por Mari Chelo

# Entornos de Desarrollo
# Bloque 1
# Tema 7: Introducción al control de versiones. Herramientas git

## 1.6.1. Definición

El 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 se puedan recuperar versiones específicas más adelante. Se puede usar no solo para la gestión de versiones de software sino para cualquier tipo de documento.

## 1.6.2. Utilidad

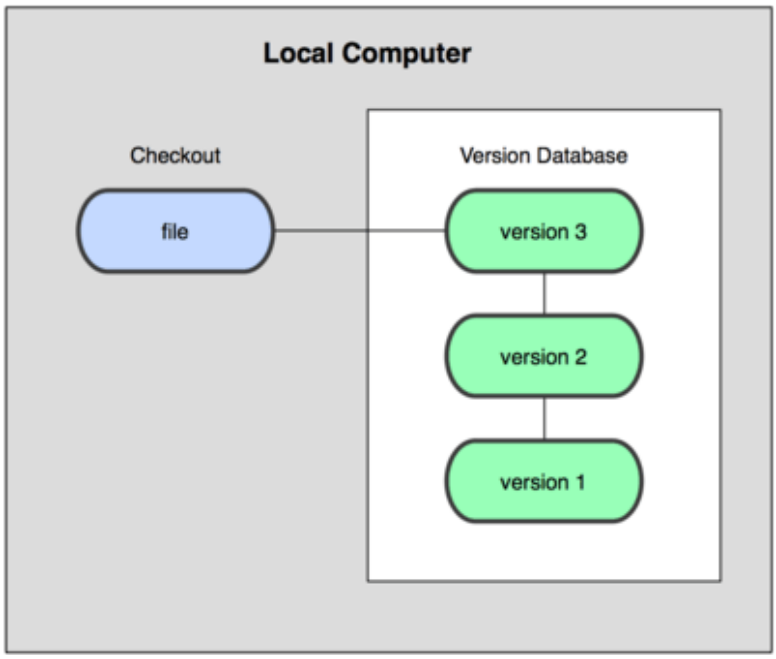
Un sistema de control de versiones (*Version Control System VCS*) permite revertir archivos o incluso el proyecto entero a un estado anterior, comparar cambios a lo largo del tiempo, ver quién modificó por última vez algo que puede estar causando problemas y mucho más. Usar un VCS también significa generalmente que si fastidias o pierdes archivos, puedes recuperarlos fácilmente.

## 1.6.3 Sistemas de control de versiones locales

Un método que se utiliza mucho por su sencillez es simplemente realizar una copia de seguridad del proyecto periódicamente indicando la fecha y hora de esa copia. De esa forma en caso de tener algún problema con posterioridad a la copia, se podrán consultar e incluso recuperar archivos anteriores al error.

La ventaja de este método como he mencionado antes es su sencillez, el inconveniente es que el control se realiza manualmente con el incremento de la posiblidad de errores. Por ejemplo una persona puede olvidar en que directorio se encuentra y modificar un fichero de la copia en vez del de la versión actual.

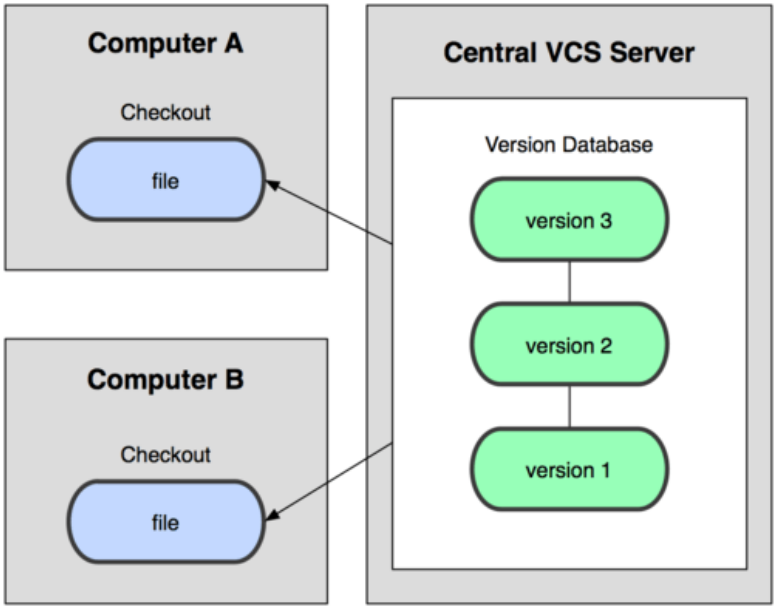
Para hacer frente a estos problemas se crearon algunas herramientas para gestionar una base de datos donde quedaban registrados los cambios de los diferentes archivos. Una de las herramientas de control de versiones más popular fue un sistema llamado rcs, que todavía podemos encontrar en muchos de los ordenadores actuales. Esta herramienta funciona básicamente guardando conjuntos de parches (es decir, las diferencias entre archivos) de una versión a otra en un formato especial en disco; puede entonces recrear cómo era un archivo en cualquier momento sumando los distintos parches.




