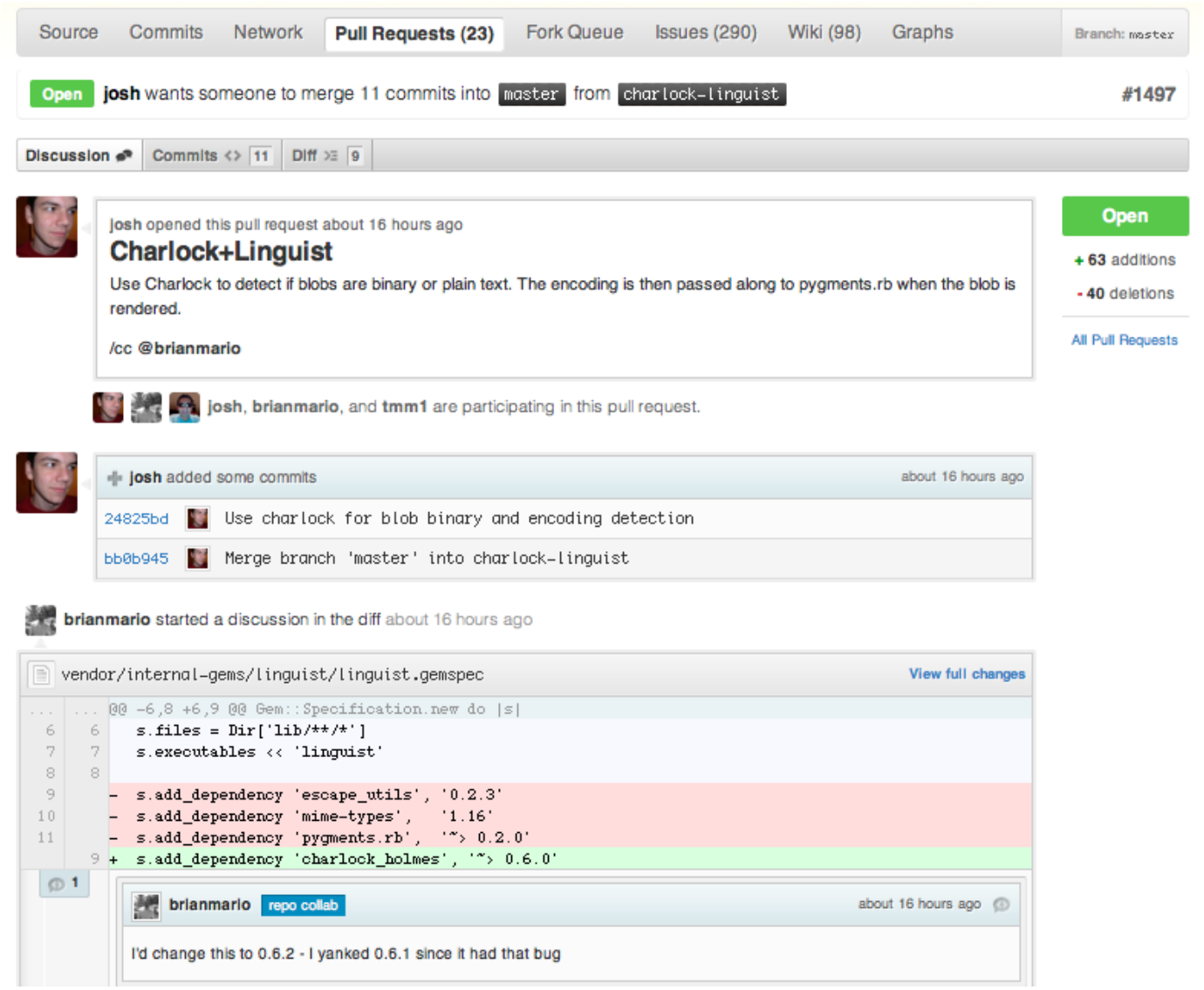
```
   87bb870..9da23f3  view-modes -> origin/view-modes
 * [new branch]      wild-renaming -> origin/wild-renaming
```

También les permite a todos ver, en la página de la Lista de sucursales de GitHub, en qué están trabajando los demás para que puedan inspeccionarlos y ver si desean ayudar con algo.

