SISTEMA DE CONTROL DE VERSIONES (SCV)

Por Rober Gutiérrez y Pablo Chico - Git y GitHub

30 DE ABRIL DE 2019

C.F.G.S DAM

ENTORNOS DE DESARROLLO

INDICE

PARTE 1

PARTE 2

1. ¿Qué es un control de versiones y para qué sirve?

Un control de versiones es un sistema que registra los cambios realizados sobre un archivo o conjunto de archivos a lo largo del tiempo de tal manera que sea posible recuperar versiones especificas más adelante.

2. ¿Qué es Git y Github?

Git es un software de control de versiones diseñado por Linus Torvalds, pensando en la eficiencia y la confiabilidad del mantenimiento de versiones de aplicaciones cuando éstas tienen un gran número de archivos de código fuente. Su propósito es llevar registro de los cambios en archivos de ordenador y coordinar el trabajo que varias personas realizan sobre archivos compartidos.

Github es una plataforma de desarrollo colaborativo para alojar proyectos utilizando el sistema de control de versiones Git. Se utiliza principalmente para la creación de código fuente de programas de ordenador.

3. ¿Qué otros sistemas de control de versiones se suelen utilizar en la actualidad?

Mercurial, Subversion, Perforce, Bazaar, etc.

Unos modelan la información que almacenan como un conjunto de archivos y las modificaciones hechas sobre cada uno de ellos a lo largo del tiempo, otros tienden a almacenar los datos como cambios de cada archivo respecto a una versión base.

Git no modela ni almacena sus datos de este modo, sino que modela sus datos más como un conjunto de instantáneas de un mini sistema de archivos. Cada vez que confirmas un cambio, o guardas el estado de tu proyecto en Git, él básicamente hace una foto del aspecto de todos tus archivos en ese momento, y guarda una referencia a esa instantánea. Para ser eficiente, si los archivos no se han modificado, Git no almacena el archivo de nuevo, sólo un enlace al archivo anterior idéntico que ya tiene almacenado.

4. ¿Qué son los sistemas centralizados y distribuidos?

Los sistemas de control de versiones se pueden clasificar en 2 grandes grupos:

Centralizados

En un sistema de control de versiones centralizado todas nuestras fuentes y sus versiones están almacenados en un único directorio (llamado repositorio de fuentes) de un ordenador (un servidor). Todos los desarrolladores que quieran trabajar con esas fuentes, deben pedirle al sistema de control de versiones una copia local para trabajar. En ella realizan todos sus cambios y cuando están listos y funcionando, le dicen al sistema de control de versiones que guarde las fuentes modificadas como una nueva versión. Algunos ejemplos son CVS y subversión.

Distribuidos

En un sistema de control de versiones distribuido no hay un repositorio central. Todos los desarrolladores tienen su propia copia del repositorio, con todas las versiones y toda la historia. Por supuesto, según van desarrollando y haciendo cambios, sus fuentes y versiones van siendo distintas unas de otras. Sin embargo, los sistemas de control de versiones distribuidos permiten que en cualquier momento dos desarrolladores cualesquiera puedan "sincronizar" sus repositorios. Si uno de los desarrolladores ha tocado determinadas fuentes y el otro no, los modificados se convierten en la versión más moderna. Ejemplos: Git y Mercurial.

5. Definiciones sobre el SCV Git

- *Repositorio*: lugar en el que se almacenan los datos actualizados e históricos de cambios, a menudo en un servidor. A veces se le denomina depósito o depot. Puede ser un sistema de archivos en un disco duro, un banco de datos, etc.

- *Rama*: representa una línea independiente de desarrollo, es como crear una nueva área de trabajo con su área de ensayo y su historial.

- *Master*: es la rama por defecto de Git.

- Comando add: agrega al repositorio los archivos que indiquemos.

- *Comando commit:* confirma los cambios realizados.

- *Head:* es una referencia a la rama donde te encuentras en cada momento.

- *Comando pull:* Unifica los comandos fetch y merge en un único comando. Fetch significa descargar los cambios realizados en el repositorio remoto y merge significa impactar en la rama en la que te encuentras parado, los cambios realizados en la rama “nombre_rama”.

- *Comando push:* sube la rama “nombre_que_queramos” al servidor remoto.

