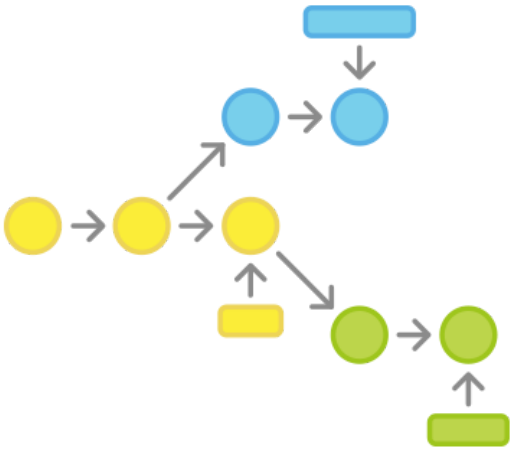
Recommend

Git Tutorials

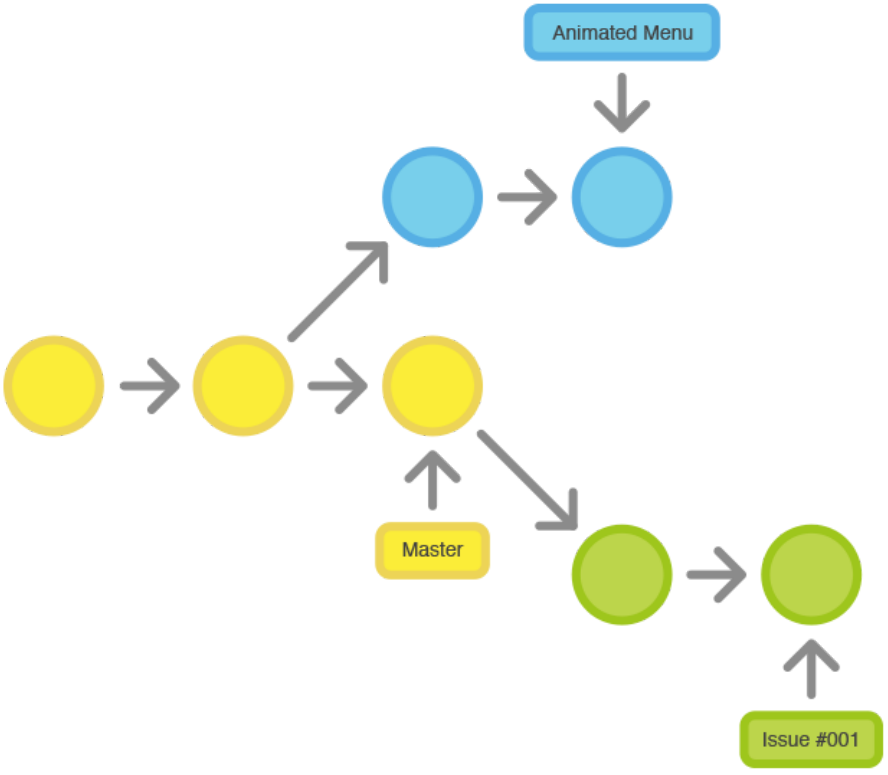
workflows Git.

La variedad de workflows posibles puede hacer que sea difícil saber por dónde empezar al usar Git en el espacio de trabajo. Esta página proporciona un punto de inicio con una visión global de los workflows más comunes para equipos de empresas.

Según vayas leyendo, recuerda que esos workflows están diseñados para ser directrices más que normas concretas. Queremos mostrarte lo que es posible, así que puedes mezclar y hacer coincidir aspectos de diferentes workflows para cumplir tus necesidades individuales.



Feature Branch Workflow



Una vez que entiendas el workflow centralizado, añadir ramas puntuales a tu proceso de desarrollo es una forma sencilla de alentar la colaboración y racionalizar la comunicación entre desarrolladores.

La idea central del Feature Branch Workflow es que todo el desarrollo de funcionalidades debe darse en una rama especializada en vez de en la rama `master`. Esta encapsulación hace que sea fácil para varios desarrolladores trabajar en una funcionalidad específica sin alterar el código base. Esto también significa que la rama `master` nunca contendrá código roto, lo que es una enorme ventaja para la integración continua de entornos.

Encapsular desarrollo de funcionalidades también hace posible hacer uso de solicitudes de recuperación, que son formas de iniciar discusiones sobre una rama. Así otros desarrolladores la oportunidad de terminar una funcionalidad antes de que se integre en el proyecto oficial. O, si se

oportunidad de terminar una funcionalidad antes de que se integre en el proyecto oficial. O, si se quedan bloqueados en medio de una funcionalidad, se puede abrir una pull request para pedir recomendaciones de tus colegas. Las solicitudes de recuperación (pull request) hacen que sea increíblemente fácil para tu equipo comentar el trabajo de los demás.

