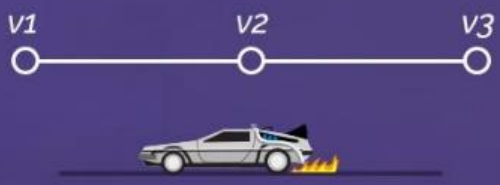
¿Sabes qué es Git?

Es un sistema de control de versiones. Lleva un registro de todos los cambios y avances de tu proyecto

Funciona como una máquina de tiempo, puedes ir al pasado de tu código o volver al presente



GitHub es un servicio que te ayuda a almacenar tu proyecto en la nube, además existen otros servicios como Gitlab o Bitucket

Todo desarrollador sin importar el lenguaje, debe dominar Git.

Git trabaja con ramas

Ayuda a que varias personas trabajen en un mismo proyecto y puedan realizar modificaciones sin afectar a los demas archivos. Una vez que esten listos todos los cambios se fusionan con la rama principal

GIT

Es un sistema de control de versiones desarrollado en 2005 por Linus Thorvalds, el creador de Linux, y publicado bajo la licencia de software libre GPLv2 de GNU. La particularidad de esta herramienta es que, aunque guarda un repositorio central para cada proyecto, todos los participantes descargan una copia local del mismo en su propio dispositivo. Cada una de estas copias constituye una copia completa de todo el contenido del repositorio, por lo que no es necesario estar conectado a la red para trabajar. Además, estos archivos sirven como copia de seguridad en caso de que el repositorio principal falle o resulte dañado. Los cambios en los archivos pueden intercambiarse con todos los demás participantes del proyecto en cualquier momento y, si corresponde, añadirse al repositorio.

Características:

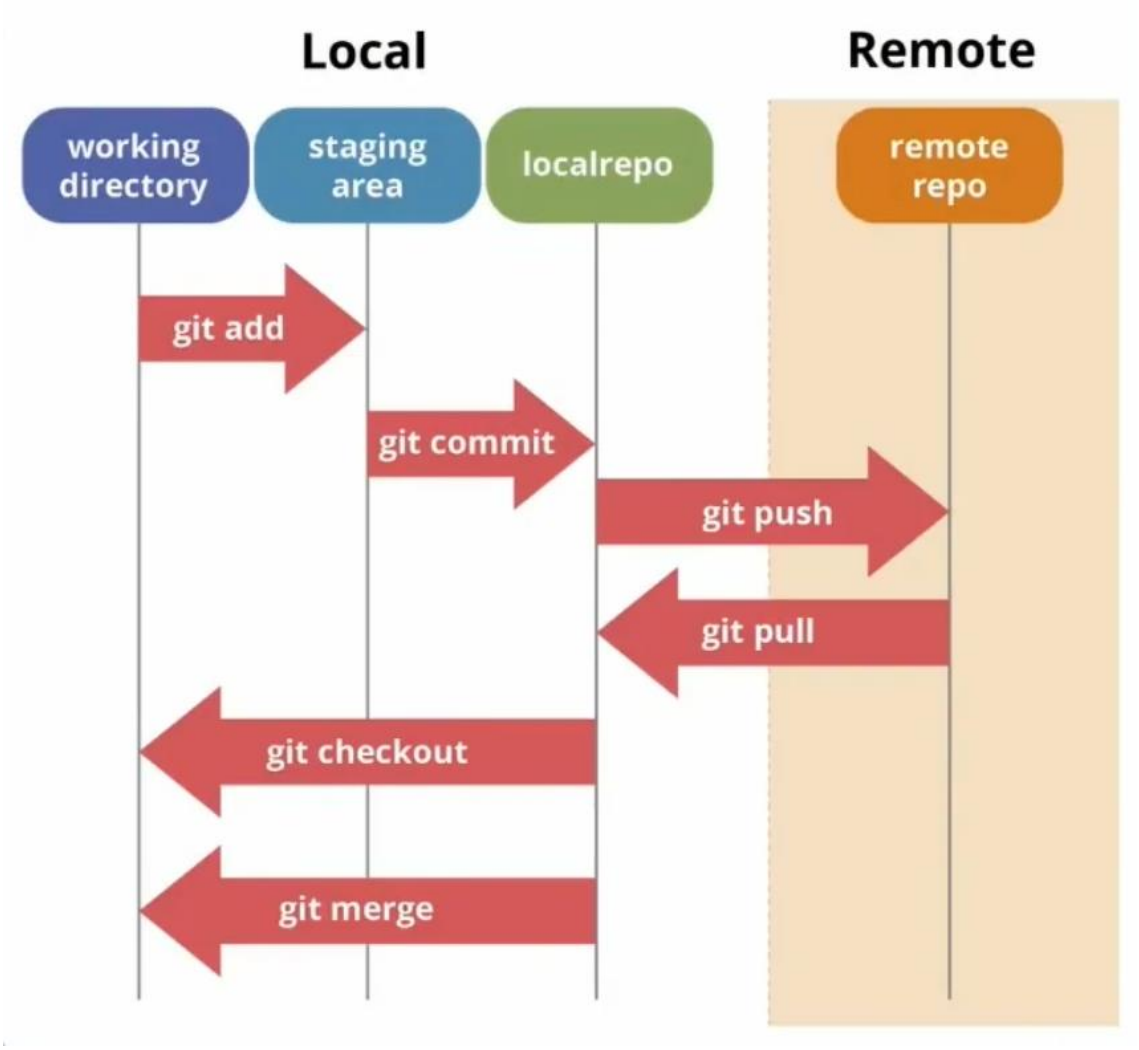
- Es capaz de mantener una gran cantidad de código distribuido, y se creó para trabajar con equipos de envergadura de forma rápida y sencilla.
- Tiene un repositorio local y un repositorio remoto:

Se pueden hacer muchas cosas en local, lo que aporta mucha tranquilidad con respecto a mergear con el repositorio remoto.

Permite modificar código en local, mergear y combinar ese código en local y todo lo que se quiera hacer con el mismo.

Si se está satisfecho con el resultado de esas modificaciones, se pueden subir al repositorio para que sea compartido con otros compañeros. En caso contrario siempre se puede hacer un rollback y volver atrás, algo que también se puede hacer en Subversion, pero siempre afecta al código, a la rama trunk, que es la que está utilizando el resto de los compañeros.

Cómo funciona Git:



¿Qué es GitHub?

GitHub es una plataforma de desarrollo colaborativo de software **para** alojar proyectos utilizando el sistema de control de versiones **Git**. El código se almacena de forma pública, aunque también se puede hacer de forma privada, creando una cuenta de pago.

¿Qué es Gitlab?

Gitlab es un servicio web de control de versiones y desarrollo de software colaborativo basado en Git. Además de gestor de repositorios, el servicio ofrece también alojamiento de wikis y un sistema de seguimiento de errores, todo ello publicado bajo una Licencia de código abierto.

Diferencias

Al margen de la similitud principal de su base común en Git, hay algunas diferencias destacables entre GitLab y GitHub.

Uno de los factores principales es, por ejemplo, el **enorme número de usuarios** de **GitHub**, seguramente el sistema de control de versiones más conocido del mundo y prácticamente el que tiene el monopolio del sector. Microsoft adquirió GitHub en 2018.

La enorme comunidad de usuarios aumenta las posibilidades de encontrar colaboradores para un proyecto propio, sobre todo, en el ámbito del código abierto. Además, la integración de repositorios de otros usuarios es más sencilla. Es la plataforma que tiene más desarrolladores trabajando en ella y actualizándola continuamente. Por ello, GitHub está considerada como plataforma más estable y potente.

Licencias e instalación en servidor propio.

Ambos programas cuentan con una versión gratuita y una versión **Enterprise** para empresas, que, a su vez, cuenta con numerosas opciones de suscripción con diferentes funciones.

En principio, ambas plataformas se pueden instalar en un servidor propio. Pero, en el caso de GitHub, para ello se requiere la versión Enterprise de pago para realizar. En cambio, GitLab también permite hospedar el programa en un servidor propio con la Community Edition gratuita.

