denny sam · Seguir
31 de enero · 7 minutos de lectura

El mejor flujo de trabajo de Git para su equipo de ingeniería de software

Trabajar con git no debería ser difícil, ¿verdad?

Foto de Roman Synkevych en Unsplash

Los desarrolladores tienen una relación de amor y odio con Git. Git es probablemente una de esas cosas que son omnipresentes en la vida de un desarrollador. Sin embargo, es algo que realmente no nos importa cuando estamos comenzando como uno.

Un par de otras cosas que damos por hecho, aunque son esenciales son:

Cubriré estos dos temas en los próximos artículos. Hay muchos conceptos erróneos con respecto a Git. Quiero escribir sobre ellos en otro número. Hazme saber si quieres saber sobre esto comentando.

Hoy analicemos el flujo de trabajo de Git que más prefiero.

TLDR

1. El flujo de trabajo que sigo es similar al flujo de trabajo de Gitflow .
2. Cree `master` rama para producción, `dev` rama para preparación y `feature` ramas para funciones individuales.
3. Cuando tenga que integrar los cambios de otro compañero de equipo en su rama, puede usar el comando git-rebase . Pero esto es un poco peligroso, así que úsalo con precaución.
4. `release` Las ramas se utilizan para fusionar un montón de cambios en la `master` rama.
5. `hotfix` Las ramas se utilizan para solucionar problemas en el servidor de producción.

Entonces, ¿qué es un flujo de trabajo de Git?

El flujo de trabajo de Git es la estrategia que sigues cuando trabajas con Git.

El flujo de trabajo de Git de un desarrollador solitario puede ser tan simple como trabajar en una sola rama, realizar compromisos con ella y usarla como la única fuente de verdad.

Pero cuando se trata de un equipo, necesitamos una estrategia de ramificación adecuada. De lo contrario, los compromisos de su colega pueden ser sobrescritos por los suyos (ese es uno entre un millón de peligros que pueden ocurrir).

Esta publicación es para aquellos que quieren entender la estrategia.

Ahora, cómo usar Git en un equipo

`git-switch and git-restore` :

Todo el mundo está familiarizado con `git-checkout` . Actualmente, `git-checkout` maneja 2 cosas

1. Restaure el contenido del archivo desde el índice (u otras fuentes) al árbol de trabajo
2. Cambiar de sucursal

Pero hace un par de años, los ingenieros de git se dieron cuenta de que un comando que maneja dos funciones muy diferentes puede ser confuso, por lo que lo dividieron en dos comandos separados.

un. `git-switch` : Manejar el cambio de ramas

```
git switch <nombre-de-sucursal>
```

B. `git-restore` : Manejar la restauración de archivos de árbol de trabajo

```
git restore <ruta-archivo>
```

Esta actualización fue parte de la versión git 2.23. Las notas de la versión para el mismo se pueden encontrar aquí :

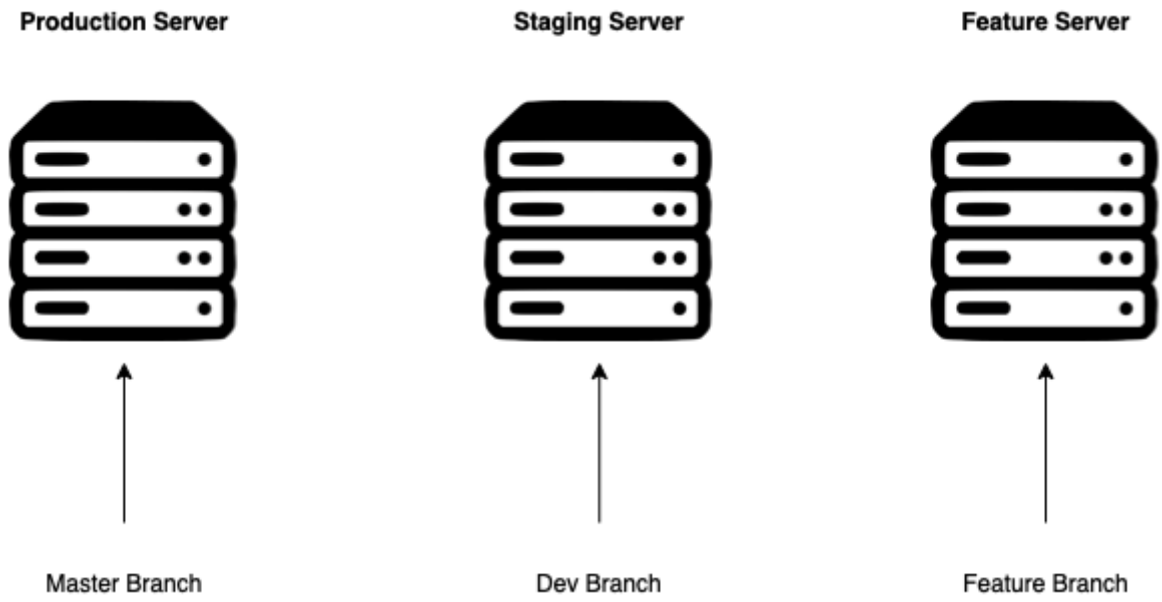
`git-rebase` :