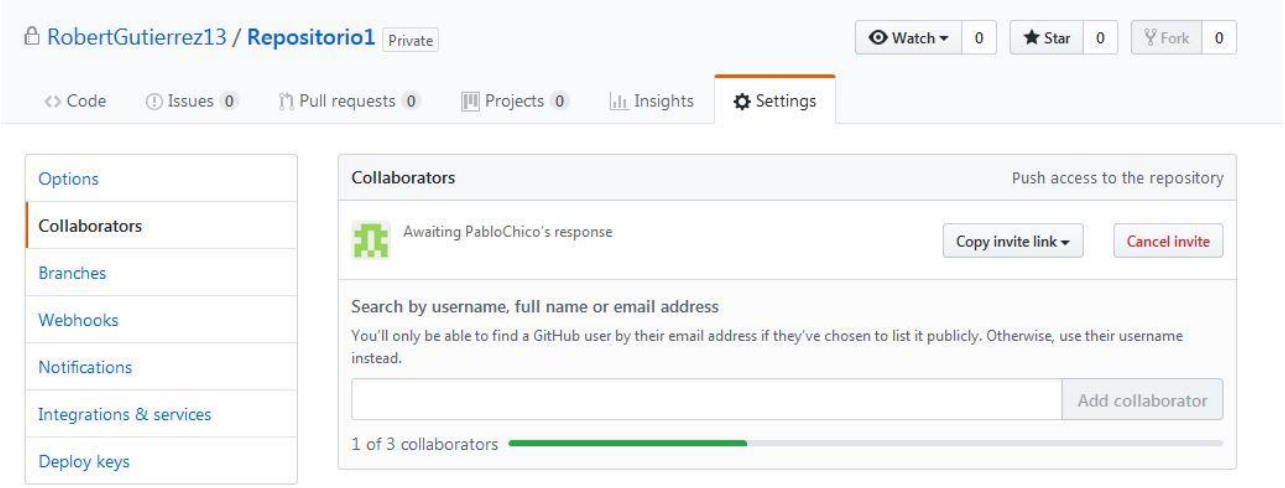
- *Comando clone:* clona un proyecto de Git en la carpeta NombreProyecto.

- *Comando status:* muestra el estado actual de la rama, como los cambios que hay sin hacer commit.

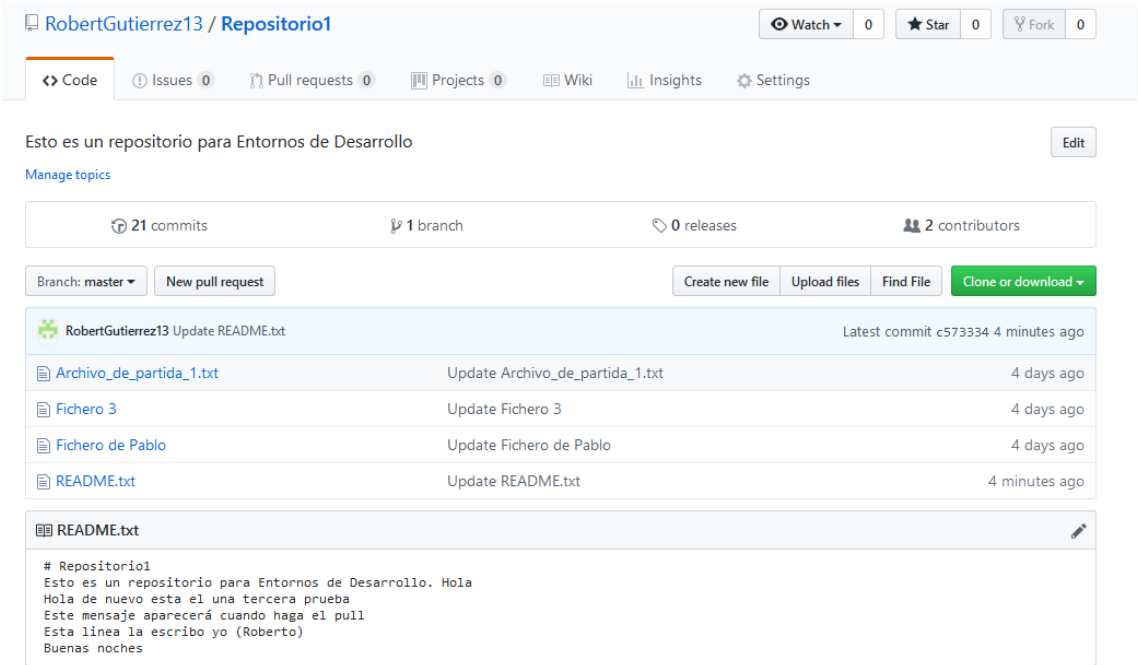
- *Fichero Git:* es un sistema de control de versiones, una herramienta para administrar el historial del código fuente, diferenciándose así de GitHub que es un servicio de alojamiento para repositorios Git, es decir que Git es la herramienta y GitHub es el servicio para proyectos que usan Git.