Cómo funciona

El Feature Branch Workflow también usa un repositorio central, y `master` sigue siendo la historia oficial del proyecto. Pero, en vez de confirmar directamente en su rama local `master`, los desarrolladores crean una nueva rama cada vez que empiezan a trabajar en una nueva funcionalidad. Las ramas puntuales deben tener nombres descriptivos, como `animated-menu-items` o `issue-#1061`. La idea es dar un propósito claro y especializado a cada rama.

Git no hace distinciones técnicas entre la rama `master` y las ramas puntuales, así que los desarrolladores pueden editar, preparar y confirmar cambios a la rama de la funcionalidad como hicieron en el workflow centralizado.

Además, las ramas de funcionalidades pueden (y deben) enviarse al repositorio central. Esto hace que sea posible compartir una funcionalidad con otros desarrolladores sin tocar nada de código oficial. Ya que `master` es la única rama "especial" almacenar varias ramas puntuales en el repositorio central no supone ningún problema. Por supuesto, esta tan bien es una forma útil para hacer una copia de seguridad de los commits locales de todos los desarrolladores.

Solicitudes de recuperación (pull request)

Aparte de aislar el desarrollo de funcionalidades, las ramas hacen posible discutir cambios a través de pull requests. Una vez que alguien acaba una funcionalidad, no la fusiona inmediatamente con `master`. En vez de eso, envía la rama puntual al servidor central y presenta una pull request para pedir que se fusionen sus añadidos con `master`. Esto da a otros desarrolladores la oportunidad de revisar los cambios antes de que sean parte del código base principal.

Revisar código es la ventaja más importante de las solicitudes de recuperación (pull request), pero en realidad están diseñadas como un modo genérico de hablar sobre código. Puedes pensar en las solicitudes de recuperación (pull request) como un discusión sobre una rama en particular. Esto significa que también pueden usarse mucho antes en el proceso de desarrollo. Por ejemplo, si un desarrollador necesita ayuda con una funcionalidad en particular, lo que tiene que hacer es rellenar una pull request. También se notificará de inmediato a las partes interesadas, y podrán ver la pregunta justo al lado de los commits pertinentes.

Una vez que se acepta una pull request, el acto real de publicar una funcionalidad viene a ser el mismo que en el workflow centralizado. Primero, hay que asegurarse de que la rama `master` está sincronizada con la rama `master` del repositorio central. Después, fusionar (merge) la rama puntual con `master` y enviar (push) la rama `master` actualizada el repositorio central.

Las solicitudes de recuperación (pull request) pueden conseguirse en soluciones de gestión de productos de repositorios como Bitbucket o Stash. Mira la documentación de pull requestde Stash para ver un ejemplo.

Ejemplo

El ejemplo a continuación muestra una solicitud de recuperación como forma de revisión de código, pero recuerda que pueden servir para muchos otros propósitos.

Mary empieza una nueva funcionalidad

Antes de que empiece a desarrollar una funcionalidad, Mary necesita una rama aislada para trabajar. Puede solicitar una nueva rama con el siguiente comando:

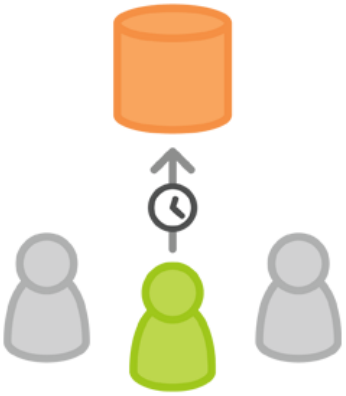
```
git checkout -b marys-feature master
```

```
git checkout -b marys-feature master
```

Esto comprueba una rama con el nombre `marys-feature` basada en `master`, y `-b` En esta rama, Mary edita, prepara y confirma cambios de la forma habitual, y crea la funcionalidad con tantos commits como sea necesario:

```
git status
git add <some-file>
git commit
```

Mary va a comer



Mary añade unos cuantos commits a su funcionalidad durante la mañana. Antes de salir a comer, es buena idea enviar su rama de funcionalidad al repositorio central.. Esto sirve como copia de seguridad, pero si Mary colaboraba con otros desarrolladores, esto también les daría acceso a sus commits iniciales.

```
git push -u origin marys-feature
```

Este comando envía `marys-feature` al repositorio central (`origin`), y la opción `-u` lo añade a la rama remota de seguimiento. Después de configurar la rama de seguimiento, Mary puede usar `git push` sin ningún parámetro para enviar su funcionalidad.