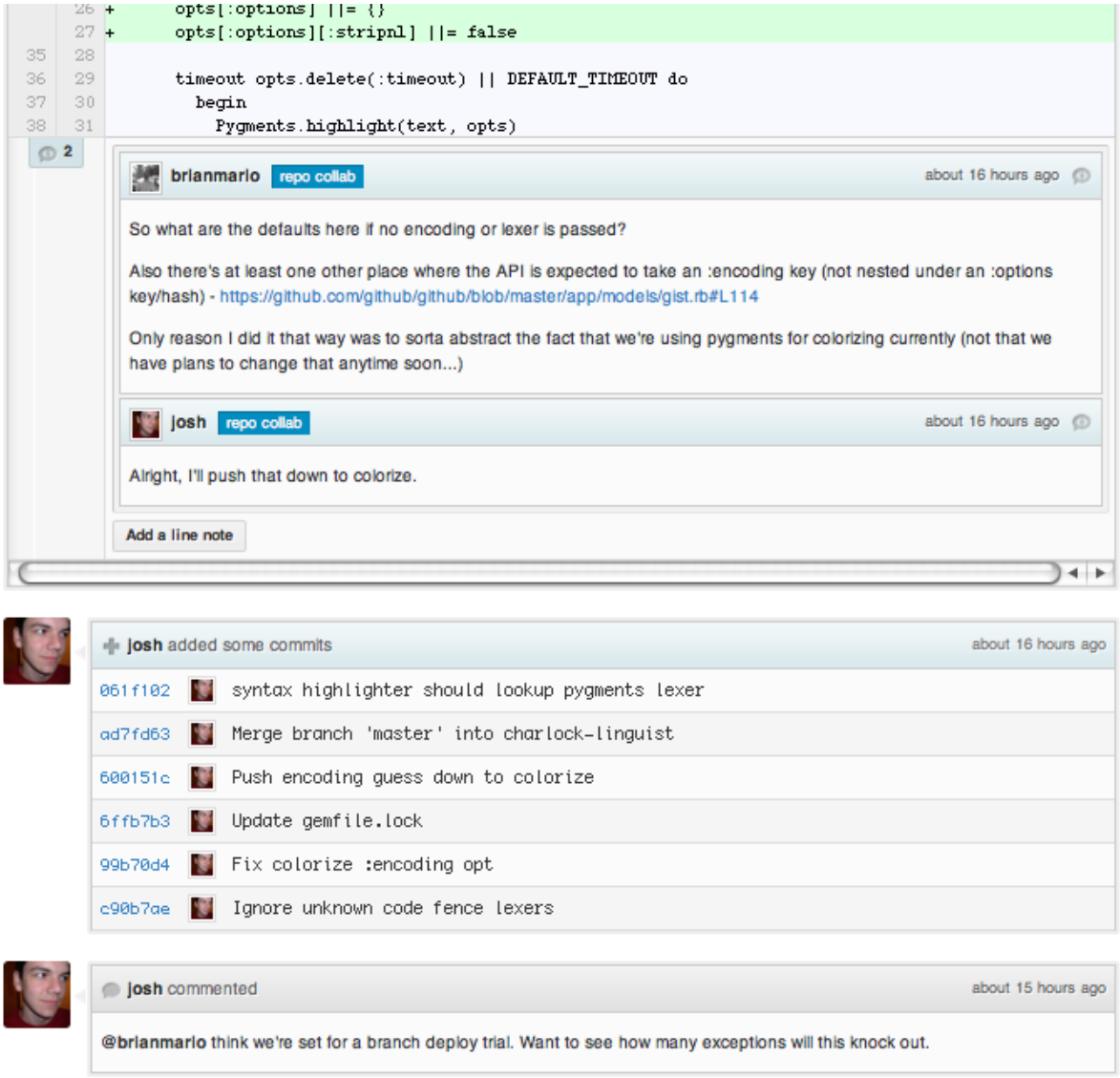
 # 4 - abre una solicitud de extracción en cualquier momento

GitHub tiene un increíble sistema de revisión de código llamado Pull Requests que me temo que no conocen suficientes personas. Muchas personas lo usan para trabajos de código abierto: horca un proyecto, actualiza el proyecto, envía una solicitud de extracción al responsable. Sin embargo, también se puede usar fácilmente como un sistema interno de revisión de código, que es lo que hacemos.

