UNIVERSIDAD TECNOLÓGICA DEL CENTRO DE VERACRUZ

INGENIERÍA EN DESARROLLO Y GESTIÓN DE SOFTWARE

9NO CUATRIMESTRE VS

LUIS FERNANDO LEYVA LUNA

FRANCISCO TRUJILLO ROMERO

04 de junio del 2022

Introducción

El desarrollo de software es un procedimiento bastante riguroso en el que es necesario llevar un control sobre lo que se está desarrollando, muchas de las veces los equipos de desarrollo pueden ser pequeños o grandes, esto significa que un solo proyecto de software puede ser trabajado por uno o varios desarrolladores, por lo que se requiere de un control sobre lo que cada uno de ellos va desarrollando, aquí es donde entran los sistemas de versionamiento de software, ya que estos nos permiten tener un control total sobre nuestros proyectos, de esta forma podemos evitar incidentes dentro de los mismos, así como también podemos tener un control sobre los desarrolladores que integran cambios al proyecto y que podrían estar utilizando malas prácticas de desarrollo o bien, que no están cumpliendo con sus metas y objetivos. Los sistemas de versionamiento de software, también nos ayudan, como su nombre lo indica, a tener un control sobre las versiones de nuestros proyectos, de esta forma podemos reducir el riesgo de pérdidas de información, tiempo y presupuesto, ya que se nos es más fácil poder identificar las versiones de nuestro software en caso de que alguna de ellas llegase a fallar estando en producción.

Los sistemas de versionamiento de software nacieron para solucionar todos estos problemas, como sabemos, seguramente nos tocó trabajar en su momento de una manera bastante rudimentaria y arcaica, puesto que para manejar las versiones de nuestros proyectos de software recurríamos a la clásica práctica de tener varías copias de nuestros proyectos con las funciones nuevas que les hubiésemos implementado, esto evidentemente era un problema bastante grande, ya que en proyectos inmensos, la capacidad de almacenamiento de nuestros discos duros podría verse comprometida, pero no solo eso, sino que además, identificar la versión de software que pudiera ser estable se volvía toda una travesía, además, otro de los riesgos que enfrentábamos en ese entonces, era el de la pérdida de la información, un mal respaldo de nuestros proyectos y un formateo total de nuestros equipos y adió trabajo, a pesar de que hoy en día ese riesgo sigue existiendo, lo cierto es que se reduce considerablemente ya que estos sistemas permiten que nuestros proyectos puedan estar respaldados en diferentes equipos a la vez.

Para conocer un poco más sobre estos sistemas de versionamiento que existen actualmente, a continuación, veremos una pequeña descripción de cada uno de ellos y así poder elegir el que más se acople a nuestras necesidades.

Capítulo 1

Plataformas utilizadas para el versionamiento

GITHUB

Se trata de una de las principales plataformas para crear proyectos abiertos de herramientas y aplicaciones, y se caracteriza sobre todo por sus funciones colaborativas que ayudan a que todos puedan aportar su granito de arena para mejorar el código.

Como buen repositorio, el código de los proyectos que sean abiertos puede ser descargado y revisado por cualquier usuario, lo que ayuda a mejorar el producto y crear ramificaciones a partir de él. Y si prefieres que tu código no se vea, también pueden crearse proyectos privados.

GITLAB

GitLab es un repositorio de gestión de proyectos y, como se puede deducir desde el nombre, está construido sobre Git. Es decir, proporciona el código para generar un servidor y gestionar los clientes, sus operaciones y los servicios que ofrece.

A través de GitLab puedes administrar grupos, personas y los permisos que quieras que tengan los usuarios dentro de los grupos o proyectos a los que pertenezcan. Si estás buscando cómo aumentar la productividad laboral en tu empresa, esta herramienta puede ayudarte.

También, te permite llevar a cabo un seguimiento del estado actual y del histórico de los proyectos, logrando ver todos los cambios y modificaciones producidas en el tiempo de desarrollo.

Teniendo presente que, GitLab es un software libre que puedes descargar e instalar en cualquier servidor, permitirá que lo uses en tu equipo personal, de manera profesional y en los ordenadores de tu organización sin ningún costo.