## 1.6.4 Sistemas de control de versiones Centralizados

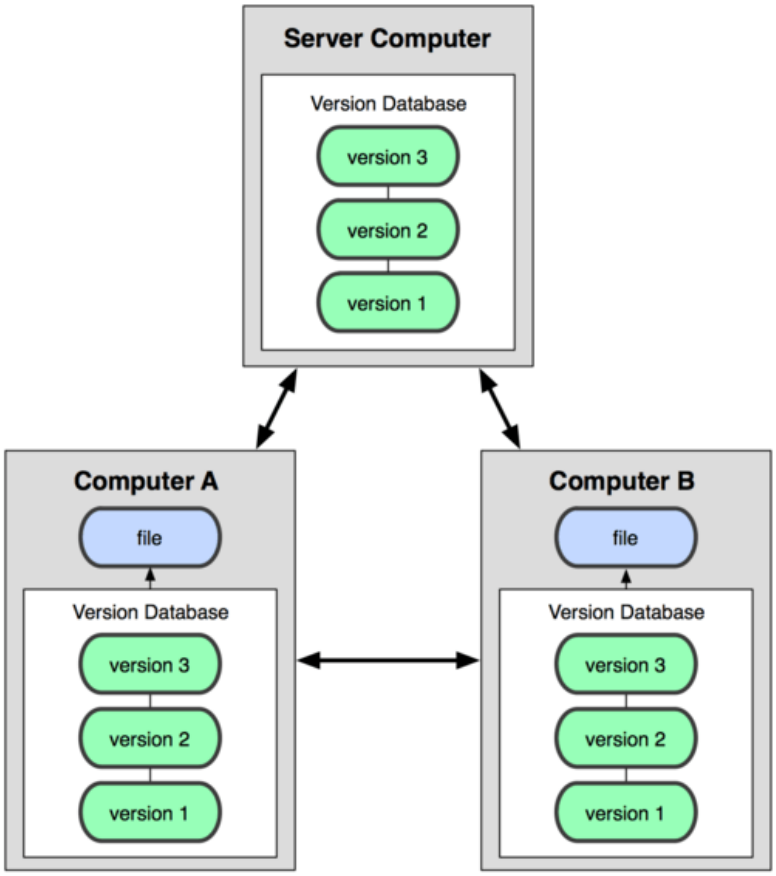
Los VSC locales presentan un gran inconveniente cuando se quiere realizar trabajo colaborativo. Para solventar este problema, se desarrollaron los sistemas de control de versiones centralizados (Centralized Version Control Systems o CVCS). Estos sistemas tienen un único servidor que contiene todos los archivos en sus distintas versiones, y varios clientes que descargan los archivos desde ese lugar central. Durante muchos años éste fue el estándar para el control de versiones.

Este estándar supuso un avance frente a los sistemas locales pero presenta el gran inconveniente de que en caso de que el servidor falle, durante el tiempo del fallo nadie puede trabajar o si falla el disco duro y no se tienen las copias de seguridad adecuadas se podría perder el proyecto al completo.



## 1.6.5 Sistemas de control de versiones Distribuidos

Estos sistemas (Distributed Version Control Systems o DVCS) surgieron precisamente para solventar el inconveniente de los centralizados. En un DVCS (como Git, Mercurial, Bazaar o Darcs), los clientes no sólo descargan la última instantánea de los archivos sino que replican completamente el repositorio. Así, si un servidor falla, y estos sistemas estaban colaborando a través de él, cualquiera de los repositorios de los clientes puede copiarse en el servidor para restaurarlo. Cada vez que se descarga una instantánea, en realidad se hace una copia de seguridad completa de todos los datos.




## 1.6.6 Git

*Git* fue desarrollado por la comunidad que desarrollaba el núcleo de Linux tras su ruptura con la compañía propietaria de la herramienta que usaban anteriormente *BitKeeper*. A partir de las carencias que detectaron de esta última, decidieron que los algunos de los objetivos del nuevo sistema fueran los siguientes:

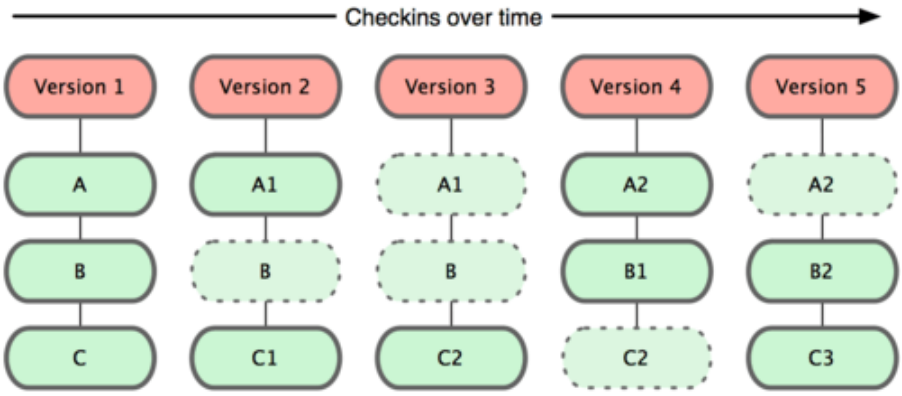
- Velocidad
- Diseño sencillo
- Fuerte apoyo al desarrollo no lineal (miles de ramas paralelas)
- Completamente distribuido
- Capaz de manejar grandes proyectos (como el núcleo de Linux) de manera eficiente (velocidad y tamaño de los datos)

Desde su nacimiento en 2005, Git ha evolucionado y madurado para ser fácil de usar y aún conservar estas cualidades iniciales. Es tremendamente rápido, muy eficiente con grandes proyectos, y tiene un increíble sistema de ramificación (branching) para desarrollo no lineal.

## 1.6.6.1 Fundamentos de Git

### 1.6.6.1.1 Modelado de datos

Git a diferencia de otros sistenas modela sus datos más como un conjunto de instantáneas de un mini sistema de archivos, en vez de guardar una lista de cambios en los archivos. Cada vez que confirmas un cambio, o guardas el estado de tu proyecto en Git, básicamente se 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.




Esta es una distinción importante entre Git y prácticamente todos los demás VCSs. Hace que Git reconsidere casi todos los aspectos del control de versiones que muchos de los demás sistemas copiaron de la generación anterior. Esto hace que Git se parezca más a un mini sistema de archivos con algunas herramientas tremendamente potentes construidas sobre él, que a un VCS.

### 1.6.6.1.2 Opera en local

La mayoría de las operaciones en Git sólo necesitan archivos y recursos locales para funcionar. Como la historia del proyecto está en local la mayoría de las operaciones parecen prácticamente inmediatas, además de permitir trabajar en un proyecto aunque se esté desconectado. Se acumulan los cambios en local y cuando vuelve la conexión se actualizan en el repositorio remoto.

