

GIT

<https://www.codigonexo.com/blog/programacion/entornos-de-trabajo/la-guia-definitiva-para-iniciarse-en-git/>

Una de las claves a la hora de trabajar profesionalmente en desarrollo web es tener un entorno profesional que te permita sacar el máximo partido al trabajo en equipo y mediante el control de versiones del código fuente de nuestro proyecto web. Como expertos en desarrollo web, os queremos presentar GIT, la herramienta que os facilitará mucho vuestro flujo de trabajo. Pero, **¿qué es 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 estas tienen un gran número de archivos de código fuente. Wikipedia

Gracias a GIT podremos trabajar con total comodidad en equipo, tener copias de seguridad de nuestro proyectos gracias a los repositorios, proporciona una mejor auditoría de eventos de ramificación y fusión... Muchas posibilidades para mejorar nuestro flujo de trabajo, pero si no nos hemos enfrentado antes a ellas, pueden resultar un poco confusas de configurar. Para ello hemos elaborado esta pequeña guía que os iniciará en el mundo de GIT.

Comenzando a usar GIT

El primer paso para comenzar a utilizar GIT es darnos de alta en un cliente virtual. Para ello podremos acceder a Gitlab, Github, Bitbucket u otro sistema de los que hay disponibles en internet. Aunque su finalidad es la misma, existen pequeñas diferencias en los servicios que ofrecen. Podéis ver la comparativa entre estos 3 clientes en [esta tabla](#).

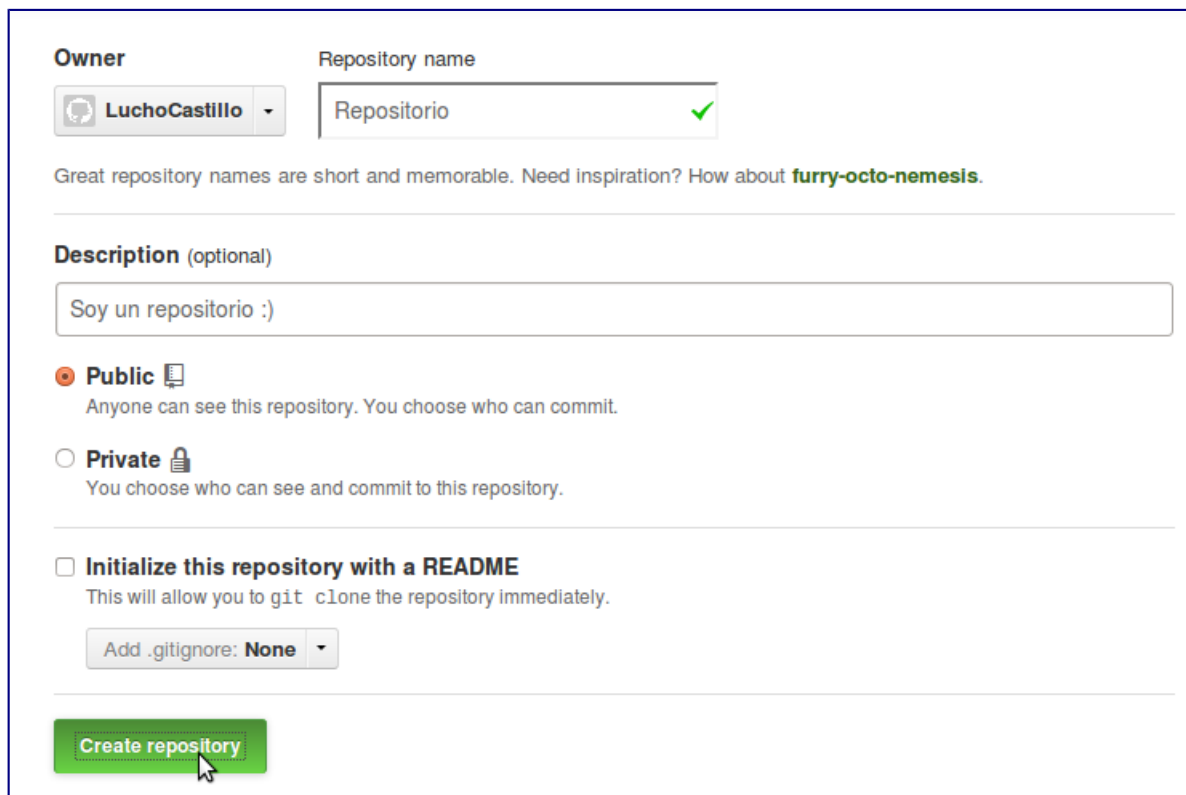
Configurando el entorno

Una vez dado de alta en un cliente virtual, el siguiente paso es descargar un programa de gestión local que nos va a permitir comunicarnos con la herramienta remota. En principio, con usar la consola nos es más que suficiente para sacarle todo el jugo a GIT, pero bien es cierto que existen alternativas interesantes para que, de forma más visual, podamos usar todas las funcionalidades de este sistema. Nuestros expertos en desarrollo nos recomiendan Netbeans, el cual lo podréis descargar [aquí](#) y gracias a su sistema visual, podréis lograr todas las funcionalidades que más adelante explicaremos.