La estabilidad del servidor de la variante hospedada de GitLab es ligeramente peor que la de GitHub, por lo que puede resultar muy ventajosa la instalación en un servidor propio.

GitHub no permite integración continua propia.

GitHub es compatible con muchos programas que facilitan el trabajo en equipo, por ejemplo, Docker, herramientas CI/CD o aplicaciones de gestión de proyectos. En el ámbito de la integración continua es imprescindible, ya que GitHub no ofrece herramientas de integración continua propias. Aquí, GitLab toma la delantera y ofrece integración continua gratuita de fábrica.

Más derechos de usuario en la versión gratuita de GitLab.

Durante mucho tiempo, la gran ventaja de GitLab era que ofrecía infinitos repositorios gratuitos a sus usuarios, pero GitHub vio su desventaja y ahora también ofrece esta característica. Aun así, GitHub sigue contando con más restricciones en su versión gratuita.

Ambos programas permiten el uso de ramificaciones protegidas, es decir, ramas de desarrollo a las que solo pueden acceder determinados usuarios, pero **GitHub solo ofrece esta posibilidad con repositorios públicos, mientras que GitLab también permite el uso de esta función con repositorios privados**. La limitación de como máximo tres desarrolladores por cada repositorio privado en GitHub es todavía más restrictiva. Sise **trabaja en equipos más grandes**, hay que contratar una suscripción y dar el salto a la versión Enterprise.

En general, **GitHub ofrece menos derechos de usuario**, **mientras que GitLab ofrece una gestión de derechos con varios roles** por defecto, GitHub exige la suscripción de equipo de pago para acceder a una función similar. Además, GitLab cuenta con un registro de contenedores, en el que los usuarios almacenan sus imágenes Docker creadas con las herramientas CI para luego gestionarlas como parte del repositorio GitLab.

Diferencias terminológicas entre las herramientas, tal y como demuestra la siguiente tabla:

GitHub	GitLab	Significado
Solicitud pull	Solicitud de combinación	Solicitud para integrar una ramificación en el maestro
Gist	Snippet	Trozo de código
Repositorio	Proyecto	Contenedor con el repositorio, los anexos y los ajustes específicos del proyecto
Organización	Grupo	Nivel en el que se asignan proyectos a usuarios

El término "**repositorio**" suele generar confusión a la hora de hacer el cambio, ya que muchos usuarios lo usan como sinónimo de "proyecto", aunque en el caso de GitHub incluye los repositorios Git y los activos del proyecto. Por ello, en GitLab se le llama "proyecto" a este contenedor, con el fin de indicar que incluye todos los datos de proyecto importantes.

Manejo e interfaz de usuario

A primera vista GitLab es más **ordenado y claro** gracias a su interfaz de usuario bien estructurada, razón por la que muchos usuarios afirman que su manejo es más sencillo e intuitivo. En GitLab, los elementos no solo se indican en lista, sino que también se pueden organizar y gestionar en una vista de escritorio.

Otra gran ventaja frente a GitHub es que la interfaz de usuario (UI) de GitLab es **escalable** y se puede adaptar al tamaño de la pantalla de forma flexible, mientras que GitHub solo ofrece un tamaño estándar fijo. Por ello, en caso de visualización en terminales móviles**,** GitLab suele ser la mejor elección como alternativa a GitHub.

La comparativa también revela que la edición y creación de códigos es un poco más sencilla en GitLab, ya que la herramienta cuenta con un **entorno de desarrollo integrado** (IDE). En cambio, GitHub solo cuenta con un editor de textos muy minimalista.

