Entrega continua de soluciones InterSystems utilizando GitLab - Parte I: Git

Artículo
Ricardo Paiva · Sep 30, 2019 Lectura de 7 min

Entrega continua de soluciones InterSystems utilizando GitLab - Parte I: Git

Todo el mundo tiene un entorno para realizar pruebas.

Algunas personas tienen la suerte de tener un entorno totalmente separado para Producción.

-- Anónimo

En esta serie de artículos, me gustaría presentar y discutir varios métodos posibles para el desarrollo de software, con las tecnologías de InterSystems y GitLab. Trataré temas como:

- Git 101
- Git flow (development process)
- Instalación de GitLab
- Flujo de trabajo de GitLab
- GitLab CI/CD (Integración Continua/Entrega Continua)
- CI/CD (Integración Continua/Entrega Continua) con contenedores

En esta primera parte se abordará la piedra angular del desarrollo de software moderno - el sistema de control de las versiones de Git y varios flujos de Git.

Git 101

Aunque el tema principal que discutiremos es sobre el desarrollo de software, en general, y cómo GitLab puede ayudarnos en ese esfuerzo, Git utiliza conceptos de alto nivel en lugar de varios conceptos subyacentes en su diseño, los cuales son importantes para mejorar la compresión de los conceptos que veremos posteriormente.

Dicho esto, Git es un sistema de control de versiones, basado en estas ideas (existen muchas más, aunque estas son las más importantes):

- **Desarrollo no lineal**: significa que mientras nuestro software es resultado del lanzamiento de la versión 1 a la 2 y después a la 3, sin que nos demos cuenta, el paso de la versión 1 a la 2 se realiza en paralelo, y varios desarrolladores se ocupan de un número de características/correcciones de errores simultáneamente
- **Desarrollo distribuído** significa que el desarrollador es independiente de un servidor central o de otros desarrolladores, y puede establecerse en su propio entorno fácilmente
- **Fusión** - las dos ideas anteriores nos llevan a una situación donde existen muchas versiones diferentes de la verdad, simultáneamente, y necesitamos unirlas de nuevo en un estado completo

No estoy diciendo que Git inventó estos conceptos. No. Más bien Git hizo que fueran fáciles y populares, y junto con varias innovaciones relacionadas, por ejemplo, la infraestructura definida por código y/o la colocación en

contenedores, cambiaron la manera en que se desarrolla el software.

Principales términos de Git

Repositorio es un proyecto que almacena datos y meta-información sobre los datos.

- "físicamente" es un directorio que se encuentra en un disco
- almacena archivos y directorios
- también almacena un historial completo de los cambios que hubo en cada archivo

El repositorio se puede almacenar:

- Localmente, en nuestro propio equipo
- De forma remota, en un servidor remoto

Pero no existe ninguna diferencia particular entre los repositorios locales y remotos, desde el punto de vista de git.

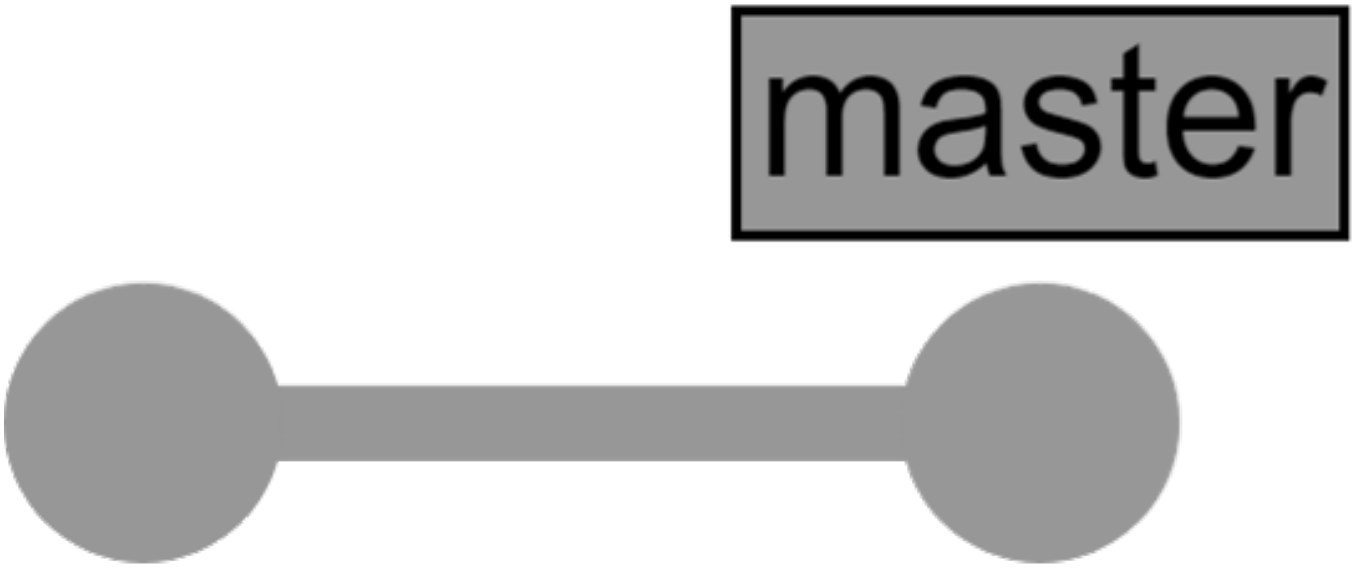
Commit es un estado fijo del repositorio. Obviamente, si cada commit almacenara el estado completo del repositorio, nuestro repositorio crecería mucho y muy rápidamente. Por lo que un commit almacena un **diff** , que es una diferencia entre el commit actual y su **commit padre**.