### 1.6.6.1.3 Integridad

Utiliza el algoritmo *hash* SHA-1 para guardar toda la información, con lo que está siempre verificado y es imposible cambiar nada sin que Git lo detecte.

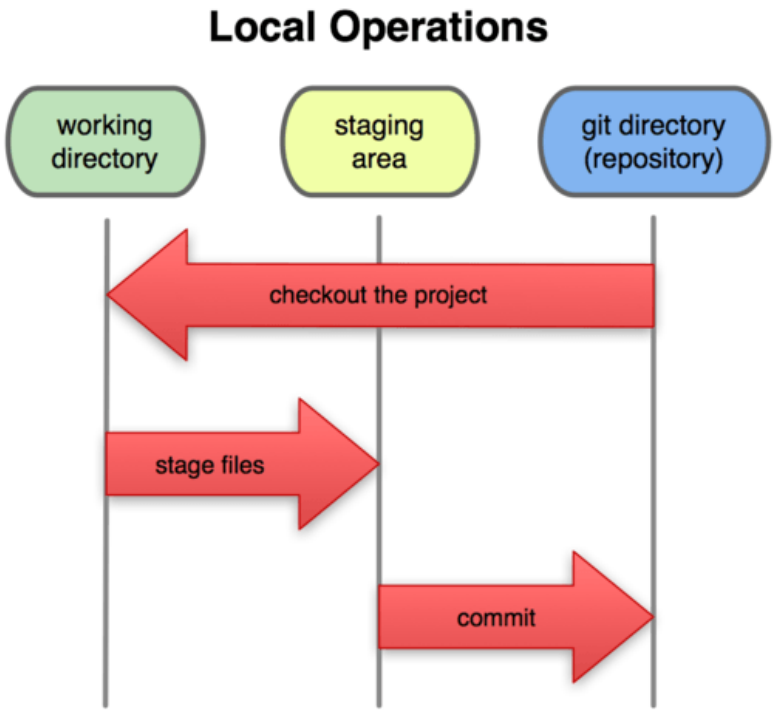
### 1.6.6.1.4 Solo añade información

Todas las operaciones de Git suponen añadir información con lo que es muy difícil conseguir que el sistema haga algo que no se pueda deshacer, o que de algún modo borre información. Siempre pueden borrarse o estropearse datos no confirmados como en cualquier VCS, pero después de confirmar una instantánea en Git es muy difícil perder información.

#### 1.6.6.1.5 Los estados del proyecto

Git tiene tres estados principales en los que se pueden encontrar los archivos de un proyecto: **confirmado** (*committed*), **modificado** (*modified*), y **preparado** (*staged*). Confirmado significa que los datos están almacenados de manera segura en la base de datos local, modificado significa que se ha modificado el archivo pero todavía no se ha confirmado a la base de datos y preparado significa que se ha marcado un archivo modificado en su versión actual para que vaya en la próxima confirmación. Esto implica que existen tres secciones en Git:

- **El directorio de Git** (*Git directory*): donde Git almacena los metadatos y la base de datos de objetos del proyecto. Es la parte que se copia cuando se clona un repositorio desde otro ordenador.
- **El directorio de trabajo** (*working directory*): Es una copia de una versión del proyecto. Estos archivos se sacan de la base de datos comprimida en el directorio de Git, y se colocan en disco para que se puedan usar.
- **El área de preparación** (*staging area*): Es un archivo simple, generalmente en el directorio de Git, que almacena información acerca de lo que va a ir en la próxima confirmación. Denominado también como índice.



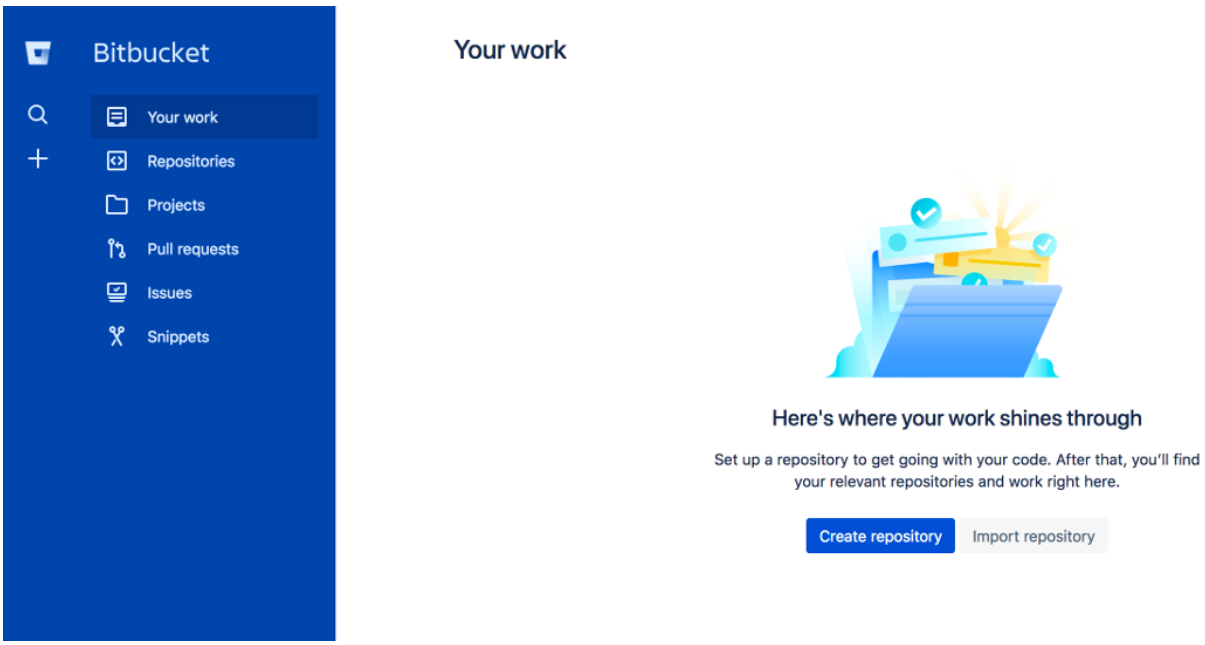

## 1.6.6.2 Repositorio

Recordemos ahora para que se usa un sistema de control de versiones, y por tanto para que vamos a utilizar Git. Normalmente el control de versiones se usa para desarrollar un proyecto más o menos grande

