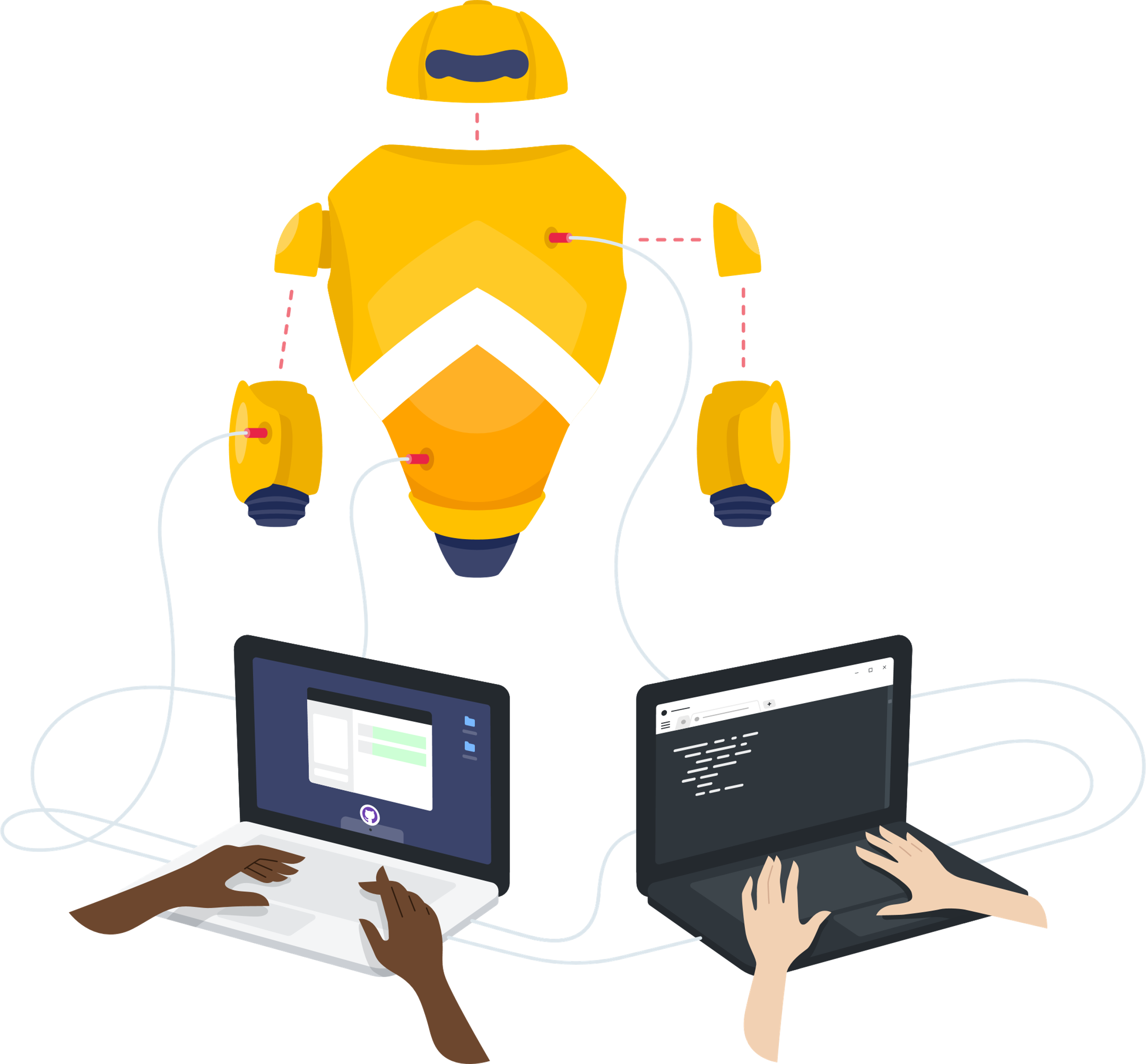
**C.F.G.S**

**DAM Y DAW .ENTORNOS DE DESARROLLO**

**TEMA 4. Control de versiones**

Tarea SCV

sistema de control de versiones



Daniel García Retamar

Francisco Gómez Durán

Raúl Del Pino Martín

**PRÁCTICA DE CONTROL DE VERSIONES. (SCV)**

**Índice**

**Parte 1**

¿Qué es un control de versiones? ............................................................................................ 3

¿Qué es Git y GitHub? ............................................................................................................... 3

¿Qué otros sistemas de control de versiones se usan? ............................................................ 3

¿Qué son los sistemas centralizados y distribuidos? ................................................................ 4

Definición de conceptos ........................................................................................................... 4-5

**Parte 2**

Creación de cuenta, repositorio y subida de archivos GitHub ................................................. 6-7

Añadir colaborador al proyecto ............................................................................................... 8

Añadir carpetas y subirlas al repositorio ................................................................................. 9

Commit y push del archivo ...................................................................................................... 10

Modificar un archivo distinto .................................................................................................. 10

Descarga de modificaciones en equipo local ........................................................................... 11

Conflictos ............................................................................................................................... 12-13

# PARTE 1

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

Un programa para el control de versiones es una aplicación ideada para gestionar ágilmente los cambios en el código fuente de los programas y poder revertirlos. Registrando 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.

