TRABAJO FINAL PRIMER TRIMESTRE

ENTORNOS DE DESARROLLO

Realizado por: Jesús Rodríguez Burrero.

1. ¿qué es GIT? y qué debemos saber sobre GIT.

Git es un sistema de gestión de código fuente, inventado en el año 2005 por Linus Torvalds, creador también del sistema operativo Linux.

La principal característica de Git es que es un sistema distribuido, esto quiere decir que toda la historia de versiones del código fuente, no reside en un servidor, sino que lo tiene cada uno de los desarrolladores que utilizan dicho repositorio.

Un repositorio es el elemento base en Git, y representa la biblioteca donde se almacenan todas las versiones de los archivos de una aplicación o proyecto. Una app o una página web serian ejemplos de archivos que podrían ubicarse dentro de un repositorio, en caso de aplicaciones de mayor volumen estas se distribuyen en multiples repositorios como sería el caso del código fuente de Linux.

Git utiliza tres estados principales en los que se pueden encontrar los archivos, Commited, Modified y Staged. Commited significa que los datos están almacenados localmente, Modified significa que se ha modificado el archivo pero todavía no se ha confirmado en la base de datos y Staged significa que hemos preparado un archivo modificado para que este vaya en la próxima actualización.

Git también trabaja por ramas: Las ramas son utilizadas para desarrollar funcionalidades aisladas unas de otras. La rama master es la rama principal o "por defecto" cuando creas un repositorio.

Lo que pasa cuando creamos una rama es que Git toma el estado en el que se encuentra el archivo y hace una derivación a otro espacio, de forma que se puede trabajar en él sin afectar al archivo inicial. Más adelante podemos juntarlos.

Las ramas nos pueden servir para muchos casos de uso y son muy útiles cuando se trabaja con equipos de varias personas, ya que, de esta manera no pisaremos el trabajo de nuestros compañeros. Un ejemplo sería la creación de una funcionalidad que queramos integrar en un programa y para la cual no queremos que la rama principal se vea afectada. Esta función experimental se puede realizar en una rama independiente, de modo que, aunque tardemos varios días o semanas en terminarla, no afecte a la producción del código que tenemos en la rama principal y que permanecerá estable.

Si en un momento dado el trabajo con una rama nos ha resultado interesante, útil y se encuentra estable, entonces podemos fusionar ramas para incorporar las modificaciones del proyecto en su rama principal. Lo más probable es que surjan conflictos.

2. Github, Gitlab y GIT. Definiciones y diferencias.

Tanto Github como Gitlab son plataformas creadas para facilitar el desarrollo colaborativo de software, basados en GIT. Estos nos permiten alojar proyectos en la web gratuitamente de forma pública, aunque podemos alojar proyectos también en modo privado, gratis para Gitlab y con una pequeña suscripción en el caso de Github.

Las diferencias más destacables podríamos decir que es el enorme número de usuarios en Github, que seguramente sea el sistema de control de versiones más conocido del mundo, esto a la hora de trabajar puede ser una gran ventaja ya que la enorme comunidad de usuarios permite la posibilidad de encontrar colaboradores para un proyecto propio, por ello Github está considerada como la plataforma más estable y potente.

Tanto Github como Gitlab cuentan con una versión gratuita y otra Enterprise para empresas, En principio ambas se pueden instalar en un servidor propio, pero en el caso de Github se requiere la versión de pago para esto.

Github en diferencia con Gitlab no ofrece herramientas de integración continua propias en este punto Gitlab tomaría la delantera, 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.

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.

TABLA DE LAS 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

3. Comandos existentes y para qué sirven.

→git config

Uno de los comandos más usados en git es git config, que puede ser usado para establecer una configuración específica de usuario.

git config --global user.email sam@google.com

→git init

Se usa para crear un nuevo repositorio en git.

```
git init nombrerepo
```

→git add

Manda los archivos a la stage y se quedan pendientes para subir.

```
git add temp.txt
```

→git clone

Nos clona un repositorio del servidor a nuestro entorno local.

```
git clone https://github.com/jesuly94/entornoydesarrollo.git
```

→git commit

Es el mensaje que pondremos a la hora de haber subido los cambios para el repositorio.

```
git commit –m “Message to go with the commit here”
```

→git status

Nos muestra el estado de los ficheros.

```
git status
```

→git push

Es uno de los comandos más básicos, este envía los cambios locales en la rama principal o en la que estemos a los repositorios remotos.

```
git push  origin master
```

→git checkout

Se usa para crear ramas y cambiar entre ellas.

```
git checkout -b <banch-name>
```

Y para cambiar de rama: `git checkout <branch-name>`

→git branch

Se usa para ver todas las ramas o para borrarlas.