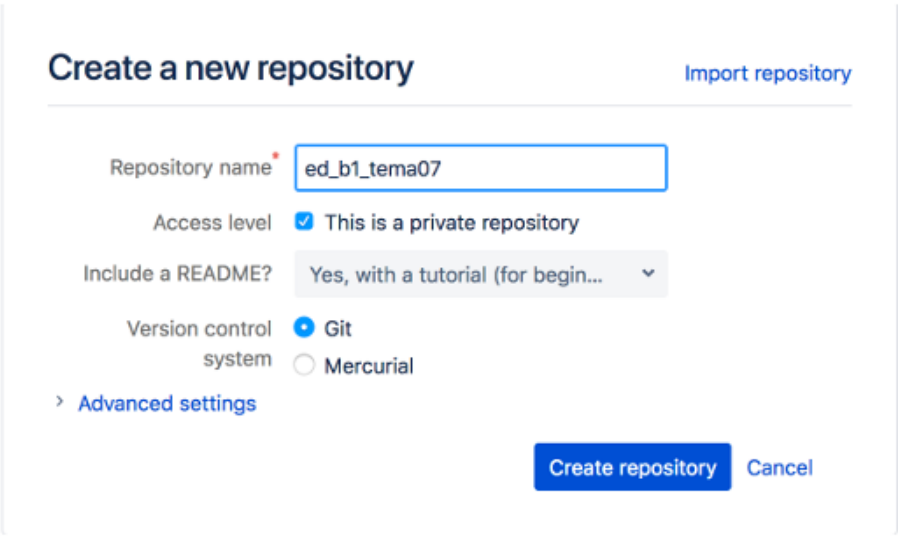
en solitario o lo que es más común de forma colaborativa. Tanto si solo una persona va a desarrollar el proyecto, como si va a ser un equipo, lo normal es querer tener una copia remota para evitar disgustos. Mediante git vamos a realizar el control de versiones de nuestro proyecto y para ello necesitamos primero tener un repositorio donde esté nuestra copia remota.

Podemos crear nuestro repositorio en GitHub, Bitbucket u otros. Para esta prueba lo vamos a hacer en Bitbucket que nos permite tener repositorios privados de forma gratuita frente a GitHub que requiere pago para repositorios privados o registrarse como estudiante.

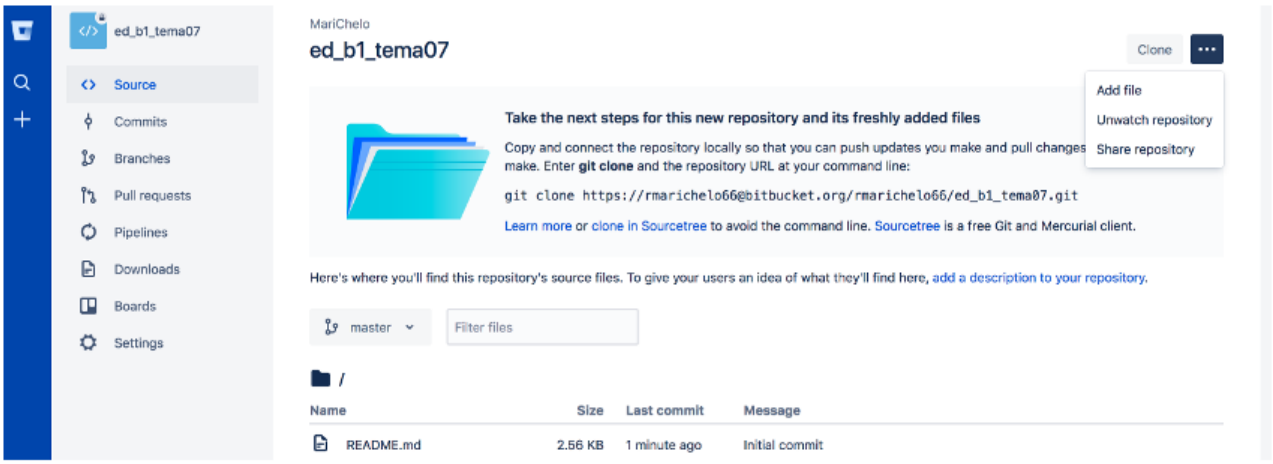
Para crear el repositorio en Bitbucket, entramos en la página https://bitbucket.org/product y nos creamos una cuenta. Una vez creada la cuenta, nos pedirá un nombre de usuario para crear nuestro espacio para los repositorios y nos saldrá una página para crear nuestro nuevo repositorio.




Creamos nuestro repositorio privado para usarlo con Git.




Esto nos creará nuestro repositorio remoto que podremos compartir con el resto del equipo.




Como se puede ver en la imagen anterior tenemos arriba a la derecha un menú que nos permitirá compartir el repositorio remoto y el botón de clonar que sería el siguiente paso a realizar para empezar a trabajar.

Clonar el repositorio se corresponde con el comando de **git clone** (como se puede ver en la imagen) y lo que hace es hacer la copia en local del repositorio.

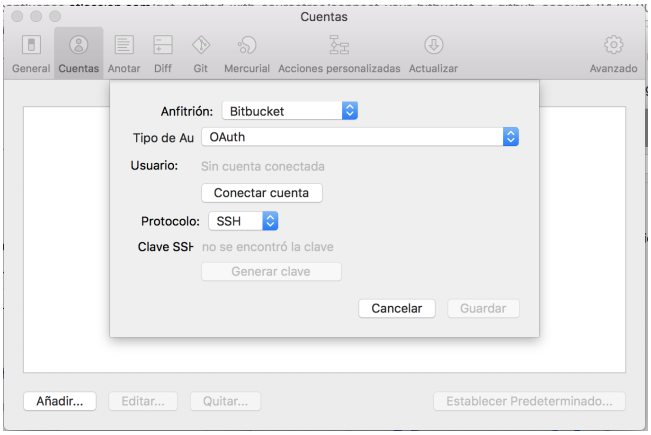
Antes de clonar el repositorio necesitamos tener Git instalado en nuestro equipo. En nuestro caso lo vamos a hacer con la herramiente Sourcetree.

