Asignatura

Programación 3

Tema

Tarea 3

Docente

Kelyn Tejada

Estudiante

Yameli Martínez Taveras

Matrícula y Sección

2023-1390 / Grupo #1

Índice

¿Qué es Git?

Hoy en día, Git es, con diferencia, el sistema de control de versiones moderno más utilizado del mundo. Git es un proyecto de código abierto maduro y con un mantenimiento activo que desarrolló originalmente Linus Torvalds, el famoso creador del kernel del sistema operativo Linux, en 2005. Un asombroso número de proyectos de software dependen de Git para el control de versiones, incluidos proyectos comerciales y de código abierto. Los desarrolladores que han trabajado con Git cuentan con una buena representación en la base de talentos disponibles para el desarrollo de software, y este sistema funciona a la perfección en una amplia variedad de sistemas operativos e IDE (entornos de desarrollo integrados).

Git, que presenta una arquitectura distribuida, es un ejemplo de DVCS (sistema de control de versiones distribuido, por sus siglas en inglés). En lugar de tener un único espacio para todo el historial de versiones del software, como sucede de manera habitual en los sistemas de control de versiones antaño populares, como CVS o Subversion (también conocido como SVN), en Git, la copia de trabajo del código de cada desarrollador es también un repositorio que puede albergar el historial completo de todos los cambios.

Además de contar con una arquitectura distribuida, Git se ha diseñado teniendo en cuenta el rendimiento, la seguridad y la flexibilidad.

Rendimiento

Las características básicas de rendimiento de Git son muy sólidas en comparación con muchas otras alternativas. La confirmación de nuevos cambios, la ramificación, la fusión y la comparación de versiones anteriores se han optimizado en favor del rendimiento. Los algoritmos implementados en Git aprovechan el profundo conocimiento sobre los atributos comunes de los auténticos árboles de archivos de código fuente, cómo suelen modificarse con el paso del tiempo y cuáles son los patrones de acceso.

A diferencia de algunos programas de software de control de versiones, Git no se deja engañar por los nombres de los archivos a la hora de determinar cuál debería ser el almacenamiento y el historial de versiones del árbol de archivos; en lugar de ello, se centra en el contenido del propio archivo. Al fin y al cabo, los archivos de código fuente se cambian de nombre, se dividen y se reorganizan con frecuencia. El formato de objeto de los archivos del repositorio de Git emplea una combinación de codificación delta (que almacena las diferencias de contenido) y compresión, y guarda explícitamente el contenido de los directorios y los objetos de metadatos de las versiones.

Su arquitectura distribuida también permite disfrutar de importantes ventajas en términos de rendimiento.

Seguridad

Git se ha diseñado con la principal prioridad de conservar la integridad del código fuente gestionado. El contenido de los archivos y las verdaderas relaciones entre estos y los directorios, las versiones, las etiquetas y las confirmaciones, todos ellos objetos del repositorio de Git, están protegidos con un algoritmo de hash criptográficamente seguro llamado "SHA1". De este modo, se salvaguarda el código y el historial de cambios frente a las modificaciones accidentales y maliciosas, y se garantiza que el historial sea totalmente trazable.

Con Git, puedes tener la certeza de contar con un auténtico historial de contenido de tu código fuente.

Algunos otros sistemas de control de versiones carecen de protección contra las modificaciones ocultas realizadas con posterioridad, algo que puede suponer una grave vulnerabilidad de seguridad de la información para cualquier organización que se base en el desarrollo de software.

Flexibilidad

Uno de los objetivos clave de Git en cuanto al diseño es la flexibilidad. Git es flexible en varios aspectos: en la capacidad para varios tipos de flujos de trabajo de desarrollo no lineal, en su eficiencia en proyectos tanto grandes como pequeños y en su compatibilidad con numerosos sistemas y protocolos.

Git se ha ideado para posibilitar la ramificación y el etiquetado como procesos de primera importancia (a diferencia de SVN) y las operaciones que afectan a las ramas y las etiquetas (como la fusión o la reversión) también se almacenan en el historial de cambios. No todos los sistemas de control de versiones ofrecen este nivel de seguimiento.