BITBUCKET

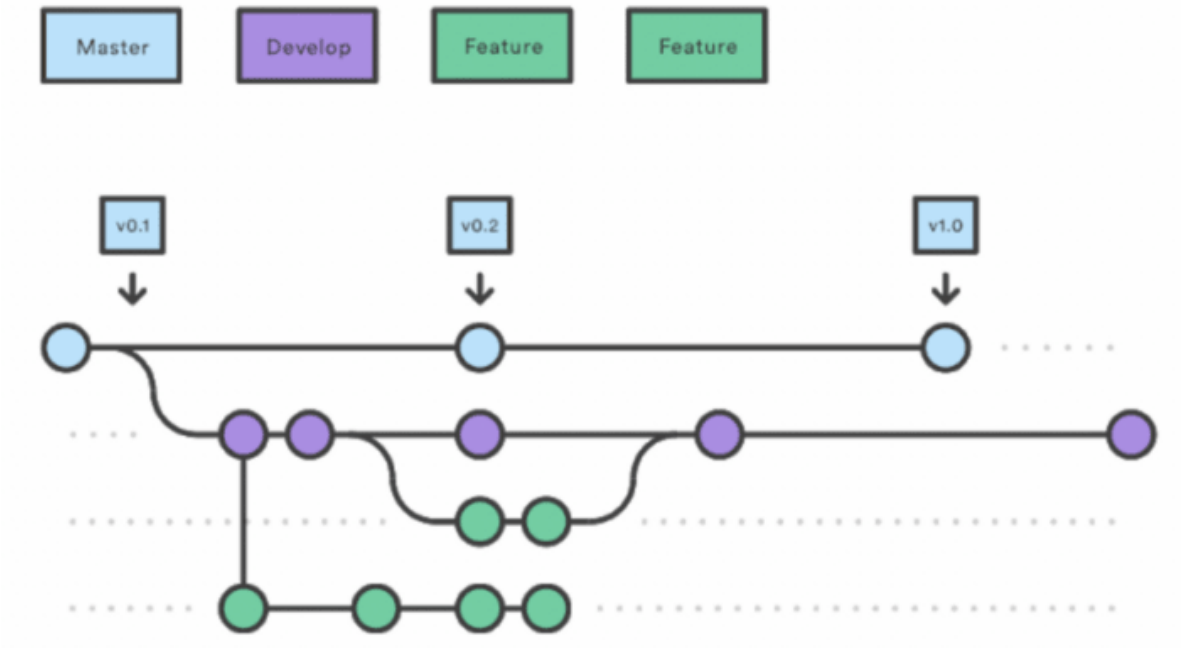
Bitbucket es la herramienta ideal para compartir datos, implementar y crear código, así como automatizar pruebas. Se puede acceder a Bitbucket Cloud mediante una URL o también permite el alojamiento local, utilizando infraestructuras on-premisses. Bitbucket Server, por otro lado, es compatible con Bamboo, automatizando procesos sin límite de licencia.

Ahora, si te interesa conocer una solución corporativa alojada en varios servidores agrupados, Bitbucket Data Center es lo ideal. Permite un rendimiento escalable, duplicaciones inteligentes y alta disponibilidad.