Esto se considera un comando peligroso. Pero es muy útil en situaciones. Este comando se usa para aplicar confirmaciones de una rama a otra rama.

> ⚠ **Una palabra de precaución** : NO use este comando para modificar el historial en ninguna sucursal pública.

Demostraré cómo uso esto en mi propio flujo de trabajo.

Mi flujo de trabajo Git



Su flujo de trabajo depende en gran medida de cuántos tipos de servidores tenga su empresa.

Muchas empresas siguen la siguiente configuración:

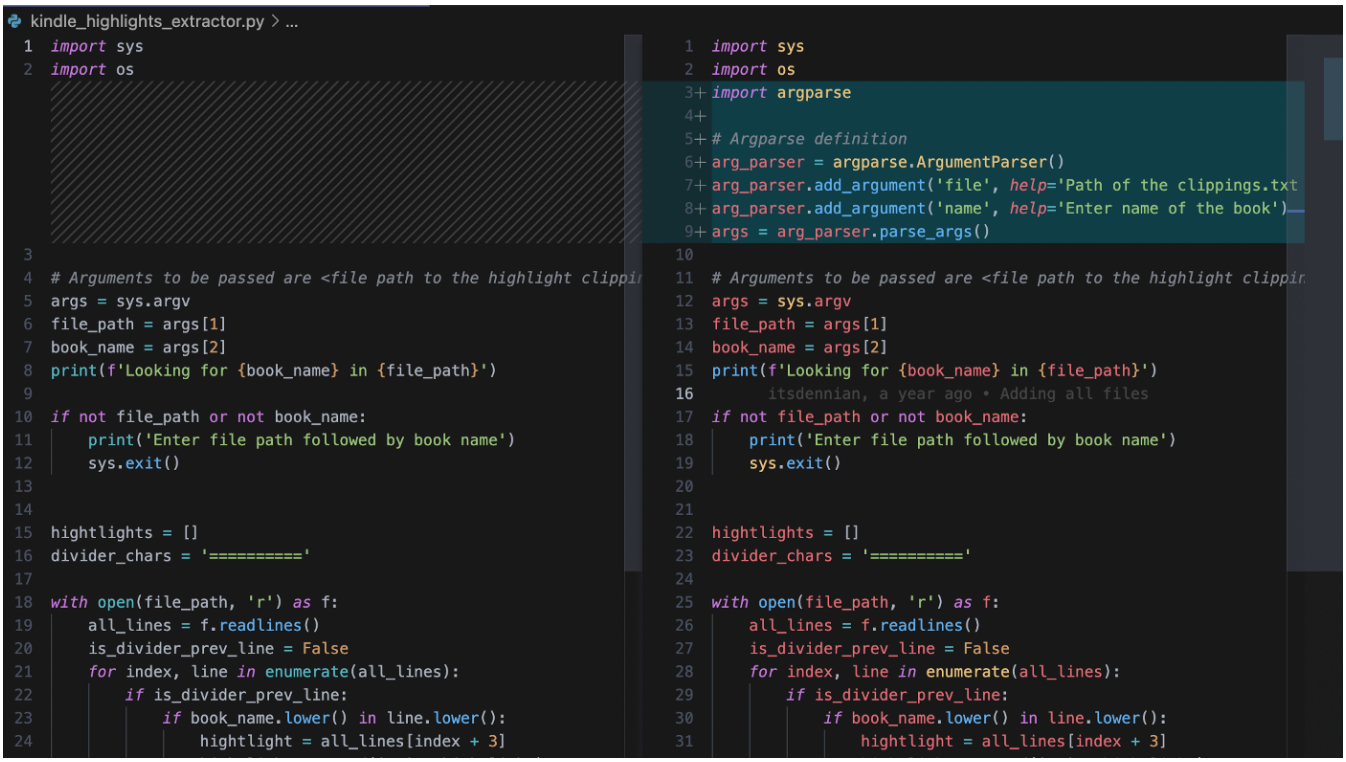
1. **Servidor de producción** , el público, servidor de cara al cliente. La implementación en este servidor se realiza después de realizar pruebas rigurosas en el servidor de ensayo.
2. **Servidor** de prueba, donde los miembros del equipo realizan pruebas antes de lanzarlo al público.
3. **Servidores de prueba** de funciones (generalmente creados para cada función), donde los desarrolladores individuales prueban las nuevas funciones antes de enviarlas al servidor de ensayo.

