

por Mari Chelo

Entornos de desarrollo

Bloque 1

Tema 7: Introducción al control de versiones. Herramientas de Git

1.7.1. Introducción al control de versiones

1.7.1.1. Definición

Los sistemas de control de versiones (VCS) son herramientas que pueden registrar cualquier cambio en cualquier archivo o conjunto de archivos a lo largo del tiempo, para que podamos recuperar fácilmente cualquier versión anterior. Se pueden utilizar no solo con archivos fuente, sino también con cualquier otro tipo de archivo.

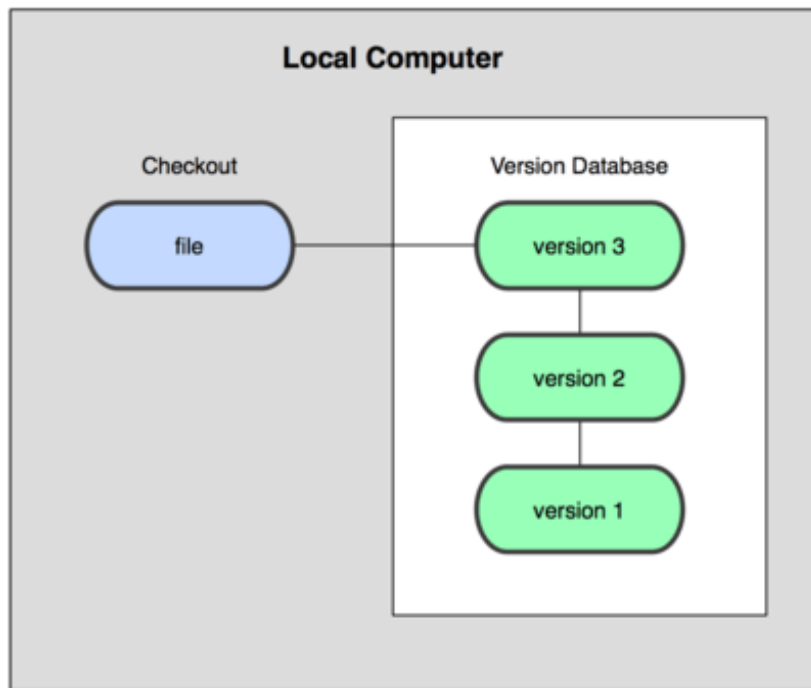
Un VCS nos permite revertir el estado de cualquier archivo o incluso de un proyecto completo, comparar archivos a lo largo del tiempo, determinar quién cambió el archivo en una marca de tiempo determinada y mucho más. Además, si algún archivo se daña o se pierde, podemos volver a una versión anterior en el historial y recuperarlo nuevamente.

1.7.1.2. Tipos de VCS: VCS local

VCS se puede utilizar en línea o en modo local. Este último modo es particularmente útil porque podemos crear fácilmente una copia de seguridad de un proyecto y almacenarlo localmente, para poder restaurarlo más tarde si es necesario (en caso de error, por ejemplo) y volver a una versión estable.

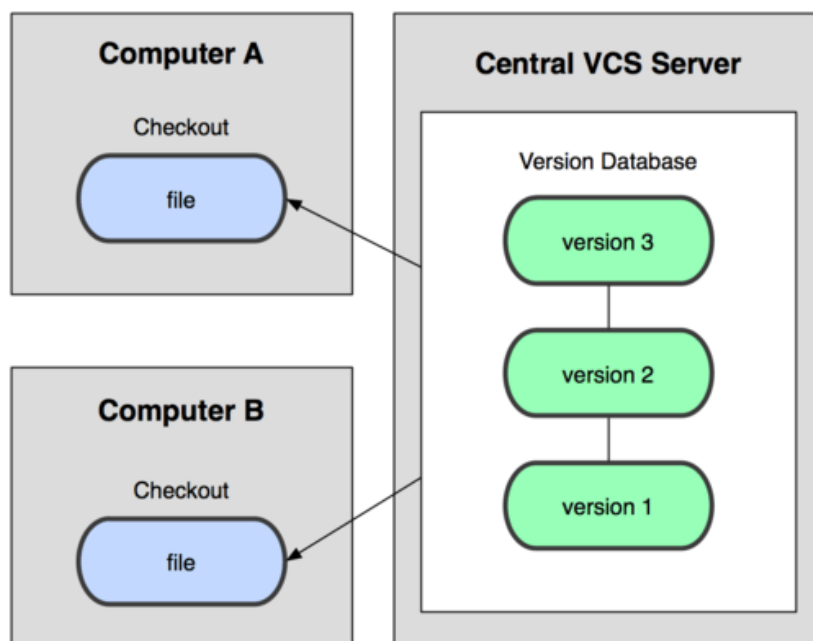
La principal ventaja de esto es su sencillez, y el principal inconveniente es que debemos gestionar el control de versiones de forma manual, por lo que podemos cometer algunos errores en este proceso. Por ejemplo, debemos olvidar que estamos en la carpeta incorrecta y luego modificar el archivo de respaldo en lugar del actual.

Para afrontar estos problemas, existen algunas herramientas interesantes que nos ayudan a gestionar los archivos y cambios. Uno de los más populares es un sistema llamado *rcs*, que todavía se puede encontrar en muchas computadoras. Básicamente, esta herramienta almacena un conjunto de parches o diferencias entre archivos de una versión a la siguiente. Estos cambios se almacenan en un tipo de archivo especial, y luego el sistema puede recuperar cualquier estado anterior de cualquier archivo, agregando o restando los parches correspondientes.



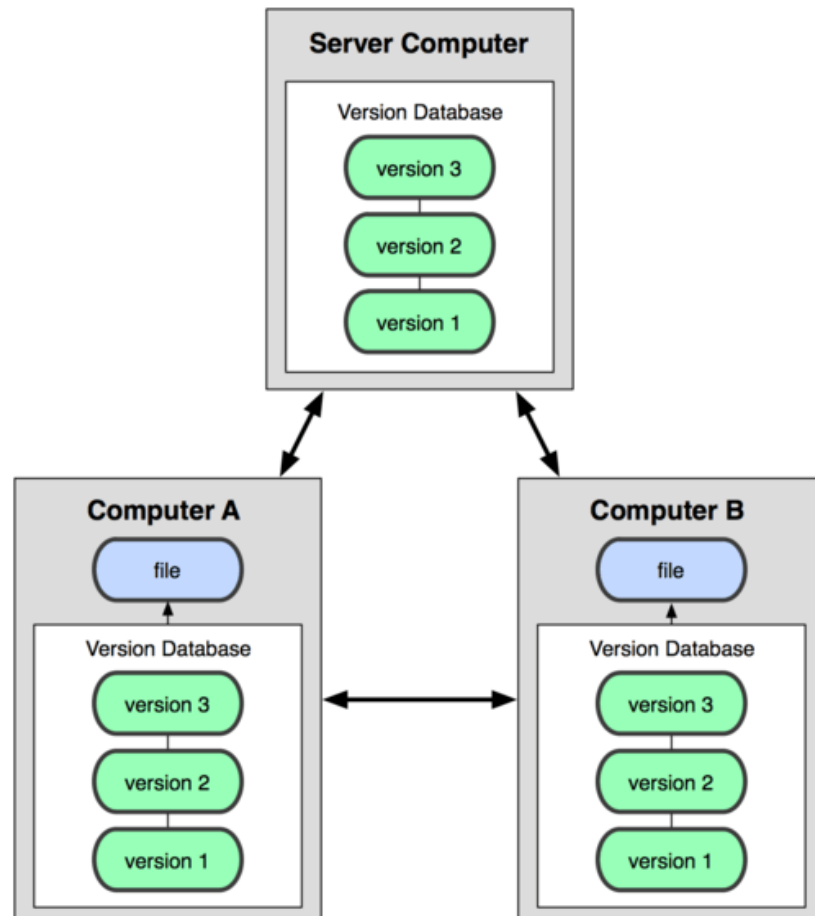
1.7.1.3. Tipos de VCS: VCS centralizado

Los VCS locales no son adecuados cuando necesitamos colaborar con otros miembros del equipo. Para solucionar este problema, también existen VCS centralizados (CVCS). Estos sistemas se instalan en un único servidor que contiene todos los archivos y sus diferentes versiones. Entonces, muchos clientes pueden conectarse a este servidor y descargar / cargar cambios en estos archivos. Esta segunda forma de controlar versiones fue estándar durante muchos años, ya que tenía grandes ventajas sobre los sistemas CVS locales, pero su principal inconveniente es que, si falla el servidor, podríamos perder todo el proyecto.



