Resumen de GIT y GitHub

Resumen de la charla del día 9/2/22

Alejandro Barrachina Argudo

Grado en Ingeniería informática
Facultad de Informática
Universidad Complutense de Madrid

2021-2022

Índice general

1 Introducción

1.1. ¿Qué es GIT?

GIT es un sistema de control de versiones que nos permite gestionar código y sus diferentes versiones a lo largo de una producción. En concreto al hablar de GIT estamos hablando de un sistema de control de versiones **descentralizado o distribuido**. Esto significa que cada usuario tendrá su propio repositorio y podrán intercambiar y mezclar versiones entre ellos. Normalmente habrá un repositorio central que sirva de sincronización entre los distintos usuarios.

1.2. Pack de estudiantes de GitHub

Las ventajas principales del pack de estudiantes son:

- Repositorios privados ilimitados.
- Herramientas disponibles solo para repositorios públicos (gráficos de contribución, ramas protegidas, GitHub Pages, etc) en repositorios privados.
- Cursos ofrecidos por otras compañías.

Para conseguir estas ventajas solo hay que ir a `https://education.github.com/` y registrarte con la cuenta de correo de la universidad.

En caso de que ya tengas cuenta con otro correo, inicia sesión como lo harías normalmente, ve a la página `https://github.com/settings/emails` y añade un correo adicional.

1.3. Creación de un repositorio

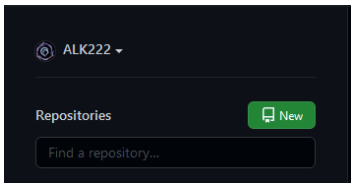

Figura 1.3.1: Botón de creación de repositorio

Para crear un repositorio nuevo iremos a la página principal (`https://github.com`) y buscaremos en la parte superior derecha el botón verde de creación.

Tras esto, se nos ofrecerán una serie de opciones a la hora de crearlo, como si queremos que sea público o privado, el nombre del mismo, o las opciones de licencia, *readme* y *gitignore*.

Estas opciones son:

- **Nombre del repositorio:** se explica por sí solo. Será el nombre de la carpeta al clonarla en nuestro PC. Conviene sustituir los espacios en blanco por guiones o barras bajas.
- **Descripción:** Resumen del objetivo del repositorio, opcional.
- **Público/Privado:** Puedes escoger si el resto de gente puede ver tu repositorio o no. Esto se puede cambiar tras la creación del repositorio en cualquier momento.
- **README:** archivo donde das una información mas detallada sobre el repositorio. Normalmente la gente no lo lee.
- **.gitignore:** archivo que va a determinar que archivos **no** se suben al repositorio. Puede editarse a mano, pero los predefinidos para cada lenguaje por github están bastante completos.
- **Licencia:** establece las condiciones de uso del resto de usuarios.

1.4. Añadir colaboradores a un repositorio

Si queremos trabajar con un equipo en un repositorio privado, tendremos que darles acceso al mismo. Para este ejemplo usaremos un usuario de nombre *A* que tiene un repositorio de nombre *prueba*.

Para añadir gente a *prueba* podemos ir a la propia página del repositorio desde nuestro perfil, debajo de la barra vista en 1.3.1 (Botón de creación de repositorio) y buscarlo ahí.

Una vez entremos en la página del repositorio podemos ir a la configuración del mismo en la barra que tendremos justo debajo del nombre del repositorio. Para hacerlo de forma más directa, iríamos a `https://github.com/A/prueba`.

Una vez en la configuración iremos a **collaborators**. Una vez en esa pestaña se nos pedirá nuestra contraseña para continuar. Después de introducir la contraseña buscaremos el botón verde de **add people** y podremos buscarles por nombre de usuario o correo.

Para que los usuarios invitados tengan acceso, deberán seguir los pasos que se nombran en el correo que les llegará a su correo principal de GitHub.

2 Trabajar sobre un repositorio remoto de manera local

Para modificar el código de la web tendremos dos opciones. Ninguna de ellas va a ser editarlo desde la propia web ya que es poco eficiente y tedioso.

Durante toda esta sección mostraré como hacer las acciones tanto con GitKraken como con GitHub Desktop.

Para explicar mejor todo, se hará referencia a dos figuras presentadas en el último apartado. (Figuras 4.0.1 y 4.0.1)

2.1. Clonar el repositorio

Clonar un repositorio significa bajar el código que hay en GitHub a nuestro PC.