Diferentes commits pueden tener un número distinto de padres:

- 0 - el primer commit en el repositorio no tiene padres
- 1 - lo más habitual - nuestro commit cambió algo en el repositorio a como estaba durante el commit padre
- 2 - cuando tenemos dos estados diferentes en el repositorio podemos unirlos en un estado nuevo. Y tanto ese estado, como ese commit tendrían dos padres.
- >2 - puede suceder cuando unimos más de 2 estados diferentes del repositorio en uno nuevo. Esto no debería ser particularmente importante para nuestra discusión, pero existe

Ahora, en el caso de un padre, cada commit que sea diferente de él se llama un **commit hijo**. Cada commit padre puede tener un número ilimitado de commits hijos**.**

La **rama** es una referencia (o puntero) para un commit. Así es como se ve:

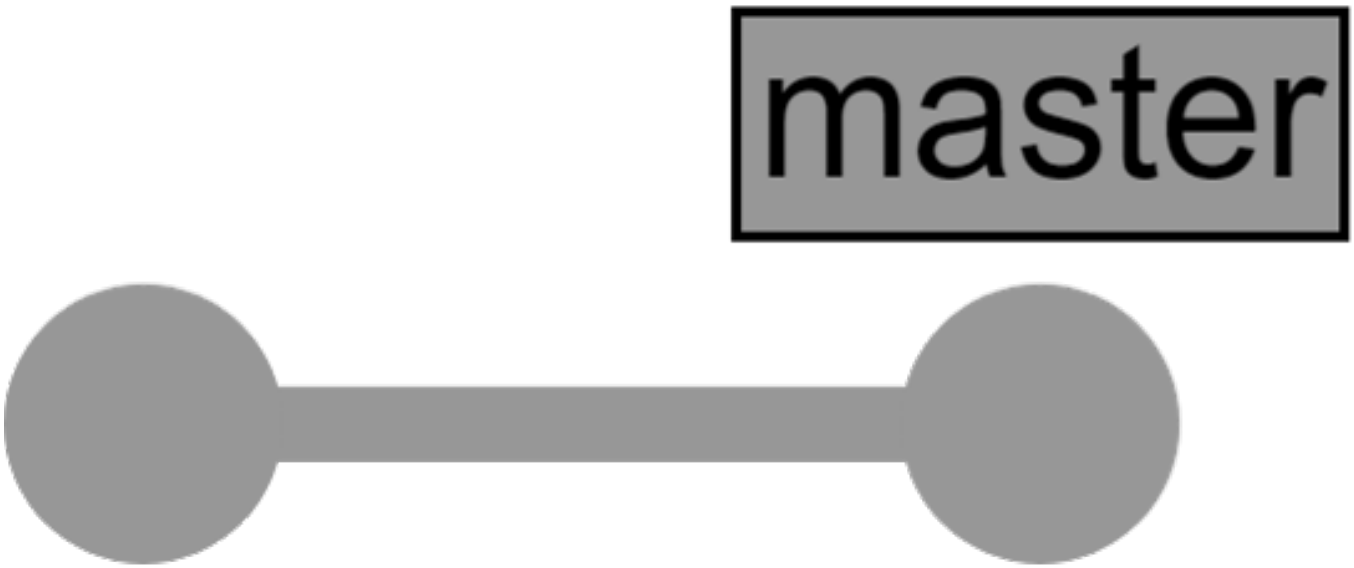


En esta imagen puede verse el repositorio con dos commits (círculos grises), el segundo es la cabeza de la rama maestra. Después de agregar más commits, nuestro repositorio comienza a verse así:

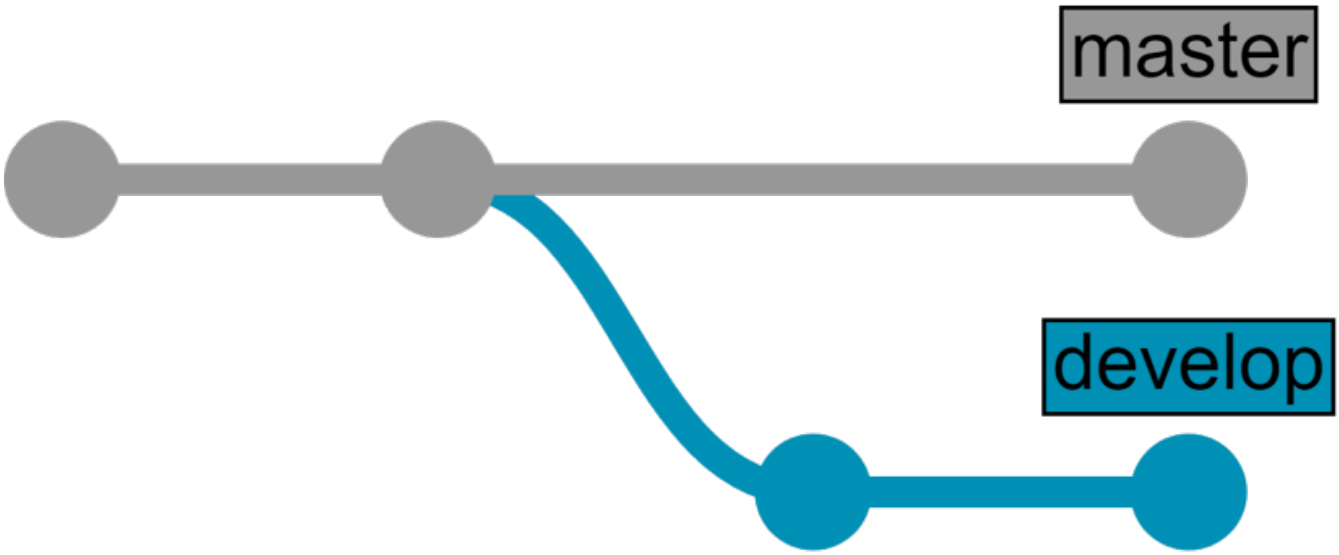
Este es el caso más sencillo. Un desarrollador trabaja en un cambio a la vez. Sin embargo, normalmente existen muchos desarrolladores trabajando simultáneamente en diferentes características y necesitamos un **árbol de commit** para mostrar lo que sucede en nuestro repositorio.

Árbol de commit

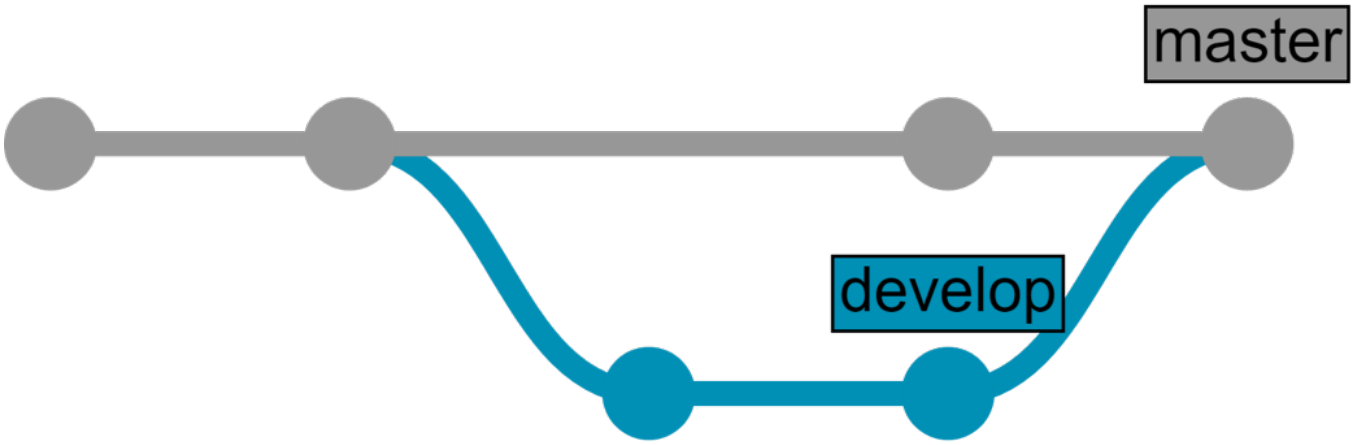
Empezaremos desde el mismo punto de partida. Este es el repositorio con dos commits:



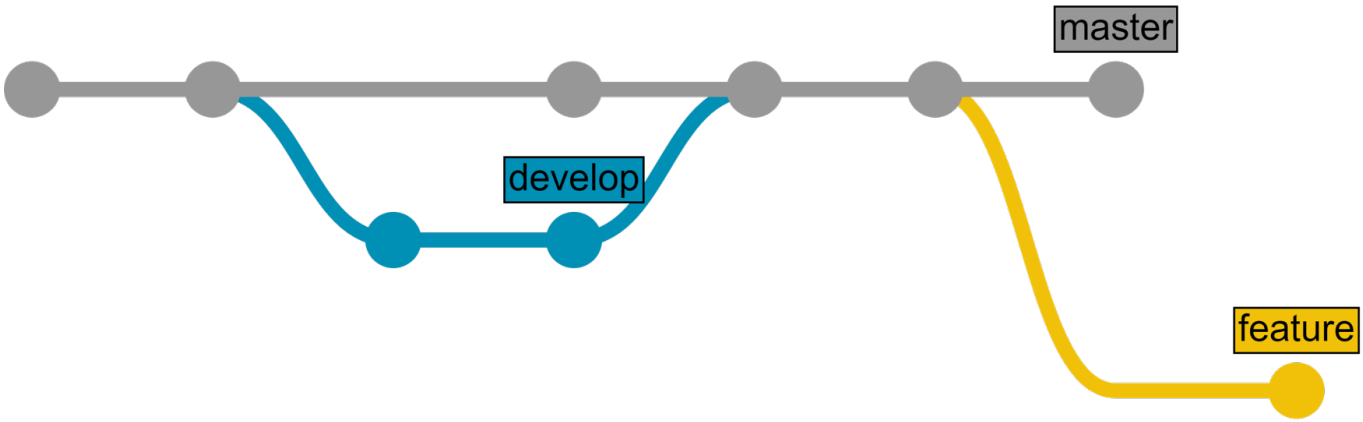
Pero en este momento, dos desarrolladores están trabando al mismo tiempo y para que no se interfieran el uno con el otro, ambos trabajan en ramas separadas:



Después de un tiempo necesitan unir los cambios realizados y para ello crean una **solicitud de fusión** (también llamada **pull request**), que es exactamente lo que parece, es una solicitud para unir dos estados diferentes del repositorio (en nuestro caso queremos fusionar la rama de desarrollo dentro de la rama maestra) en un nuevo estado. Una vez que se revisó y aprobó correctamente, nuestro repositorio se ve de la siguiente manera:



Y el desarrollo continúa:



Git 101 - Resumen

Conceptos principales:

- **Git** es un sistema de control distribuido (no lineal) de las versiones
- **Repositorio** almacena datos y meta-información sobre los datos
- **Commit** es un estado fijo del repositorio
- La **rama (Branch)** es una referencia hacia un commit
- **Solicitud de fusión** (también llamada **pull request**) - es una solicitud para unir dos estados diferentes del repositorio en uno nuevo.

Si desea leer más sobre Git, aquí puede encontrar varios libros sobre Git.