## 1.6.6.3 Interfaz gráfico Sourcetree

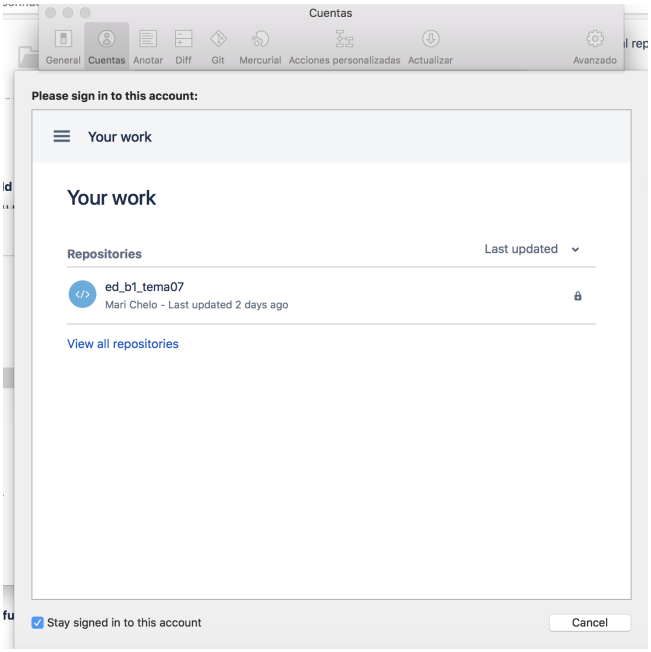
Para aprender a utilizar Git, vamos a hacerlo mediante un interfaz gráfico llamado Sourcetree. https://www.sourcetreeapp.com/ Este interfaz gráfico simplifica la interacción con los repositorios Git para que no necesitemos aprendernos todos los comandos de consola de Git. Descargamos de https://www.sourcetreeapp.com/ el instalador adecuado a nuestro sistema operativo y lo instalamos. Una vez instalado, al abrirlo nos pedirá que accedamos con nuestra cuenta Atalasian (justo la que hemos creado antes para el repositorio) y que nos conectemos a Bitbucket o GitHub (esta aplicación puede utilizarse con ambos). En nuestro caso elegiremos el Bitbucket Cloud y nos conectamos a nuestra cuenta. Documentación de la página de Bitbucket

Una vez hecho esto ya podemos darle a clonar el repositorio que habíamos creado. O podemos clonar desde el mismo SourceTree. Para hacerlo desde SourceTree, debemos añadir la cuenta (o cuentas) para que nos conecte con el repositorio.

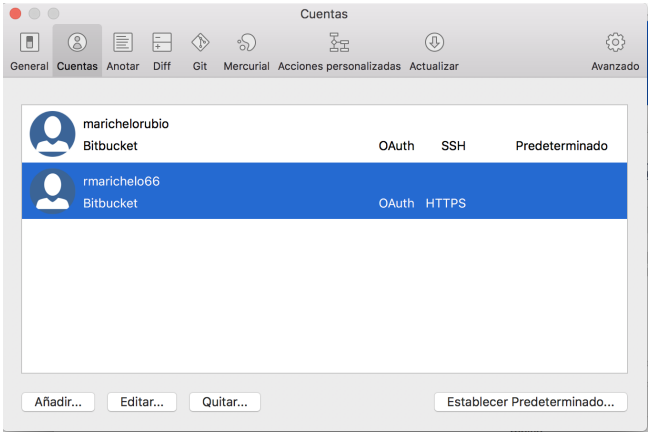

Elegimos remoto de las pestañas de arriba a la izquierda y le damos a cuentas en el menú de arriba a la derecha. Tras esto nos sale una pantalla de cuentas donde dándole al botón de añadir de abajo a la izquierda nos permitirá conectar con nuestra cuenta para acceder al repositorio remoto. Al darle al botón de conectar cuenta, nos accede al repositorio remoto y nos abre la web de Bitbucket que ya hemos visitado cuando creamos el repositorio.



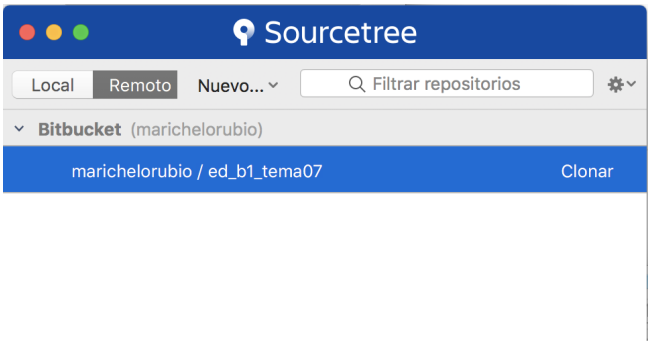

Desde aquí entramos en nuestro repositorio y accederemos a la pantalla donde estaba el botón de clonar. Y desde aquí también podemos compartir con el resto del equipo de desarrollo el repositorio con la opción ***share*** que aparece en el menú contiguo al botón de clonar.




También podemos clonar nuestro repositorio directamente desde SourceTree, una vez añadida y conectada la cuenta.