6. Creación de una cuenta en GitHub y un repositorio

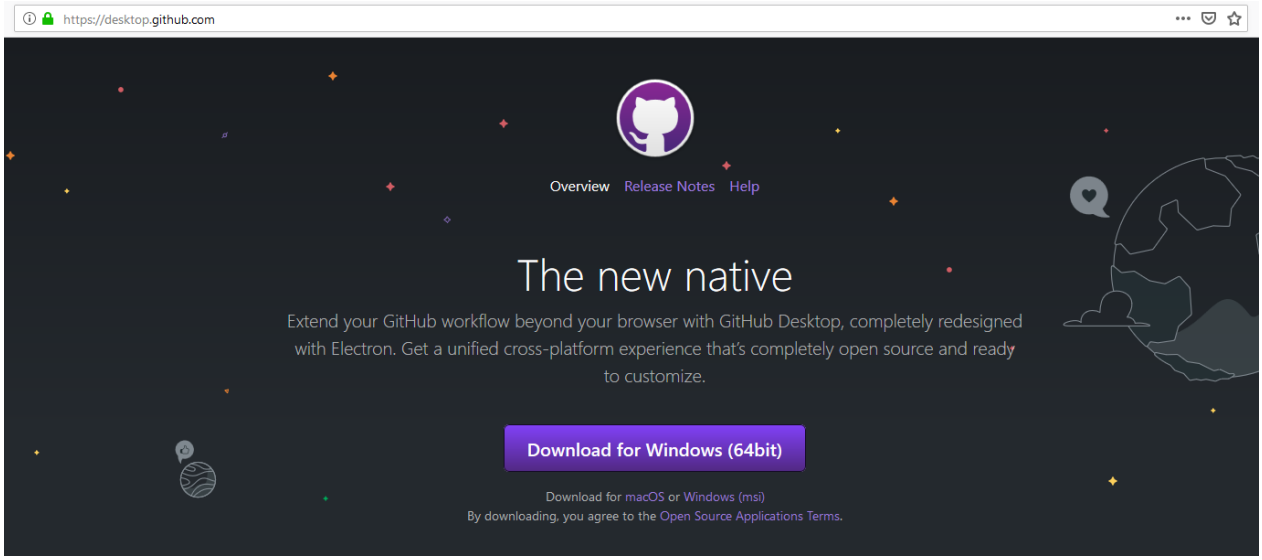
- Lo primero que hacemos es dirigirnos a la página https://github.com donde creamos una cuenta con nuestro correo electrónico y una contraseña, después confirmamos desde nuestro correo y ya podemos entrar en la página para crear un repositorio.
- Después de crear el repositorio añado un colaborador, que también tiene que crear una cuenta en GitHub y confirmarla.



- Hemos creado varios archivos donde hemos ido modificando por separado

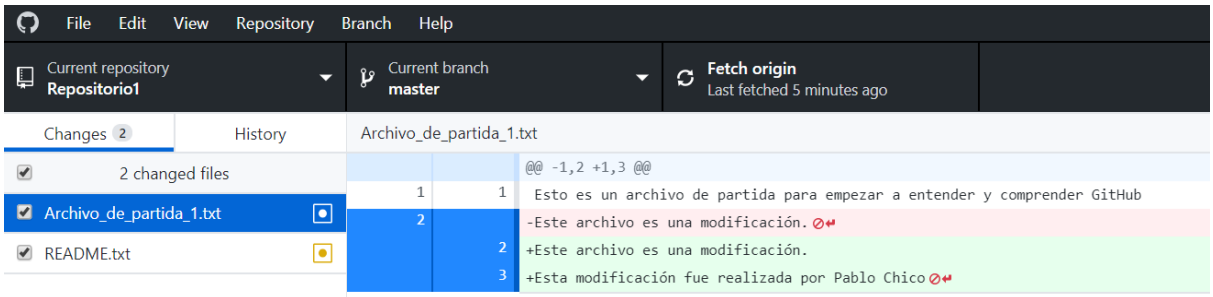


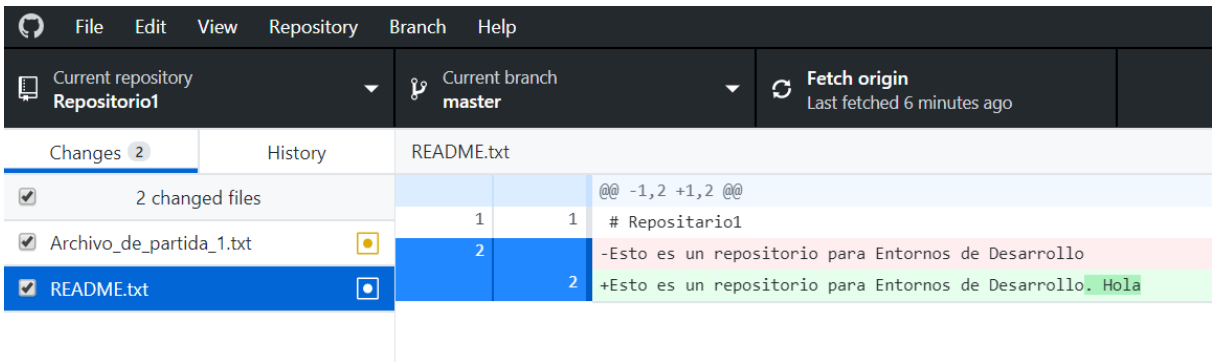
- El siguiente paso es descargar la aplicación GitHub en nuestros ordenadores para lo que nos dirigimos a la siguiente página https://desktop.github.com donde pinchamos en el botón Download for Windows (64 bit) y una vez descargado procedemos a su instalación.