Flujos de Git

Ahora que el lector se familiarizó con los términos y conceptos básicos de Git, hablaremos de cómo se puede administrar parte del desarrollo del ciclo de vida del software utilizando Git. Existe una serie de prácticas (llamada flujos) que describen el proceso de desarrollo utilizando Git, pero únicamente examinaremos dos de ellas:

- Flujo de GitHub
- Flujo de GitLab

Flujo de GitHub

El flujo de GitHub puede describirse con la frase "más fácil imposible". Aquí está:

1. Cree una rama desde el repositorio
2. Utilice commit para mover los cambios hacia su nueva rama
3. Envíe una solicitud pull request desde su rama con los cambios propuestos para iniciar una discusión
4. Utilice commit para añadir más cambios en su rama según sea necesario. Su solicitud pull request se actualizará automáticamente
5. Fusione la solicitud pull request una vez que la rama esté lista para fusionarse

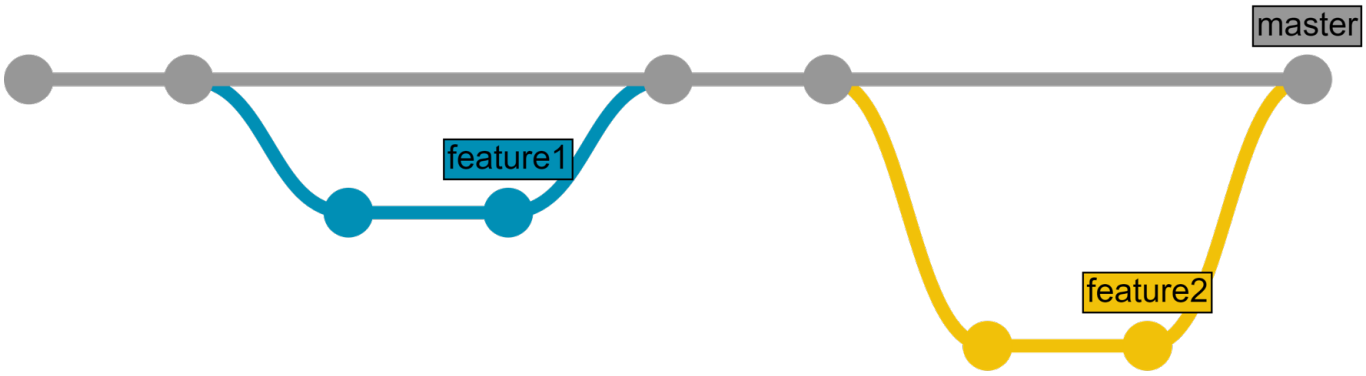
Existen varias reglas que debemos seguir:

- La rama maestra siempre esta en punto de desplegarse (¡y funciona!)
- No existe ningún desarrollo que vaya directamente a la rama maestra
- El desarrollo se realiza en las ramas de las funciones
- Maestro == entorno* de producción**
- Necesita implementarlo para la producción tan frecuentemente como sea posible

* El entorno es un sitio configurado donde se ejecuta su código- podría ser un servidor, una máquina virtual o incluso un contenedor.

** No confundir con "Producciones de Ensemble". Aquí "producción" significa EN VIVO.

Así es como se ve:



Puede aprender más sobre el flujo de GitHub aquí>>. O en esta guía>>.

El flujo de GitHub es bueno para proyectos pequeños y para probarlo si apenas comienza a familiarizarse con los flujos de Git. GitHub los utiliza, no obstante, también puede ser una opción válida para proyectos grande.

Flujo de GitLab

Si no está listo para implementarlo en producción inmediatamente, el flujo de GitLab ofrece el flujo de GitHub + entornos. Así es como funciona: se desarrolla en las ramas de las funciones, según lo definido anteriormente, se fusiona en la rama maestra pero aquí hay un cambio: la rama maestra solo equivale al entorno de prueba. Además, tiene "ramas de entorno", las cuales se enlazan con los otros entornos que pueda tener.