Para crear un repositorio en GitHub, solo hay que seleccionar el botón “New Repo”, de la barra de herramientas, habiendo entrado a [GitHub](#) con tu cuenta:

Ahora habrá que llenar dos datos:

1. Nombre del repositorio
2. Descripción del repositorio (opcional)



The screenshot shows the GitHub 'Create repository' form. At the top, the 'Owner' is set to 'LuchoCastillo' and the 'Repository name' is 'Repositorio', which is marked as valid with a green checkmark. Below this, a message suggests repository names should be short and memorable, with an example 'furry-octo-nemesis'. The 'Description' field contains 'Soy un repositorio :)'. Under the 'Visibility' section, 'Public' is selected, indicating that anyone can see the repository. The 'Initialize this repository with a README' checkbox is unchecked. At the bottom, there is a dropdown for '.gitignore' set to 'None' and a green 'Create repository' button with a mouse cursor hovering over it.

¡Listo! Repositorio creado, ahora lo vas a poder ver en tu perfil.

Inicializar y sincronizar el repositorio local

En este punto se nos pueden dar dos casos. El primero de ellos es que en nuestro repositorio local (nuestro ordenador) no tengamos ningún archivo aún, es decir, algún compañero a empezado el proyecto y nosotros queremos unirnos a él. El otro caso es el contrario, que tengamos un proyecto local que queremos compartir con nuestros compañeros.

Si lo que **queremos es unirnos a un repositorio**, lo primero que **tenemos que hacer es crear el ‘vínculo’ entre nosotros y el repositorio remoto**. Para ello, abriendo nuestra consola teclearemos lo siguiente:

git clone : Nos conecta con el repositorio remoto para leer los datos que tiene como últimos. En este caso, nos creará una carpeta con todos los archivos que contenga.

Desde este momento ya tenemos cargados los últimos ficheros sincronizados en el servidor remoto y nuestro servidor local con GIT.

Si estamos en la segunda opción, que es la de **compartir nuestro repositorio**, lo que **tenemos que conseguir es inicializar GIT para poder conectar los ficheros que tenemos con el repositorio**. Para ello, utilizaremos la siguiente línea de comandos:

Accederemos mediante comandos a la carpeta destino y escribiremos **git init**. Este comando inicializa directamente un control en nuestro servidor local.

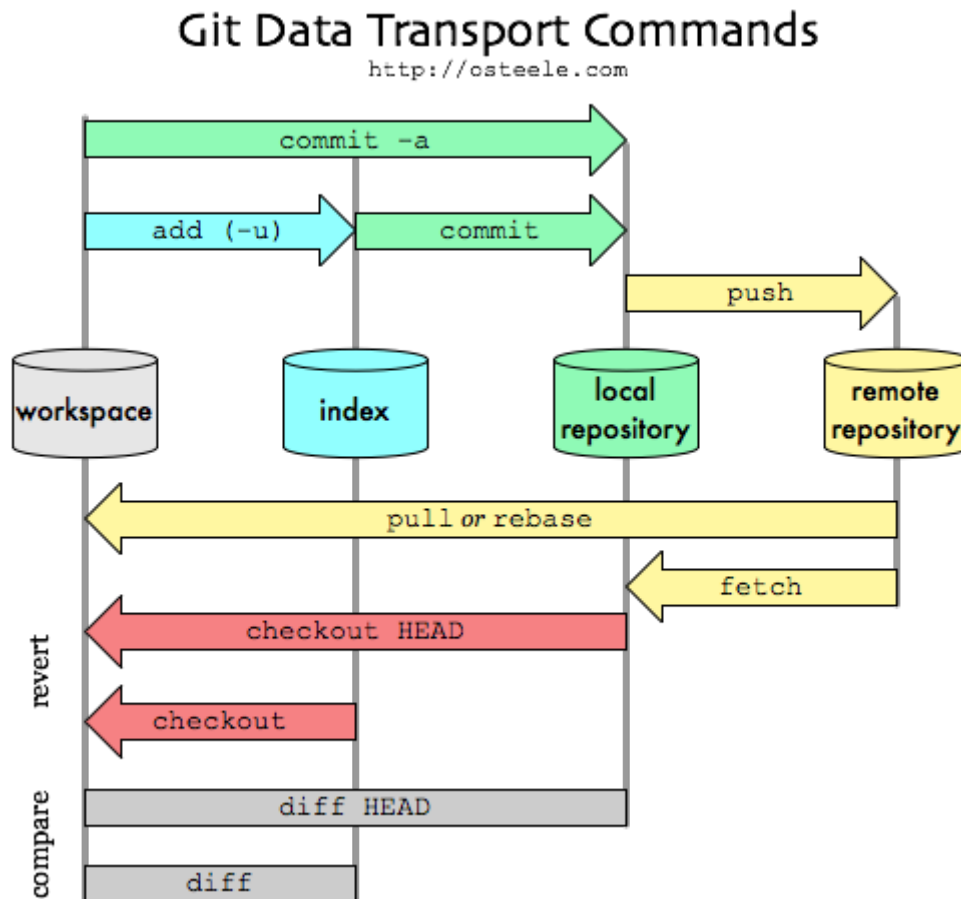
Lo que tenemos que hacer ahora es decirle a donde quiere que apunte, es decir, con que repositorio tiene que interactuar. Para ello, teclearemos lo siguiente **git remote add origin** . Desde este momento ya tenemos sincronizado nuestro ordenador con el repositorio remoto.