* ¿Qué es git y Github?

**Git**

Git es un sistema de control de versiones distribuido. Git fue impulsado por Linus Torvalds y el equipo de desarrollo de del Kernel de Linux. Debido a que el software utilizado anteriormente para tal tarea se convirtió en software propietario, el equipo de Linux creo su propio software también distribuido que aportase lo mejor de los sistemas existentes hasta el momento. Así nació Git, un sistema de control de versiones de código abierto y multiplataforma. Existen varias interfaces gráficas, pero se recomienda usarlo con líneas de comandos.

**Github**

Es un servicio para el alojamiento de repositorios de software gestionados por el sistema de control de versiones de Git. A diferencia de Git, Github es algo mas particular: un sitio web que usa Git para ofrecer a la comunidad de desarrolladores repositorios de software.

* ¿Qué otros sistemas de control de versiones se suelen actualizar en la actualidad?

CVS: Es un sistema de control de versiones con el que puedes guardar el historial de archivos fuente y documentos.

Subversion: Herramienta de control de versiones open source basada en un repositorio cuyo funcionamiento se asemeja al de un sistema de ficheros.

Git: Software de control de versiones fácil de aprender. Puede manejar desde pequeños proyectos a proyectos enormes con velocidad y eficiencia.

Mercurial: Software de control de versiones rápido y fácil de usar.

Bazaar: Es un sistema de control de versiones que ayuda a seguir el historial del proyecto a lo largo del tiempo y es fácil de colaborar con otros.

* ¿Qué son los sistemas centralizados y distribuidos?

**Sistemas centralizados:**

Este tipo de sistemas tienen un único servidor que contiene todos los archivos versionados, y varios usuarios pueden descargar los archivos desde ese lugar central. Una de las desventajas de estos sistemas son el único servidor, ya que si el servidor se cae ningún usuario podrá colaborar o realizar cambios en los archivos

Dos ejemplos de sistemas centralizados son CVS y Subversion.

**Sistemas distribuidos:**

Estos sistemas permiten a los usuarios replicar completamente el repositorio, lo que permite que, si un servidor cae y los sistemas estaban colaborando a través de él, cualquiera de los repositorios de los usuarios puede copiarse al servidor para restaurarlo. Cada clon es una copia completa de todos los datos.

Dos ejemplos de sistemas distribuidos son Mercurial y Bazaar.

Con este sistema ya no tenemos el problema del servidor central.

• ¿Qué son las ramas (branches)? ¿Qué es master?

Una rama Git es un apuntador móvil apuntando a una de las confirmaciones (commit). La rama por defecto es la rama master, con la primera confirmación de cambios que realicemos, se creará esta rama principal master apuntando a dicha confirmación. En cada confirmación de cambios que realicemos, la rama ira avanzando automáticamente. Y la rama master apuntara a la última confirmación realizada. En resumen, una rama es una modificación de los datos que difiere de la rama principal o rama master.

Directorio .git

El directorio .git es el directorio raíz de nuestro proyecto. Algunos de los archivos que contiene son:

• COMMIT\_EDITMSG: Contiene la descripción del último commit

• config: Contiene las opciones de configuración de este repositorio.

• Description: Esta descripción se mostrará cuando estés viendo tu repositorio o la lista de repositorios.

• FETCH\_HEAD: Los SHAs de todas las ramas que fueron actualizadas por el último git fetch.

• HEAD: El ref actual. En la mayoría de los casos éste es probablemente refs/heads/master.

• index: El área de staging conteniendo información acerca de los archivos que serán incluidos en el próximo commit.

• packed-refs: refs inactivos.

• ORIG\_HEAD: Durante una integración, este es el SHA de la rama en la que se está realizando la operación.

• MERGE\_HEAD: Durante una integración, este es el SHA de la rama desde la cual se están tomando los cambios a integrar.

• MERGE\_MODE: Se utiliza para pasar opciones de un git merge al siguiente git commit cuando ocurre un conflicto durante la integración y se requiere un git commit para finalizar la operación. Actualmente --no-ff es la única opción que se pasa de esta forma.

• MERGE\_MSG: Enumera los conflictos que se produjeron en el último git merge.

• RENAMED-REF: Está relacionado con errores ocurridos al grabar refs.

• hooks: Contiene distintos scripts que son ejecutados en determinados momentos al trabajar con Git, como por ejemplo después de cada commit o antes de cada rebase.

• info: Relativamente sin importancia, a excepción del archivo exclude que se encuentra dentro de este directorio.

• logs: Contiene la historia de las distintas ramas.

• objects: El depósito interno de objetos, indexados por SHAs.

• rebase-apply: La base para el comando rebase y también para git am.