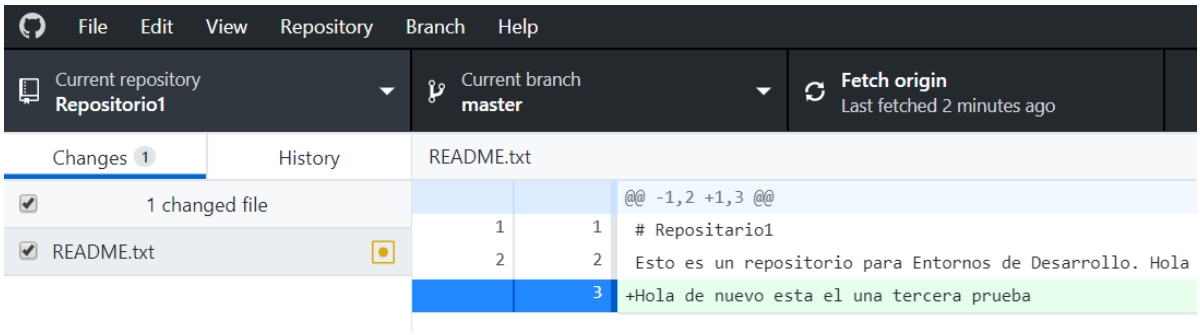


7. Practica con ficheros desde ordenadores diferentes y usando el mismo repositorio

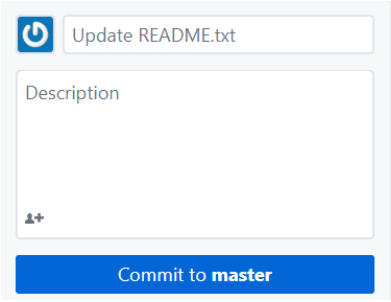
- Ahora vamos con la práctica, a continuación, las siguientes capturas de modificación de ficheros en tiempo real.



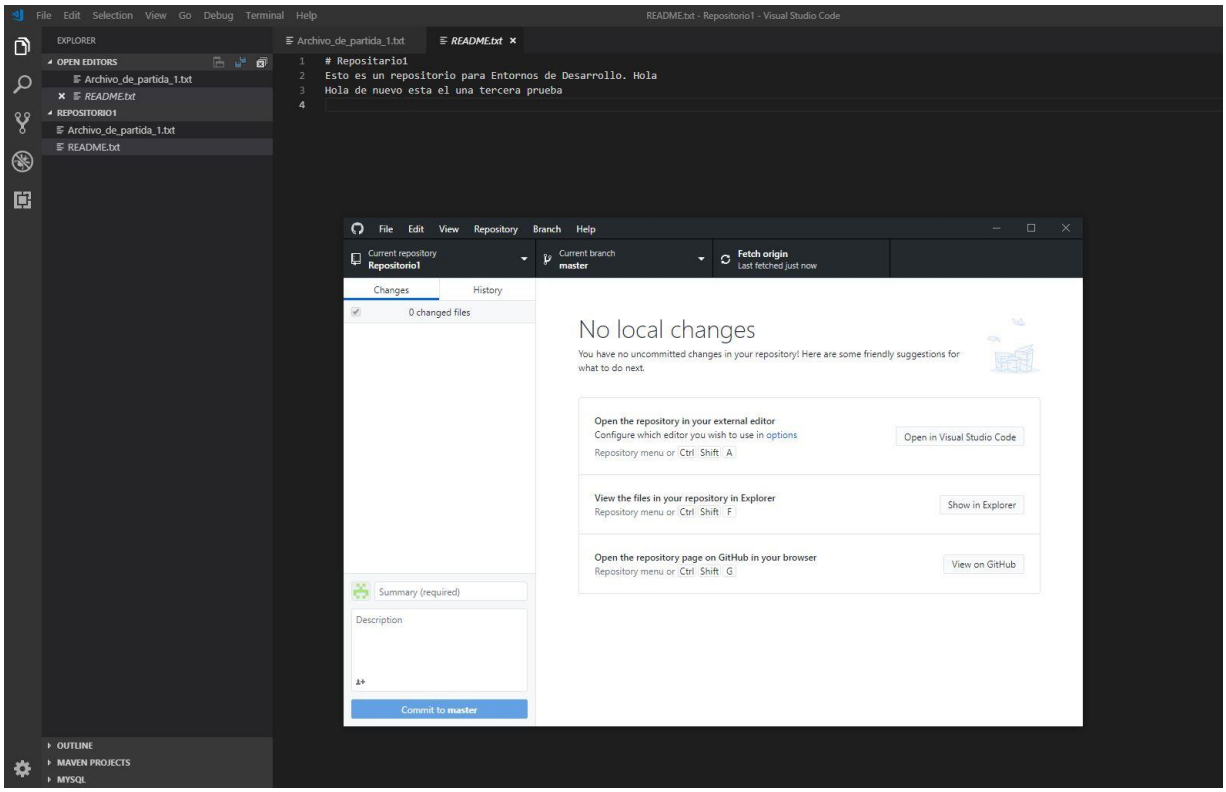




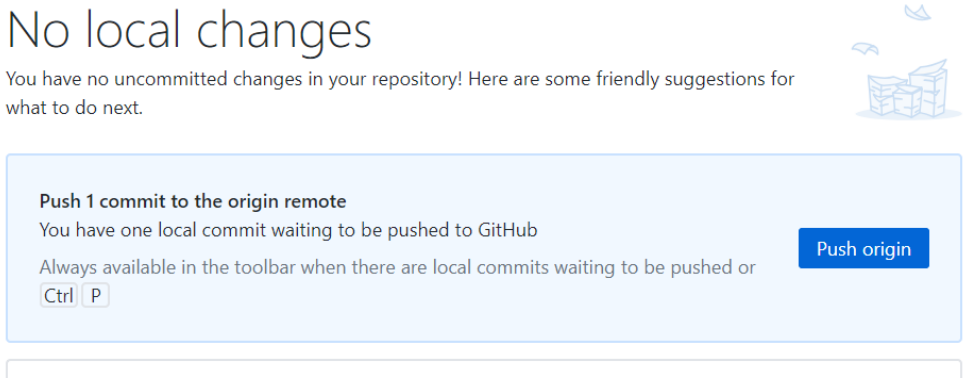
- A continuación, pinchamos en Commit to master para subir los cambios a GitHub