Ahora solo nos queda “empujar” todos los archivos que tenemos a dicho repositorio, para ello usaremos el comando **git push -u origin master**. En este caso le estamos diciendo que suba todo lo que hay actualmente en esta carpeta desde ‘origen’ a la rama ‘master’ de nuestro repositorio.

Flujo de Trabajo de Git

Tu repositorio local está compuesto por tres “árboles” administrados por git. El primero es tu Directorio de trabajo que contiene los archivos, el segundo es el Index que actúa como una zona intermedia, y el último es el HEAD que apunta al último commit realizado.

En el siguiente gráfico podéis entender a la perfección el flujo de trabajo de Git.



[Fuente](#)

Funciones Básicas en Git.

Trabajando en nuestro Workspace, acumularemos X cambios, respecto a estructura, funcionalidad, estilo... Para registrar esos cambios y añadirlos al index usaremos **git add <filename>** donde filename es el nombre de los archivos que hemos modificado. Si lo que queremos es añadir todos los cambios que hemos hecho, usaremos el comando **git add**.

Este es el primer paso en el flujo de trabajo básico. Para hacer commit a estos cambios usa **git commit -m "Commit message"**. En Commit message puedes escribir un mensaje, normalmente se usa para poder especificar qué cambios se han realizado en el último commit.

Ahora el archivo está incluido en el HEAD, pero aún no en tu repositorio remoto, por lo que para incluir estos archivos en nuestro repositorio remoto debemos enviar esos archivos. Para enviar todos los cambios que se encuentran en local repository, debemos ejecutar **git push origin master**, donde master corresponde a la rama de trabajo en la que nos encontramos, si nos encontramos en otra rama, por ejemplo desarrollo, nuestro comando sería **git push origin desarrollo**.

Sistema de Ramas

Como si de un árbol se tratase, git basa su funcionamiento en ramas, utilizadas para desarrollar funcionalidades distintas unas de otras, es decir, en nuestro flujo de trabajo podemos desarrollar las distintas partes de nuestro proyecto en distintas ramas que convergen en un punto que fusionamos para obtener el proyecto final. Por defecto Git crea la rama master, pero podemos crear tantas ramas como necesitemos y poder fusionarlas a la rama principal cuando terminemos.

Para crear una nueva rama y saltar a ella usa **git checkout -b ramanueva**

Para volver a la rama principal **git checkout master**

Para borrar la rama **git branch -d ramanueva**

recuerda que si trabajas en equipo, una rama no estará disponible para los demás a no ser que la subas a el repositorio remoto, para ello usa git push origin

Actualizar y fusionar ramas

Si trabajamos en equipo y otros desarrolladores se encuentran haciendo “push”, necesitaremos “bajar” esas versiones, para ello necesitamos actualizar nuestro repositorio local al commit más nuevo, usamos **git pull** para poder actualizar y fusionar los últimos cambios en remoto.

Si lo que deseamos es fusionar una rama a nuestra rama activa (en la que nos encontramos trabajando), necesitaremos usar git merge para poder fusionar las ramas. Git automáticamente fusionará los cambios realizados, pero puede darse la casuística de que hayamos trabajado en la misma línea de código que otros desarrolladores, apareciendo los famosos conflictos, que git te mostrará y bajo los que tendrás un total control, permitiendo eliminar el código duplicado o fusionarlo, siempre manualmente.

Una vez fusionados, necesitamos marcarlos y actualizar mediante **git add y git push**.

Antes de fusionar los cambios podemos revisarlos usando **git diff <source_branch><target_branch>**

Reemplazar cambios locales

Puede ser que durante el desarrollo de nuestro proyecto, hagamos algo indebido. Pero gracias al comando **git checkout --<filename>** podremos reemplazar cambios locales, permitiendo recuperar el último contenido.

Si quisiéramos acceder a la última versión del servidor, deshaciendonos de todos los cambios locales y commits usamos **git fetch origin / git reset --hard origin/master**.

Ya no hay excusas para usar GIT en vuestro día a día. Si tenéis alguna duda, no dudéis en consultarnos.

Source Tree

<https://jadcode.wordpress.com/2014/06/09/source-tree/>

En este tutorial vamos a aprender un poco más sobre los sistemas de control de versiones utilizando una nueva herramienta llamada Source Tree.

¿Qué es Source Tree?