• refs: La copia maestra de todos los refs que existen en tu repositorio, ya sean stashes, tags, ramas locales o remotas.

•¿Qué hace el comando add y commit?

**add**: Añade los ficheros modificados

**commit**: Confirma los ficheros enviados

**pull**: Descarga el proyecto del repositorio

**push**: Envía el repositorio local

**clone**: Clona un repositorio

**status**: Devuelve el estado de un repositorio y sus modificaciones

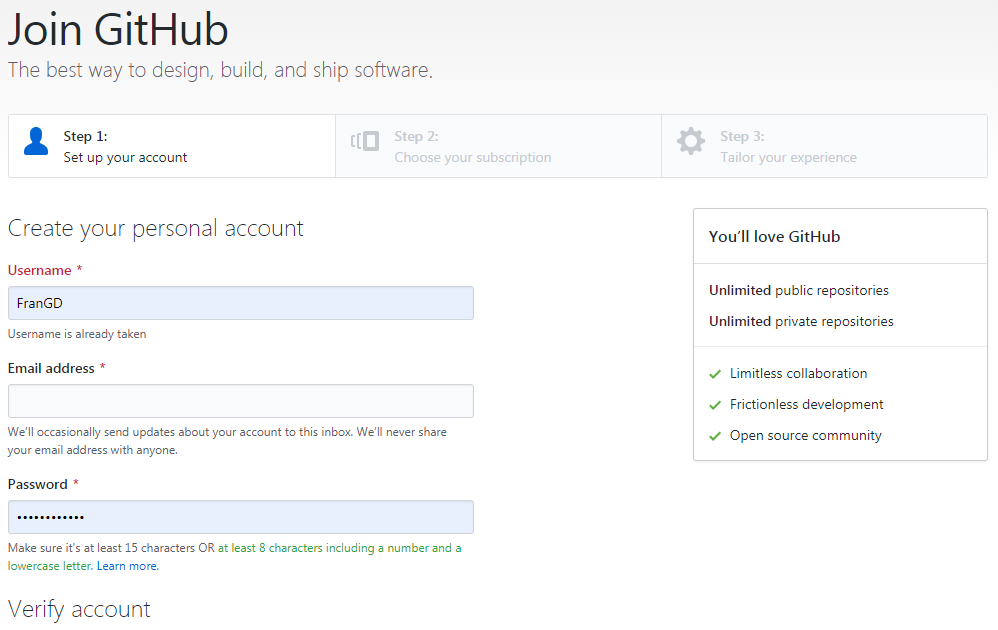
**C.F.G.S DAM Y DAW .ENTORNOS DE DESARROLLO**

**TEMA 4. Control de versiones**

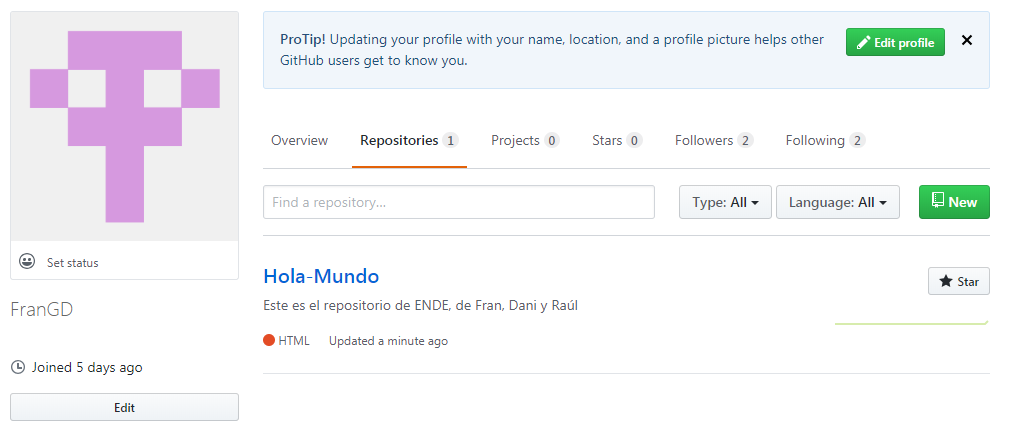
## **PARTE 2**

Para que puedas tener en tu ordenador una copia de un sistema de control de versiones y puedas trabajar en paralelo con un compañero, haremos lo siguiente:

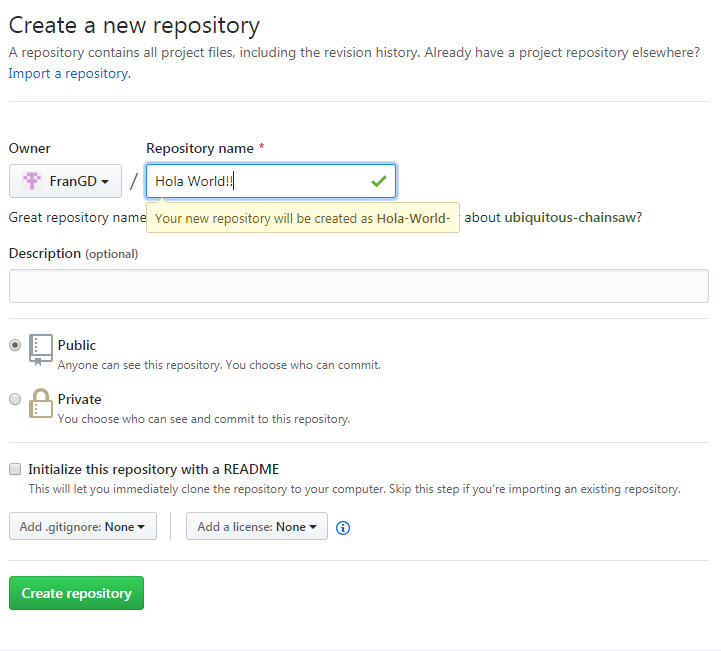
Lo primero que necesitaremos será crear una cuenta de GitHub donde crearemos los repositorios para ello accedemos a github.com y pulsamos en la opción de sign up e introducimos nuestros datos.

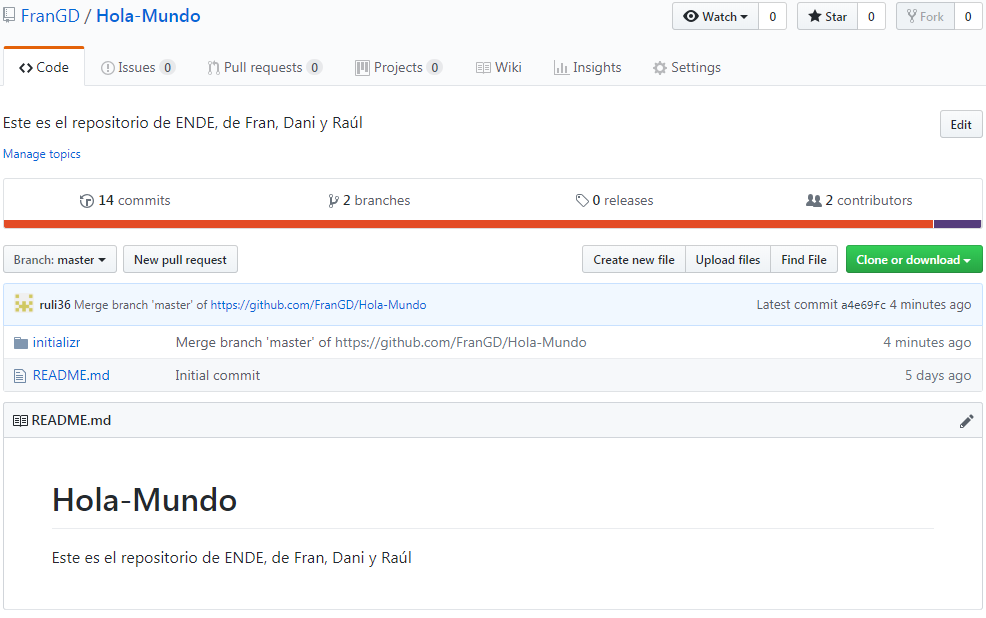


Una vez creada la cuenta crearemos nuestro primer repositorio para ello en página del perfil creado y accedemos a la pestaña Repositories. En la pestaña pulsamos en new.