En realidad, lo usamos más como una conversación de sucursal que como una solicitud de extracción. Puede enviar solicitudes de extracción de una rama a otra en un único proyecto (público o privado) en GitHub, de modo que puede usarlas para decir "Necesito ayuda o revisar sobre esto" además de "Por favor combine esto".



Aquí puede ver a Josh cc'ing Brian para que lo revise y Brian viene con algunos consejos sobre una de las líneas de código. Más abajo podemos ver a Josh reconociendo las preocupaciones de Brian y empujando más código para abordarlas.

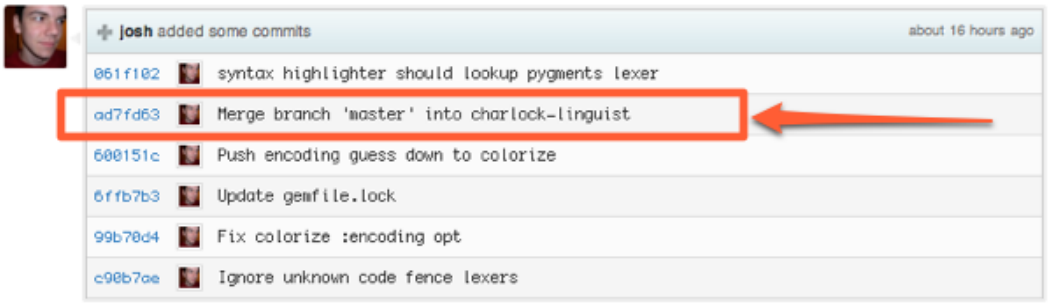


Finalmente, puede ver que todavía estamos en la fase de prueba; esta aún no es una rama lista para implementar, utilizamos las Solicitudes de extracción para revisar el código mucho antes de que realmente desee fusionarlo `master`para la implementación.

Si está atrapado en el progreso de su función o sucursal y necesita ayuda o consejo, o si es un desarrollador y necesita un diseñador para revisar su trabajo (o viceversa), o incluso si tiene poco o ningún código pero una captura de pantalla comps o ideas generales, abre una solicitud de extracción. Puede cc personas en el sistema GitHub agregando un @nombre de usuario, por lo que si desea la revisión o comentarios de personas específicas, simplemente cc en el mensaje PR (como lo vio a Josh arriba).

Esto es genial porque la función de solicitud de extracción le permite comentar líneas individuales en la diferencia unificada, en compromisos únicos o en la solicitud de extracción en sí misma y todo lo arrastra en línea a una única vista de conversación. También le permite continuar presionando a la sucursal, por lo que si alguien comenta que olvidó hacer algo o que hay un error en el código, puede arreglarlo y enviarlo a la sucursal, GitHub mostrará los nuevos commits en la vista de conversación y puedes seguir iterando en una rama así.