1.7.1.4. Tipos de VCS: VCS distribuido

El VCS distribuido (DVCS) surgió para resolver el principal inconveniente de CVCS. En un DVCS (como Git, Mercurial, Bazaar o Darcs), los clientes no solo se conectan al servidor, sino que también descargan todo el repositorio. Entonces, si un servidor falla, cualquiera de los repositorios locales de los clientes se puede copiar al servidor nuevamente y el proyecto se puede restaurar. Cada vez que descargamos algo del repositorio, estamos haciendo una copia de seguridad completa de los datos.



1.7.2 Git

Git fue desarrollado por el equipo de Linux una vez que rompieron la relación con *BitKeeper*, la herramienta que usaban antes para el control de versiones. A partir de las carencias observadas en esta herramienta, decidieron desarrollar algunos de los principales objetivos del nuevo sistema:

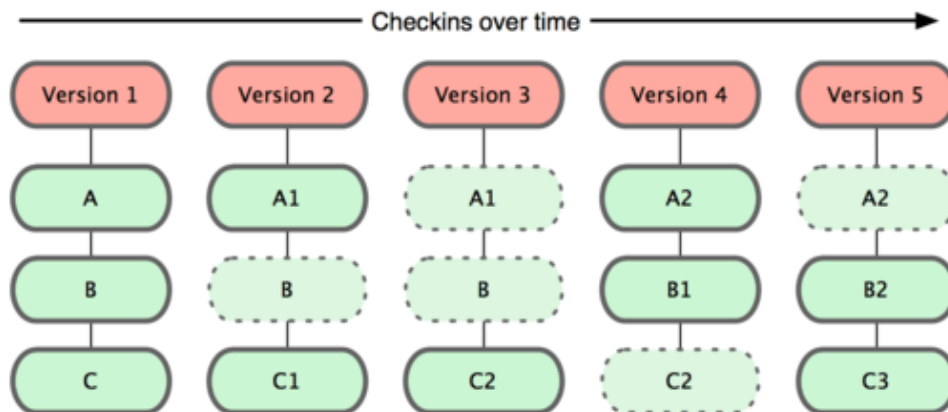
- Velocidad
- Diseño sencillo
- Fuerte apoyo al desarrollo no lineal (miles de sucursales paralelas) Completamente distribuido
-
- Adecuado para grandes proyectos (como el núcleo de Linux) Eficiencia (en
- términos de velocidad y tamaño de datos)

Desde su nacimiento en 2005, Git ha evolucionado y se ha vuelto cada vez más fácil de usar. Es realmente rápido y eficiente con grandes proyectos, y tiene un excelente sistema de ramificación.

1.7.2.1 Fundamentos de Git

1.7.2.1.1 Modelado de datos

Git almacena algún tipo de conjunto de instantáneas de su sistema de archivos, en lugar de almacenar una lista de cambios. Cada vez que cargamos un nuevo cambio, básicamente toma una foto de cada archivo en ese momento y almacena una referencia a esta instantánea. Si el archivo no ha sido modificado, entonces Git no guarda una copia, solo un enlace a una versión anterior idéntica.



Esta es una diferencia importante entre Git y casi todos los demás VCS, y hace que Git reconsidere estos aspectos de las generaciones anteriores de VCS. Por tanto, se parece más a un sistema de archivos pequeños con algunas herramientas útiles que a un VCS.

1.7.2.1.2 Trabajo local

La mayoría de las funciones de Git solo necesitan archivos y recursos locales para funcionar. Como el historial del proyecto se almacena localmente, muchas operaciones son inmediatas y nos permite trabajar en un proyecto incluso si no estamos conectados a Internet. Los cambios se almacenan localmente y, en cuanto tengamos conexión, se podrá actualizar el repositorio externo.

1.7.2.1.3 Integridad

Usos de Git *picadillo* Algoritmo SHA-1 para almacenar la información, por lo que los datos siempre se verifican y, en caso de que se modifiquen, Git lo notaría.

1.7.2.1.4 Solo agrega información

Cada operación de Git consiste en agregar alguna información, por lo que todo se puede deshacer fácilmente (la información no se borra). Después de confirmar una instantánea, la información se almacena de forma segura.

1.7.2.1.5 Estados del proyecto

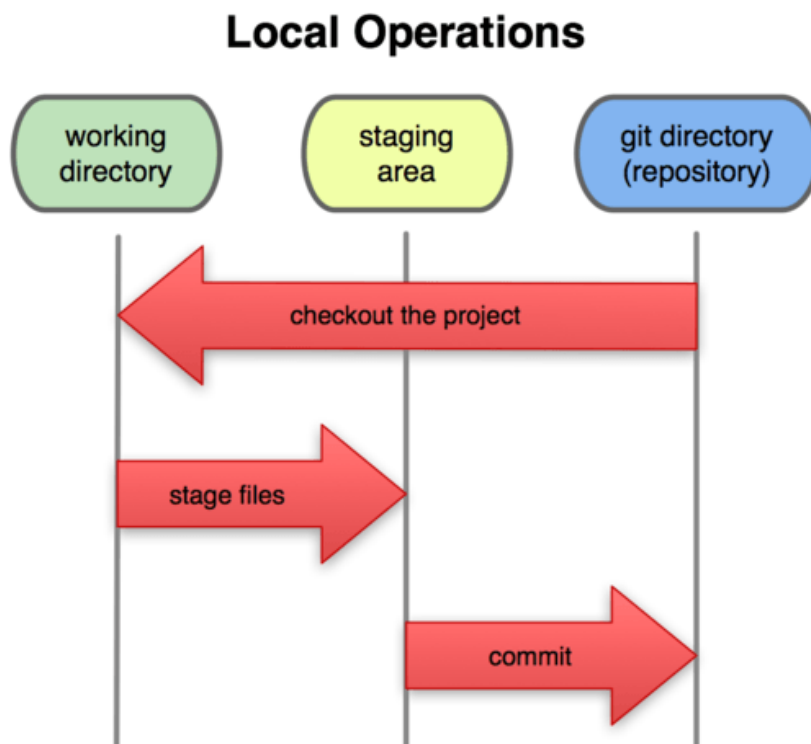
Git tiene tres estados principales en los que cada archivo de un proyecto puede ser:

- **Modificado:** los datos se han modificado localmente, pero aún no se han confirmado.

- **Escentificado:** los datos han sido etiquetados para ser enviados en la próxima confirmación.
- **Comprometido:** los datos se almacenan de forma segura en un almacenamiento local

Por lo tanto, hay tres secciones en Git:

- **Directorio de Git:** donde Git almacena los metadatos y la base de datos de los elementos del proyecto. Esta parte es lo que copiamos cuando clonamos el repositorio de otra computadora.
- **Directorio de trabajo:** es una copia de una versión del proyecto. Estos archivos se extraen de la base de datos de Git y se colocan en una carpeta, listos para ser utilizados.
- **Área de ensayo:** es un archivo simple almacenado en el directorio Git que contiene información sobre los archivos que se enviarán en la próxima confirmación. También es llamado *índice*.

