**Ejercicio 7 – Jesús Cobos Labrada**

**Curso: 1º DAW**

**Objetivo:** Realizar un documento PDF que refleje mediante texto, esquemas e imágenes toda la información sobre git que conozcas (al menos 6 páginas).

1. **¿Qué es Git y debemos saber sobre él?**

Git es una herramienta que realiza una función del control de versiones de código de forma distribuida, de la que destacamos varias características: fue diseñada por Linus Torvalds; no depende de un repositorio central, es software libre, y con ella podemos mantener un historial completo de versiones, e incluso podemos movernos, como si tuviéramos un puntero en el tiempo, por todas las revisiones de código y desplazarnos una manera muy ágil; tiene un sistema de trabajo con ramas que lo hace especialmente potente, y en cuanto a la funcionalidad de las ramas, las mismas están destinadas a provocar proyectos divergentes de un proyecto principal, para hacer experimentos o para probar nuevas funcionalidades; las ramas pueden tener una línea de progreso diferente de la rama principal donde está el core de nuestro desarrollo. En algún momento podemos llegar a probar algunas de esas mejoras o cambios en el código y hacer una fusión a nuestro proyecto principal, ya que todo esto lo maneja Git de una forma muy eficiente.

1. **Definiciones y Diferencias**

· GitHub: Es un sistema de gestión de proyectos y control de versiones de código, así como una plataforma de red social diseñada para desarrolladores. ¿Pero para qué se usa GitHub? Bueno, en general, permite trabajar en colaboración con otras personas de todo el mundo, planificar proyectos y realizar un seguimiento del trabajo.

GitHub es también uno de los repositorios online más grandes de trabajo colaborativo en todo el mundo.

· GitLab: Es una herramienta basada en Git, que usas de la misma manera que cualquier otra herramienta similar. Generalmente usas Git a través de la línea de comandos, o a través de programas de interfaz gráfica, o del propio editor de código. Toda esa operativa que ya conoces y que hemos explicado en el Manual de Git, no cambia.

Además del hosting remoto para repositorios GitLab ofrece una interfaz web para controlar el repositorio y muchas otras herramientas. Ofrece la posibilidad de examinar el código en cualquiera de sus versiones, realizar acciones relacionadas con el sistema de repositorios como mergear el código de versiones de proyecto o gestionar las "pull request" (que en GitLab se llaman "merge request"), gestionar problemática de tu software diversa, automatizar procesos como el despliegue o la ejecución de pruebas del software, etc. Toda esta operativa la realizas, o configuras, en GitLab por medio de una web.

Por tanto, para usar GitLab simplemente necesitas las mismas herramientas que ya utilizas en tu día a día, el terminal o un programa de interfaz gráfica para gestionar tu repositorio, así como el navegador web para acceder a el ecosistema de herramientas disponible en el sitio de GitLab. Por supuesto, todas estas herramientas las puedes usar desde cualquier ordenador conectado a Internet, independientemente de su sistema operativo.

· Git: Es un sistema de control de versiones desarrollado por Linus Torvalds (el hombre que creó Linux). Y un sistema de control de versiones es cuando los desarrolladores hacen un nuevo proyecto, siempre continúan haciéndole modificaciones al código. Incluso después de la puesta en marcha de los proyectos, todavía necesitan actualizar las versiones, corregir errores, agregar nuevas funciones, etc.

El sistema de control de versiones ayuda a registrar los cambios realizados al código. Aún más, registra quién realizó los cambios y puede restaurar el código borrado o modificado.

· Diferecias: Apenas les veo diferencias, porque GitLab surgió de GitHub y ambos surgieron de Git. Pero 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. No es ninguna casualidad que Microsoft adquiriera GitHub en 2018.

Está claro que tanto el tamaño como la posición que GitHub ocupa en el mercado también se reflejan a nivel práctico. 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 tiene más desarrolladores trabajando en ella y actualizándola continuamente. Por ello, GitHub está considerada como plataforma más estable y potente.

1. **Comandos existentes y para qué sirven:**

-Configurar Nombre que salen en los commits:

git config --global user.name "omar"

-Configurar Email:

git config --global user.email omar69@gmail.com

-Marco de colores para los comando:

git config --global color.ui true

-Iniciamos GIT en la carpeta donde está el proyecto:

git init

-Clonamos el repositorio de github o bitbucket:

git clone <url>

-Añadimos todos los archivos para el commit:

git add .

-Hacemos el primer commit:

git commit -m "Texto que identifique por que se hizo el commit"

-Subimos al repositorio:

git push origin master

-Clonamos el repositorio de github o bitbucket:

git clone <url>

-Añadimos todos los archivos para el commit:

git add .