Eso significa:

1. rama maestra → Para servidor de producción
2. rama dev/staging → Para servidor de staging
3. rama de características → Para el servidor de prueba de características

El flujo de trabajo que seguimos en nuestra empresa y que voy a mostrar aquí está estrechamente relacionado con el popular flujo de trabajo de Gitflow .

> Cuando se trata de comparar cambios de git, no soy un gran admirador de la terminal.
>
> Preferiría ver los cambios en mi propio editor de VS Code, donde puedo comparar el antes y el después de manera efectiva.
>
> Echa un vistazo a la siguiente captura de pantalla de mi editor de VS Code que muestra los cambios.



¡La forma gráfica de ver los cambios es más intuitiva!

Mi flujo de trabajo de git consta de varias ramas y ciertas reglas adjuntas a estas ramas.

Los cambios realizados en esta rama se implementan en el servidor de producción.

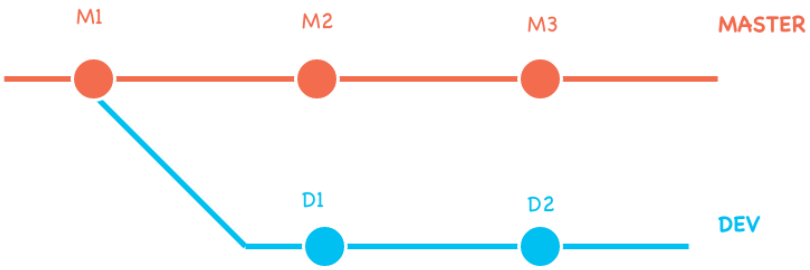
Se supone que no debe realizar ningún cambio en esta rama directamente. También está prohibido crear ramas a partir de esto para crear características hasta que sea un error de producción (en cuyo caso creará ramas de revisión, que he explicado en detalle a continuación).

master branch with 3 commits

👍 Rama de desarrollo

Una vez que haya configurado la `master` rama, debe derivar una `dev` rama de ella.

Esta rama se usa para probar todas las funciones antes de implementarlas en producción. Los cambios en esta rama se implementan en el servidor de ensayo y se prueban.



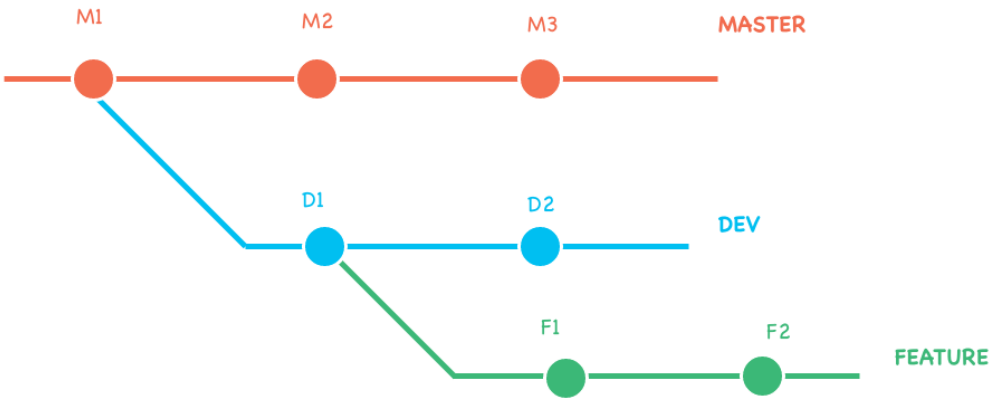
dev branch merged out of master at the M1 commit (first commit on master) & having it's own commits

🤟 Ramas destacadas

Cada característica tendrá una rama separada.