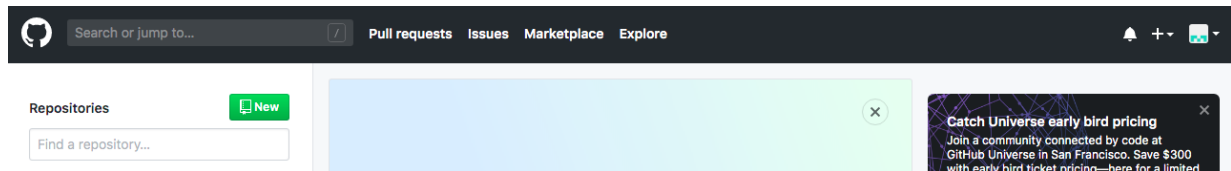


1.7.2.2 Repositorio

Un VCS se utiliza normalmente para almacenar proyectos que pueden ser desarrollados por muchas personas. Si desarrollamos el proyecto por nuestra cuenta o con otras personas, es posible que necesitemos tener una copia remota del mismo, para poder restaurarlo si hay algún problema con nuestra copia local. Para hacer esto, necesitamos tener un repositorio donde se almacenará nuestra copia remota.

Podemos crear nuestro repositorio en [GitHub](#), [Bitbucket](#) u otras plataformas. En este caso, usaremos GitHub, que es el más popular. Además, nos permite crear repositorios tanto públicos como privados.

En primer lugar, debemos registrarnos en GitHub (si aún no tenemos una cuenta). Esta es la página principal una vez que iniciamos sesión.




Luego, si queremos crear un repositorio GitHub, debemos hacer clic en el *Nuevo* en la esquina superior izquierda, y especificamos el nombre del repositorio y algunas de sus configuraciones generales: si queremos que sea público o privado, y si queremos agregar una inicial **LÉAME** archivo (recomendado).

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Owner **Repository name ***

 nachoiborrallES /

Great repository names are short and memorable. Need inspiration? How about **probable-couscous**?

Description (optional)

☒ **Public**
Anyone can see this repository. You choose who can commit.

☐ **Private**
You choose who can see and commit to this repository.

Skip this step if you're importing an existing repository.

☐ **Initialize this repository with a README**
This will let you immediately clone the repository to your computer.

Add .gitignore: **None** | Add a license: **None** ⓘ

Create repository

Si hacemos clic en el nombre del repositorio en el panel izquierdo de la vista principal, podemos ingresar a este repositorio. Desde esta página podemos, por ejemplo, clonar o descargar el repositorio, o ver el historial de confirmaciones.

Learn Git and GitHub without any code!

Using the Hello World guide, you'll start a branch, write comments, and open a pull request.

Read the guide

nachoiborrales / test

Unwatch 1
Star 0
Fork 0

<> Code
Issues 0
Pull requests 0
Projects 0
Wiki
Security
Insights
Settings

No description, website, or topics provided. Edit

Manage topics

1 commit
1 branch
0 releases
1 contributor

Branch: master
New pull request

Create new file
Upload files
Find File
Clone or download

nachoiborrales Initial commit
Latest commit #7e5a1b 3 minutes ago

README.md
Initial commit
3 minutes ago

Si hacemos clic en el *Configuraciones* enlace, podemos cambiar algunas configuraciones. Desde esta página, podemos agregar colaboradores del *Colaboradores* menú a la izquierda (esto es, otros usuarios de GitHub) a nuestro proyecto, para que también puedan hacer cambios en él. También podemos eliminar el repositorio o cambiar su visibilidad (público / privado).

nachoiborrales / test

Unwatch 1
Star 0
Fork 0

<> Code
Issues 0
Pull requests 0
Projects 0
Wiki
Security
Insights
Settings

Options

Collaborators

Branches

Webhooks

Notifications

Integrations & services

Deploy keys

Moderation

Interaction limits

Settings

Repository name

Rename

☐ **Template repository**

Template repositories let users generate new repositories with the same directory structure and files. Indicate if nachoiborrales/test can be used as a template for creating other repositories.

Social preview

Upload an image to customize your repository's social media preview.

Images should be at least 640×320px (1280×640px for best display).

[Download template](#)

Antes de clonar el repositorio, necesitamos tener una herramienta Git instalada en nuestra computadora, para que podamos confirmar, presionar y extraer los cambios. En este caso, vamos a utilizar una herramienta llamada GitKraken.

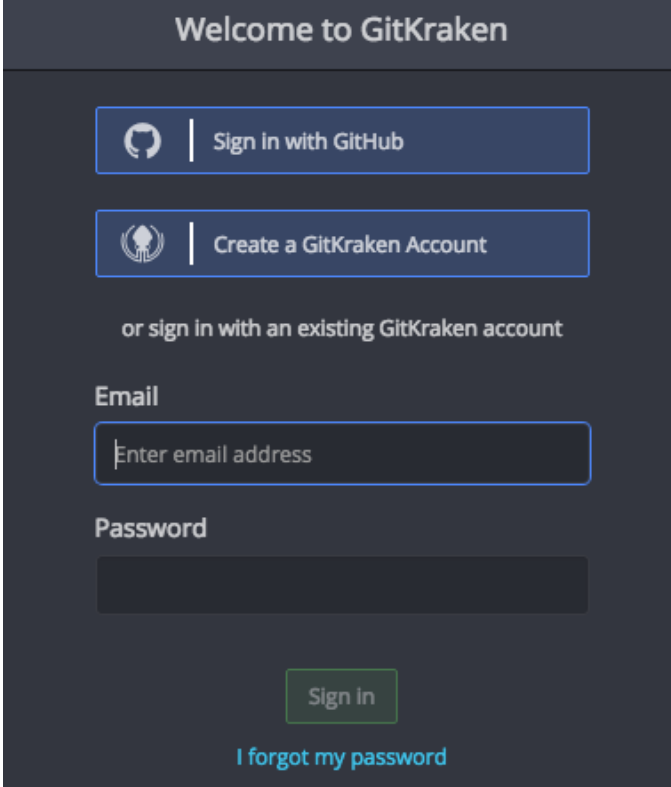
1.7.3. Herramientas de Git

Finalmente, en esta sección vamos a hablar sobre algunas herramientas útiles que podemos utilizar para trabajar con repositorios Git.

1.7.3.1. GitKraken

GitKraken es una herramienta de git gratuita que se puede ejecutar en Windows, Linux o Mac OSX. Tiene también una versión comercial, si queremos tratar con repositorios privados, o necesitamos algunas funcionalidades avanzadas.

Se puede descargar desde su [sitio web oficial](#). Después de la instalación, podemos iniciar la aplicación. La primera vez que lo lancemos, nos pedirá que nos registremos, ya sea con nuestra cuenta de GitHub (si ya tenemos una), o creando nuestra propia cuenta de GitKraken. Podemos seguir esta opción si no tenemos ninguna cuenta de GitHub, pero si nos registramos con GitHub, podemos clonar fácilmente nuestros repositorios de GitHub más adelante. Si elegimos esta opción, GitKraken le pedirá que se conecte a GitHub desde su sitio web principal.

The image shows the GitKraken login interface. At the top, it says "Welcome to GitKraken". Below this, there are two main buttons: "Sign in with GitHub" (with the GitHub logo) and "Create a GitKraken Account" (with the GitKraken logo). Underneath these, it says "or sign in with an existing GitKraken account". Then, there are input fields for "Email" (with a placeholder "Enter email address") and "Password". At the bottom, there is a "Sign in" button and a link "I forgot my password".