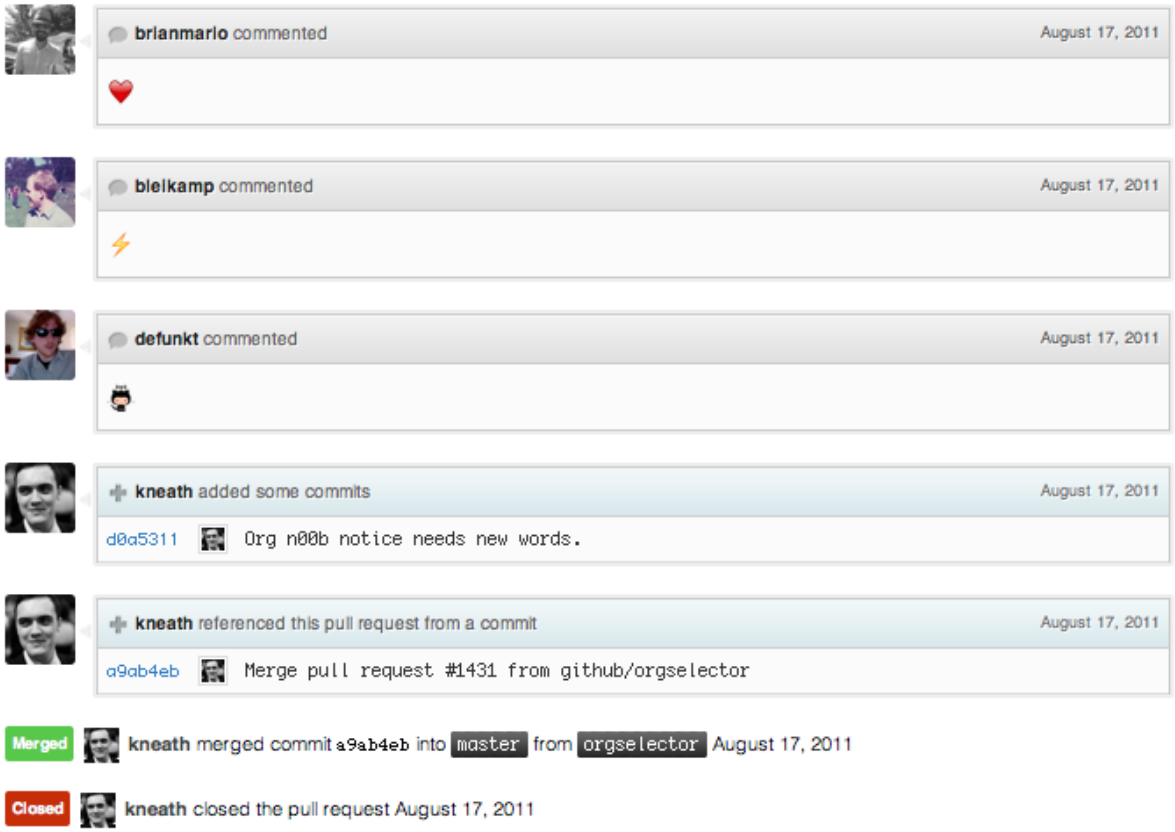
Si la sucursal ha estado abierta demasiado tiempo y sientes que se está desincronizando con la rama principal, puedes fusionar el master en tu rama de tema y continuar. Puede ver fácilmente en la discusión de solicitud de extracción o en la lista de confirmación cuando la rama se actualizó por última vez con el 'maestro'.



Cuando todo se hace realmente y realmente en la sucursal y sientes que está listo para implementarse, puedes pasar al siguiente paso.

 # 5 - fusionar solo después de la revisión de solicitud de extracción

Simplemente no trabajamos directamente `master`o trabajamos en una rama temática y la fusionamos cuando creemos que se ha realizado; tratamos de obtener la aprobación de otra persona de la empresa. En general, se trata de un comentario de +1 o emoji o ": shipit:", pero intentamos que otra persona lo mire.



Una vez que logramos eso, y la sucursal pasa a CI, podemos fusionarlo en maestro para su implementación, lo que automáticamente cerrará la Solicitud de extracción cuando lo presionemos.

 # 6: implementar inmediatamente después de la revisión

Finalmente, su trabajo está hecho y fusionado en la `master`sucursal. Esto significa que incluso si no lo implementa ahora, las personas basarán el nuevo trabajo en él y la próxima implementación, que probablemente ocurrirá en unas pocas horas, lo impulsará. Entonces, como realmente no quieres que alguien más presione algo que escribiste que rompe las cosas, las personas tienden a asegurarse de que realmente se mantenga estable cuando se fusiona y las personas también tienden a impulsar sus propios cambios.