-Añadimos el archivo para el commit:

git add <archivo>

-Añadimos todos los archivos para el commit omitiendo los nuevos:

git add --all

-Añadimos todos los archivos con la extensión especificada:

git add \*.txt

-Añadimos todos los archivos dentro de un directorio y de una extensión específica:

git add docs/\*.txt

-Añadimos todos los archivos dentro de un directorios

git add docs/

-Cargar en el HEAD los cambios realizados

git commit -m "Texto que identifique por que se hizo el commit"

-Agregar y Cargar en el HEAD los cambios realizados

git commit -a -m "Texto que identifique por que se hizo el commit"

-De haber conflictos los muestra

git commit -a

-Agregar al último commit, este no se muestra como un nuevo commit en los logs. Se puede especificar un nuevo mensaje

git commit --amend -m "Texto que identifique por que se hizo el commit"

-Subimos al repositorio

git push <origien> <branch>

-Subimos un tag

git push --tags

-Muestra los logs de los commits

git log

-Muestras los cambios en los commits

git log --oneline --stat

-Muestra los cambios realizados a un archivo

git diff

-Agregar repositorio remoto

git remote add origin <url>

-Cambiar de remote

git remote set-url origin <url>

-Remover repositorio

git remote rm <name/origin>

-Muestra lista repositorios

git remote -v

-Muestra los branches remotos

git remote show origin

-Crea un branch

git branch <nameBranch>

-Lista los branches

git branch

-Comando -d elimina el branch y lo une al master

git branch -d <nameBranch>

-Elimina sin preguntar

git branch -D <nameBranch>

-Muestra una lista de todos los tags

git tag

-Crea un nuevo tags

git tag -a <verison> - m "esta es la versión x"

-Lista un estado actual del repositorio con lista de archivos modificados o agregados

git status

-Quita del HEAD un archivo y le pone el estado de no trabajado

git checkout -- <file>

-Crea un branch en base a uno online

git checkout -b newlocalbranchname origin/branch-name

-Busca los cambios nuevos y actualiza el repositorio

git pull origin <nameBranch>

-

Cambiar de branch

git checkout <nameBranch/tagname>

-

Une el branch actual con el especificado

git merge <nameBranch>

-Verifica cambios en el repositorio online con el local

git fetch

-Borrar un archivo del repositorio

git rm <archivo>

-Descargar remote de un fork

git remote add upstream <url>

-Merge con master de un fork

git fetch upstream

git merge upstream/master

-Saca un archivo del commit

git reset HEAD <archivo>

-Devuelve el último commit que se hizo y pone los cambios en staging

git reset --soft HEAD^

-Devuelve el último commit y todos los cambios

git reset --hard HEAD^

-Devuelve los 2 último commit y todos los cambios

git reset --hard HEAD^^

-Rollback merge/commit

git log

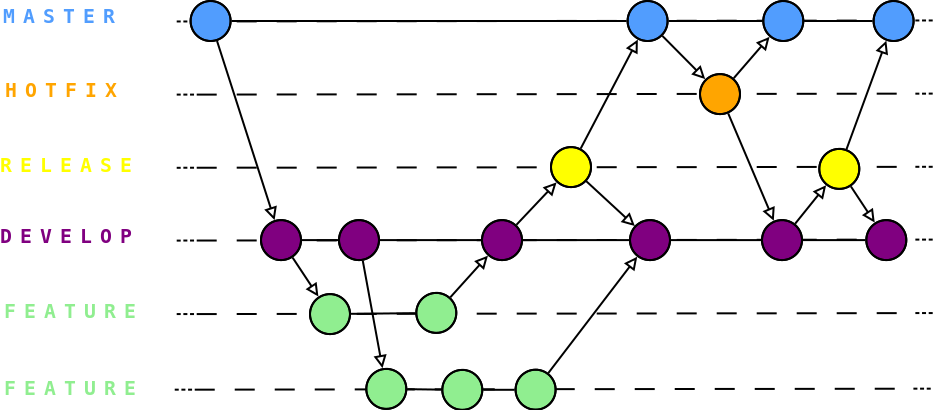
git reset --hard <commit\_sha>

1. **Ramas, tipos y usos:**

Branch, que se traduce como rama, es una copia de tu repositorio. Puedes utilizar la rama cuando quieras hacer un desarrollo de forma aislada.

Trabajar en una rama no afectará el repositorio central u otras ramas. Si has completado el trabajo, puedes combinar tu rama con otras ramas y con el repositorio central mediante una pull request.

Y existen varios tipos de ramas, las cuales son las siguientes:



Y cada una de ella, complementan el proceso que se le asignan para actualizar el código de la rama principal “master”, sin que se carguen la versión. Extraen el código de la rama master, en el desarrollo, luego entre varios usuarios con ramas propias, se dedican a actualizar el código, y luego de ello, se vuelve a la rama develop, para subir la alpha del programa a relase, y luego a master, pero como estas versiones suelen ser inestables, se suelen meter en hotfix, para reparar fallos medios/graves, y luego se manda a desarrollo para solidificar la versión, y luego se manda a master. Este es el ciclo que sufre cada programa. Esto nos ayuda a poder desarrollar una nueva versión conjunto varios usuarios, y sin destruir la rama principal “master”.

1. **Resolución de conflictos:**

Si bien cubrimos algunos conceptos básicos para resolver conflictos de fusión en Principales Conflictos que Pueden Surgir en las Fusiones, para conflictos más complejos, Git proporciona algunas herramientas para ayudarlo a descubrir qué está sucediendo y cómo lidiar mejor con el conflicto.

En primer lugar, si es posible, intente asegurarse de que su directorio de trabajo esté limpio antes de realizar una fusión que pueda tener conflictos. Si tiene un trabajo en progreso, hágale commit a una rama temporal o stash.