Control de versiones con Git

Git es la mejor opción para la mayoría de los equipos de software actuales. Aunque cada equipo es diferente y debería realizar su propio análisis, aquí recogemos los principales motivos por los que destaca el control de versiones de Git con respecto a otras alternativas:

Git es una excelente herramienta

Git tiene la funcionalidad, el rendimiento, la seguridad y la flexibilidad que la mayoría de los equipos y desarrolladores individuales necesitan. Estas cualidades de Git se detallan más arriba. En comparación directa con gran parte de las demás alternativas, Git resulta muy ventajoso para muchos equipos.

Git es un estándar de facto

Git es la herramienta con el mayor índice de adopción de su clase, lo que la hace muy atractiva por las siguientes razones. En Atlassian, casi todo el código fuente de nuestros proyectos se gestiona en Git.

Git es un proyecto de código abierto de calidad

Git es un proyecto de código abierto muy bien respaldado con más de una década de gestión de gran fiabilidad. Los encargados de mantener el proyecto han demostrado un criterio equilibrado y un enfoque maduro para satisfacer las necesidades a largo plazo de sus usuarios con publicaciones periódicas que mejoran la facilidad de uso y la funcionalidad. La calidad del software de código abierto resulta sencilla de analizar y un sinnúmero de empresas dependen en gran medida de esa calidad.

Git goza de una amplia base de usuarios y de un gran apoyo por parte de la comunidad. La documentación es excepcional y para nada escasa, ya que incluye libros, tutoriales y sitios web especializados, así como podcasts y tutoriales en vídeo.

¿Para qué sirve el comando git init?

El comando `git init` crea un nuevo repositorio de Git. Puede utilizarse para convertir un proyecto existente y sin versión en un repositorio de Git, o para inicializar un nuevo repositorio vacío. La mayoría de los demás comandos de Git no se encuentran disponibles fuera de un repositorio inicializado, por lo que este suele ser el primer comando que se ejecuta en un proyecto nuevo.

Al ejecutar `git init`, se crea un subdirectorio de `.git` en el directorio de trabajo actual, que contiene todos los metadatos de Git necesarios para el nuevo repositorio. Estos metadatos incluyen subdirectorios de objetos, referencias y archivos de plantilla. También se genera un archivo `HEAD` que apunta a la confirmación actualmente extraída.

Aparte del directorio de `.git`, en el directorio raíz del proyecto, se conserva un proyecto existente sin modificar (a diferencia de SVN, Git no requiere un subdirectorio de `.git` en cada subdirectorio).

De manera predeterminada, `git init` inicializará la configuración de Git en la ruta del subdirectorio de `.git`. Puedes cambiar y personalizar dicha ruta si quieres que se encuentre en otro sitio. Asimismo, puedes establecer la variable de entorno `$GIT_DIR` en una ruta personalizada para que `git init` inicialice ahí los archivos de configuración de Git. También puedes utilizar el argumento `--separate-git-dir` para conseguir el mismo resultado. Lo que se suele hacer con un subdirectorio de `.git` independiente es mantener los archivos ocultos de la configuración del sistema (`.bashrc`, `.vimrc`, etc.) en el directorio principal, mientras que la carpeta de `.git` se conserva en otro lugar.

Uso

En comparación con SVN, el comando `git init` permite crear de manera increíblemente sencilla nuevos proyectos con control de versiones. Con Git, no es necesario que crees un repositorio, importes los archivos ni extraigas una copia de trabajo. Además, Git no precisa de la existencia previa de ningún servidor o privilegios de administrador. Basta con que utilices el comando cd en el subdirectorio de tu proyecto y ejecutes `git init` para que tengas un repositorio de Git totalmente funcional.

```
git init
```

Transforma el directorio actual en un repositorio de Git. De esta forma, se añade un subdirectorio de `.git` al directorio actual y se crea la posibilidad de empezar a registrar las revisiones del proyecto.