Nuestro campfire bot, hubot, puede hacer implementaciones para cualquiera de los empleados, por lo que es simple:

```
hubot depoy github to production
```

desplegará el código y el tiempo de inactividad cero reiniciará todos los procesos necesarios. Puedes ver qué tan común es esto en GitHub:

Rick	hubot deploy github/oauth_cors to production	Tue, Aug 2
Rick	hubot lock deploy github migration db	Tue, Aug 2
Rick	hubot deploy github/oauth_cors to staging	Tue, Aug 2
tmm1	hubot deploy github to the cloud	Tue, Aug 2
tmm1	hubot deploy github to the cloud	Tue, Aug 2
atmos	hubot deploy logs for github	Tue, Aug 2
atmos	hubot deploy github to production	Tue, Aug 2
Rick	hubot deploy github to production/fs	Tue, Aug 2
Rick	hubot deploy github to production/arch1	Tue, Aug 2
Rick	hubot deploy github to production/fs	Tue, Aug 2
Rick	hubot deploy github to production/fs	Tue, Aug 2
sr	hubot deploy github to production/fe	Tue, Aug 2
sr	hubot deploy github to production	Tue, Aug 2
sr	hubot deploy github to production	Tue, Aug 2
sr	hubot deploy github/ghost-town to production	Tue, Aug 2
ekkub	hubot deploy github to production	Tue, Aug 2
sr	hubot deploy github/ghost-town to staging	Tue, Aug 2
sr	hubot deploy github to production	Tue, Aug 2
atmos	hubot deploy github to production	Tue, Aug 2
sr	hubot deploy github to production	Tue, Aug 2
atmos	hubot deploy github to production	Tue, Aug 2
atmos	hubot deploy github to production	Tue, Aug 2
atmos	hubot deploy github to production	Tue, Aug 2
meron	hubot deploy github to production	Tue, Aug 2
atmos	hubot deploy github to staging	Tue, Aug 2
atmos	hubot deploy github to production/smtp1	Tue, Aug 2
atmos	hubot deploy github to production/fe,fs,aux1	Tue, Aug 2
atmos	hubot deploy github to production/fe,fs,aux1	Tue, Aug 2
atmos	hubot deploy github to production	Tue, Aug 2

Puede ver a 6 personas diferentes (incluido un técnico de soporte y un diseñador) implementar aproximadamente 24 veces en un día.

He hecho esto para las sucursales con una confirmación que contiene un cambio de una línea. El proceso es simple, directo, escalable y poderoso. Puede hacerlo con ramas de características con 50 commits en ellas que tomaron 2 semanas, o 1 commit que tomó 10 minutos. Es un proceso tan simple y sin fricciones que no te molesta que tengas que hacerlo incluso para 1 commit, lo que significa que las personas rara vez intentan saltear o eludir el proceso a menos que el cambio sea tan pequeño o insignificante que simplemente no importa .

Este es un proceso increíblemente simple y poderoso. Creo que la mayoría de la gente estaría de acuerdo en que GitHub tiene una plataforma muy estable, que los problemas se abordan rápidamente si alguna vez surgen y que las nuevas características se introducen a un ritmo rápido. No hay compromiso de calidad o estabilidad para que podamos obtener más velocidad o simplicidad o menos proceso.

Conclusión

Git en sí mismo es bastante complejo de comprender, por lo que el flujo de trabajo que utilizas con él es más complejo de lo necesario, simplemente agrega más gastos generales mentales para todos los días. Siempre recomendaría utilizar el sistema más simple posible que funcionará para su equipo y hacerlo hasta que no funcione más y luego agregar complejidad solo cuando sea absolutamente necesario.

Para los equipos que tienen que hacer lanzamientos formales en un intervalo de más largo plazo (de algunas semanas a algunos meses entre lanzamientos), y ser capaces de hacer correcciones urgentes y ramas de mantenimiento y otras cosas que surgen del envío con poca frecuencia, git-flow hace sentido y yo recomendaría su uso.

Para los equipos que han establecido una cultura de envío, que presionan a la producción todos los días, que están constantemente probando e implementando, recomendaría elegir algo más simple como GitHub Flow.

Discusión, enlaces y tweets

Soy un desarrollador en GitHub. Sígueme en Twitter ; disfrutarás mis tweets Me encargo de crear cuidadosamente cada uno. O al menos pretenden hacerte reír. O ofendido Una de esas dos ... aún no he decidido cuál.

Tweet Follow @chacon