Por lo general, existen tres entornos (se pueden crear más si se necesitan):

- Entorno de prueba == rama maestra
- Entorno de preproducción == rama de preproducción
- Entorno de producción == rama de producción

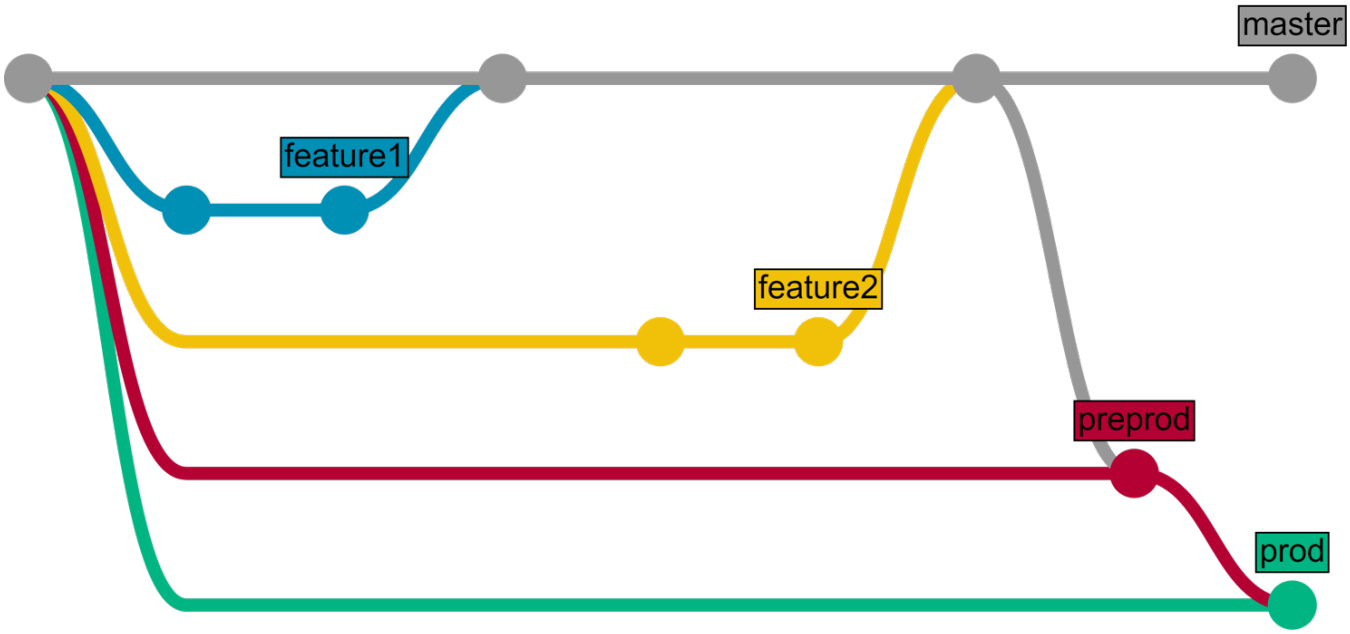
El código que llega a una de las ramas del entorno debe moverse inmediatamente al entorno correspondiente, lo cual puede hacerse:

- Automáticamente (se hará en las partes 2 y 3)
- Semiautomáticamente (igual que en la forma automática, excepto que debe presionarse un botón que autorice la implementación)
- Manualmente

El proceso completo es así:

1. La función se desarrolla en la rama de las funciones
2. La rama de la función pasa por una revisión y se fusiona con la rama maestra
3. Después de un tiempo (una vez que se fusionaron varias funciones) la rama maestra se fusiona con la preproducción
4. Poco después (pruebas de usuario, etc.), la preproducción se fusiona con la producción
5. Mientras realizábamos las fusiones y las pruebas, varias funciones nuevas se desarrollaron y fusionaron en la rama maestra, por lo tanto VAYA al paso

Así es como se ve:



Puede aprender más sobre el flujo de GitLab aquí>>

Conclusión

- Git es un sistema distribuido (no lineal) de control de versiones
- El flujo de Git puede utilizarse como una guía para el ciclo de desarrollo del software, existen varios flujos para elegir

Enlaces

- Libro sobre Git
- Flujo de GitHub
- Flujo de GitLab
- Flujo de Driessen (flujo más completo, como comparación)
- Código para este artículo

Preguntas para debatir

- ¿Utiliza algún flujo de git? ¿Cuál?
- ¿Cuántos entornos tiene para un proyecto de tamaño medio?

Continuará...

En la siguiente parte, veremos:

- Cómo instalar GitLab
- Hablaremos sobre algunas modificaciones recomendadas
- Debatiremos sobre le flujo de trabajo con GitLab Workflow (no confundirse con el flujo de GitLab).

¡Manténganse informado!

#Change Management #Administración del sistema #Contenedorización #Despliegue #Docker #Git #Integración continua #Principiante #Caché
00 2 0 0 228

Mensajes relacionados