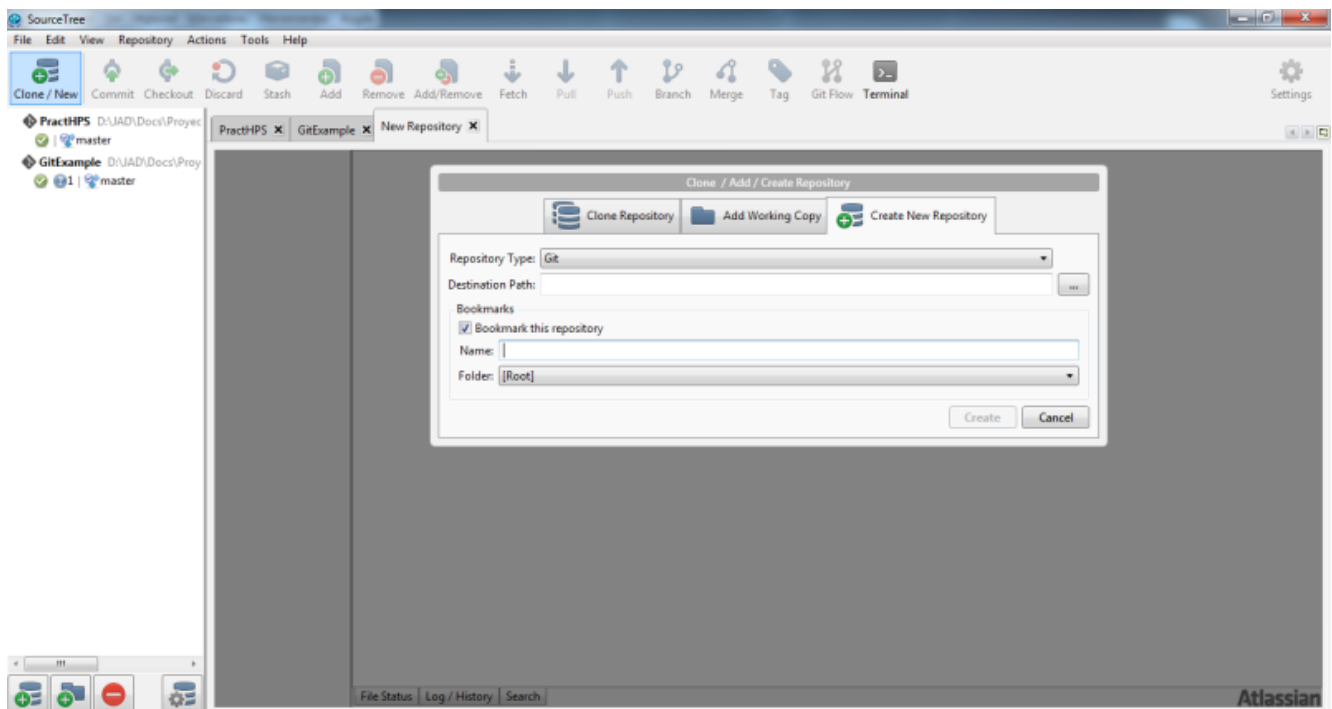
Source Tree es un potente GUI (Graphical User Interface – Interfaz Gráfica de Usuario) para gestionar todos tus repositorios ya sean Git o Mercurial. Con Source Tree podemos crear, clonar, hacer commit, push, pull, merge y algunas cosas más de una forma bastante fácil. Desarrollado por Atlassian e inicialmente solo para Mac, también cuenta con su versión para Windows y lo puedes descargar desde este enlace <http://www.sourcetreeapp.com/>

Damos al botón Clone/New para que aparezca la siguiente ventana. La opción Clone Repository es para clonar un repositorio ya existente; la opción Add Working Copy es para añadir un repositorio que tengas local (en tu equipo); y Create New Repository para crear un nuevo repositorio. Vamos a crear uno nuevo:

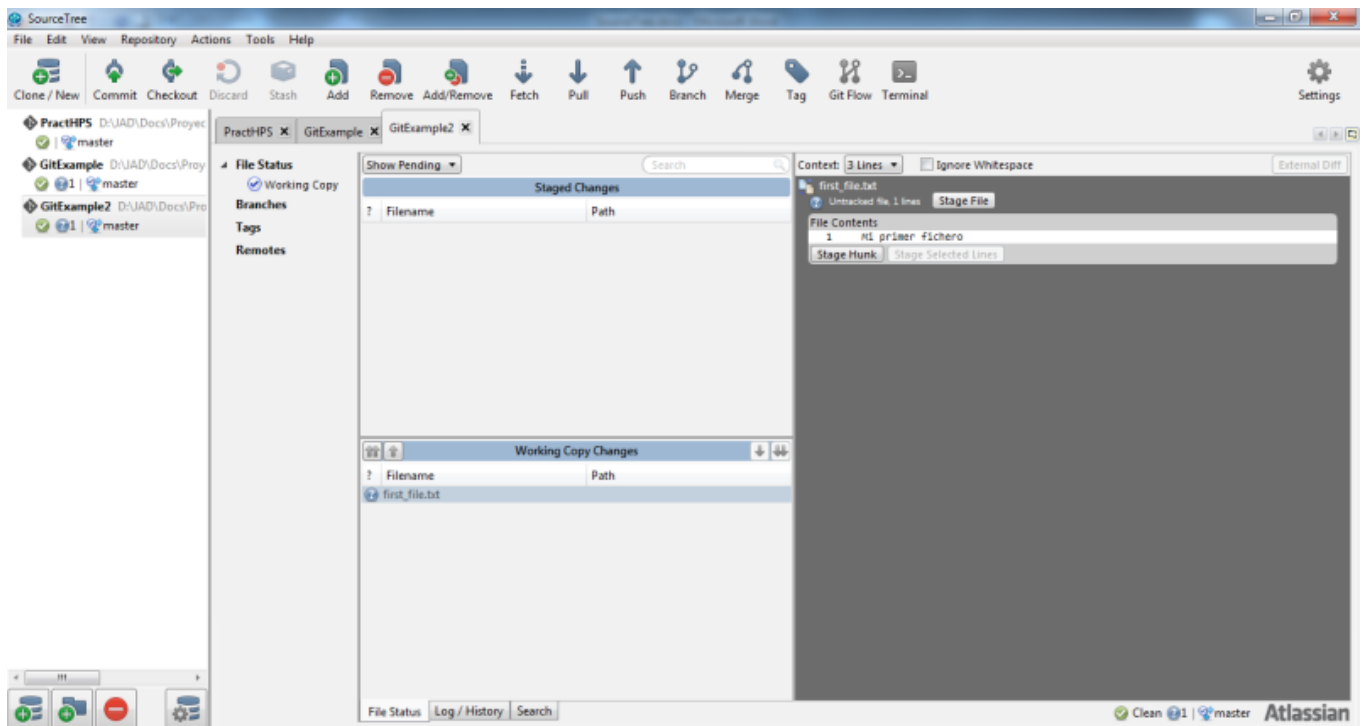
En Repository Type tenemos la opción de elegir tanto repositorios Git como repositorios Mercurial, en este caso elegimos Git.