Welcome to GitKraken

Sign in with GitHub

Create a GitKraken Account

or sign in with an existing GitKraken account

Email

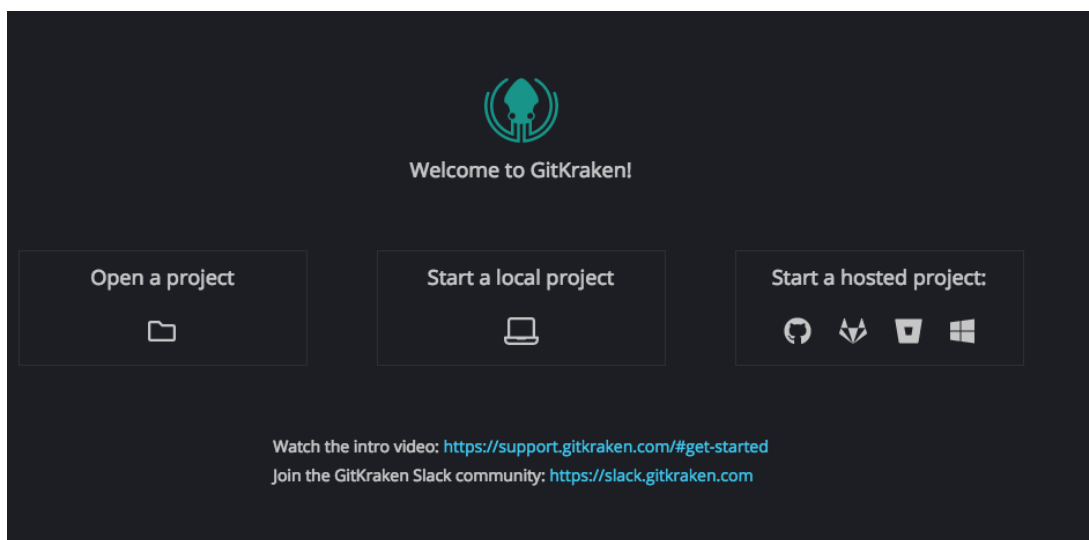
Enter email address

Password

Sign in

[I forgot my password](#)

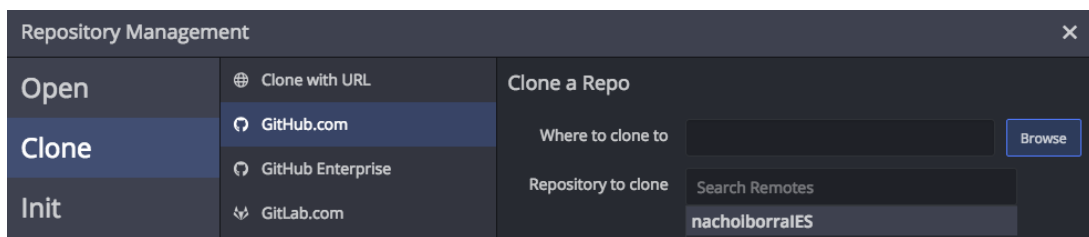
Después de cantar, podemos ver la pantalla de bienvenida:



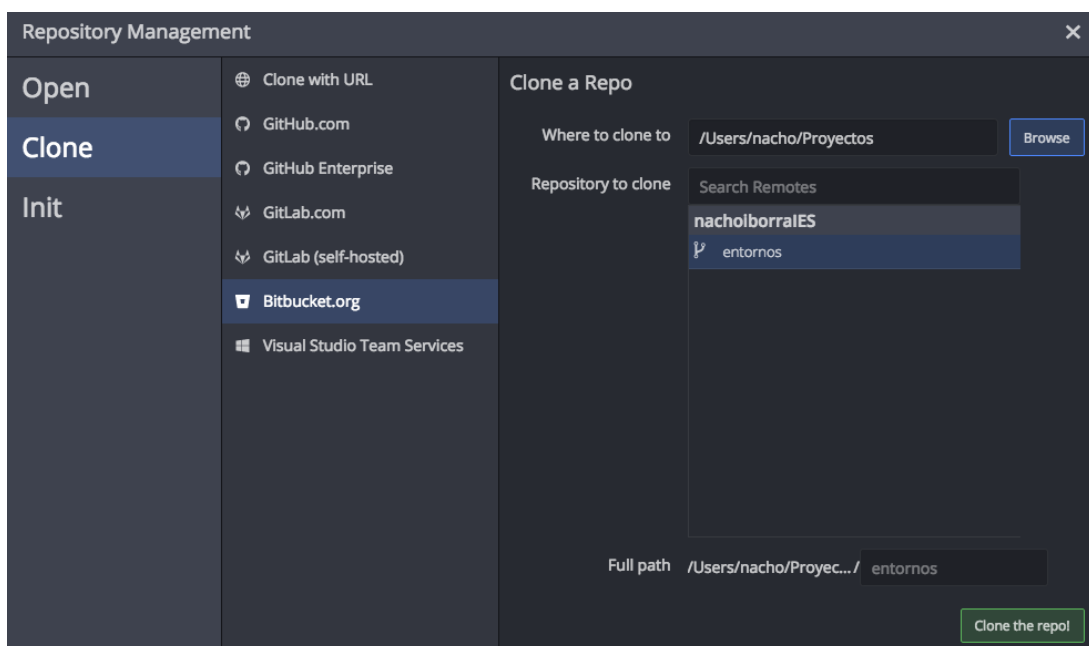
- Desde la opción de la izquierda (*Abrir un proyecto*) podemos conectarnos a un repositorio remoto (de GitHub o Bitbucket, por ejemplo) y descargarlo.
- Desde la opción del medio (*Iniciar un proyecto local*) podemos iniciar un nuevo proyecto local y crear archivos en él. Luego, podremos confirmar y cargar los cambios en un repositorio remoto.
- De la opción correcta (*Iniciar un proyecto alojado*) podemos crear un repositorio remoto en una de las plataformas permitidas (GitHub, Bitbucket y alguna otra), para que se conecte a este repositorio y lo descargue también.

Supongamos que ya hemos creado un repositorio remoto, por lo que elegimos la primera opción (*Abra un proyecto*).

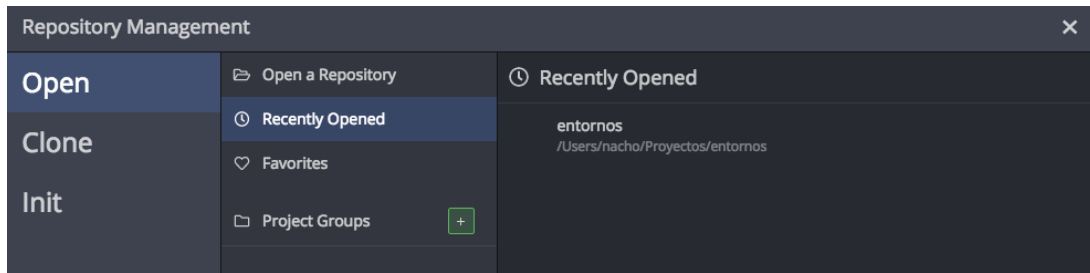
Luego, debemos elegir la plataforma git con la que conectarnos. En nuestro caso, elegimos GitHub, por lo que debemos hacer clic en el botón correspondiente.



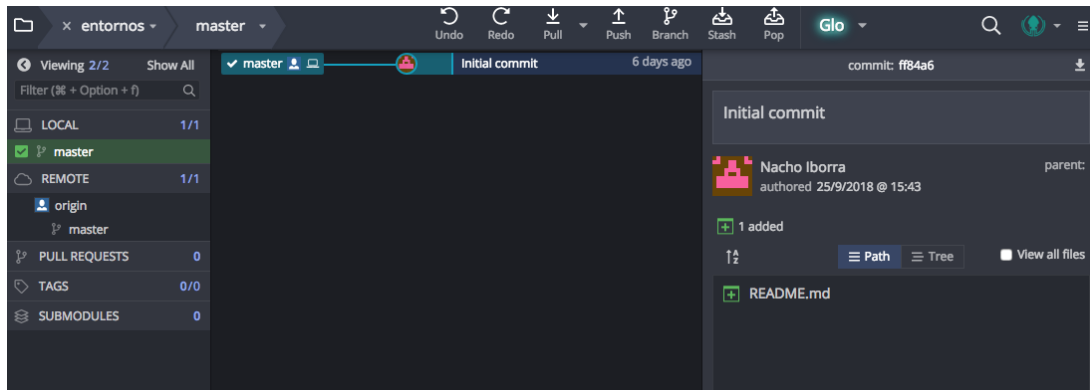
Debemos elegir el repositorio a clonar, y la carpeta donde queremos descargar el proyecto, en el panel derecho. Luego, podemos hacer clic en el *¡Clona el repositorio!* botón en la esquina inferior derecha.



Una vez clonado nuestro repositorio, podemos explorarlo desde el *Abrir un proyecto* opción, o haciendo clic en el icono de la carpeta en la esquina superior izquierda de la ventana de GitKraken.