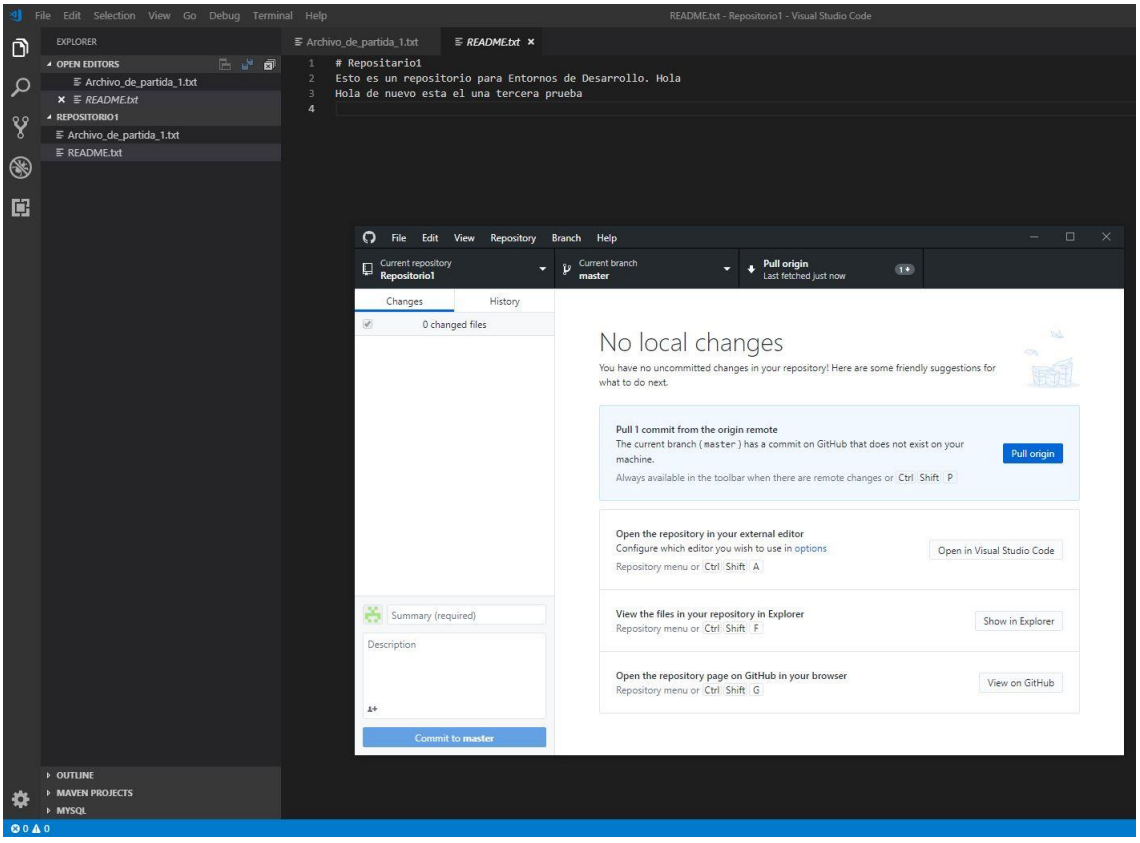
- Mientras tanto en el ordenador de Roberto estamos en Visual Studio Code que tenemos instalado previamente y al que hemos accedido desde la aplicación GitHub, encontrándonos ya en nuestro repositorio local después de haberlo clonado, así podemos trabajar en tiempo real sin problema.