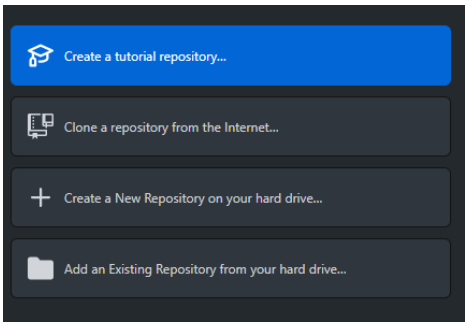

Figura 2.1.1: Pantalla vacía de GitHub Desktop

Si vas a clonar un repositorio desde GitHub Desktop hay dos opciones: si no tienes más repositorios te saldrá como una opción en grande, en caso de tener más repositorios solo tienes que pulsar *Ctrl + Shift + O*.

Desde GitKraken solo tienes que pulsar *Ctrl + N* para clonarlo.

En ambos casos te pedirán que escojas el repositorio a clonar y donde clonarlo, la carpeta donde se descargará el código. En el caso de GitKraken tendrás que escoger GitHub como el proveedor.

2.2. Subir código

Parte importante de colaborar en un proyecto es modificar código y subir tus cambios.

Desde GitHub Desktop tendremos a la izquierda una pestaña con todos los archivos modificados (Figura 4.0.1: Cuadro 1). Al lado de estos archivos podemos marcar cuáles queremos subir y cuáles no. Una vez escogidos cuáles subir pondremos una descripción (a poder ser más explicativa que *Update file*) y le daremos al botón azul de *Commit to main*. Tras esto solo falta irnos a la barra superior y darle a *Push origin* (Figura 4.0.1: Cuadro 2). Esto último hará que subamos el código a GitHub y esté disponible para el resto del equipo.

En caso de usar GitKraken, las cosas son muy similares. A la izquierda tenemos una pestaña con los archivos modificados(Figura 4.0.2: Cuadro 1), tendremos que marcar uno por uno cuáles queremos subir dando click derecho al archivo y seleccionando *Stage file*, en caso de querer seleccionarlos todos tendremos un botón verde que dirá *Stage all changes*. Tras esto pondremos una descripción y daremos al botón verde de *Stage files to commit*. Solo falta ir a la barra de arriba y darle al botón de *Push* (Figura 4.0.2: Cuadro 2)para subir los cambios al origen.

2.3. Bajar cambios

Antes de ponernos a picar código conviene mirar a ver qué han hecho nuestros compañeros en el tiempo en que no estábamos y bajarnos sus cambios.

En GitHub Desktop esto lo haremos desde la barra superior, en un botón en el que pondrá *Fetch origin* (Figura 4.0.1: Cuadro 2). En caso de haber cambios el botón cambiará a *Pull origin* (Figura 4.0.1: Cuadro 2). Si pulsamos esta vez nos bajará los cambios que haya en el repositorio.

En GitKraken se actualizará automáticamente en cierto intervalo de tiempo o al abrir la aplicación. Sabremos si hay algo nuevo porque en el gráfico saldrá un punto nuevo con una etiqueta ligeramente más oscura que la del gráfico principal. Una vez más para bajarnos el código solo tenemos que ir a la barra de arriba y pulsar en *Pull* (Figura 4.0.2: Cuadro 3). En caso de que queramos actualizar los cambios y no se haya pasado el intervalo de tiempo, solo hay que ir al botón de *Pull*, darle a la flecha que hay al lado y seleccionar *Fetch all*.

2.4. Qué hacer cuando hay un *commit* erróneo

Todos cometemos errores, así que no está de más saber cómo arreglar un *commit* que rompe el programa.

Aquí explicaremos un comando: *git reset*.

El comando **reset** nos permite resetear, como bien indica su nombre, la cabeza del repositorio hasta un *commit* anterior. Esto se hará en tres posibles modos:

- **Soft:** mueve la cabeza del repositorio al *commit* seleccionado, pero no altera el repositorio. Los cambios entre el *commit* y la antigua cabeza se quedarán confirmados para un nuevo *commit*.
- **Mixed:** genera una copia de los datos posteriores al *commit* al que se quiere volver.
- **Hard:** descarta todos los cambios posteriores al *commit* deseado.

En GitKraken podremos seleccionar los modos dando click derecho a un commit y buscando la opción *Reset main to this commit*.

GitHub Desktop y GitHub en su interfaz web no tienen, al momento de escribir esto, soporte para esta opción. La otra opción para hacer esto desde tu repositorio local sería hacerlo desde la línea de comandos.

3 | Ramas

Una característica muy importante de este tipo de sistemas de gestión de versiones son las ramas. Una rama puede definirse como un conjunto independiente de *commits*.

Al crear un repositorio tendremos una rama principal, normalmente llamada *Master*(casi desechado en repositorios nuevos) o *Main*. A partir de aquí podemos seleccionar un commit concreto y crear una rama a través de él.