Destination Path se elige el directorio donde se creará el repositorio.

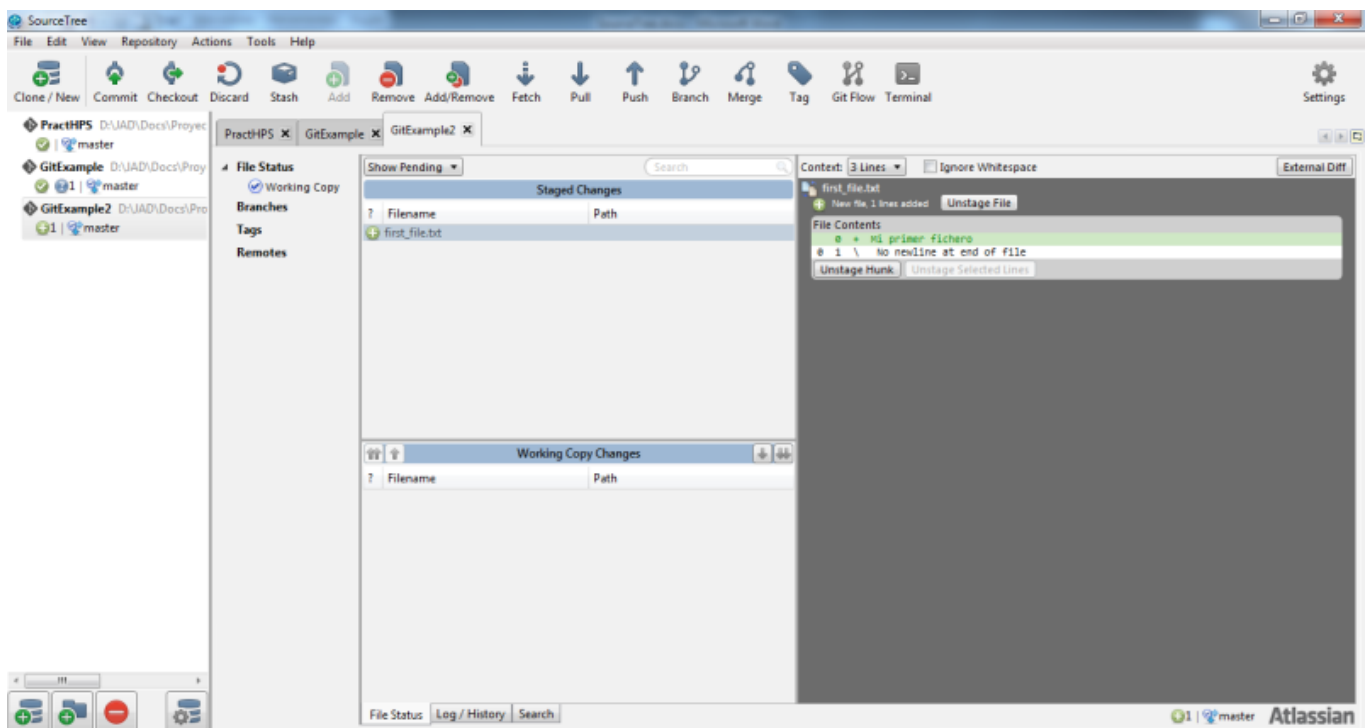
Dejamos marcada la opción Bookmark this repository (para que aparezca en el listado de nuestros repositorios) y en Name elegimos un nombre para el repositorio. Damos a Create y tendremos nuestro repositorio creado para trabajar.