Después de elegir la carpeta del proyecto (o el propio proyecto si lo hemos abierto recientemente), podemos ver su contenido:

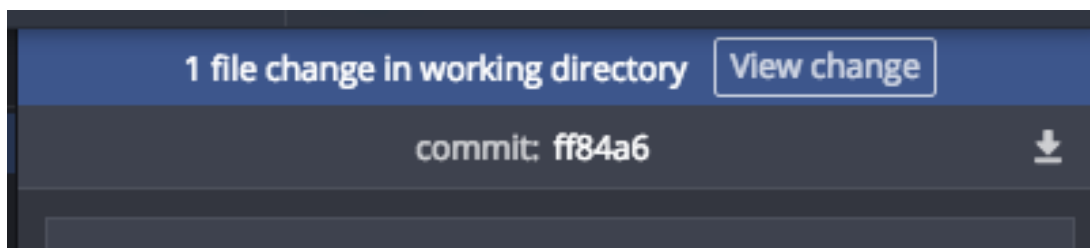


En el panel derecho podemos ver el contenido del proyecto (archivos y carpetas). Podemos verlos como caminos o como un árbol, pinchando en los botones correspondientes de esta parte derecha. En el panel del medio tenemos el historial de operaciones sobre el repositorio (confirmaciones, etc.). Finalmente, en el panel de la izquierda tenemos una lista de ramas disponibles.

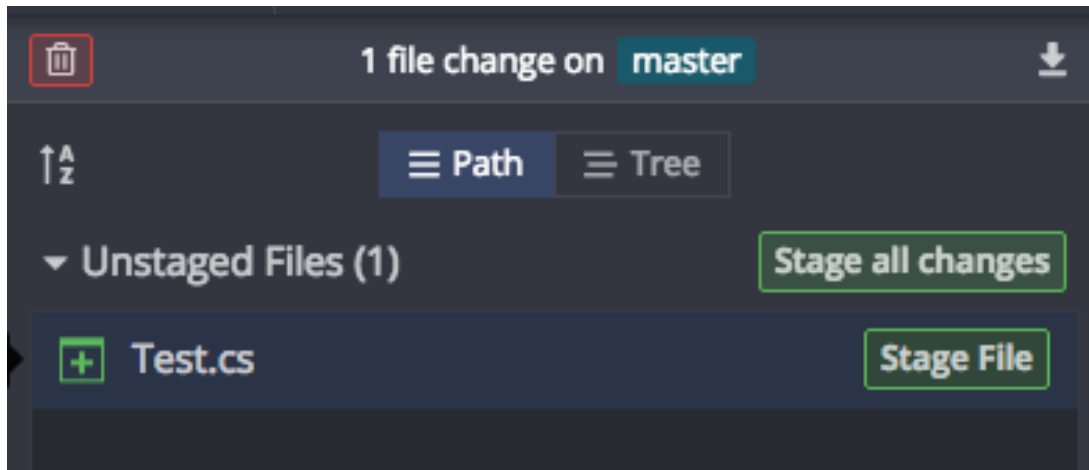
1.7.3.1.1. Comprometiendo nuevos cambios

Si agregamos contenido nuevo al proyecto (o editamos / eliminamos contenido existente), todos los cambios se mostrarán en el panel derecho automáticamente.

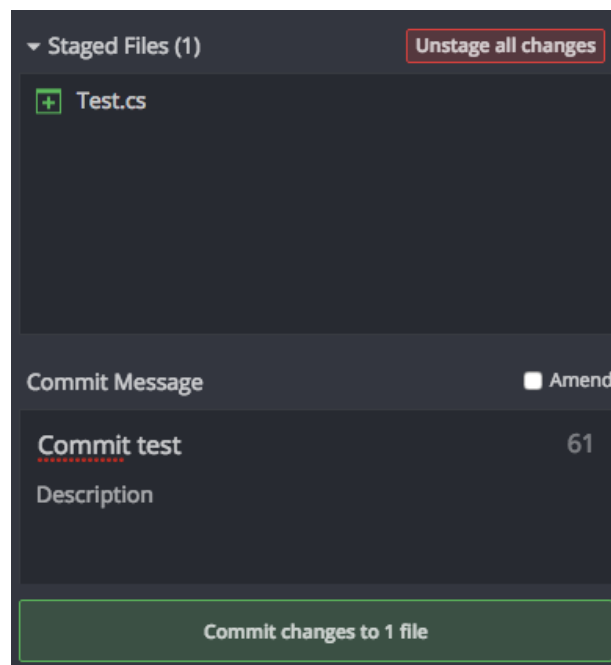
Por ejemplo, si agregamos un nuevo archivo fuente, podemos verlo de esta manera:



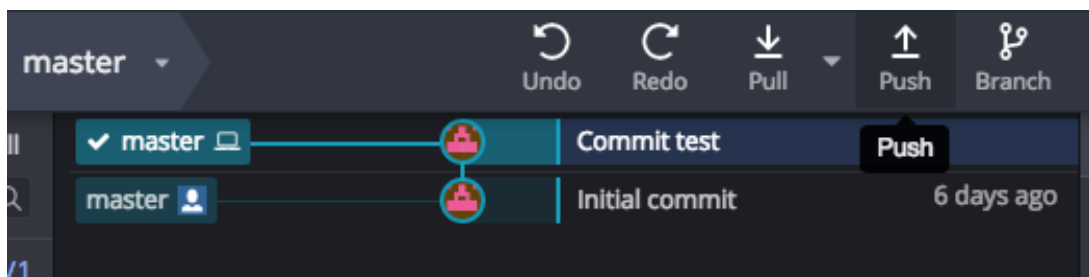
Luego, debemos hacer clic en el *Ver cambios* para ver los archivos que han cambiado. Podemos organizarlos individualmente (es decir, marcarlos para que se confirmen en la próxima operación de confirmación), o prepararlos todos a la vez haciendo clic en el botón *Organizar todos los cambios* botón.



El siguiente paso consiste en confirmar los cambios. Debemos agregar un comentario de confirmación (explicando las novedades de esta confirmación) y luego hacer clic en el *Cometer cambios* archivos en la parte inferior.



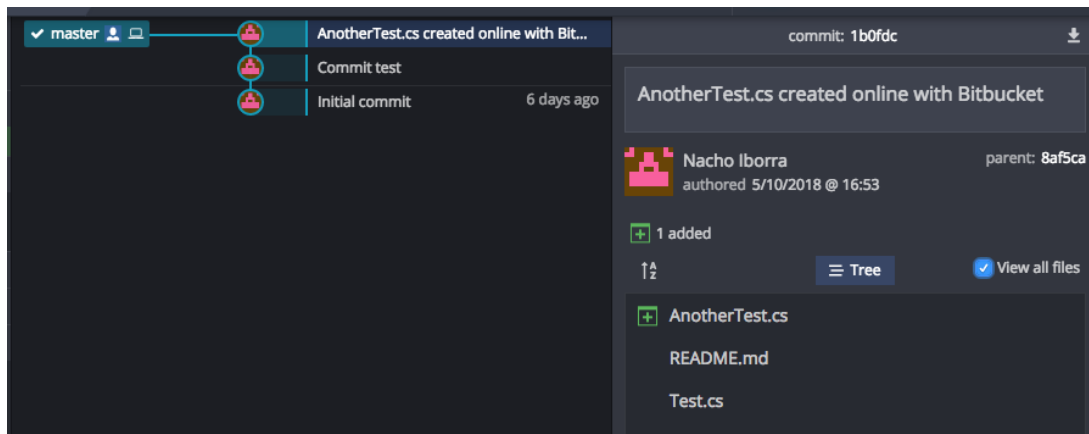
Finalmente, necesitamos enviar los cambios al repositorio remoto. Para hacer esto, vamos al panel central, elegimos el compromiso que queremos empujar y luego hacemos clic en el *empujar* en la barra de herramientas superior.