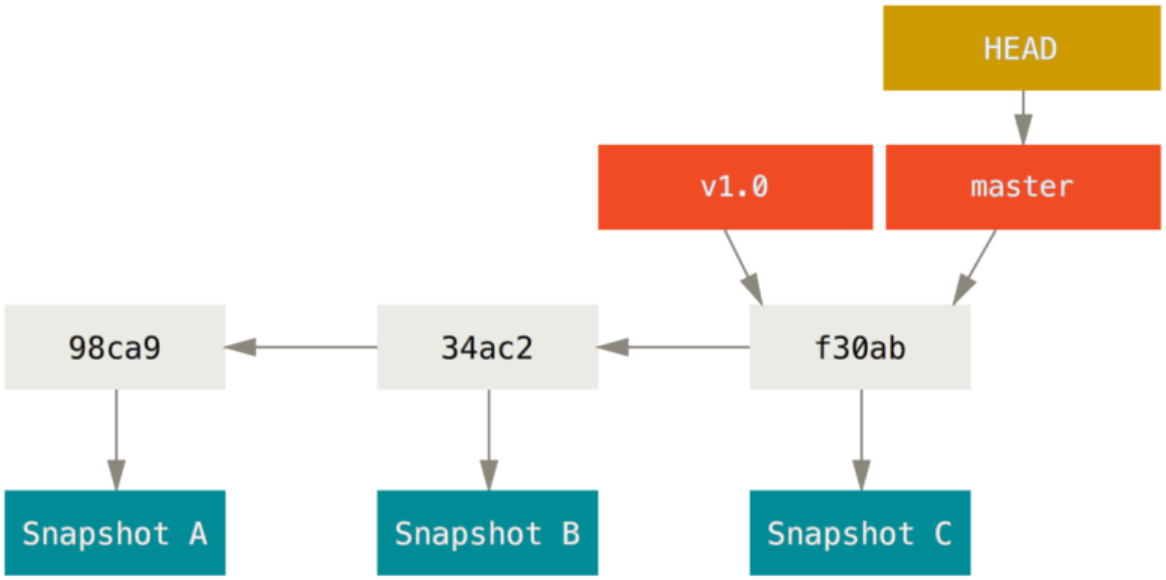
```
git init <directory>
```

Crea un repositorio de Git vacío en el directorio especificado. Al ejecutar este comando, se creará un nuevo subdirectorio llamado ＜directory＞ que nada más contendrá el subdirectorio `.git`.

Si ya has ejecutado `git init` en el directorio de un proyecto y este contiene un subdirectorio de `.git`, puedes volver a ejecutar `git init` de forma segura en el mismo directorio del proyecto. No se reemplazará la configuración de `.git` existente.

¿Qué es una rama en Git?

Una rama Git es simplemente un apuntador móvil apuntando a una de esas confirmaciones. La rama por defecto de Git es la rama master. Con la primera confirmación de cambios que realicemos, se creará esta rama principal master apuntando a dicha confirmación. En cada confirmación de cambios que realicemos, la rama irá avanzando automáticamente.