Ahora nos creamos cualquier fichero dentro del directorio del repositorio, en mi caso he creado el fichero fist_file.txt (dentro del fichero escribe cualquier cosa), en Working Copy Changes se nos mostrara el fichero y en la parte derecha se podrá ver el contenido del fichero.

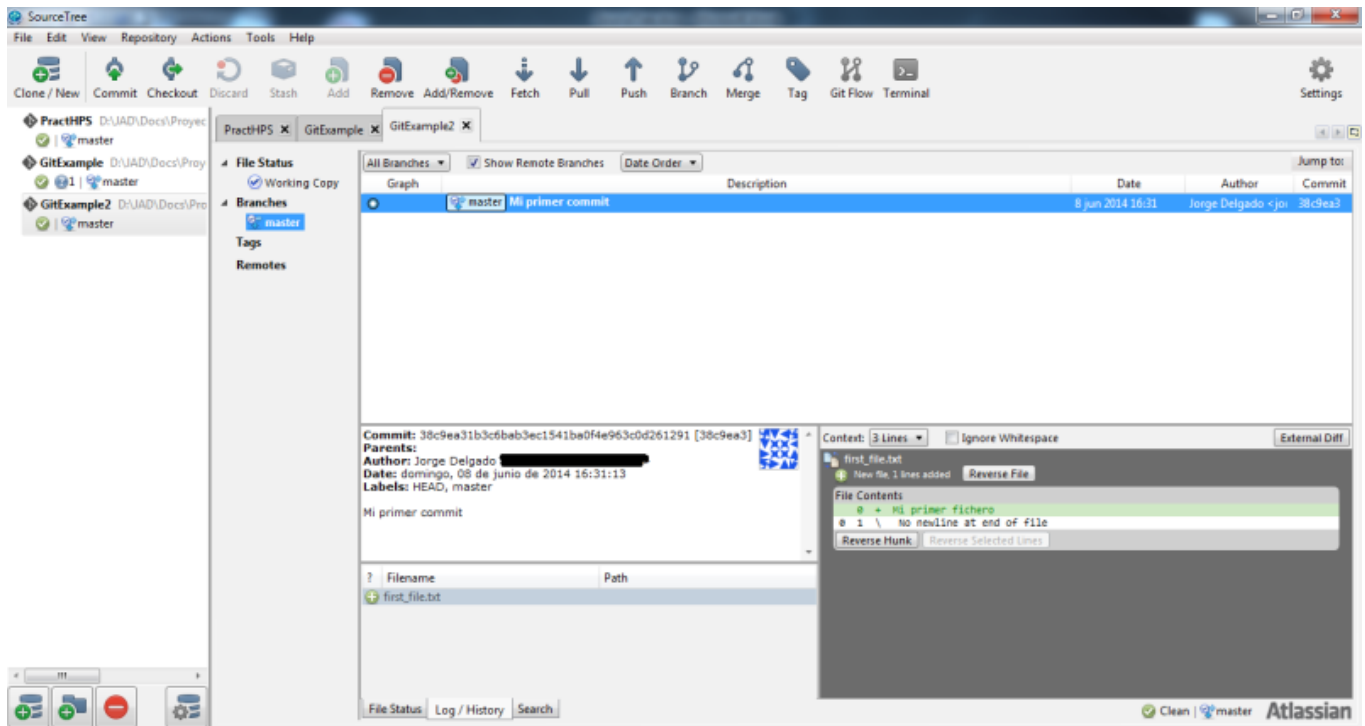


Esto demuestra que la herramienta está continuamente buscando dentro del directorio del repositorio y de esta forma hará un buen seguimiento de nuestros proyectos. Para poder hacer un commit tenemos que hacer previamente una subida a Staged Changes. Para ello pulsamos en Add (en las opciones de arriba) y veremos el fichero en Staged Changes marcado como añadido y listo para hacer un commit.