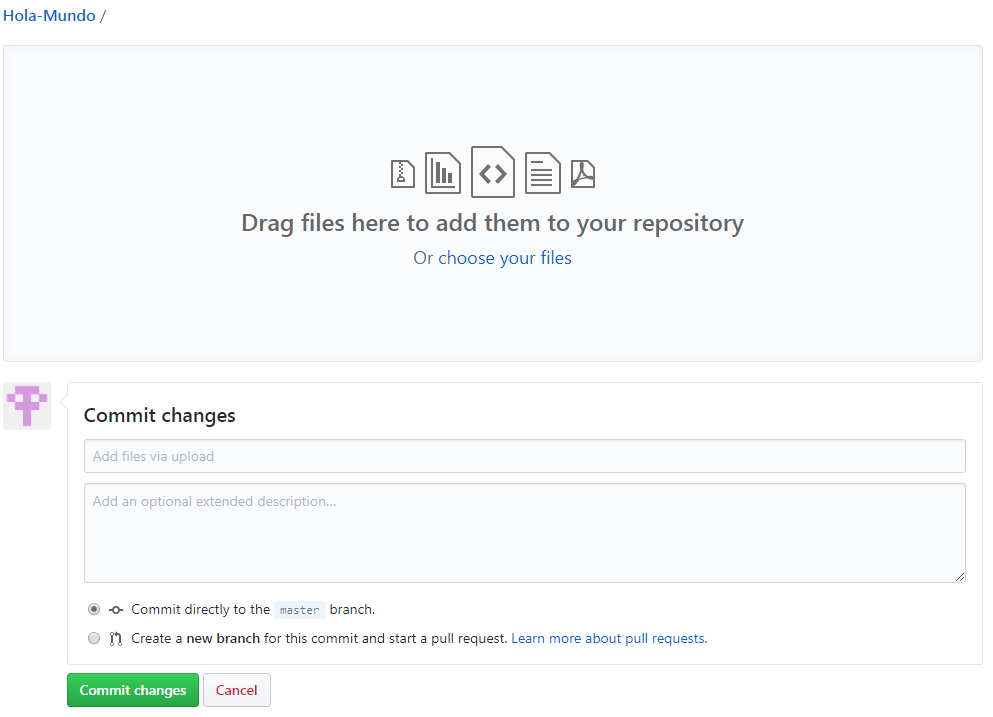


La nueva ventana nos pedirá el nombre del repositorio, una descripción opcional y si queremos que este sea público o privado.

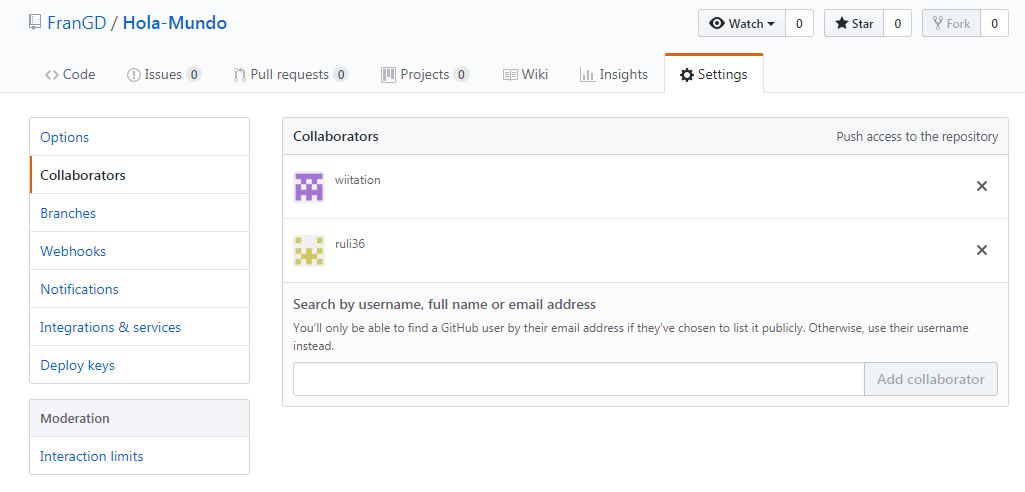


Dentro del repositorio pulsamos en la opción upload files donde se nos preguntará por el archivo que deseamos subir.

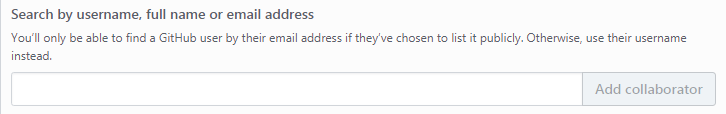
Arrastramos el archivo y pulsamos commit.



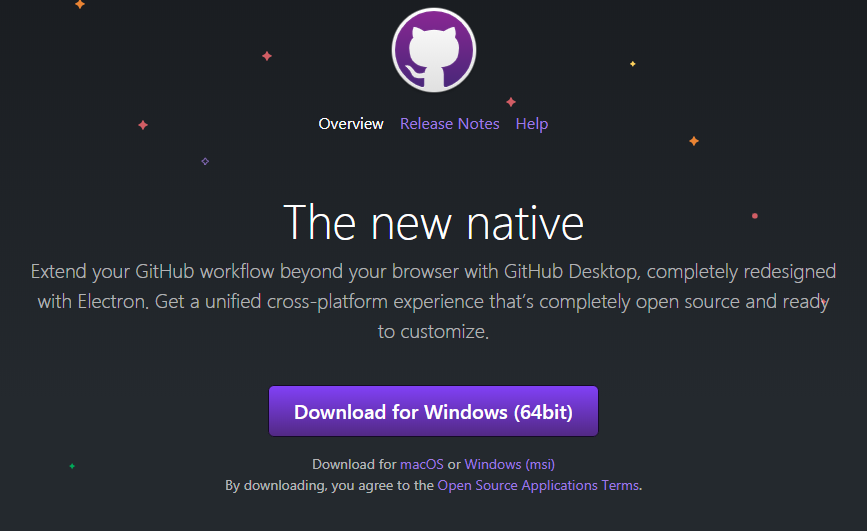
Para añadir un colaborador seleccionamos la pestaña settings en la ventana del repositorio. Abajo a la izquierda en la lista de opciones pinchamos en Collaborators.



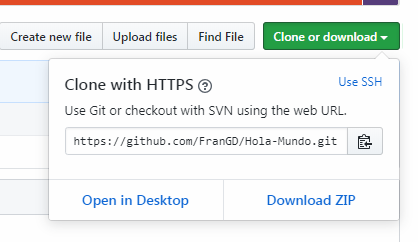
Buscamos a nuestro compañero de proyecto y lo añadimos.