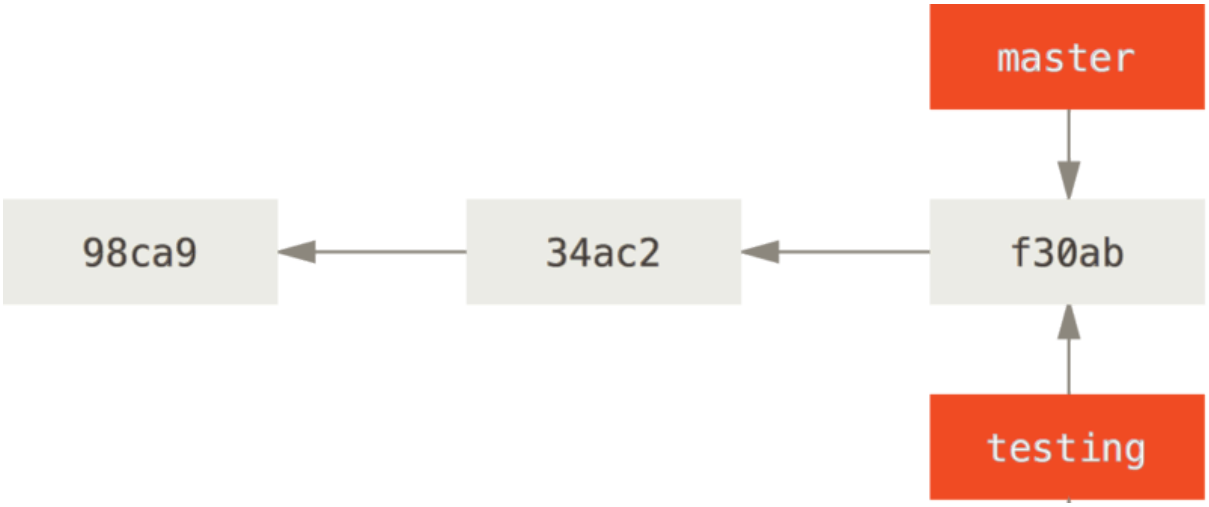


Crear una Rama Nueva

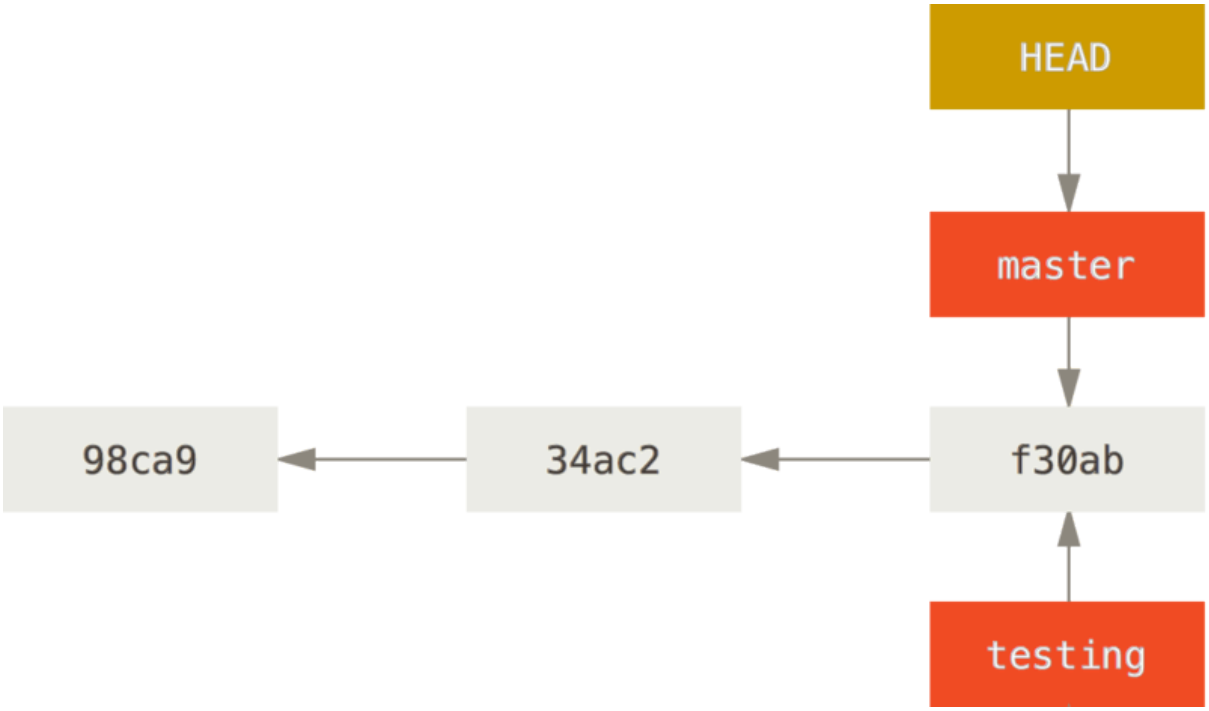
¿Qué sucede cuando creas una nueva rama? Bueno…, simplemente se crea un nuevo apuntador para que lo puedas mover libremente. Por ejemplo, supongamos que quieres crear una rama nueva denominada "testing". Para ello, usarás el comando `git branch`:

```
$ git branch testing
```

Esto creará un nuevo apuntador apuntando a la misma confirmación donde estés actualmente.



Y, ¿cómo sabe Git en qué rama estás en este momento? Pues…, mediante un apuntador especial denominado HEAD. Aunque es preciso comentar que este HEAD es totalmente distinto al concepto de HEAD en otros sistemas de control de cambios como Subversion o CVS. En Git, es simplemente el apuntador a la rama local en la que tú estés en ese momento, en este caso la rama `master`; pues el comando `git branch` solamente crea una nueva rama, pero no salta a dicha rama.



¿Cómo saber en cuál rama estoy trabajando?

El comando `git branch` tiene más funciones que las de crear y borrar ramas. Si lo lanzas sin parámetros, obtienes una lista de las ramas presentes en tu proyecto:

```
$ git branch
  iss53
* master
  testing
```

El carácter * delante de la rama `master`: nos indica la rama activa en este momento (la rama a la que apunta `HEAD`). Si hacemos una confirmación de cambios (commit), esa será la rama que avance.