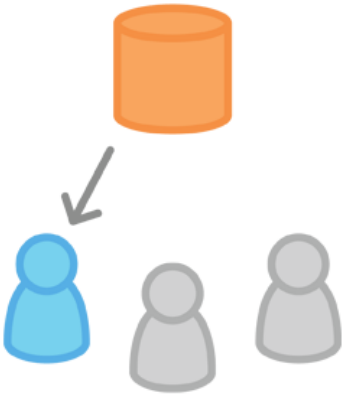
Mary acaba su funcionalidad

Cuando vuelve de comer, acaba la funcionalidad. Antes de fusionarla con `master`, tiene que rellenar una pull request para decir al resto del equipo que ha acabado. Pero primero tiene que asegurarse de que en el repositorio central están sus commits más recientes:

```
git push
```

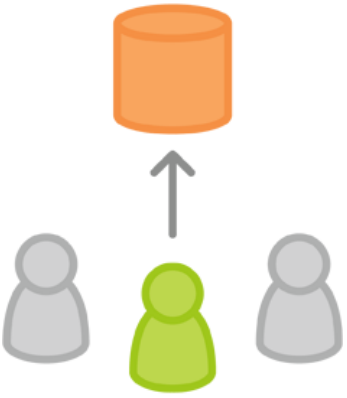
Entonces ella rellena las solicitudes de recuperación (pull request) en su Git GUI para pedir fusionar (merge) `marys-feature` con `master`, y se notifica inmediatamente a los miembros del equipo. Lo bueno de las solicitudes de recuperación (pull request) es que muestran los comentarios al lado de los commits pertinentes, así que es fácil preguntar sobre grupos de cambio.

Bill recibe la pull request



A Bill le llegan las solicitudes de recuperación (pull request) y mira `marys-feature`. Decide que quiere hacer varios cambios antes de integrarla en el proyecto oficial, y él y Mary hablan a través de la pull request.

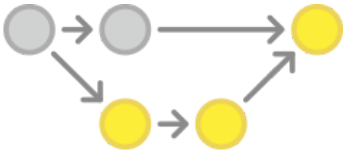
Mary hace los cambios



Para hacer los cambios, Mary usa exactamente el mismo proceso que usó para crear la primera iteración de su funcionalidad. Edita, prepara, confirma y envía cambios al repositorio central. Se muestra toda su actividad en la pull request, y Bill puede hacer comentarios.

Si quisiera, Bill podría recuperar `marys-feature` en su repositorio y trabajar en él por su cuenta. Si él añade cualquier commit, también se mostraría en el pull request.

Mary publica su funcionalidad



Una vez que Bill está preparado para aceptar la pull request, alguien ha de fusionar (merge) la funcionalidad con el proyecto estable (esto lo pueden hacer Bill o Mary):

```
git checkout master
git pull
git pull origin marys-feature
git push
```

Primero, el que haga la fusión tiene que comprobar su rama `master` y asegurarse de que está actualizada. Entonces, `git pull origin marys-feature` fusiona el repositorio central y la copia de `marys-feature`. También se puede usar un simple `git merge marys-feature`, pero el comando que se muestra arriba se asegura de que siempre recibes (pull) la versión más actualizada de la rama de la funcionalidad. Por último, la rama actualizada `master` ha de enviarse de vuelta a `origin`.

Este proceso normalmente resulta en una fusión confirmada. A algunos desarrolladores les gusta porque es como una unión simbólica de la funcionalidad con el resto del código base. Pero si prefieres una historia lineal, es posible reorganizar (rebase) la funcionalidad a la punta de la rama `master` antes de ejecutar la fusión, lo que daría una fusión con avance rápido.

Algunos GUI automatizan el proceso de aceptación de la pull request al ejecutar todos estos comandos solo con pulsar en el botón “Aceptar”. Si el tuyo no lo hace, al menos será capa de cerrar automáticamente las pull request cuando la rama de la funcionalidad se fusione con la rama `master`

Mientras tanto, John hace exactamente lo mismo

Mientras Mary y Bill trabajan en `marys-feature` y hablando sobre ello en su pull request, John hace exactamente lo mismo con su propia rama puntual. Al aislar funcionalidades en ramas separadas, todos pueden trabajar de forma independiente, pero sigue siendo fácil compartir cambios con otros desarrolladores cuando sea necesario.

A dónde ir desde aquí

Hasta ahora, podrás ver que las ramas puntuales son una forma de multiplicar casi literalmente las funcionalidades de una sola rama `master` que se use en el workflow centralizado. Además, las ramas puntuales también facilitan las solicitudes de recuperación (pull request), lo que hace posible discutir commits específicos dentro de la versión de control GUI.

El workflow de ramas puntuales es una forma increíblemente flexible de desarrollar un proyecto. El problema es que a veces es demasiado flexible. Para equipos más grandes, suele ser beneficioso asignar más roles específicos a diferentes ramas. El workflow Gitflow es un patrón común para gestionar el desarrollo de funcionalidades, lanzar preparación y mantenimiento.

PREVIOUS
Centralized Workflow

NEXT
Gitflow Workflow

Recommend this if you found it useful!

Regístrate para más artículos y recursos & de Git:

Your email address

Sign up

Nuestros últimos post del blog de Git

What’s new in Git 2.1

Following the git 2.0.0 release two-and-a-half months ago we’re being treated to a new minor version of git, 2.1.0, with a host of exciting new features! The

full release notes are available he ...

Leer en el blog de Git