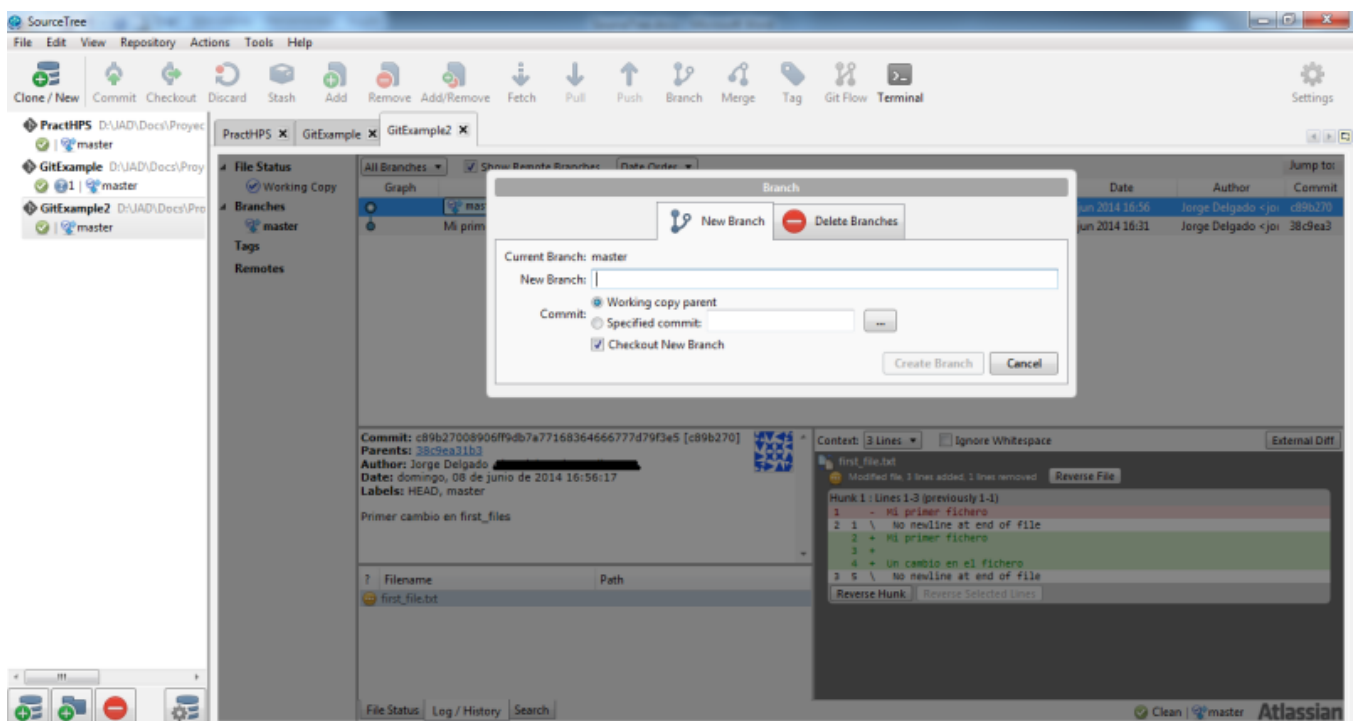
Ahora le damos a Commit (opciones de arriba) y nos saldrá la ventana para introducir un mensaje en Commit message. Esto es prácticamente igual que en [Git en NetBeans – Parte 1](#). Escribimos lo que queramos (en mi caso “Mi primer commit”); veremos los ficheros para este commit; también si nos fijamos en la parte derecha donde podemos ver el contenido del fichero contamos con el botón Unstage File, por si queremos cancelar el commit y regresar el fichero al Working Copy Changes; dejamos todo tal cual y hacemos el commit.

La ventana se cerrará y si vamos a Branches podemos ver la rama master con el primer commit. Cada vez que vayamos hacer un commit o trabajar con diferentes ramas, es muy importante ir a esta opción pues será el historial de nodos del proyecto.



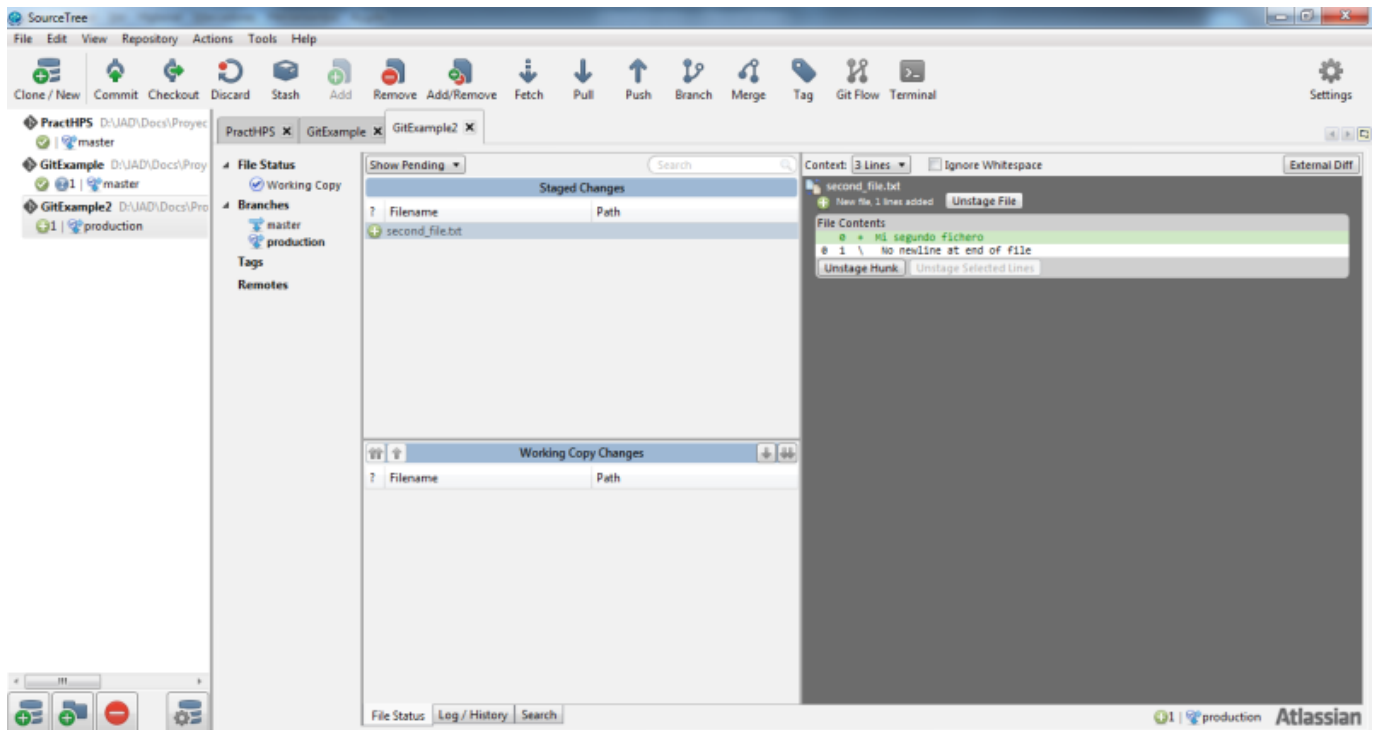
Hacemos otro cambio en el fichero para poder apreciar mejor lo que acabo de explicar. En Files Status y después en Working Copy vemos que Source Tree ha captado los cambios que hemos hecho. Añadimos para preparar el segundo commit, realizamos el commit (podremos apreciar lo que tenía antes en rojo, a lo que se tiene ahora en verde) y al volver a Branches vemos los dos commits y master apuntando al segundo commit.