¿Quién creó Git?

Git es un software de control de versiones diseñado por **Linus Torvalds**, pensando en la eficiencia, la confiabilidad y compatibilidad del mantenimiento de versiones de aplicaciones cuando estas tienen un gran número de archivos de código fuente. Su propósito es llevar registro de los cambios en archivos de computadora incluyendo coordinar el trabajo que varias personas realizan sobre archivos compartidos en un repositorio de código.

¿Cuáles son los comandos esenciales de Git?

git add
Mueve los cambios del directorio de trabajo al área del entorno de ensayo. Así puedes preparar una instantánea antes de confirmar en el historial oficial.
rama de git

Este comando es tu herramienta de administración de ramas de uso general. Permite crear entornos de desarrollo aislados en un solo repositorio.

Git Checkout

Además de extraer las confirmaciones y las revisiones de archivos antiguos, git checkout también sirve para navegar por las ramas existentes. Combinado con los comandos básicos de Git, es una forma de trabajar en una línea de desarrollo concreta.

git clean

Elimina los archivos sin seguimiento de tu directorio de trabajo. Es la contraparte lógica de git reset, que normalmente solo funciona en archivos con seguimiento.

git clone

Crea una copia de un repositorio de Git existente. La clonación es la forma más habitual de que los desarrolladores obtengan una copia de trabajo de un repositorio central.

git commit

Confirma la instantánea preparada en el historial del proyecto. En combinación con git add, define el flujo de trabajo básico de todos los usuarios de Git.

git commit --amend

Pasar la marca --amend a git commit permite modificar la confirmación más reciente. Es muy práctico si olvidas preparar un archivo u omites información importante en el mensaje de confirmación.

git config

Este comando va bien para establecer las opciones de configuración para instalar Git. Normalmente, solo es necesario usarlo inmediatamente después de instalar Git en un nuevo equipo de desarrollo.

git fetch

Con este comando, se descarga una rama de otro repositorio junto con todas sus confirmaciones y archivos asociados. Sin embargo, no intenta integrar nada en el repositorio local. Esto te permite inspeccionar los cambios antes de fusionarlos en tu proyecto.

git init

Inicializa un nuevo repositorio de Git. Si quieres poner un proyecto bajo un control de revisiones, este es el primer comando que debes aprender.

git log

Permite explorar las revisiones anteriores de un proyecto. Proporciona varias opciones de formato para mostrar las instantáneas confirmadas.

Git merge

Es una forma eficaz de integrar los cambios de ramas divergentes. Después de bifurcar el historial del proyecto con git branch, git merge permite unirlo de nuevo.

git pull

Este comando es la versión automatizada de git fetch. Descarga una rama de un repositorio remoto e inmediatamente la fusiona en la rama actual. Este es el equivalente en Git de svn update.

git push

Enviar (push) es lo opuesto a recuperar (fetch), con algunas salvedades. Permite mover una o varias ramas a otro repositorio, lo que es una buena forma de publicar contribuciones. Es como svn commit, pero envía una serie de confirmaciones en lugar de un solo conjunto de cambios.

git rebase

Un cambio de base con git rebase permite mover las ramas, lo que ayuda a evitar confirmaciones de fusión innecesarias. El historial lineal resultante suele ser mucho más fácil de entender y explorar.

git rebase -i

La marca -i se usa para iniciar una sesión de cambio de base interactivo. Esto ofrece todas las ventajas de un cambio de base normal, pero te da la oportunidad de añadir, editar o eliminar confirmaciones sobre la marcha.

git reflog

Git realiza el seguimiento de las actualizaciones en el extremo de las ramas mediante un mecanismo llamado registro de referencia o reflog. Esto permite volver a los conjuntos de cambios aunque no se haga referencia a ellos en ninguna rama o etiqueta.

git remote

Es un comando útil para administrar conexiones remotas. En lugar de pasar la URL completa a los comandos fetch, pull y push, permite usar un atajo más significativo.

git reset

Deshace los cambios en los archivos del directorio de trabajo. El restablecimiento permite limpiar o eliminar por completo los cambios que no se han enviado a un repositorio público.

git revert

Permite deshacer una instantánea confirmada. Si descubres una confirmación errónea, revertirla es una forma fácil y segura de eliminarla por completo del código base.

git status

Muestra el estado del directorio en el que estás trabajando y la instantánea preparada. Lo mejor es que lo ejecutes junto con git add y git commit para ver exactamente qué se va a incluir en la próxima instantánea.