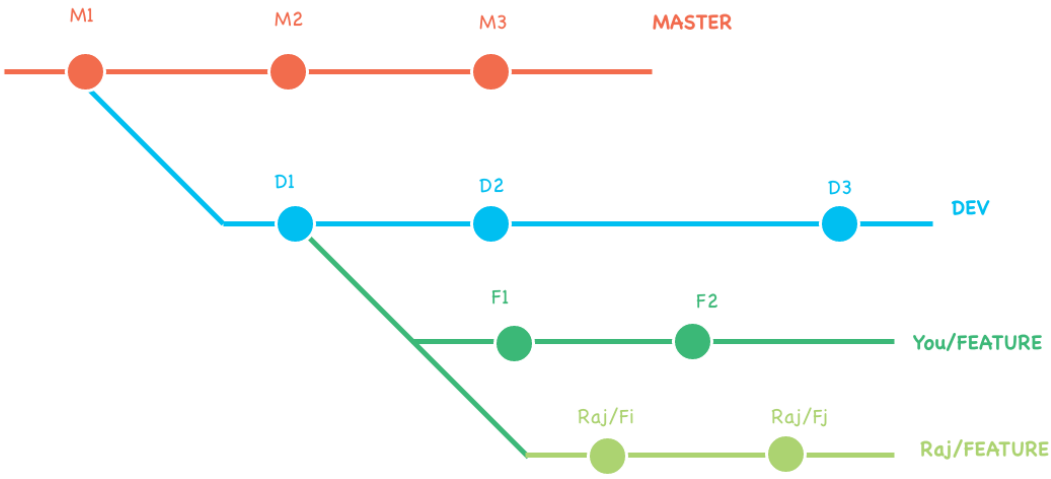
Las ramas de función se ramifican fuera de la `dev` rama y se vuelven a fusionar con la `dev` rama una vez que se completa la función.

Tomemos una característica y construyamos una rama para ella.



Aquí, ramificaste una `feature` rama y creaste 2 confirmaciones en ella, `F1` & `F2`

Ahora suponga que su colega Raj, que también está trabajando en la misma función, ha creado 2 confirmaciones en su rama de función local.



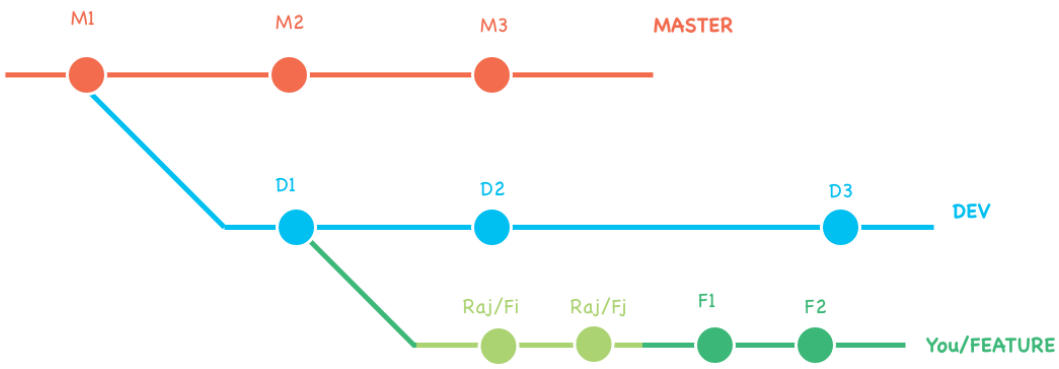
Hmm, esto se está poniendo complicado. ¿Cómo integra sus cambios en su rama antes de enviarlos a la rama de desarrollo?

Aquí es donde usamos el `git-rebase` comando mencionado anteriormente. En cierto modo, está modificando la base de sus cambios en su rama para que la rama de características esté en una estructura lineal de confirmaciones (observe que tanto su

Por su parte, va a ingresar el comando

```
git rebase <rama de Raj>
```

Veamos cómo se ven nuestras sucursales ahora.

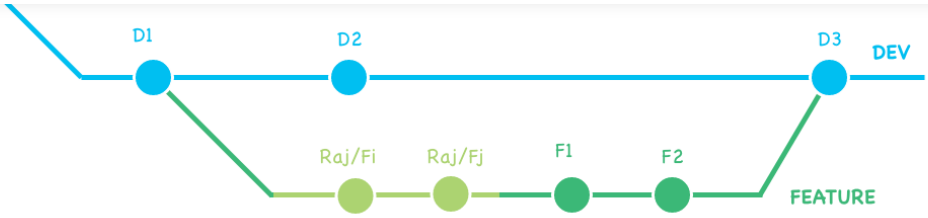


Tus confirmaciones han sido 'añadidas' a sus confirmaciones. Ahora tiene sus cambios y los cambios de Raj.

> Tenga en cuenta que está bien reorganizar de esta manera, ya que no está editando ninguna de las ramas principales. En cuanto a Raj, puede tomar un tirón de la `feature` rama una vez que la empujas al control remoto y también puede ver tus confirmaciones.

La razón por la que usamos `rebase` el comando en lugar del comando de git `merge` es para tener una rama de git simple y lineal, que no se puede lograr con la combinación.