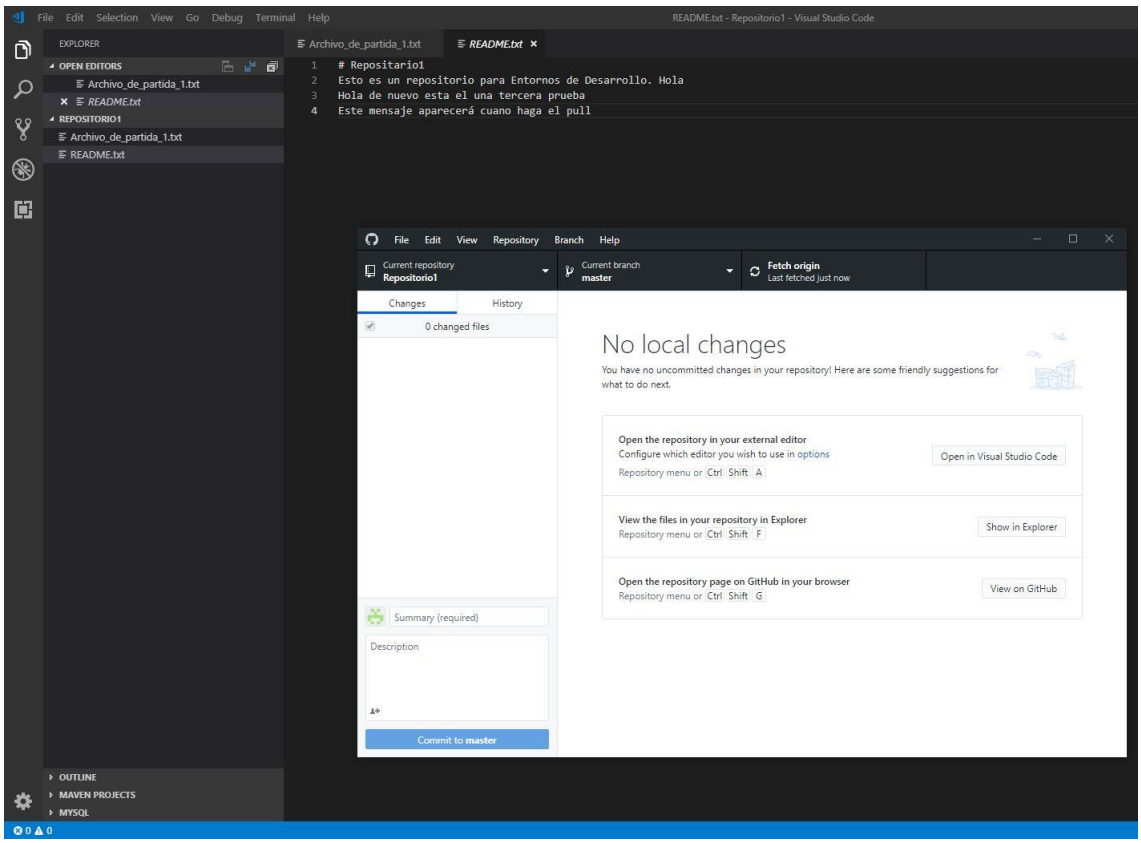
- Ahora vemos en el ordenador de Pablo



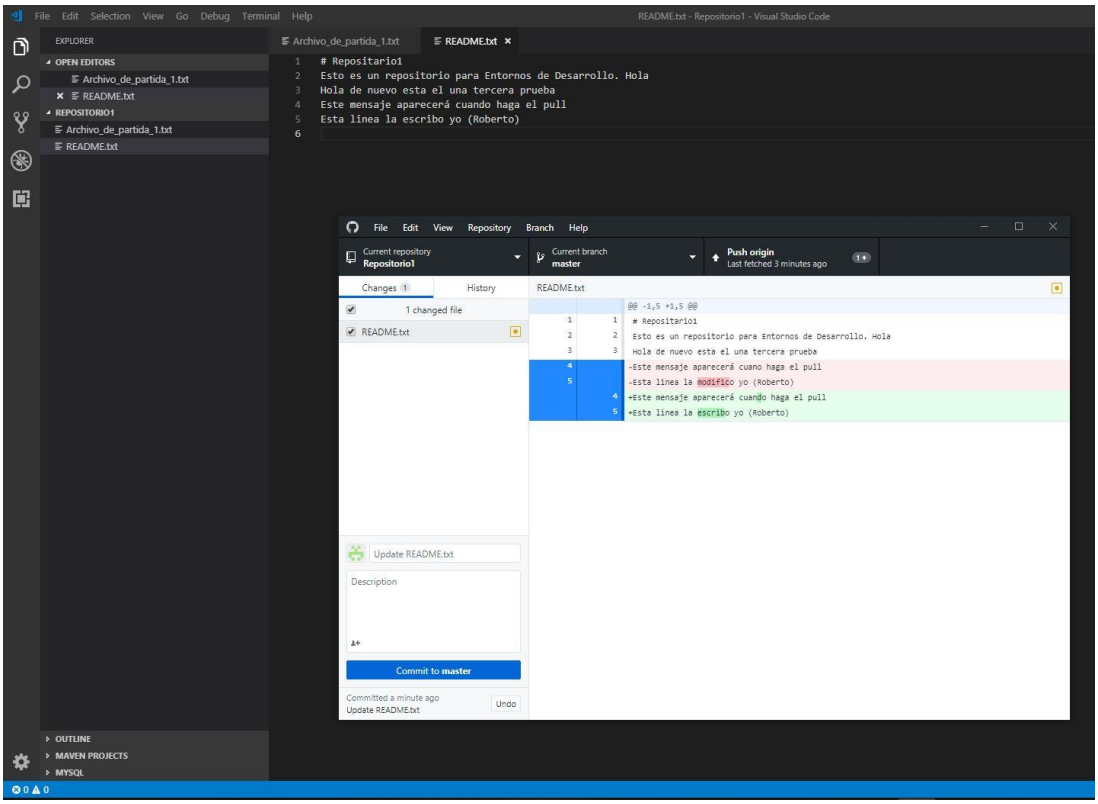
- Y ya vemos en el ordenador de Roberto que tras hacer fetching nos aparece el botón pull origin y también vemos que nos indica el botón superior de la aplicación GitHub pull origin 1+.