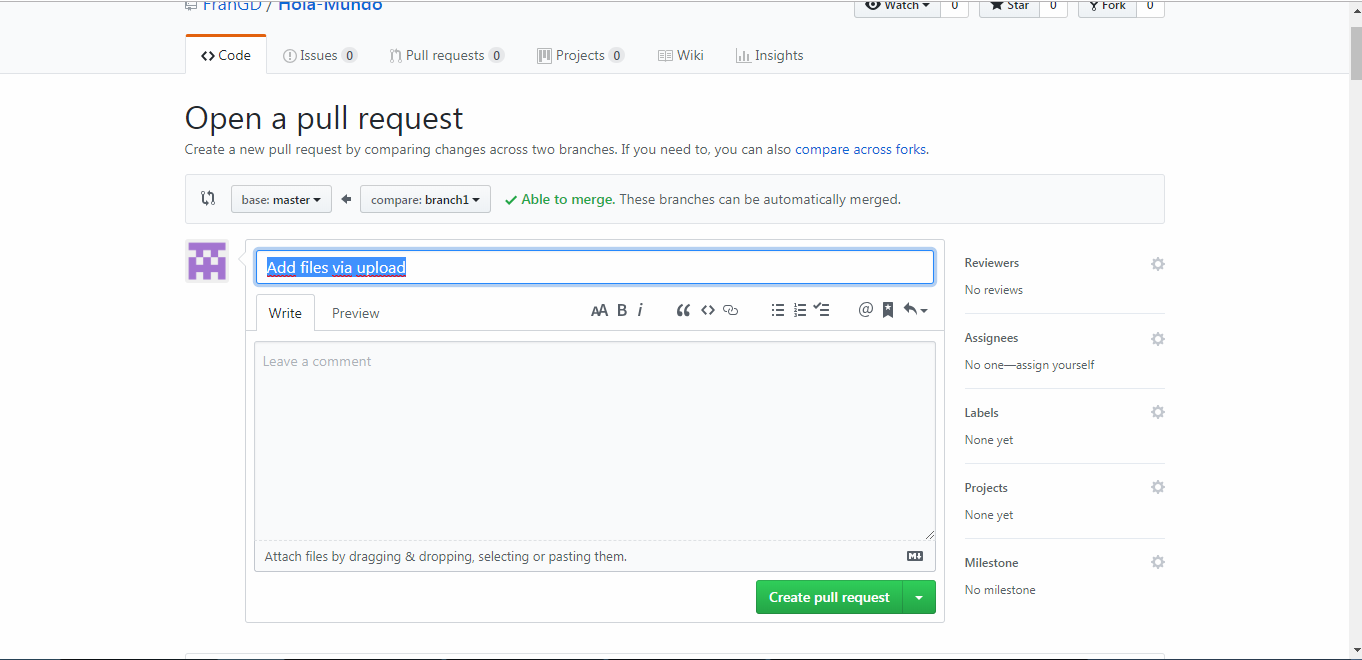
Para descargar Github desktop nos dirigimos al siguiente enlace: <https://desktop.github.com/> aquí encontraremos la versión desktop de GitHub y procederemos con la descarga.

