Git

Podemos definir a Git como un control de versiones pensado para la eficiencia y la confiabilidad del mantenimiento de versiones de aplicaciones que tienen un gran número de archivos de código fuente. Su propósito es llevar el registro de los cambios en archivos locales, compartirlos en un repositorio remoto y así, coordinar el trabajo que varias personas realizan sobre archivos compartidos.

Fue diseñado por Linus Torvalds, que se basó en BitKeeper y en Monotone. Actualmente el núcleo del proyecto Git se ha vuelto un sistema de control de versiones completo, utilizable de forma directa y mantiene una enorme cantidad de código distribuida y gestionada por mucha gente.

Github

Github es una plataforma de desarrollo colaborativo donde se pueden subir proyectos utilizando el sistema de control de versiones Git. Su utilidad principal es para la creación de código fuente de programas de ordenador. Almacena principalmente el código de los proyectos de forma pública, por eso es la plataforma más importante de colaboración para proyectos de código abierto (Open Source).

Fue desarrollado por Chris Wanstrath, P. J. Hyett, Tom Preston-Werner y Scott Chacon usando Ruby on Rails,

Gitlab

Gitlab es un servicio web de control de versiones y desarrollo de software colaborativo basado en Git, y es usado por organizaciones como la NASA, el CERN, IBM o Sony. Además de ser un gestor de repositorios, también ofrece alojamiento de wikis y un sistema de seguimiento de errores, todo ello publicado bajo una licencia de código abierto. Resumidamente, es una suite completa que permite gestionar, administrar, crear y conectar los repositorios con diferentes aplicaciones y hacer todo tipo de integraciones con ellas.

Fue desarrollado por los programadores Dmitriy Zaporozhets y Valery Sizov. Actualmente su arquitectura tecnológica incluye Go, Ruby on Rails y Vue.js.

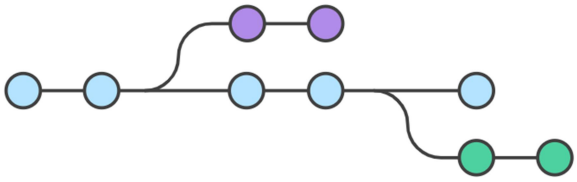
Principales diferencias

- GitHub solo te da la opción de lectura o escritura en un repositorio, mientras que con GitLab puedes proporcionar acceso a determinadas partes del proyecto.
- GitLab permite incluir adjuntos en un issue o problema.
- GitLab permite marcar un proyecto como que está en desarrollo, lo que avisará a otros usuarios de la situación del mismo. De esta manera, se evita que otros usuarios integren tu proyecto antes de tiempo.
- GitHub tiene una comunidad de usuarios y una cantidad de proyectos muy superior a GitLab.

Comandos

- git config: establece una configuración específica de usuario.
- git init: crea un nuevo repertorio GIT.
- git add: agrega archivos al index.
- git clone: revisa repositorios.
- git commit: cambia la cabecera.
- git status: muestra la lista de los archivos que se han cambiado junto con los archivos que están por ser añadidos o comprometidos.
- git push: envía los cambios que se han hecho en la rama principal de los repositorios remotos que están asociados con el directorio en el que se está trabajando.
- git checkout: se puede usar para crear ramas o cambiar entre ellas.
- git remote: se usa para conectar a un repositorio remoto.
- git branch: lista, crea o borra ramas.
- git pull: fusionar todos los cambios que se han hecho en el repositorio local trabajando.
- git merge: fusiona una rama con otra rama activa.
- git diff: hace una lista de conflictos.
- git tag: marca commits específicos con una etiqueta.
- git log: muestra una lista de commits en una rama junto con todos los detalles.
- git reset: resetear el index y el directorio que está trabajando al último estado comprometido.
- git rm: remueve archivos del index y del directorio que está trabajando.
- git stash: ayuda a salvar cambios que no están por ser comprometidos inmediatamente, pero temporalmente.
- git show: muestra información sobre cualquier objeto git.
- git fetch: busca todos los objetos de un repositorio remoto que actualmente no reside en el directorio local en el que se está trabajando.

Ramas



Podemos clasificarlas en ramas principales (rama Master y rama Develop) y en ramas auxiliares (rama Feature, rama Release y rama Hotfix).