Supongamos que estamos en un equipo desarrollando una aplicación de gestión de bibliotecas. El encargado de la funcionalidad de alquilar libros no quiere interferir con el trabajo de su compañero que se está dedicando a hacer las consultas a la base de datos. Para esto, cada uno de ellos creará una rama para ir implementando su funcionalidad concreta. Cuando acaben pueden hacer un *merge* a la rama principal, o en otras palabras, fusionar su código con la última versión de la rama principal.

Aunque este es un caso factible, las ramas se suelen usar también para el desarrollo independiente de versiones o para parches.

Tras entender lo que son las ramas, vamos a ver como gestionarlas

3.1. Crear una rama

Para poder gestionar una rama, primero tenemos que tenerla.

En GitKtraken crearemos una rama nueva seleccionando un *commit* y con click derecho seleccionando *Create branch here*. En GitHub Desktop nos iremos al botón de ramas y ahí escogeremos la opción de crear una nueva.

3.2. Fusionar ramas

Antes se ha mencionado el fusionar una rama con la principal, pero también se pueden fusionar dos ramas secundarias. A esto lo llamaremos un *merge*.

Durante un *merge* puede haber conflictos entre archivos. Esto ocurrirá cuando dos fragmentos de código no se puedan solapar automáticamente y se nos pedirá arreglarlo. GitKraken te permite seleccionar las líneas de código a mantener entre los posibles conflictos desde su propia interfaz. Tanto GitKraken como GitHub Desktop te permitirán arreglar esto en editores externos como Code, Vim o Atom, por nombrar unos ejemplos.

Desde GitKraken nos dará la opción de fusionar dos ramas si arrastramos la etiqueta de la rama que queremos fusionar hacia la etiqueta de la rama que va a recibir el *merge*.

En GitHub Desktop nos iremos a la barra de opciones de arriba de la pantalla y en ramas podremos ver la opción *Merge into current branch*. Esto significa que tendremos que ejecutar esta opción desde la rama que va a recibir el *merge* y seleccionar la rama a fusionar.

3.3. Cherry Pick

Cherry Pick es el nombre que recibe la acción de aplicar solo cierto *commit* de una rama a otra.

Esto puede ser útil en el desarrollo de una programa si en cierto punto de una rama se detecta un *bug* que afecta al resto.

Esta opción no se da en GitHub ni en su versión web ni en su aplicación de escritorio, pero en GitKraken es tan fácil como hacer click derecho en el *commit* y seleccionar *Cherry pick commit*.

3.4. Reorganizar ramas

El comando **rebase** puede ser útil en algunas situaciones. Es parecido a *merge*, pero mientras que éste último cierra la rama al terminar la fusión, *rebase* solo fusiona el código manteniendo ambas ramas como independientes.