1.7.3.1.2. Extrayendo nuevos contenidos del servidor

Si estamos trabajando en equipo, o si solo queremos actualizar los cambios a otra computadora, es posible que necesitemos extraer el contenido del repositorio. Si, por ejemplo, alguien ha subido nuevos archivos al repositorio (o

archivos existentes modificados), si hacemos clic en el *Halar* de la barra de herramientas superior, veremos los archivos involucrados. También podemos elegir la confirmación correspondiente en la lista (panel central) y luego ver todos los archivos en el panel derecho



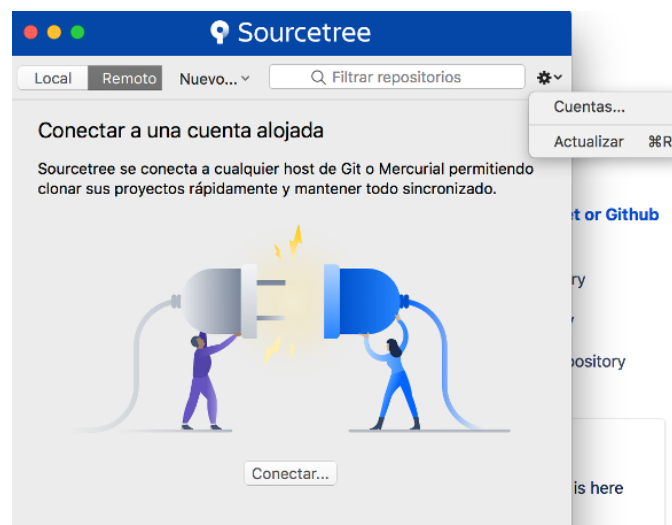
1.7.3.2. SourceTree

También podemos usar una herramienta llamada SourceTree para aprender a usar Git. Puedes conseguirlo en [es sitio web o oficial](#). Esta GUI simplifica la interacción con los repositorios de Git, como lo hace GitKraken, por lo que no necesitamos aprender todos los comandos de la consola de Git. Está disponible en sistemas Windows y MacOSX (no hay una versión de Linux disponible).

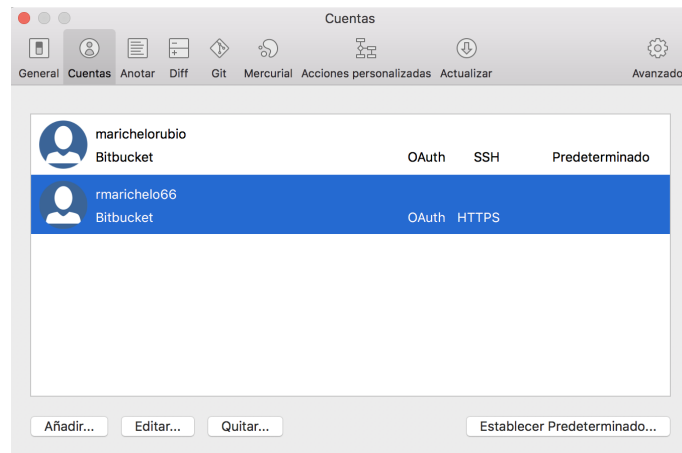
Una vez que descarguemos SourceTree (elija el instalador de acuerdo a su sistema operativo), simplemente lo instalamos y ejecutamos. Luego, nos pedirá que ingresemos a nuestra cuenta de Atlassian (la que acabamos de crear al registrarnos en Bitbucket o GitHub). Después de iniciar sesión, veremos una pantalla de configuración la primera vez que ejecutemos la aplicación. Podemos omitir esta configuración (haga clic en el *Omitir la configuración* en este caso).

[Documentación de Bitbucket sobre cómo usar SourceTree](#)

Desde la pantalla principal de SourceTree, podemos conectarnos a nuestra (s) cuenta (s) y administrar los repositorios. Debemos elegir *Remoto* en las pestañas superiores izquierdas, y luego elegimos *Cuentas* en el menú superior derecho.



Luego, veremos una pantalla de cuentas, y podremos agregar nuestra cuenta haciendo clic en el *Añadir* botón en la esquina inferior izquierda. Después de hacer clic en el *Conectar cuenta* botón, veremos nuestra cuenta en la lista de cuentas.

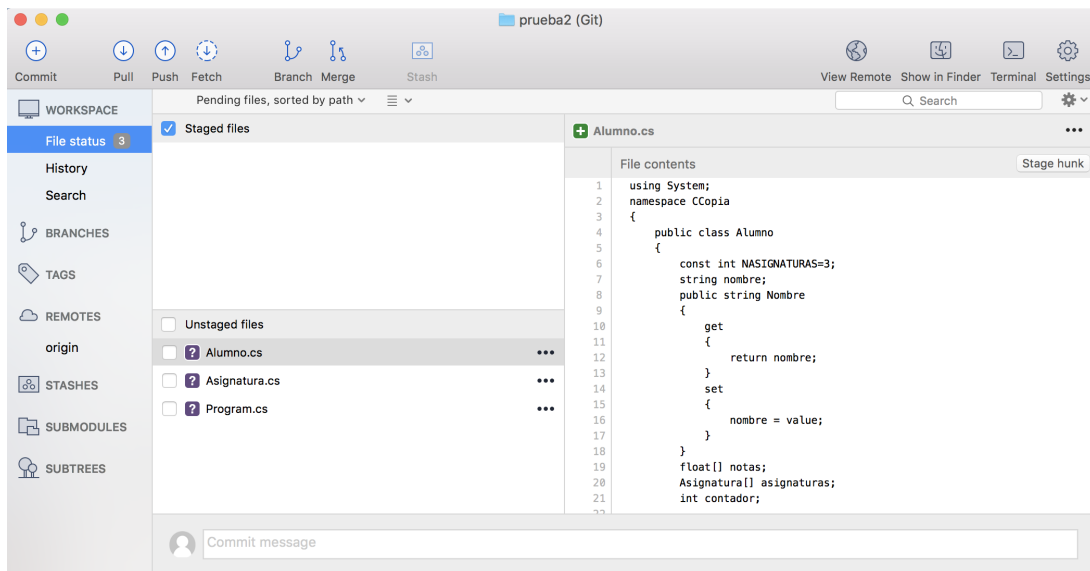


Si cerramos el *cuentas* pantalla y volver a la pantalla principal, veremos nuestro repositorio remoto y el *Clon* opción:

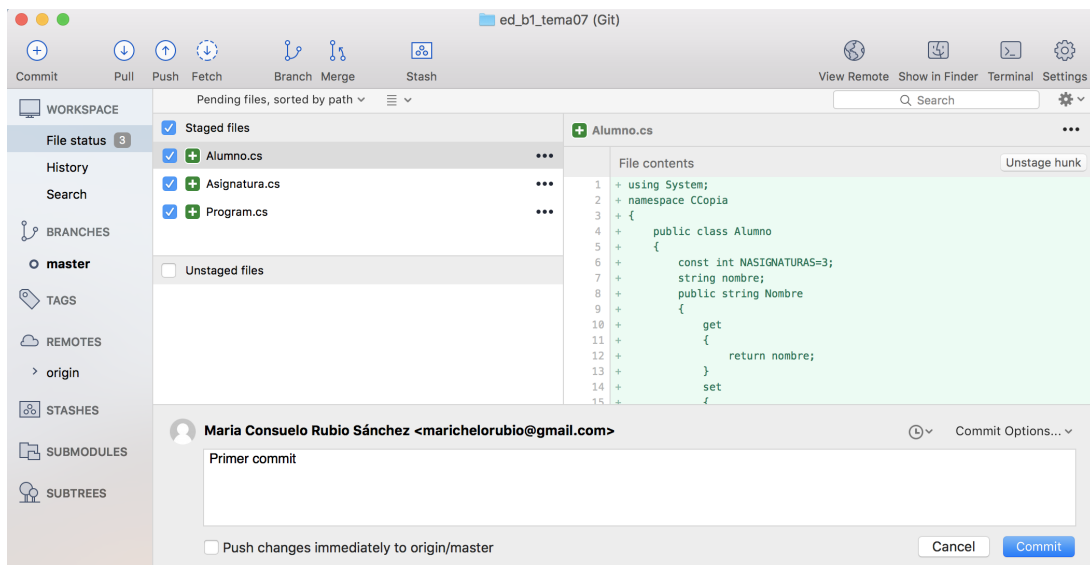


Si hacemos clic en el *Clon* opción, luego elegiremos la ubicación para descargar el repositorio. Luego, podemos comenzar a trabajar en el proyecto usando Git para el control de versiones. Solo necesitamos movernos a la carpeta donde se ha descargado.