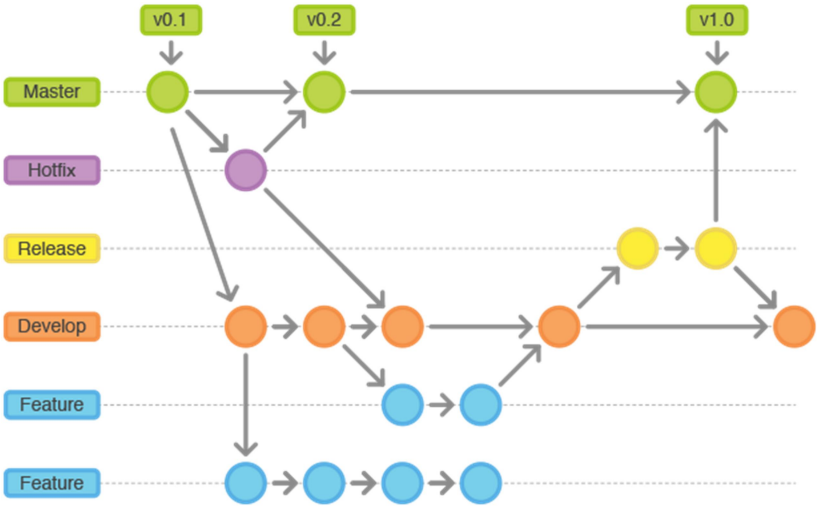
- La rama Master, es la principal y en la que nos encontramos automáticamente al iniciar un proyecto con Git.

- La rama Develop, es la principal de desarrollo en la que se trabajará el código y se ramificará dependiendo de las necesidades del proyecto.

- La rama Feature, para nuevas características, nuevos requisitos o nuevas historias de usuario.

- La rama Release, para estandarizar o cortar una serie de código que ha estado desarrollándose en la rama Develop, se saca una rama de este tipo, se mergea y ahí se depura.

- La rama Hotfix, que habitualmente se utiliza para código para depurar el código que venga de producción, por haberse detectado un defecto crítico en producción que deba resolverse, al que se le va a hacer una Release puntual para corregirlo.



Resolución de conflictos

A veces múltiples desarrolladores podrían intentar editar el mismo contenido. Si el primer desarrollador intenta editar código que el segundo desarrollador está editando, podría producirse un conflicto. Para evitar que ocurran ciertos conflictos, los desarrolladores trabajarán en ramas aisladas separadas. La función principal del comando git merge es combinar ramas separadas y resolver las ediciones conflictivas.

Supongamos que un directorio tenemos un archivo index.html con el siguiente contenido:

```
<!DOCTYPE HTML>
<html>
  <head>
    <title>Titulo</title>
  </head>
  <body>
    <p>Contenido de la web</p>
  </body>
</html>
```

Inicializaremos en él un repositorio haciendo git init y luego haremos nuestro primer commit con git add texto.txt y luego git commit -m “commit inicial”.

Vamos a crear una nueva rama para añadir algo de contenido: git checkout -b contenido.

Con ello ya estamos en la nueva rama y ahora vamos a cambiar el título.

Guardamos los cambios y hacemos commit en esa rama: git commit -a -m “cambios en el título”.

Nos movemos de nuevo a la rama master: git checkout master.

Hacemos otros cambios en el archivo incluyendo el título y luego commit de los cambios: git commit -a -m “se añade más contenido”.

Ahora intentamos hacer merge con la rama creada anteriormente: git merge contenido.

En este caso no podrá hacer el merge y nos mostrará que hay un conflicto que no nos permitirá continuar hasta que se resuelva:

```
Auto-merging index.html
CONFLICT (content): Merge conflict in index.html
Automatic merge failed; fix conflicts and then commit the result.
```

Git nos proporciona una ayuda diciéndonos que archivo tiene el conflicto, el cual al abrirlo nos muestra cuáles son los cambios tanto de una rama como de la otra:

```
<!DOCTYPE HTML>
<html>
  <head>
<<<<<<< HEAD
    <title>Nuevo Titulo</title>
=======
    <title>Nuevo Titulo para la web</title>
>>>>>>> contenido
  </head>
  <body>
    <p>Contenido de la web</p>
    <p>Nuevo párrafo de la página</p>
  </body>
</html>
```

Tenemos que elegir entre lo que está entre <<<<<<< HEAD y ======= que es contenido que tenemos en la rama donde estamos haciendo el merge (master) o entre ======= y >>>>>>> contenido donde están los cambios hechos en la rama que queremos unir (contenido).

Para ello arreglamos el archivo con los cambios elegidos, guardamos, agregamos y hacemos commit de los cambios (git commit -a) y de esta manera logramos hacer merge con éxito.