GitLab vs. GitHub principales diferencias:

GitHub	GitLab
Los elementos se pueden rastrear en varios repositorios	Los elementos no se pueden rastrear en varios repositorios
Los repositorios privados exigen la versión de pago	Los repositorios privados se permiten en la versión gratuita
No hay opción gratuita de hospedaje en servidor propio	Opción gratuita de hospedaje en servidor propio
Integración continua solo mediante herramientas de terceros como Travis CI, CircleCI etc.	Integración continua gratuita incluida
No cuenta con plataforma de implementación integrada	Implementación de software a través de Kubernetes
Rastreo completo de comentarios	Sin rastreo de comentarios
No hay opción de exportación de elementos como archivo CSV	Opción de exportación de elementos como archivo CSV por correo electrónico
Panel personal para rastrear elementos y solicitudes pull	Panel de análisis para planificar y supervisar proyectos

- **git config:** Uno de los comandos más usados en git es git config, que puede ser usado para establecer una configuración específica de usuario, como sería el caso del email, un algoritmo preferido para diff, nombre de usuario y tipo de formato, etc… Por ejemplo, el siguiente comando se usa para establecer un email: *git config --global user.email sam@google.com*

- **git init:** Este comando se usa para crear un nuevo repertorio GIT: *git init*

- **git add:** Este comando puede ser usado para agregar archivos al index. Por ejemplo, el siguiente comando agrega un nombre de archivo temp.txt en el directorio local del index: git add temp.txt

- **git clone:** Este comando se usa con el propósito de revisar repertorios. Si el repertorio está en un servidor remoto se tiene que usar el siguiente comando: git clone alex@93.188.160.58:/path/to/repository

Pero si estás por crear una copia local funcional del repertorio, usa el comando:

git clone /path/to/repository

- **git commit:** El comando commit es usado para cambiar a la cabecera. Ten en cuenta que cualquier cambio comprometido no afectara al repertorio remoto. Usa el comando:

  git commit –m "Message to go with the commit here"

- **git status:** Este comando muestra la lista de los archivos que se han cambiado junto con los archivos que están por ser añadidos o comprometidos: *git status*

- **git push:** Este es uno de los comandos más básicos. Un simple push envía los cambios que se han hecho en la rama principal de los repertorios remotos que están asociados con el directorio que está trabajando. Por ejemplo:
  git push origin master
- **git checkout:** El comando checkout se puede usar para crear ramas o cambiar entre ellas. Por ejemplo, el siguiente comando crea una nueva y se cambia a ella:

  command git checkout -b <banch-name>
  Para cambiar de una rama a otra solo usa:

  git checkout <branch-name>

- **git remote:**
  El comando git se usa para conectar a un repositorio remoto. El siguiente comando muestra los repositorios remotos que están configurados actualmente:
  git remote -v

Este comando te permite conectar al usuario con el repositorio local a un servidor remoto:

git remote add origin <93.188.160.58>

- **git branch**: Este comando se usa para listar, crear o borrar ramas. Para listar todas las ramas se usa: git branch Para borrar la rama:
  git branch -d <branch-name>
- **git pull:** Para poder fusionar todos los cambios que se han hecho en el repositorio local trabajando, el comando que se usa es: *git pull*
- **git merge:** Este comando se usa para fusionar una rama con otra rama activa:
  git merge <branch-name>
- **git diff:** Este comando se usa para hacer una lista de conflictos. Para poder ver conflictos con el archivo base usa: *git diff --base <file-name>*

  El siguiente comando se usa para ver los conflictos que hay entre ramas que están por ser fusionadas para poder fusionarlas sin problemas:

  git diff <source-branch> <target-branch>
  Para solo ver una lista de todos los conflictos presentes usa: *git diff*

- **git tag:** Etiquetar se usa para marcar commits específicos con asas simples. Por ejemplo: *git tag 1.1.0 <instert-commitID-here>*
- **git log:** Ejecutar este comando muestra una lista de commits en una rama junto con todos los detalles. Por ejemplo:
  commit 15f4b6c44b3c8344caasdac9e4be13246e21sadw