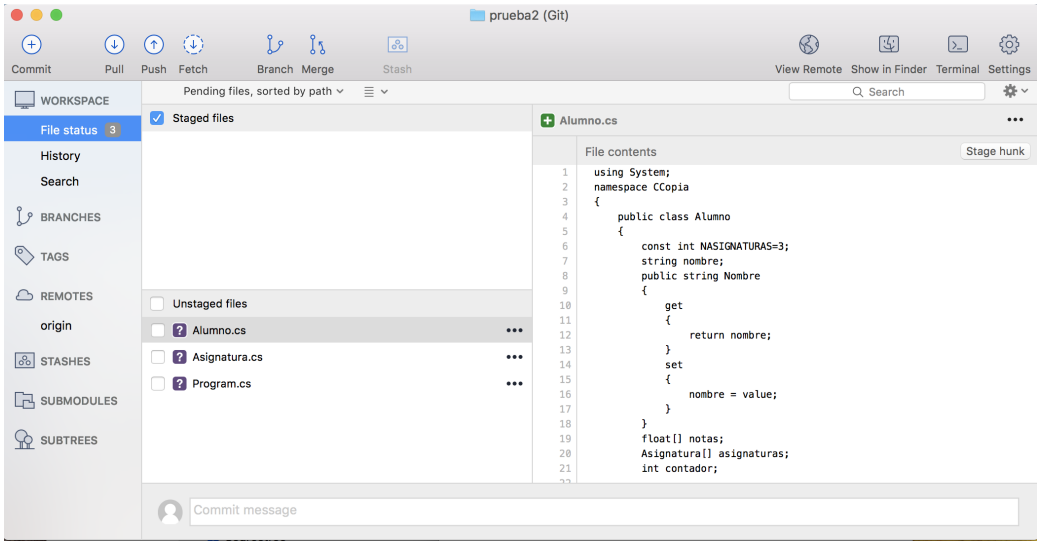



Cerrando el apartado de cuentas volvemos a la pantalla principal de Bitbucket y vemos como aparece el repositorio remoto y una opción de clonar.

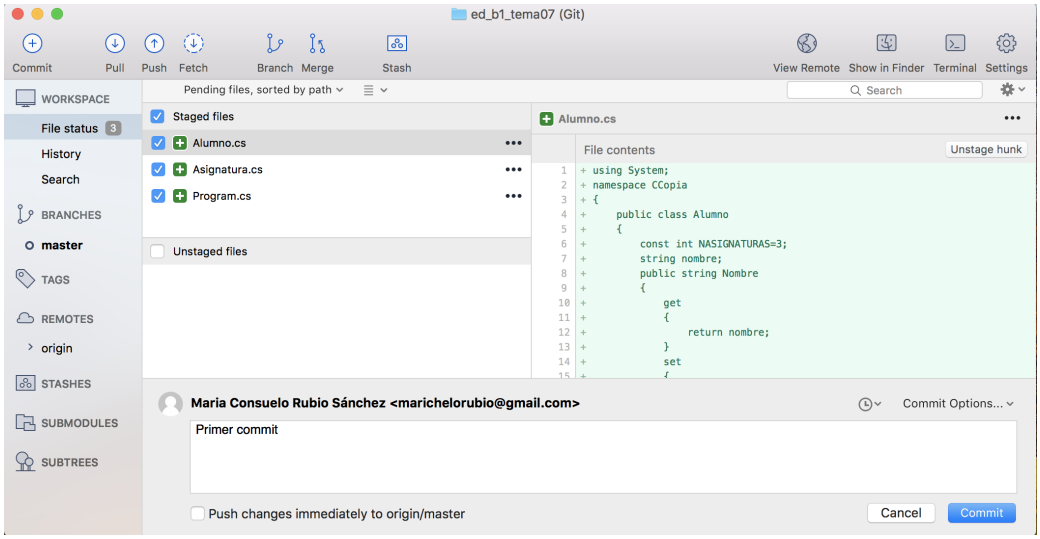



Una vez tenemos clonado el repositorio ya podemos trabajar en nuestro proyecto utilizando Git para el control de versiones. Nos habrá aparecido en nuestro equipo en la ubicación especificada la carpeta correspondiente al repositorio y ahí es donde a partir de ahora trabajaremos con nuestro proyecto.

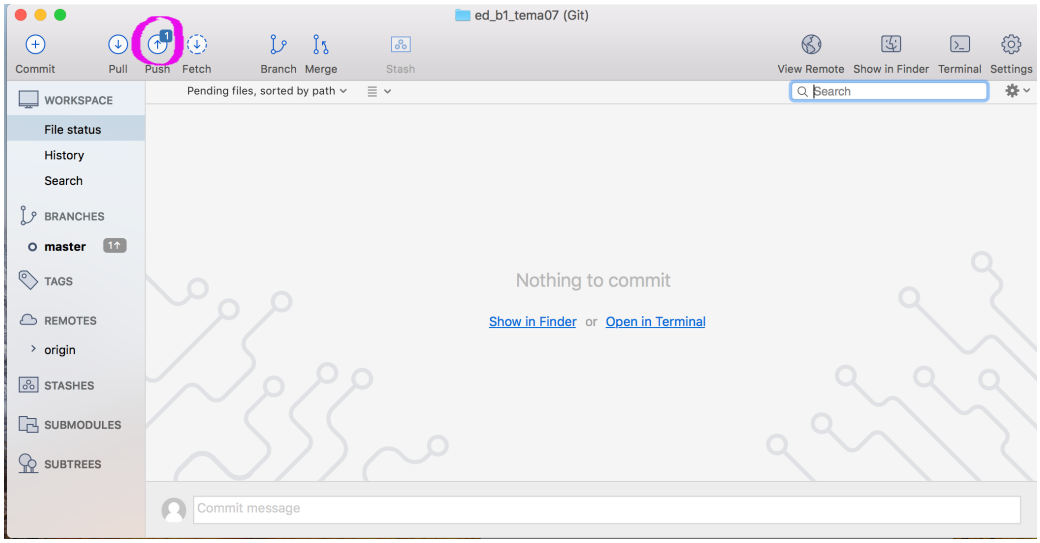
Lo primero que debemos hacer es incorporar nuestro proyecto al repositorio local y subirlo al repositorio remoto para poder compartir el proyecto con el equipo. Si abrimos el SourceTree y pulsamos en el botón local veremos los repositorios que tenemos clonados y pulsando en el que queremos operar nos saldrá una pantalla como esta.