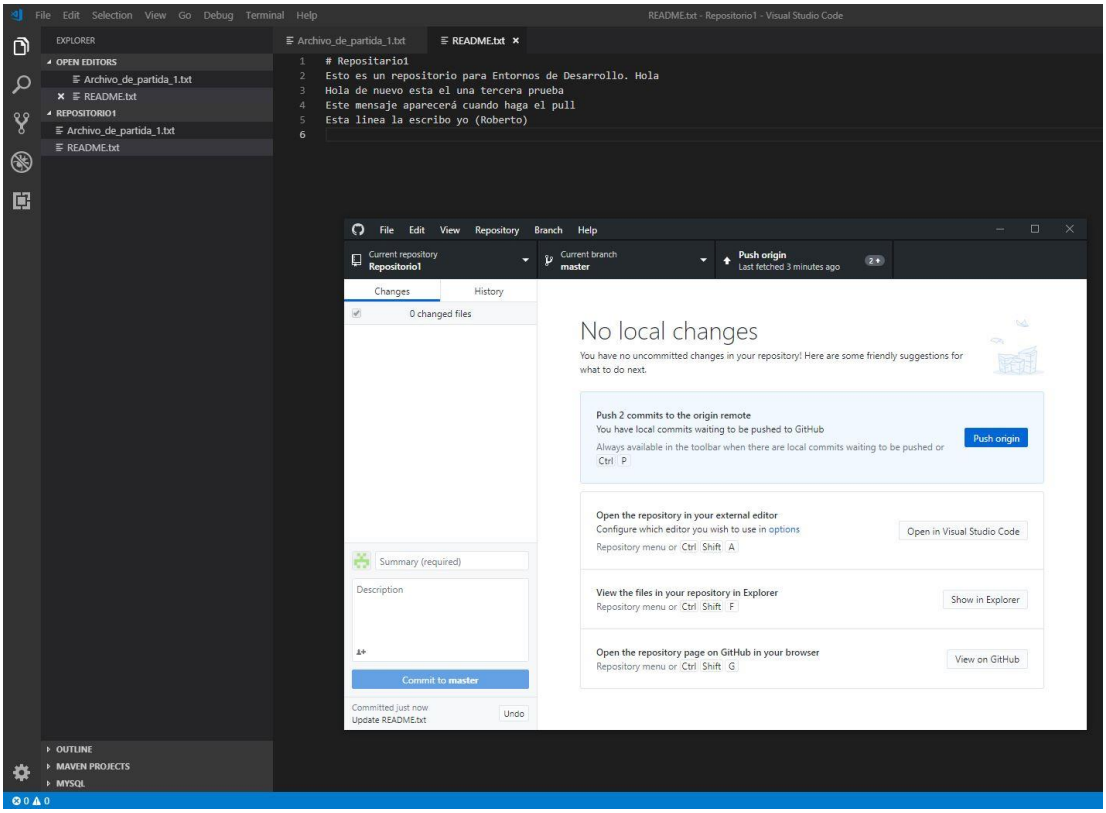
- Tras el pull vemos que tenemos el fichero readme actualizado con una linea más, la 4 concretamente que es la que ha modificado Pablo.



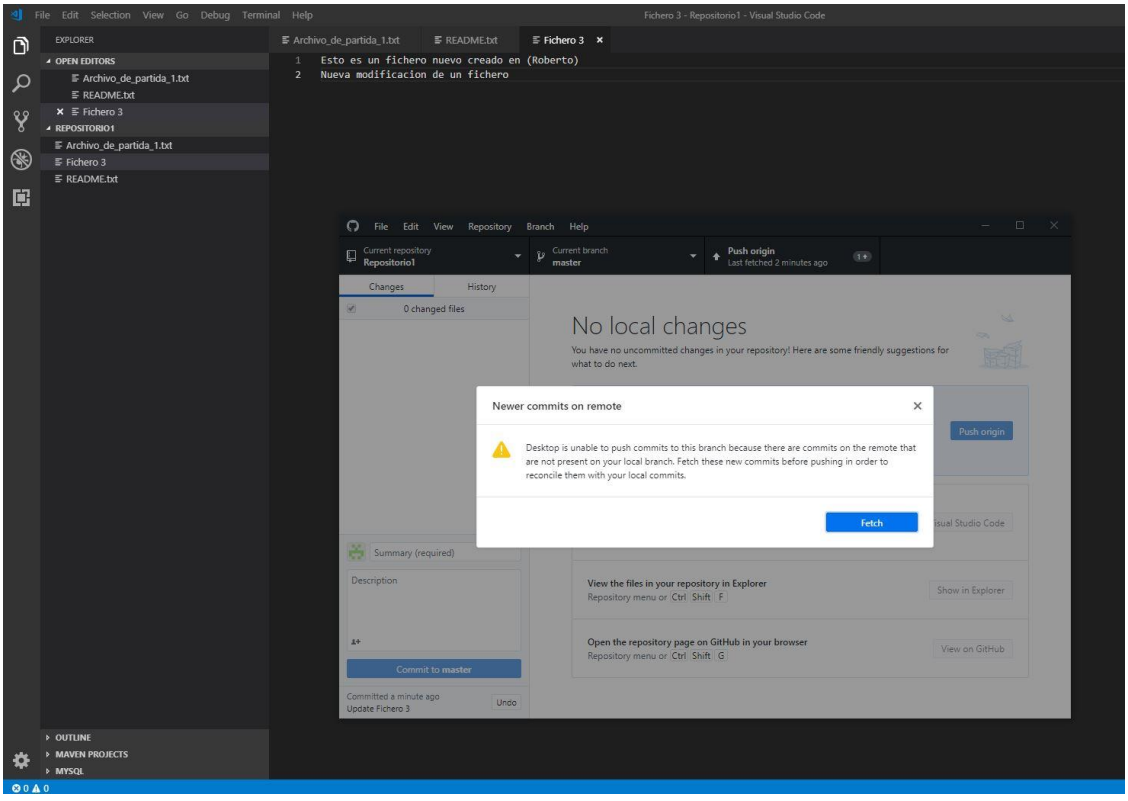
- Ahora modifico yo y envío de nuevo haciendo Commit para que se hagan los cambios, llegados a este punto debo decir que si hay un pull pendiente a mi no me deja de subir cambios ya que borraría lo que ha hecho Pablo, con lo que veo que este sistema es totalmente perfecto y rápido.