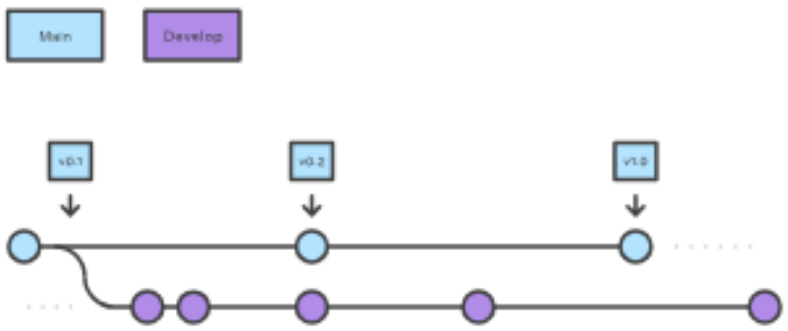
¿Qué es Git Flow?

Gitflow es un modelo alternativo de creación de ramas en Git en el que se utilizan ramas de función y varias ramas principales.

Fue Vincent Driessen en nvie quien lo publicó por primera vez y quien lo popularizó. En comparación con el desarrollo basado en troncos, Gitflow tiene diversas ramas de más duración y mayores confirmaciones. Según este modelo, los desarrolladores crean una rama de función y retrasan su fusión con la rama principal del tronco hasta que la función está completa. Estas ramas de función de larga duración requieren más colaboración para la fusión y tienen mayor riesgo de desviarse de la rama troncal. También pueden introducir actualizaciones conflictivas.

Gitflow puede utilizarse en proyectos que tienen un ciclo de publicación programado, así como para la práctica recomendada de DevOps de entrega continua. Este flujo de trabajo no añade ningún concepto o comando nuevo, aparte de los que se necesitan para el flujo de trabajo de ramas de función. Lo que hace es asignar funciones muy específicas a las distintas ramas y definir cómo y cuándo deben estas interactuar. Además de las ramas de función, utiliza ramas individuales para preparar, mantener y registrar publicaciones. Por supuesto, también puedes aprovechar todas las ventajas que aporta el flujo de trabajo de ramas de función: solicitudes de incorporación de cambios, experimentos aislados y una colaboración más eficaz.

Funcionamiento



Ramas principales y de desarrollo

En lugar de una única rama `main`, este flujo de trabajo utiliza dos ramas para registrar el historial del proyecto. La rama `main` o principal almacena el historial de publicación oficial y la rama `develop` o de desarrollo sirve como rama de integración para las funciones. Asimismo, conviene etiquetar todas las confirmaciones de la rama `main` con un número de versión.

El primer paso es complementar la rama `main` predeterminada con una rama `develop`. Una forma sencilla de hacerlo es que un desarrollador cree de forma local una rama `develop` vacía y la envíe al servidor:

```
git branch develop
git push -u origin develop
```

Esta rama contendrá el historial completo del proyecto, mientras que `main` contendrá una versión abreviada. A continuación, otros desarrolladores deberían clonar el repositorio central y crear una rama de seguimiento para `develop`.

A la hora de utilizar la biblioteca de extensiones de git-flow, ejecutar `git flow init` en un repositorio existente creará la rama `develop`:

```
$ git flow init

Initialized empty Git repository in ~/project/.git/
No branches exist yet. Base branches must be created now.
Branch name for production releases: [main]
Branch name for "next release" development: [develop]

How to name your supporting branch prefixes?
Feature branches? [feature/]
Release branches? [release/]
Hotfix branches? [hotfix/]
Support branches? [support/]
Version tag prefix? []

$ git branch
* develop
 main
```

¿Qué es el desarrollo basado en trunk (Trunk Based Development)?