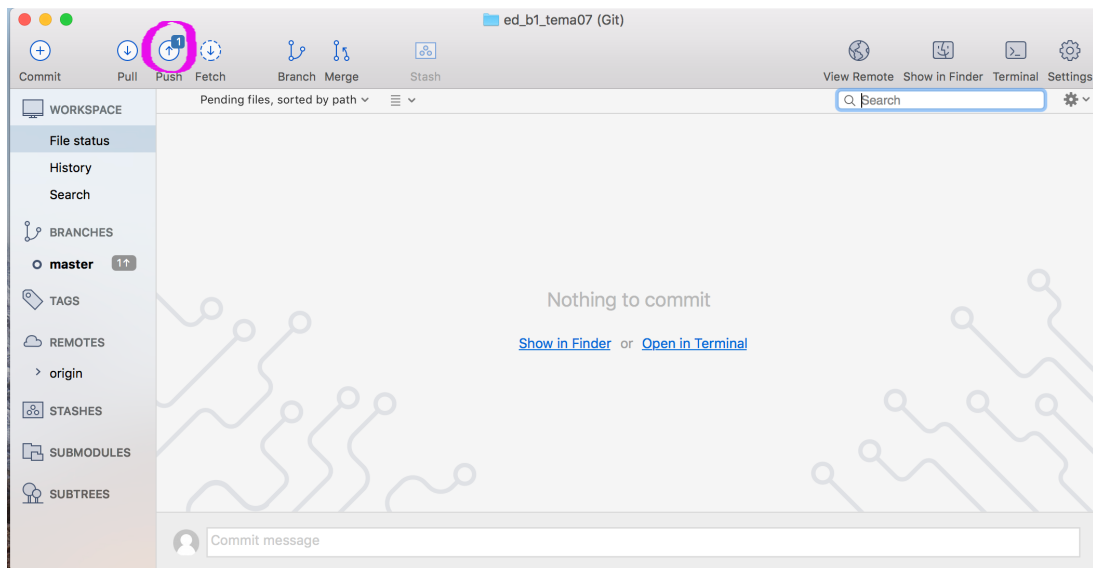
Nuestros primeros pasos deben estar enfocados en compartir el proyecto con el resto del equipo. Si abrimos SourceTree y hacemos clic en el *local*, luego vemos nuestros repositorios locales clonados. Si elegimos uno de ellos, veremos esta pantalla:



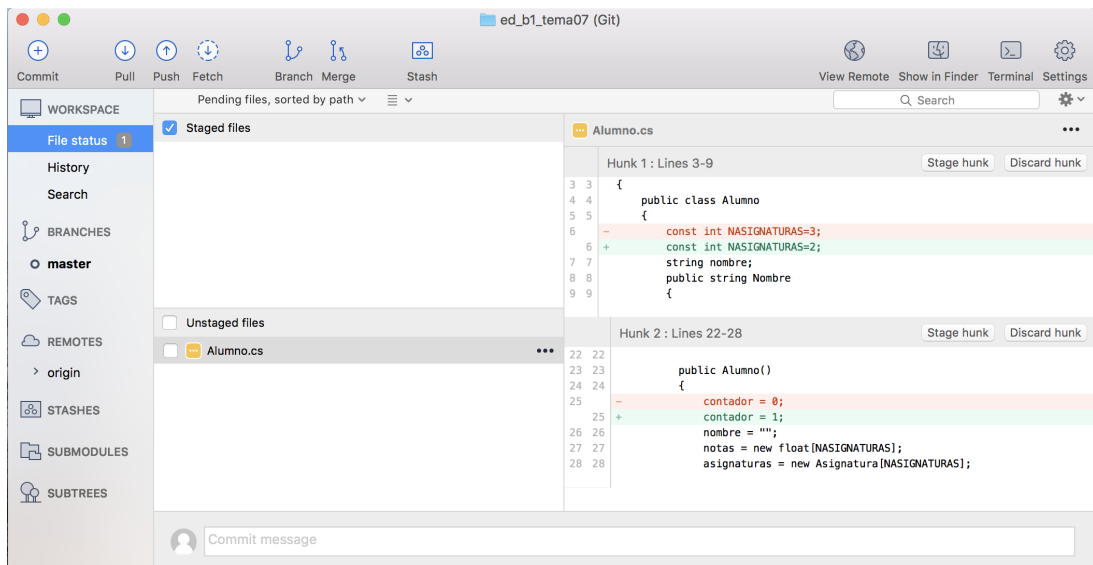
Podemos ver los archivos que aún no están organizados (*sin escenificar*). Si los revisamos, se convertirán en escenarios, de modo que si hacemos un *cometer*, serán respaldados. Entonces solo necesitamos hacer un *empujar* operación para cargar los cambios al repositorio remoto. Cuando hacemos el *cometer*, debemos ingresar un comentario que describa los cambios que hemos realizado en el proyecto.



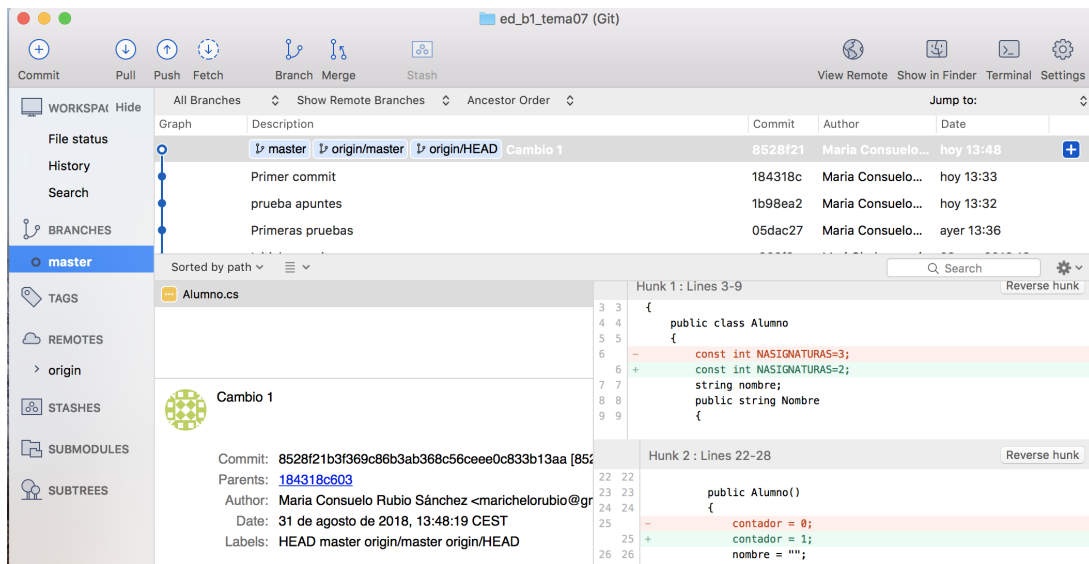
Tenga en cuenta que cuando confirmamos los cambios, no se cargarán en el repositorio remoto hasta que elijamos el *empujar* opción.



Si modificamos nuestros archivos, podemos ver los cambios / diferencias entre las versiones en el panel derecho.



Una vez que los cambios se han confirmado y cargado (enviado), podemos ver todos los cambios en el *Historia* lengüeta.



El panel muestra la fecha, hora, usuario y comentario para cada confirmación.

Ir a buscar La opción nos permite comprobar los cambios en los archivos remotos, para que podamos actualizar nuestra versión local a través de la

Halar opción. Cuando hacemos un *Halar* Operación puede haber conflictos entre nuestras copias locales y las copias remotas. SourceTree nos notificará acerca de estos conflictos para que podamos resolverlos antes de extraer los archivos.