Es hora de fusionar `feature` back to `dev` , creando una `D3` confirmación con cambios en la `feature` rama.



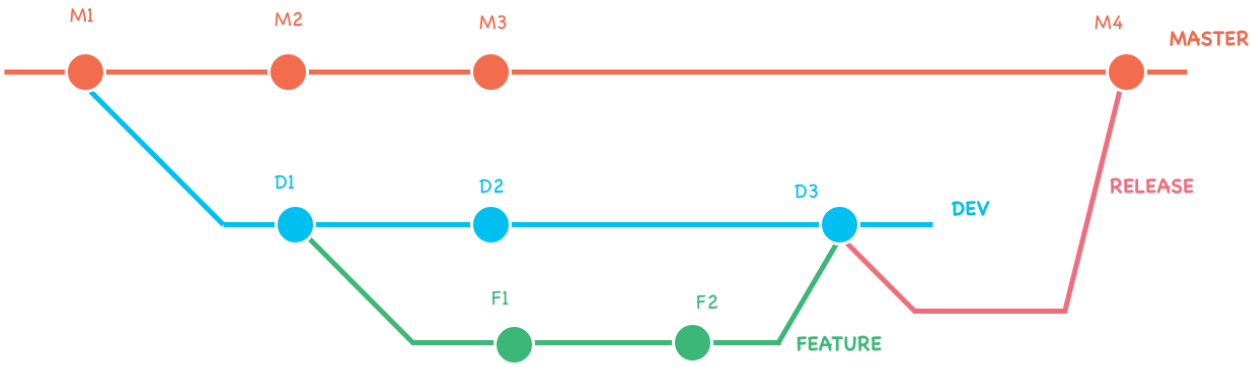
Merging feature into dev creating a D3 commit with changes of feature

🤘 Suelta sucursales

Una vez que se implementan y prueban suficientes funciones en el servidor de ensayo, puede implementarlas en el servidor de producción fusionando primero los cambios de `dev` a la `master` rama.

Para fusionarse con `master` , crea una `release` rama, fuera de la `dev` rama. Esta rama se fusionará con `master` .

Después de esto, puede fusionar la `release` rama en el archivo `dev` .



release branch merged into master creating M4 commit

🔥 Ramas de revisión

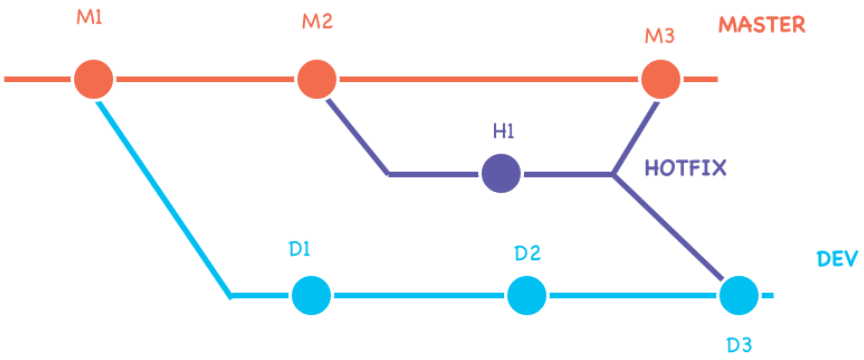
Cuando algo falla en su servidor de producción, no hay suficiente tiempo para ramificarse `dev` , fusionarlo nuevamente, crear una rama de lanzamiento, etc.

La forma más sencilla será hacer lo siguiente:

1. Rama una `hotfix` rama de `master`

4. Combinar lo mismo para `dev` ramificar

Algo como esto



Algo de etiqueta de Git para mantener

Cuando trabaja en un entorno colaborativo, debe seguir algunos protocolos al usar Git.

1. **Cada fusión con sucursales públicas debe ser revisada.** Cada vez que intente fusionarse con una `dev` rama o una rama maestra, cree una solicitud de incorporación de cambios y dígales a sus compañeros de equipo que desea fusionar algunos de sus cambios. Esto les ayudará a aprovechar y trabajar con el código más reciente.

2. **No te metas con la historia de las sucursales.** Las confirmaciones de Git son inmutables. Pero hay ciertos comandos en git que pueden crear confirmaciones utilizando las confirmaciones anteriores, haciendo que parezca que has vuelto a una confirmación anterior. Esto puede causar problemas con el código de trabajo de otros compañeros de equipo. Haga esto solo si es absolutamente necesario. Además, siga el consejo n. ° 1 después de hacer esto.

¿Quieres conectarte?