  Author: Alex Hunter <alexh@gmail.com>
  Date: Mon Oct 1 12:56:29 2016 -0600

- **git reset:** Para resetear el index y el directorio que está trabajando al último estado comprometido se usa este comando: *git reset - -hard HEAD*
- **git rm**: Este comando se puede usar para remover archivos del index y del directorio que está trabajando: *git rm filename.txt*
- **git stash:** Este es uno de los comandos menos conocidos, pero ayuda a salvar cambios que no están por ser comprometidos inmediatamente, pero temporalmente: *git stash*
- **git show:** Se usa para mostrar información sobre cualquier objeto git. Por ejemplo: *git show*
- **git fetch:** Este comando le permite al usuario buscar todos los objetos de un repositorio remoto que actualmente no reside en el directorio local que está trabajando. Por ejemplo: *git fetch origin*
- **git ls-tree:** Para ver un objeto de árbol junto con el nombre y modo de cada uno de ellos, y el valor blob´s SHA-1, se usa: *git ls-tree HEAD*
- **git cat-file:** Usando el valor SHA-1, se puede ver el tipo de objeto usando este comando. Por ejemplo:

  git cat-file –p d670460b4b4aece5915caf5c68d12f560a9fe3e4

- **git grep:** Este comando le permite al usuario buscar en los árboles de contenido cualquier frase o palabra. Por ejemplo, para buscar por www.tupaginaweb.com en todos los archivos se usaría: *git grep "www.tupaginaweb.com"*
- **gitk:** Este es la interfaz gráfica para un repositorio local que puede invocar escribiendo y ejecutando: *gitk*
- **git instaweb:** Con este comando un servidor web puede correr interconectado con el repositorio local. Un navegador web también está automáticamente dirigido a el: *git instaweb –http=webrick*
- **git gc:** Para optimizar el repositorio por medio de una recolección de basura, que limpiara archivos innecesarios y los optimizara, usa: *git hc*
- **git archive:** Este comando le permite al usuario crear archivos zip o tar que contengan los constituyentes de un solo árbol de repositorio:
  git archive – -format=tar master
- **git prune:** Con este comando los objetos que no tengan ningún puntero entrante serán eliminados: *git prune*
- **git fsck:** Para poder hacer un chequeo de integridad del sistema de archivos git, usa este comando. Cualquier objeto corrompido será detectado: *git fsck*
- **git rebase:** Este comando se usa para la re aplicación de los compromisos en otra rama. Por ejemplo: *git rebase master*

Ramas, tipos y usos

En Git las Ramas son espacios o entornos independientes para que un Desarrollador sea Back-end, Front-end, Tester, etc. pueda usar y así trabajar sobre un mismo Proyecto sin chancar o borrar el conjunto de archivos originales del proyecto, dándonos flexibilidad para desarrollar nuestro proyecto de manera mas organizada.

Master

Es la rama principal. Contiene el repositorio que se encuentra publicado en producción, por lo que debe estar siempre estable.

Development

Es una rama sacada de Master. Es la rama de integración, todas las nuevas funcionalidades se deben integrar en esta rama. Luego que se realice la integración y se corrijan los errores (en caso de haber alguno), es decir que la rama se encuentre estable, se puede hacer un merge de development sobre la rama Master.

Features

Cada nueva funcionalidad se debe realizar en una rama nueva, específica para esa funcionalidad. Estas se deben sacar de Development. Una vez que la funcionalidad esté desarrollada, se hace un merge de la rama sobre Development, donde se integrará con las demás funcionalidades.

Hotfix

Son errores de software que surgen en producción, por lo que se deben arreglar y publicar de forma urgente. Es por ello, que son ramas sacadas de Master. Una vez corregido el error, se debe hacer una unificación de la rama sobre Master. Al final, para que no quede desactualizada, se debe realizar la unificación de Master sobre Development.

Release

Las ramas de release apoyan la preparación de nuevas versiones de producción. Para ellos se arreglan muchos errores menores y se preparan adecuadamente los metadatos. Se suelen original de la rama develop y deben fusionarse en las ramas master y develop.

Resolución de conflictos

- Creamos una rama basada en otra
- Trabajamos en la nueva rama
- Guardamos los cambios
- Nos ubicamos en la rama que queremos unir los cambios
- Fusionamos los cambios

git checkout -b feature

Realizamos los cambios pertinentes

git commit -m “feature realizada”

git checkout master

git merge feature