4 | Figuras explicativas

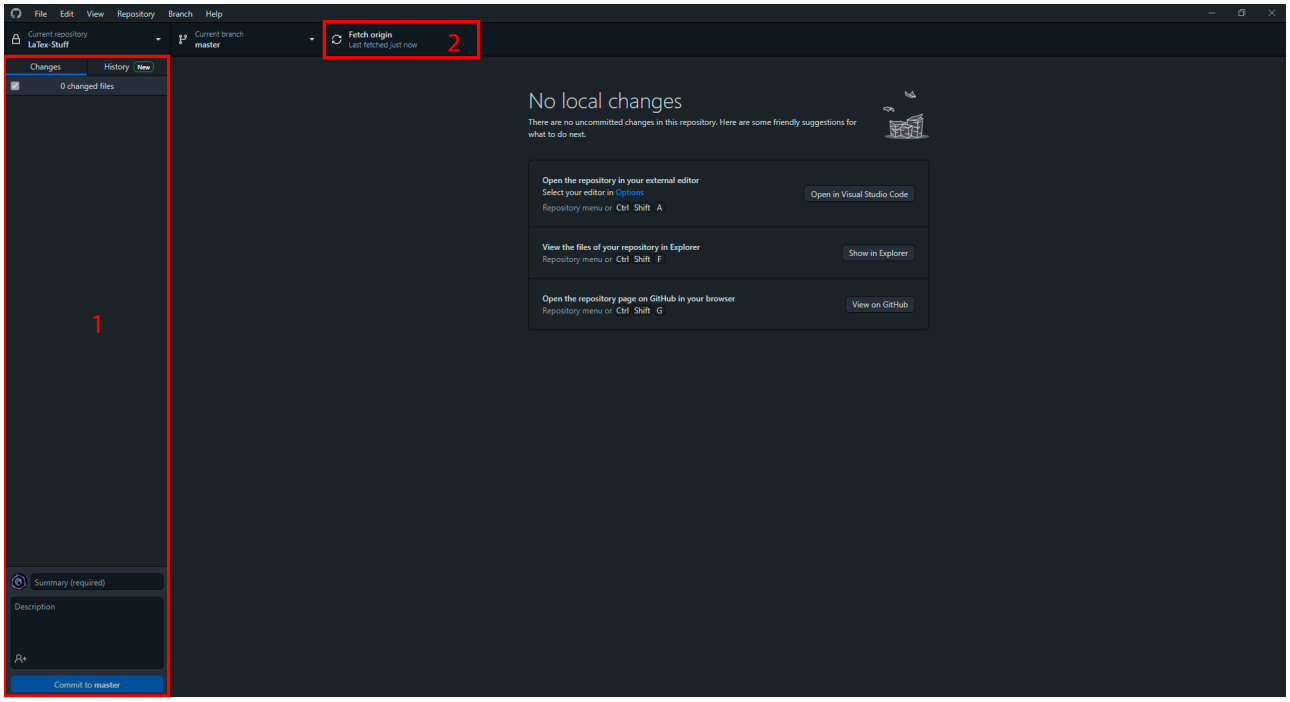

Figura 4.0.1: Captura de pantalla de GitHub Desktop

En el caso de este programa, el botón 2 cambiará de opción automáticamente si hay o no código que subir o que bajar

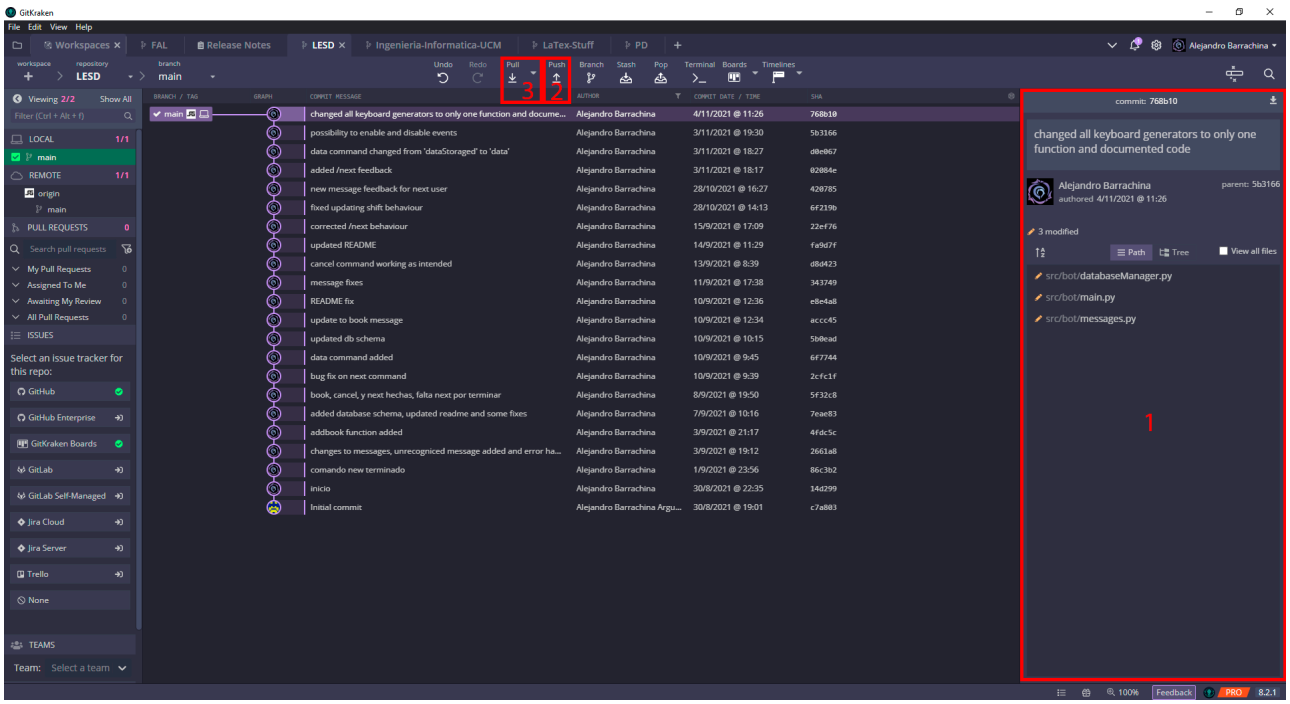

Figura 4.0.2: Captura de pantalla de GitKraken

Índice de figuras

Índice de cuadros