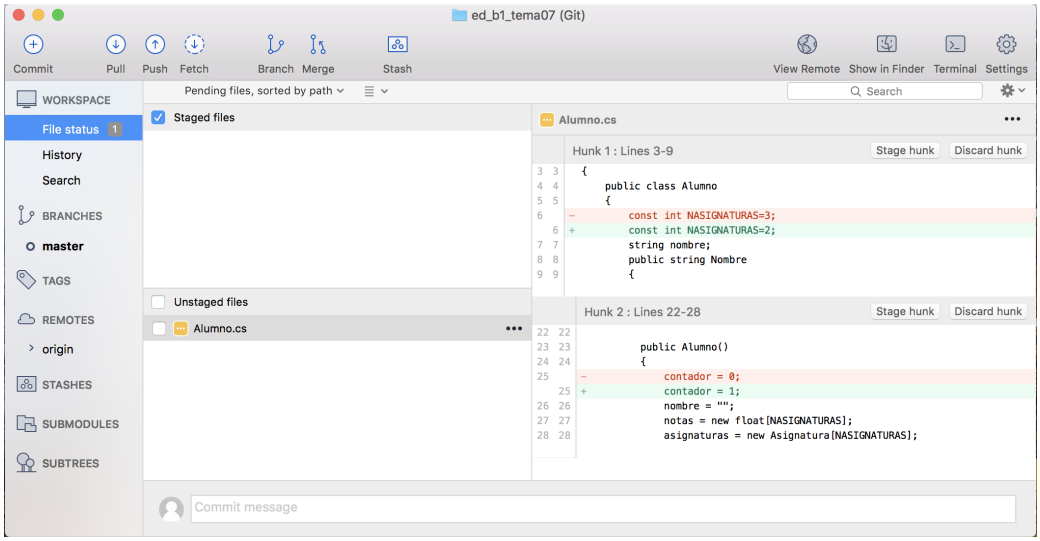

Donde observamos los archivos que hemos incorporado y están en estado no preparado (***unstaged***). Si marcamos su casilla de verificación, pasarán a preparados de forma que si hacemos un ***commit*** pasarán a estar confirmados y sólo nos faltaría hacer ***push*** para que se suban al repositorio remoto. Al hacer ***commit*** se nos pide un texto donde normalmente se indica cuales son los cambios incorporados en esta versión del proyecto.



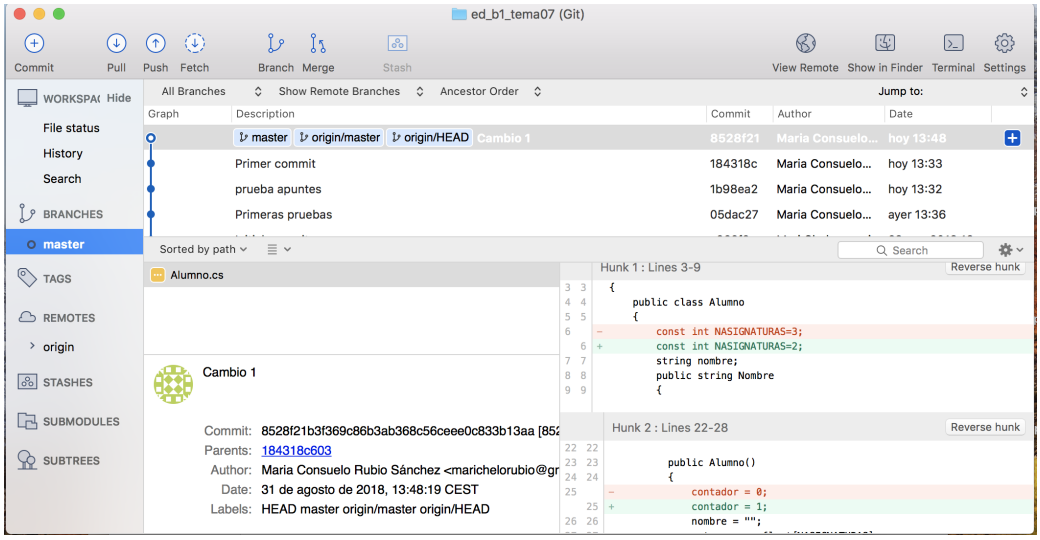

Una vez hecho el commit se nos queda marcado que hay cambios pendientes de subir y al pulsar a *push* se copiarán esos cambios en el repositorio remoto.




Si realizamos modificaciones en nuestros fuentes se quedan reflejados dichos cambios.




Y una vez confirmados y subidos podemos ver todos los cambios realizados por todo el equipo en la pestaña de *History*.




Como se puede observar se quedan reflejados la fecha, la hora, el usuario y el comentario proporcionado por el usuario al hacer el ***commit***.

La opción ***fetch*** nos permite consultar los cambios realizados por otros compañeros del equipo de desarrollo y que no tienes en tu repositorio local y si los quieres incorporar a tu carpeta local deberás utilizar la opción de ***pull***. Cuando hacemos ***pull*** podemos encontrar conflictos entre los cambios realizados por otro usuario y los cambios realizados por ti, SourceTree informará de dichos conflictos para que sean solventados antes de realizar el ***pull***.

***Branch*** nos permite iniciar una bifurcación en el desarrollo del proyecto de forma que se siga trabajando en la rama original y en otra actualización al mismo tiempo. Por ejemplo para nuevas versiones de software que requieren muchos cambios se abre una nueva rama para la nueva versión y se siguen corrigiendo errores y mandando esta actualizaciones a los usuarios mientras la nueva versión se sigue desarrollando sin interferir en el funcionamiento de la anterior. Sólo cuando la nueva versiónes esta comprobada se puede realizar el ***merge*** de las dos ramas y se combinarán las versiones en una sola a partir de ese momento.