`git branch`

Y para borrar: `git branch -d <branch-name>`

→git pull

Se usa para descargar los cambios más reciente del servidor.

`git pull`

→git merge

Sirve para fusionar una rama con otra rama activa.

`git merge <branch-name>`

→git diff

Se usa para ver los últimos cambios realizados.

`git diff`

→git tag

Sirve para marcar commits específicos.

`git tag 1.1.0 <instert-commitID-here>`

→git log

Este comando muestra una lista de commits en una rama junto con todos los detalles.

`git log` o `git log –pretty=oneline` (este último los muestra en una línea).

→git reset

Elimina los cambios que no lleven commit.

`git reset - -hard HEAD`

→git fetch

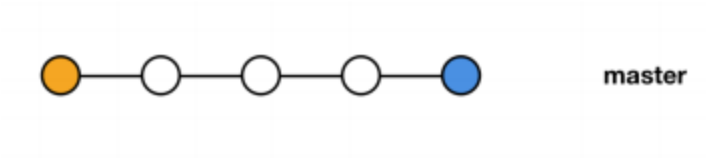
Este comando le permite al usuario buscar todos los objetos de un repositorio remoto que actualmente no reside en el directorio local que está trabajando.

`git fetch origin`

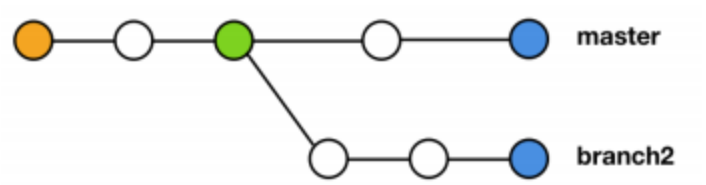
4. Ramas, tipos y usos.

En esta parte he cogido imágenes para que así quede más claro y se entienda mejor.

Según vas trabajando en un proyecto y creando más y más commit, vas construyendo un camino temporal, que se denomina rama.



Sin embargo, en cualquier punto o commit, puedes necesitar generar un camino alternativo o bifurcación, momento a partir del cual tendrás dos ramas para el mismo proyecto.



Estas bifurcaciones o forks se producen cuando vamos a cambiar bastantes cosas del proyecto, y asi se crea un entorno aislado para que no impacte al resto de desarrolladores que seguirán trabajando en la rama principal.

Cuando has acabado el desarrollo en tu rama creada y sabes que funciona correctamente se fusiona con la rama principal o master.

Las ramas, al igual que los commit, puedes dejarlas en tu repositorio local o publicarlas en el remoto para que otros puedan colaborar con sus aportaciones.

Es importante saber que cuando se genera una rama, los commit que se han a partir de entonces iran a esa rama creada, cuando trabajamos con las ramas se utilizan tres comandos principalmente:

- Git branch
- Git checkout
- Git merge

Estructura de ramas en Git:

- master: es la rama de producción de nuestro proyecto, sobre la cual nunca se deben acometer cambios directamente por ningún motivo, y que en todo momento debe ser completamente estable.

- develop: es la rama dedicada al desarrollo e integración de nuevas funcionalidades de nuestro proyecto. Cuando una nueva versión del proyecto y sus nuevas funcionalidades sean completamente estables, se fusionarán todos los cambios con la rama master.

- features: por cada nueva funcionalidad, se creará una rama de tipo feature, teniendo como origen la rama develop, que una vez estable será fusionada de nuevo con la rama develop.

- hotfix: por cada corrección o incidencia a resolver en nuestro proyecto, crearemos una rama de tipo hotfix, teniendo como origen la rama master, que una vez estable será fusionada de nuevo con la rama master.

5. Resolución de conflictos.

Cuando dos personas están trabajando en un mismo archivo, puede ocurrir, que se realicen commit por parte de ambas personas, y cuando vayas a hacer el push o pull no pueda ser automática la promoción o descarga de dicho commit. Cuando esto ocurre, tiene que haber una intervención humana, para indicarle a Git, qué cambios de cada uno de los dos commit quieres conservar y cómo, para a continuación generar un tercer commit fruto de la fusión de los cambios de los dos anteriores.

Básicamente consiste en fusionar commit para indicarle como se quiere dejar el archivo, como he explicado antes, dos personas trabajan en el mismo archivo, estas lo suben y aparece el conflicto, por lo cual o se fusiona el contenido y se pone en el tercer commit que realizamos, de ahí la intervención humana ya que nosotros veremos cómo queremos fusionarlo , una vez hecho los repositorios quedarían alineados y no nos daría error.