Capítulo 2

Flujos de trabajo para los controladores de versiones

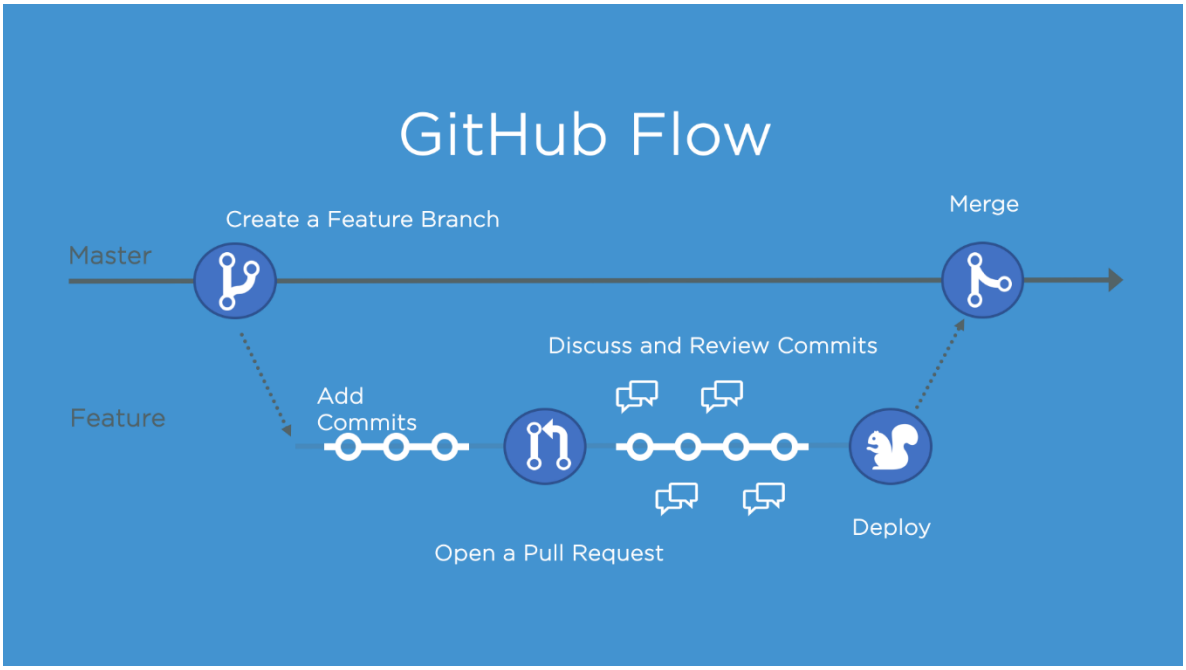
GIT FLOW



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.

GITHUB FLOW



GitHub Flow, fue creado por GitHub y es conocido en la comunidad de desarrolladores como una alternativa simple y ligera a GitFlow. GitHub Flow se basa en un flujo de trabajo basado en branches que permite a equipos de desarrollo enfocarse principalmente en la entrega continua. A diferencia de Git Flow, no existen los branches de “releases”, ya que está pensado para que la implementación en producción ocurra con frecuencia, incluso varias veces al día si es posible.

En esta estrategia de branching, en el repositorio tenemos dos tipos de branches:

- main (o master): el branch de código principal, es el que contiene el código que está listo para producción.
- features: los branches de funcionalidades que permiten el desarrollo en paralelo.

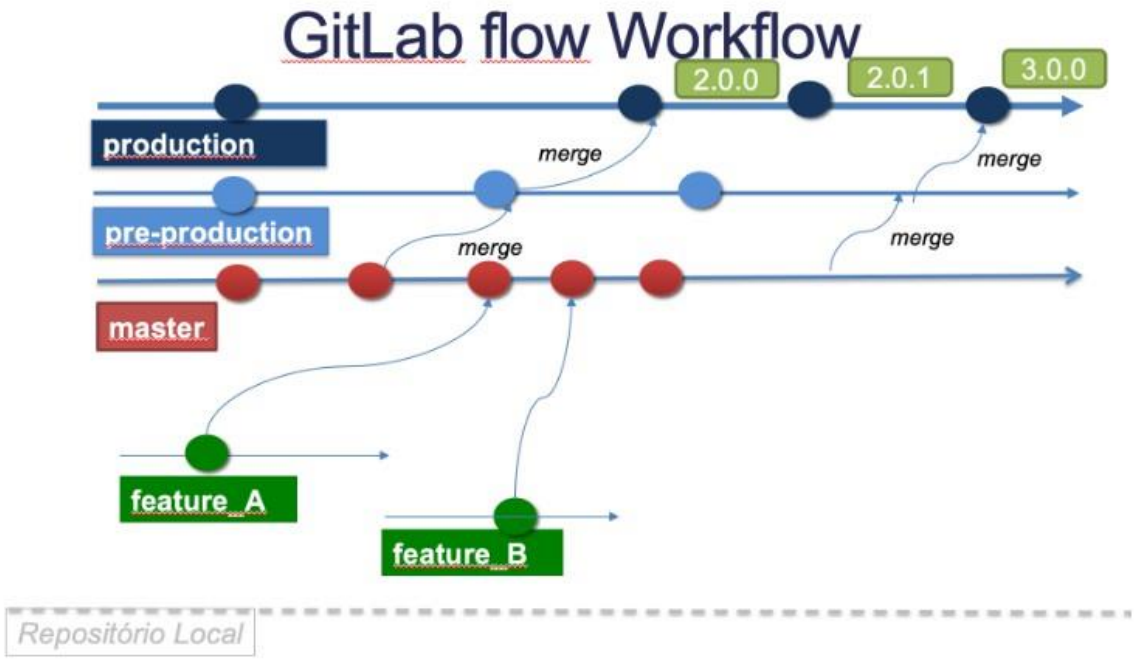
Cuando utilizas GitHub Flow, existen seis principios que debemos seguir para asegurarnos de mantener un buen código listo para producción en todo momento:

- El código que está en _main debe poder implementarse en producción en cualquier momento._
- Cuando se crean nuevos _feature branches, se deben crear con nombres descriptivos. Por ejemplo, feature/add-new-account-type._
- Primero hacer commit en local, para luego hacer push al repositorio remoto.
- Abrir pull requests para solicitar feedback o ayuda, antes de hacer merge en el branch principal.
- Hacer merge del código en el branch _main, solo después de haber recibido aprobación del pull request._

- Una vez hecho el merge del código en _main, debe implementarse de inmediato, mejor si es en un entorno de producción.