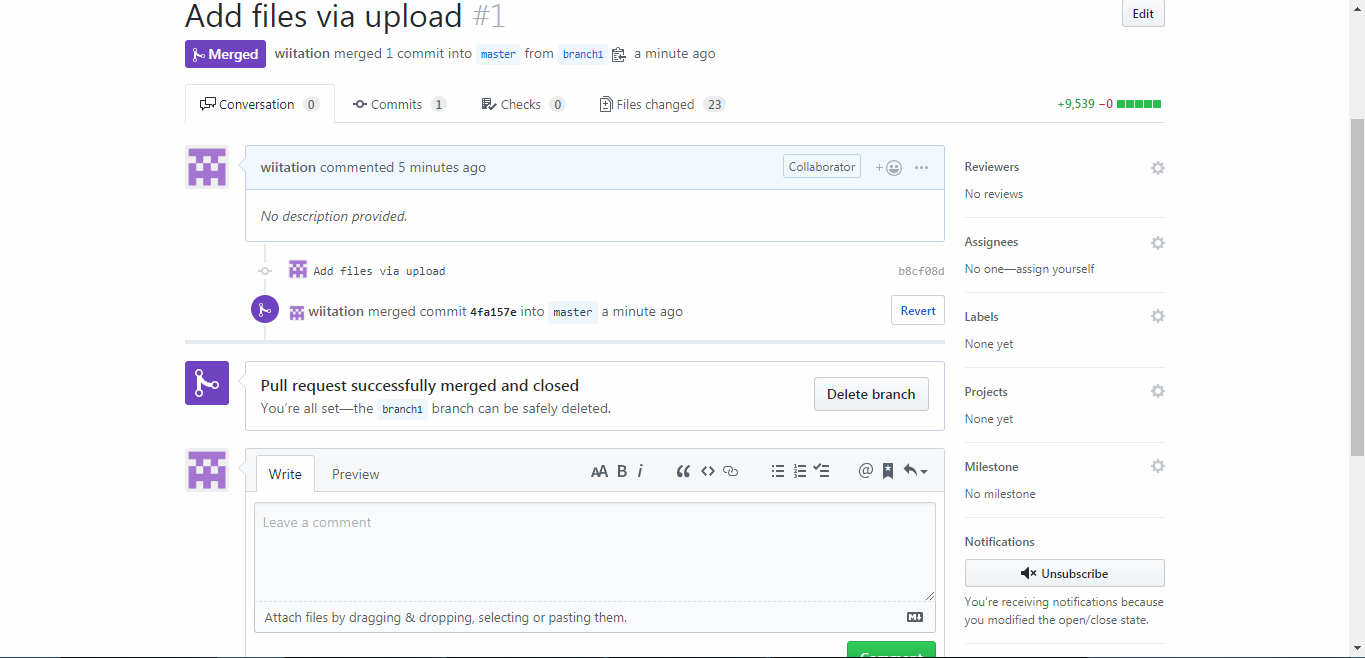


Para realizar un clone en nuestro PC solo tendremos que pulsar el botón clone or download que tenemos dentro del repositorio.



Para añadir carpetas con archivos debemos utilizar el comando pull, iniciamos una pull request y le indicamos a Github cuales son los archivos que necesitamos subir. Creamos la pull request y lo confirmamos con commit.

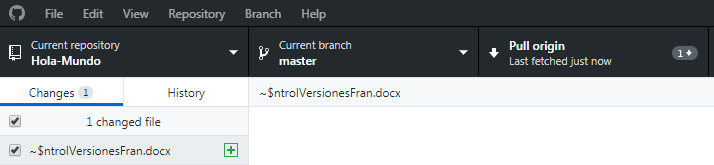




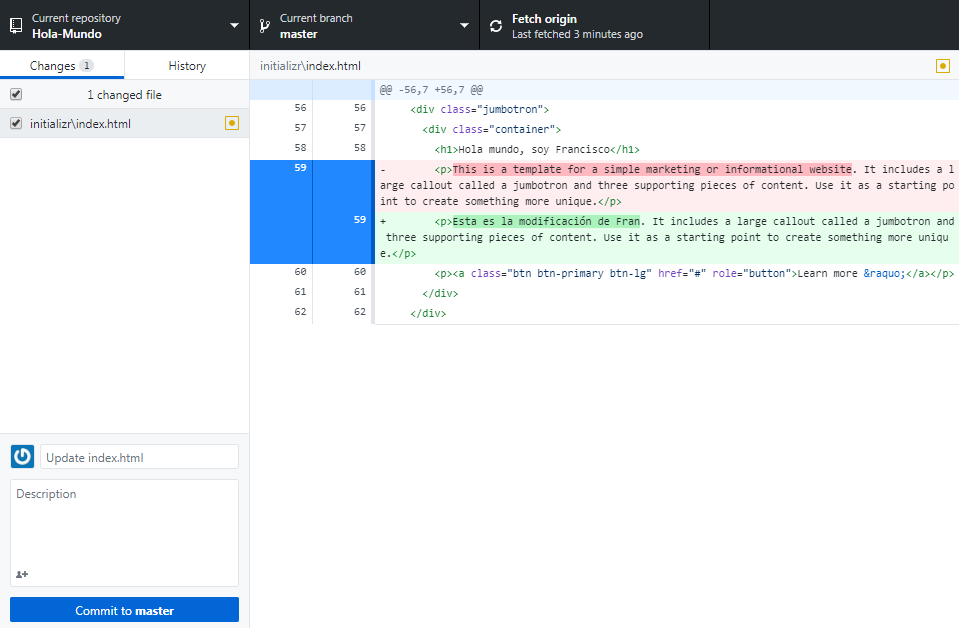
Finalmente, con el comando push lo subimos al repositorio.



En las siguientes capturas se muestra que los colaboradores tienen una copia del proyecto en su equipo local. Para ello refrescamos la página con fetch y luego pulsamos en Pull origin situado arriba a la derecha lo que descarga la última versión en nuestro equipo.



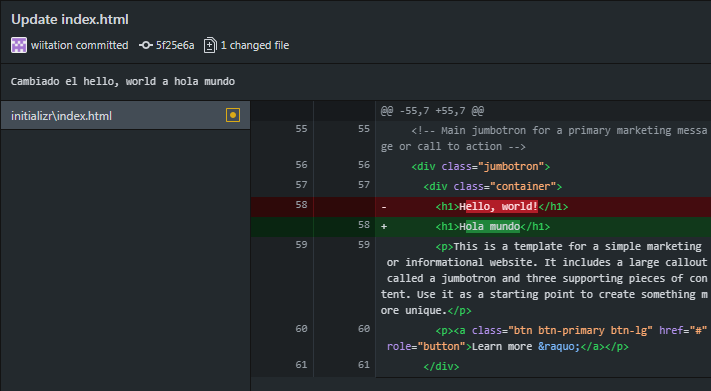
Los miembros del grupo modifican archivos diferentes, como se muestra en la captura los diferentes cambios realizados aparecen en la pestaña changes de la izquierda, debajo tendremos los diferentes archivos que se modificaran en los que podremos navegar y ver sus diferentes cambios. Mas abajo pedirá en nombre de la versión y una breve descripción de los cambios realizados.



