APUNTES DE GIT

- **QUE ES GIT?**

Git es un sistema de control de versiones distribuido, diseñado por Linus Torvalds. Está pensando en la eficiencia y la confiabilidad del mantenimiento de versiones de aplicaciones cuando estas tienen un gran número de archivos de código fuente. Git está optimizado para guardar todos estos cambios de forma atómica e incremental.

- **QUE ES UN SISTEMA DE CONTROL DE VERSIONES**

Es un sistema que registra los cambios realizados sobre un archivo o conjunto de archivos a lo largo del tiempo, de modo que puedas llevar el historial del ciclo de vida de un proyecto. Cualquier tipo de archivo que se encuentre en un ordenador puede ponerse bajo control de versiones. Solo se lleva un registro de los cambios de los archivos (cuando y donde ocurrieron y quien los hizo), sin la necesidad de guardar todo el archivo completo.

- **QUE ES GITHUB?**

Es una plataforma de desarrollo colaborativo para alojar proyectos utilizando el sistema de control de versiones Git. Se emplea principalmente para la creación de código fuente de programas de computadora. Github puede considerarse como la red social de código para los programadores y en muchos casos es visto como tu curriculum vitae, pues aquí guardas tu portafolio de proyectos de programación.

- **INSTALACION**

-Asegurarse que este seleccionado GIT BASH

-Luego podemos editar el editor texto visual de código

-Elegir como llamar la rama inicial de un repositorio

-Podemos decidir donde usar GIT. Si solo queremos desde el git BASH o

bien desde la línea de comandos nativa de windows o similares.

-Seleccionar el sistema de seguridad, por medio de openSSH

-Windows y linux guardan el enter de manera distinta. Pregunta, quieres

que nos encarguemos nosotros, te encargas tu o se lo dejamos a Dios

--1- Los saltos de lineas se los acepta en windows y cuando se lo envían

---- al repositorio se lo cambia a linux/UNIX. GENERA COMPATIBILIDAD

--2- Se hace como este indicado en el sistema operativo

--3- No se hacen conversiones, puede que genere inconvenientes.

-Quieres usar CMD o bien el emulador linux en windows MinTTY, para usar comando linuxs

-Nos pide seleccionar el comportamiento por defecto de un PULL

-Cómo vamos a gestionar las credenciales llaves dentro de windows, recomendado una librería proporcionar una autenticación HTTPS de múltiples factores sin problemas con Git

- Podemos habilitar el enlaces simbólicos de linux en windows.

File sistema caching hacer que todo corra más rápidos por se guardar en cache del sistema

- **COMANDOS LINUX** (estructura de archivos en windows y linux son distintas en windows mayuscula y minuscula son indistintas, pero en linux/max no)

*pwd: Nos muestra la ruta de carpetas en la que te encuentras ahora mismo.

*ls: Nos permite cambiar ver los archivos de la carpeta donde estamos ahora mismo. Podemos usar uno o más argumentos para ver más información sobre estos archivos (los argumentos pueden ser -- + el nombre del argumento o - + una sola letra o shortcut por cada argumento).

- ls -a: Mostrar todos los archivos, incluso los ocultos.

- ls -l: Ver todos los archivos como una lista.

*cd: Nos permite navegar entre carpetas.

- cd /: Ir a la ruta principal:

- cd o cd ~: Ir a la ruta de tu usuario

- cd carpeta/subcarpeta: Navegar a una ruta dentro de la carpeta donde estamos ahora mismo.

- cd .. (cd + dos puntos): Regresar una carpeta hacia atrás.

*mkdir: Nos permite crear carpetas (por ejemplo, mkdir Carpeta-Importante).

*touch: Nos permite crear archivos (por ejemplo, touch archivo.txt).

*rm: Nos permite borrar un archivo o carpeta (por ejemplo, rm archivo.txt). Mucho cuidado con este comando, puedes borrar todo tu disco duro.

*cat: Ver el contenido de un archivo (por ejemplo, cat nombre-archivo.txt).

- Si quieres referirte al directorio en el que te encuentras ahora mismo puedes usar cd . (cd + un punto).

*history: Ver los últimos comandos que ejecutamos y un número especial con el que podemos repetir su ejecución.

! + número: Ejecutar algún comando con el número que nos muestra el comando history (por ejemplo, !72).

clear: Para limpiar la terminal. También podemos usar los atajos de teclado Ctrl + L o Command + L.

- **COMANDO BASICOS GIT**

Creamos un proyecto

git init - inicialización el proyecto

git status - para ver el estado del repositorio

git add (arhivo) - Agregamos archivos al área de staging

git commit -m - Confirmamos los cambios con un mensaje y se agrega al repositorio local -OJO ANTES HAY QUE CONFIGURARLO

git status - Nos permite ver el estado general del repositorio

git rm -- cached - Para eliminar el archivo que agregue de staging. git ya no lee

GIT CONFIG - Nos permite configurar algunas cosas de git como ser el user y email

git config --list - con este comando vemos todas las configuraciones de git

git config --global user.email 'email'

git config --global user.name 'Nombre'

git log archivo - Indica los cambios que sufrió el archivo. El numero largo es el nombre que le pone git a la modificación que hicimos

git log --stat - cambios específicos en que archivos en el commit que hicimos

git show archivo - Nos muestra el ultimo commit HEAD- el mensaje del commit y luego una diferencia entre la ultima y la penúltima versión. Hay un índice dentro de la base de datos de git de donde están los cambios, también hay inflo de las versiones que existen y los bytes que cambiaron.

git diff idCommit1 idCommit2 - nos muestra las diferencias entre dos versiones

- **VOLVER EN EL TIEMPO CON GIT**

GIT RESET IDCOMMIT --hard - nos permite volver a una versión anterior, todo vuelve

al estado anterior, es el más peligroso pero que más se usa. Volvemos de una manera agresiva

GIT RESET IDCOMMIT --soft - volvemos a la versión anterior pero si tenemos cosas en staging eso queda ahi para el próximo commit. hacer GIT LOG para ver las diferencias. Hago un cambio subo con add y luego el reset soft y luego commit

GIT RESET HEAD ARCHIVO - Si queremos sacar un archivo de staging para poder hacer commit porque lo agregamos por accidente.

Recordar que el comando **RESET PISA AL REPOSITORIO**

GIT CHECKOUT IDCOMMIT (o bien nombre de rama) **ARCHIVO** nos permite viajar en el tiempo y volver a la versión anterior de un archivo especifico o el proyecto entero. También es la

forma de movernos entre ramas.

CHECKOUT NO PISA AL REPOSITORIO solo el contenido.

- **RAMAS EN GIT**

Cuando hacemos una rama lo que hacemos es hacernos una copia del HEAD de master y creamos una nueva línea de trabajo. Los cambios que hagamos en la rama no serán vistos por la rama master hasta que no la volvamos a fusionar con un merge.

GIT BRANCH - listamos ramas del repositorio

GIT BRANCH Nombre-rama - creamos rama

GIT CHECKOUT NOMBRE RAMA - cambiamos la rama

Para hacer el merge vamos a hacer modificaciones en develop, luego nos vemos a master y hacemos unas modificaciones ahí también. Por lo que si yo quiero traer a master lo que hay en develop lo que tengo que hacer es posicionarme sobre la rama MASTER y hacer un merge

GIT MERGE nombre rama - traemos el contenido de develop y hacemos una fusión

automáticamente hace un commit.