Rama nos permite crear una rama en el desarrollo del proyecto. De esta manera, podemos trabajar en la rama maestra original del proyecto y en otra actualización al mismo tiempo. Por ejemplo, si estamos preparando una nueva versión de software que requiere muchos cambios, podemos crear una nueva rama para esta nueva versión y comenzar a desarrollarla mientras continuamos con el desarrollo de la versión actual. Solo cuando la nueva versión esté completa, podremos *unir* ambas ramas, entonces ambas versiones se combinarán en una sola versión.

Ejercicios propuestos:

Para los siguientes ejercicios necesita tener un *GitHub* cuenta. Así que regístrese si no tiene ninguna cuenta ([aquí](#) es el sitio web oficial).

1.7.3.1. Crea un repositorio público llamado *Programas Cpp* en su cuenta de GitHub. Clónelo en una carpeta local usando GitKraken o SourceTree. Luego, copie el siguiente programa en un archivo fuente llamado *hola.cpp* dentro de esa carpeta.

```
#include <iostream>

using namespace std;

int principal ()
{
    cout << "Hola Mundo" ;
    return 0 ;
}
```


Una vez que haya terminado de copiar el archivo, confirme y envíe los cambios al repositorio remoto.

1.7.3.2. Realice algunos cambios en el archivo fuente anterior. Por ejemplo, cambie el texto que se imprime en el `cout` línea. Entonces, haz un segundo `cometer` y su correspondiente `empujar` para actualizar los cambios.

1.7.3.3. Comparta su proyecto con un compañero de clase y pídale que actualice algunos de los archivos. Luego, actualice estos archivos en su repositorio local a través de un `Halar` operación.

1.7.3.3. Integrar Git con un IDE (código de Visual Studio)

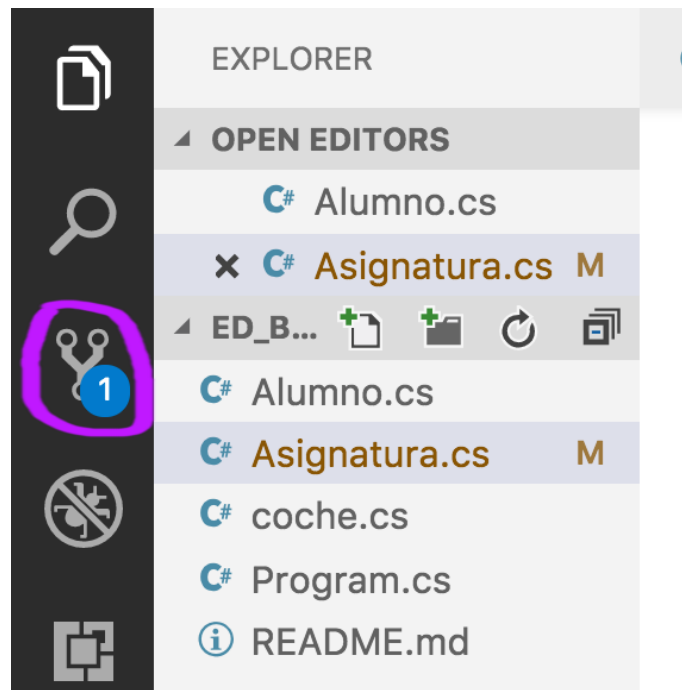
La mayoría de los IDE actuales nos permiten usar Git internamente, por lo que podemos realizar las operaciones principales de Git (confirmar, empujar, tirar, etc.) desde el propio IDE. Veamos cómo funciona con Visual Studio Code.

En primer lugar, debemos clonar nuestro repositorio remoto. Abrimos la paleta de comandos (`Control + Mayús + P` , o `Cmd + Mayús + P` en sistemas Mac) y escribimos "clonar". Entonces veremos el `clon de git` comando, nosotros seleccionémoslo y luego se nos pedirá que ingresemos la URL remota del repositorio. Debemos pegar allí la URL de nuestro repositorio remoto de GitHub o BitBucket (podemos verlo pinchando en el `Clon` botón).

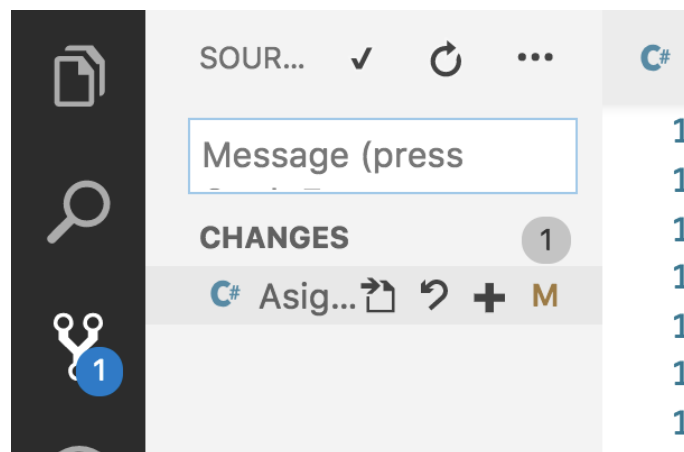


NOTA: Si `clon de git` Visual Studio no reconoce el comando, es posible que deba instalar Git antes. Para hacer esto, puedes seguir [estas instrucciones](#) , dependiendo de su sistema operativo.

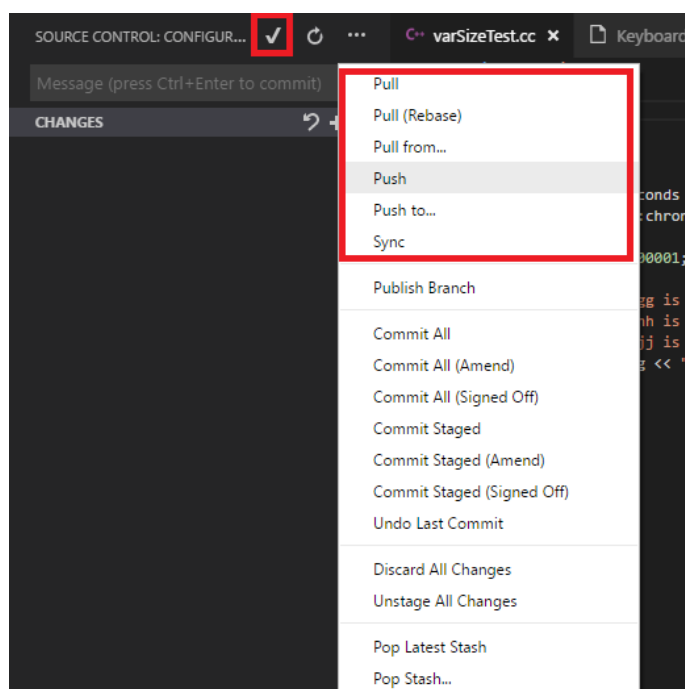
Luego, veremos los archivos del repositorio en Visual Studio Code, y si realizamos algún cambio en cualquier archivo, se marcará automáticamente con una 'M' junto al nombre del archivo. Esto significa que el archivo ha sido modificado desde la última versión cargada en el repositorio. Luego podemos preparar el archivo haciendo clic en el *Fuente de control* icono (ver imagen siguiente). De esta forma, podemos ver los cambios realizados y las opciones de Git.



En cuanto a las opciones, tenemos *etapa* (+), deshacer cambios, *cometer* (✓) y opciones más generales en el menú superior.



Si queremos empujar o tirar los cambios hacia / desde el repositorio, debemos hacer clic en el *mas opciones* botón (...) en la barra de herramientas superior izquierda y elija la opción adecuada.



Ejercicios propuestos:

1.7.3.4. Crea un nuevo repositorio público llamado *CppProgramsVSCode* en su cuenta de GitHub y luego clonarlo desde VS Code.

1.7.3.5. Repita los ejercicios 1.7.3.1 y 1.7.3.2 usando VSCode.

1.7.4. Aprender más

Puede leer más sobre Git y las herramientas de Git en los siguientes enlaces

- <https://git-scm.com/book/es/v1>
- <https://support.gitkraken.com/integrations/github/>
- <https://fluence.atlassian.com/get-started-with-sourcetree/get-started-with-sourcetree-847359026.html>