El desarrollo basado en troncos es una práctica de gestión de control de versiones en la que los desarrolladores fusionan pequeñas actualizaciones de forma frecuente en un "tronco" o rama principal. Se ha convertido en una práctica habitual entre los equipos de DevOps y parte del ciclo de vida de DevOps, ya que simplifica las fases de fusión e integración. De hecho, el desarrollo basado en troncos es una práctica obligatoria de la CI y la CD. Permite a los desarrolladores crear ramas de corta duración con pequeñas confirmaciones, a diferencia de otras estrategias de ramas de funciones de larga duración. A medida que la complejidad del código base y el tamaño del equipo van creciendo, el desarrollo basado en troncos ayuda a mantener el flujo de publicación de la producción.

Comparación de Gitflow y desarrollo basado en troncos

Gitflow es un modelo alternativo de creación de ramas en Git que utiliza ramas de función de larga duración y varias ramas principales. Gitflow tiene más ramas de mayor duración y confirmaciones más grandes que el desarrollo basado en troncos. Según este modelo, los desarrolladores crean una rama de función y retrasan su fusión con la rama principal del tronco hasta que la función está completa. Estas ramas de función de larga duración requieren más colaboración para fusionarlas, ya que presentan un mayor riesgo de desviarse de la rama troncal e introducir actualizaciones conflictivas.

Gitflow también tiene líneas de rama principales independientes para el desarrollo, las correcciones, las funciones y las publicaciones. Existen diferentes estrategias para fusionar las confirmaciones entre estas ramas. Al haber más ramas que gestionar y compaginar, suele haber más complejidad, por lo que se requieren sesiones adicionales de planificación y revisión por parte del equipo.

El desarrollo por tronco está mucho más simplificado, ya que se centra en la rama principal como fuente de correcciones y publicaciones. En este tipo de desarrollo, se da por sentado

que la rama principal permanece estable, sin incidencias, y siempre está lista para la implementación.

Ventajas del desarrollo basado en troncos

El desarrollo basado en troncos es una práctica obligatoria para la integración continua. Si los procesos de desarrollo y pruebas están automatizados, pero los desarrolladores trabajan en ramas de función aisladas y largas que se integran con poca frecuencia en una rama compartida, no se está aprovechando todo el potencial de la integración continua.

El desarrollo basado en troncos disminuye la fricción de la integración del código. Cuando los desarrolladores terminan una tarea nueva, deben fusionar el código nuevo en la rama principal; pero no deben fusionar los cambios en el tronco hasta que hayan comprobado que los pueden compilar correctamente. Durante esta fase, pueden surgir conflictos si se han realizado modificaciones desde el inicio de la tarea nueva. En concreto, estos conflictos son cada vez más complejos a medida que los equipos de desarrollo crecen y la base de código se amplía. Esto ocurre cuando los desarrolladores crean ramas independientes que se desvían de la rama de origen y otros desarrolladores están fusionando a la vez código que se solapa. Por suerte, el modelo de desarrollo basado en troncos reduce estos conflictos.

Bibliografía

Atlassian. (s/f-a). Comandos básicos de Git. Atlassian. Recuperado el 3 de abril de 2025, de https://www.atlassian.com/es/git/glossary

Atlassian. (s/f-b). Desarrollo basado en troncos. Atlassian. Recuperado el 3 de abril de 2025, de https://www.atlassian.com/es/continuous-delivery/continuous-integration/trunk-based-development

Atlassian. (s/f-c). Flujo de trabajo de Gitflow. Atlassian. Recuperado el 3 de abril de 2025, de https://www.atlassian.com/es/git/tutorials/comparing-workflows/gitflow-workflow

Atlassian. (s/f-d). git init. Atlassian. Recuperado el 3 de abril de 2025, de https://www.atlassian.com/es/git/tutorials/setting-up-a-repository/git-init

Atlassian. (s/f-e). Qué es Git. Atlassian. Recuperado el 3 de abril de 2025, de https://www.atlassian.com/es/git/tutorials/what-is-git

Git - Gestión de Ramas. (s/f). Git-scm.com. Recuperado el 3 de abril de 2025, de https://git-scm.com/book/es/v2/Ramificaciones-en-Git-Gesti%C3%B3n-de-Ramas

Git - ¿Qué es una rama? (s/f). Git-scm.com. Recuperado el 3 de abril de 2025, de https://git-scm.com/book/es/v2/Ramificaciones-en-Git-%C2%BFQu%C3%A9-es-una-rama%3F

Wikipedia contributors. (s/f). Git. Wikipedia, The Free Encyclopedia. https://es.wikipedia.org/w/index.php?title=Git&oldid=161647604