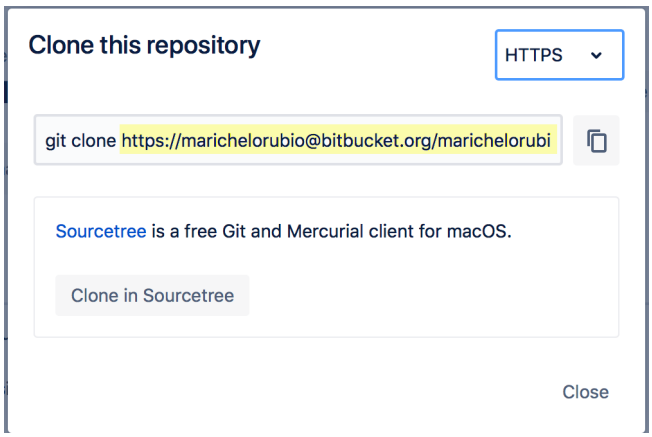
> Ejercicios propuestos:

> 1.6.6.1. Copia algunos fuentes (por ejemplo .cs de programación) a tu nuevo repositorio local. Y abre Sourcetree para ver lo sucedido. Realiza un primer ***commit*** y ***push*** para subir estos fuentes a tu repositorio remoto.
>
> 1.6.6.2. Realiza algunos cambios en alguno de los fuentes y observa como Sourcetree los detecta. Actualiza el repositorio remoto con estos cambios.
>
> 1.6.6.3. Comparte tu repositorio con un compañero y pídele que realice cambios en alguno de los fuentes. Consulta estos cambios e incorpóralos a tu repositorio local.

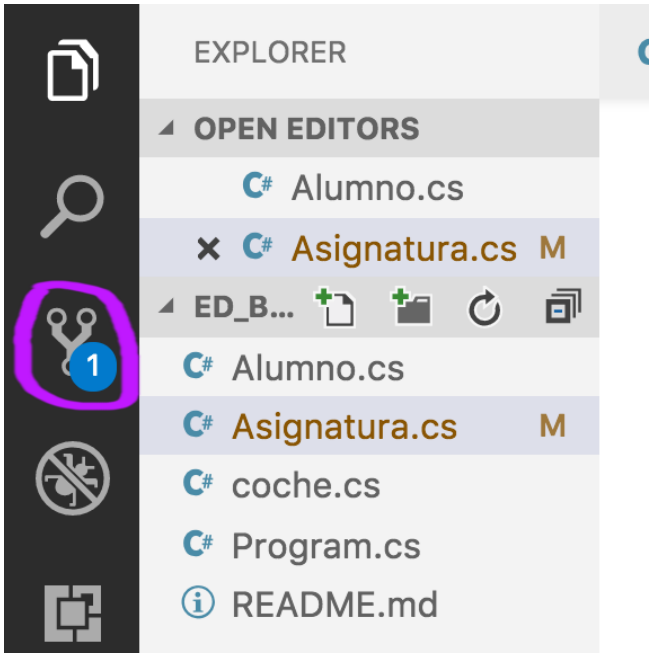
## 1.6.6.4 Integración de Git en un Entorno de Desarrollo (Visual Studio Code)

La mayoría de los entornos de desarrollo incorporan la integración con Git de forma que directamente desde el mismo editor con el que estás trabajando puedas hacer las principales acciones de Git para llevar el correcto control de versiones de tu proyecto. En concreto vamos a ver como integrar Git con Visual Studio Code.

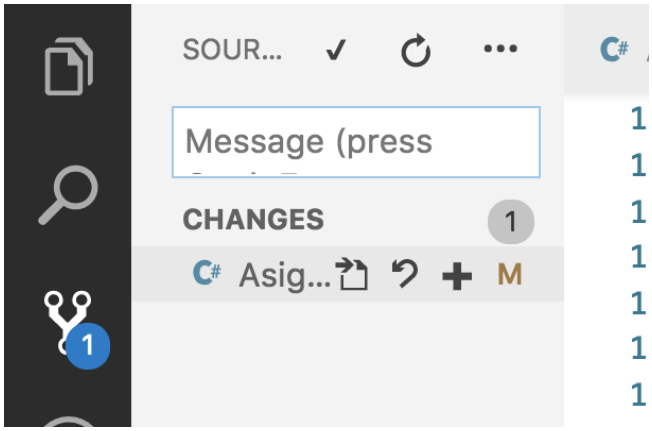
Lo primero como siempre es clonar nuestro repositorio remoto. Accedemos a la paleta de comandos de Visual Studio Code (Control-Shit-P) y escribimos clone. Nos aparecerá el comando de git clone, lo elegimos y a continuación nos pedirá la URL de nuestro repositorio remoto. Debemos poner el https:// correspondiente al repositorio que tenemos creado en Bitbucket. Nos salía cuando le dábamos a clonar desde la página de Bitbucket.




Nos saldrán los ficheros del repositorio y si realizamos algún cambio se nos queda marcado el fichero en cuestión con una M al lado del nombre. Está marcado como modificado, a continuación si queremos pasarlo a preparado (*staged*) debemos pulsar el *Source Control* (el icono destacado en el imagen siguiente) vemos los cambios realizados y salen las opciones de Git.




Tenemos para el fichero modificado las opciones de ***stage***(+), deshacer el cambio, ***commit*** (✓) y más opciones generales en el menú superior.



> Ejercicios propuestos:

> 1.6.6.4. Crea otro repositorio en tu cuenta de Bitbucket y clonalo desde Visual Studio Code.
>
> 1.6.6.5. Realiza los ejercicios del 1.6.6.1. al 1.6.6.3. utilizando Visual Studio Code.

## 1.6.7.Para profundizar

Puedes ampliar información sobre este tema en los enlaces que tienes a continuación.

- https://git-scm.com/book/es/v1
- https://confluence.atlassian.com/get-started-with-sourcetree/get-started-with-sourcetree-847359026.html