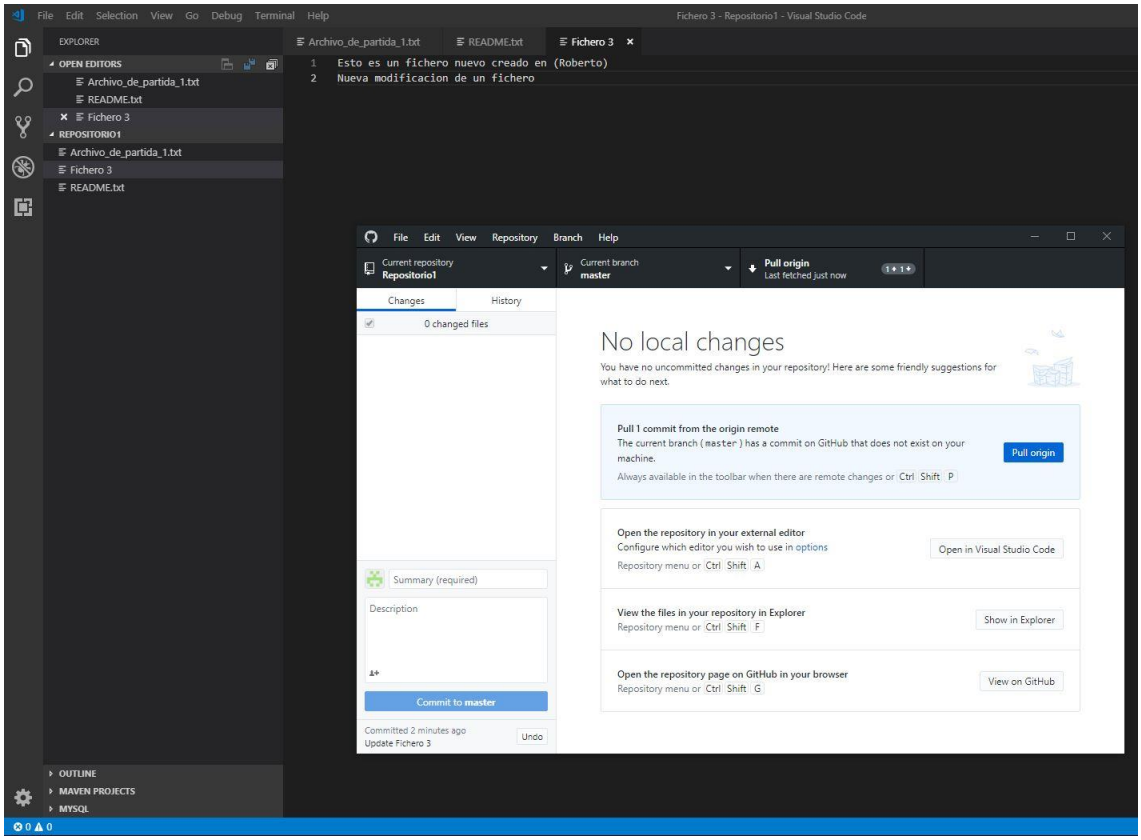
- Al hacer el Commit to master me sale lo siguiente para hacer Push y subirlo.



- Ahora nos muestra que hay nuevas entradas en la nube(Newer commits on remote).



- En esta pantalla vemos que hay modificación y descarga simultanea en tiempo real sin perdida de datos ni sobreescrituras. Podemos verlo en el botón superior Pull origin 1+ ↑ y 1+↓.



- Y por último después de hacer el pull vemos que los ficheros que ha creado Pablo ya me aparecen en el Visual Studio Code.