GitHub Flow es ideal cuando necesitas mantener una única versión del código de producción.

GITLAB FLOW



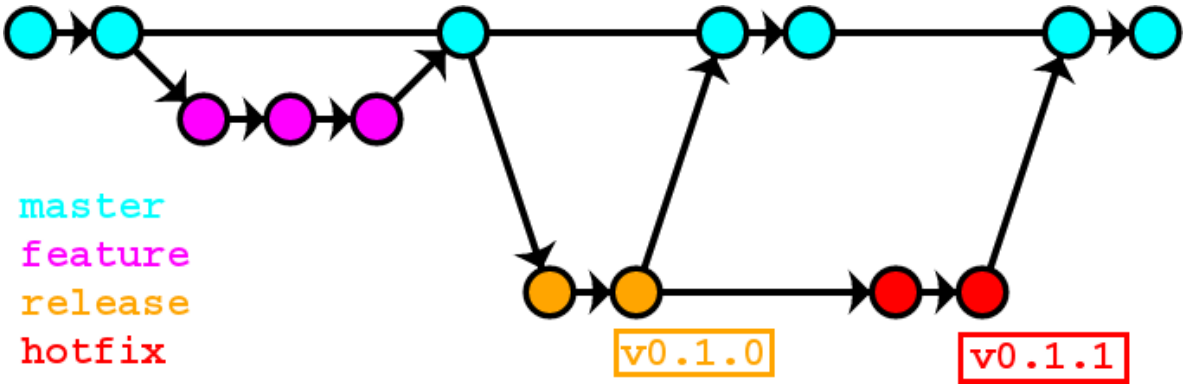
GitLab Flow se caracteriza por un flujo de trabajo bastante similar a GitHub Flow. La principal diferencia es que no solo permite el uso de release branches, sino que adiciona el uso de environment branches, por ejemplo, QA, Pre-Producción y Producción. Esto debido a que considera los casos donde una nueva funcionalidad no siempre puede implementarse en entornos de producción.

Al igual que GitHub Flow, GitLab Flow propone utilizar un master branch y feature branches. Una vez que terminamos el desarrollo en un feature branch, hacemos merge de nuevo con master.

Con respecto a los branches por entornos, estos se crean a partir de master cuando estamos listos para implementar nuestra aplicación. Tener diferentes ramas por entorno nos permite configurar herramientas de CI y CD para que se despliegue automáticamente los commits realizados en cada una de esas ramas. Si detectamos un problema crítico, lo solucionamos en el feature o en master, una vez corregido, hacemos merge con las ramas de los entornos.

Con respecto a los release branches, seguimos haciendo todo el trabajo en feature branches que luego hacemos merge a master al finalizar. Cuando nos aseguramos que master es lo suficientemente estable, es decir, ya hemos realizado todas las pruebas y corregido todos los errores, creamos un release branch que va a contener nuestra versión de código para desplegar. Si hay un problema crítico, primero lo solucionamos en master, realizamos cherry picking para hacer merge en release, es decir, seleccionamos las porciones de código que deseamos que contenga nuestro release.

ONE FLOW



OneFlow fue concebido como una alternativa más simple a GitFlow. Sin embargo, “más simple” no quiere decir que permita hacer menos.

Con OneFlow es necesario tener un master branch que tendrá una vida infinita en el repositorio de código, además, cada nueva versión de producción debe basarse en la versión anterior, una condición que la mayoría de proyectos de software cumple. La mayor diferencia entre One Flow y Git Flow es que en OneFlow no existe develop.

Si bien el flujo de trabajo aboga por tener un branch de larga duración (master), eso no significa que no haya otros branches involucrados. Lo que sí es cierto es que cada uno de estos branches de soporte, por ejemplo, feature, hotfix, etc. deben ser de corta duración, se crean a partir de master y deben, una vez finalizado el desarrollo, volver a master. Los branches de soporte, facilitan el intercambio de código entre los desarrolladores y actúan como respaldo. Sin embargo, el historial de cambios de todo el código, siempre se encontrará en el master branch.

Conclusión

Dado que Git es una herramienta flexible, permite que los desarrolladores trabajen utilizando una amplia variedad de flujos de trabajo y estrategias, lo que hace que seleccionar alguna de entre todas las opciones disponibles no sea tan fácil, especialmente para quienes no tengan mucha experiencia en Git.

Dado que Git es una herramienta flexible, permite que los desarrolladores trabajen utilizando una amplia variedad de flujos de trabajo y estrategias, lo que hace que seleccionar alguna de entre todas las opciones disponibles no sea tan fácil, especialmente para quienes no tengan mucha experiencia en Git.