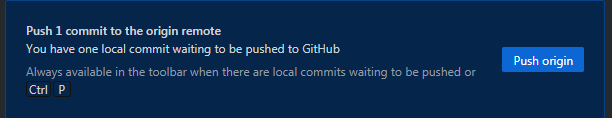
Después de completar esta información solo quedaría pulsar sobre Commit to master y hacer un push origin.



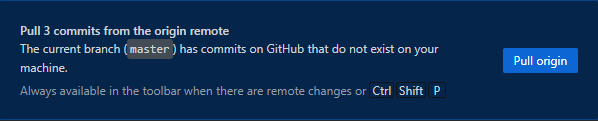
En el siguiente ejemplo podemos el nombre de la versión y su descripción tras hacer el commit.



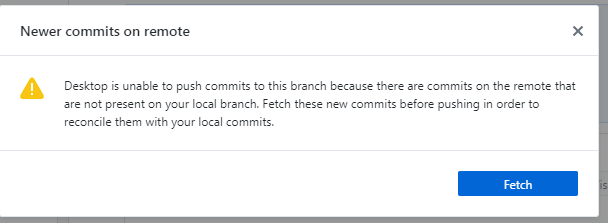
Realizamos un push para subirlo al repositorio.



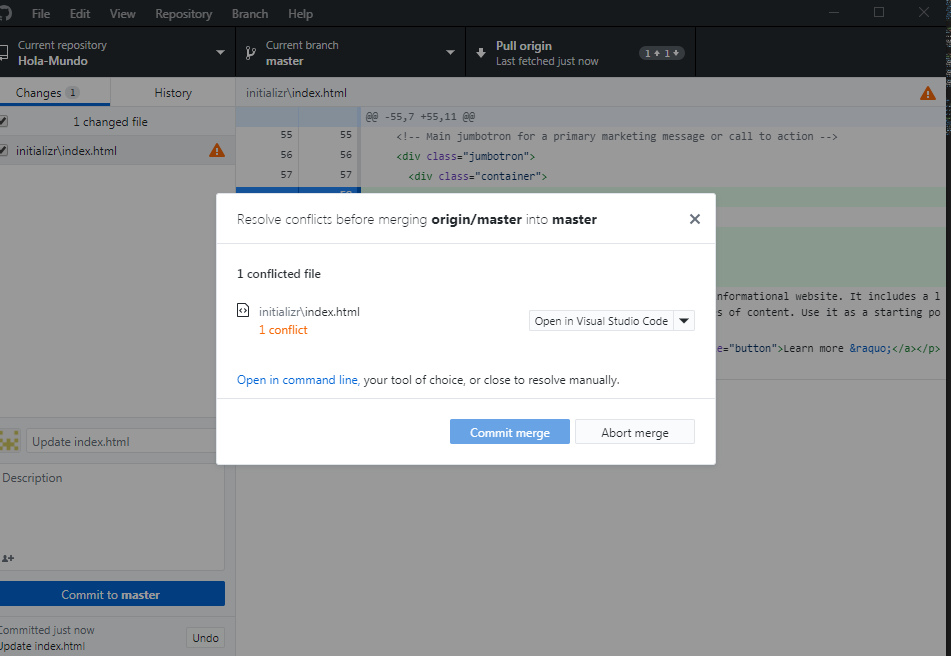
Descargamos de nuevo las modificaciones realizadas en el proyecto en nuestro equipo mediante pull origin.



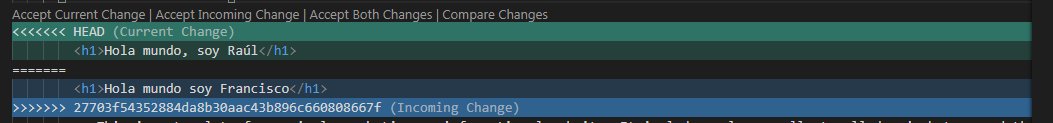
Modificamos el mismo archivo con un compañero en la última versión, hacemos commit y push. Nos notifica que se ha producido un conflicto.



El programa te advierte que no puede hacer un commit a la rama porque hay commits que no están presentes en tu versión del archivo.



Te muestra el archivo con el conflicto, al abrirlo en un editor puedes ver cuál es el problema, junto a las opciones disponibles:



* Accept current change: Acepta el cambio y sobrescribe la version subida.
* Accept Incoming change: Acepta el cambio y sobrescribe tu versión.
* Accept Both Changes: Acepta ambos cambios y los incorpora a tu documento.
* Compare changes: Compara los cambios en ambos documentos para ver la diferencia entre ambos.