Ahora vamos a ver algo importante, cómo trabar con diferentes ramas. Para ellos vamos a Branch (en las opciones de arriba) y vemos una ventana con dos opciones, para crear una nueva rama o para borrar una rama existente.



Elegimos el nombre que queramos para la rama, la siguiente opción es para elegir un commit específico o el que tenemos en Working copy parent (lo dejamos marcado en Working copy parent) y Checkout New Branch la dejamos marcada pues cuando creamos la rama nos posicionaremos en ella. Damos a Create Branch y volvemos a Branch. Lo que veremos son dos ramas, la master y la que acabamos de crear (production en mi caso) apuntando ambas al mismo commit.

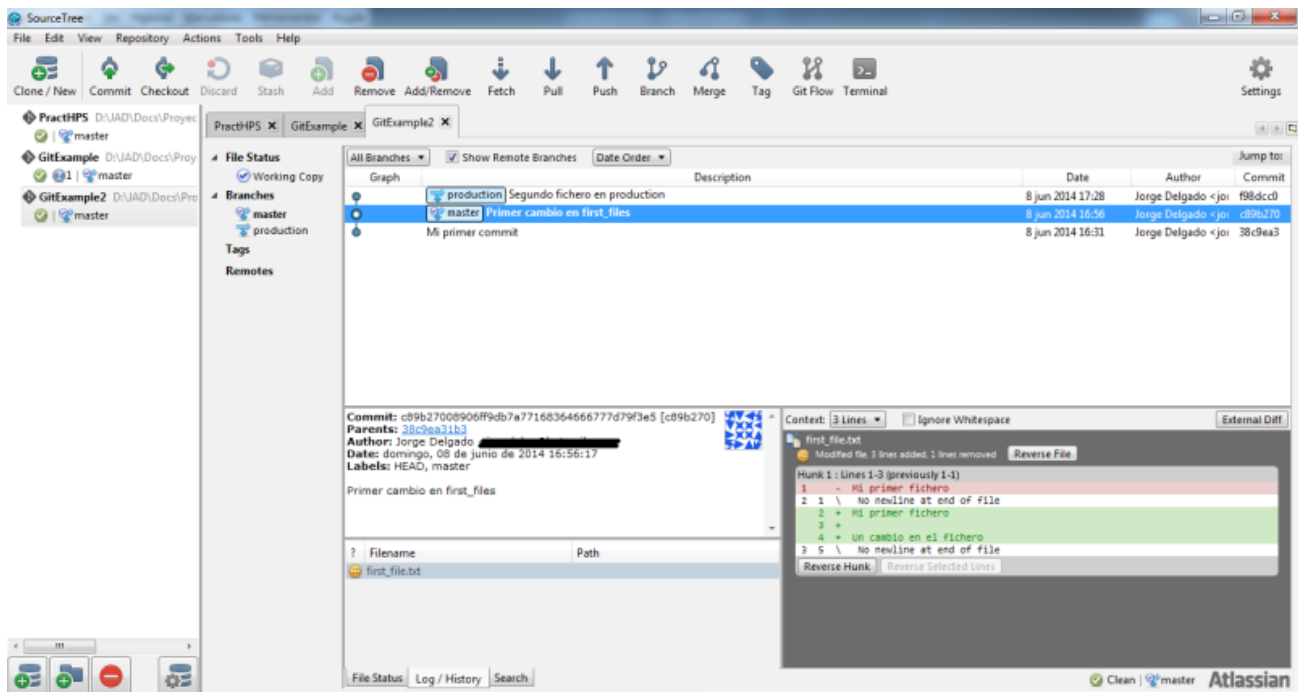
Creamos un nuevo fichero, le escribimos lo que queramos y al volver a Working Copy vemos que Source Tree ha captado el segundo fichero. Hacemos Add para que pase a Staged Changes. Si nos fijamos en Branches, vemos que la nueva rama está marcada. Eso quiere decir que actualmente estamos trabajando en ella y por tanto el siguiente commit será para esa rama.

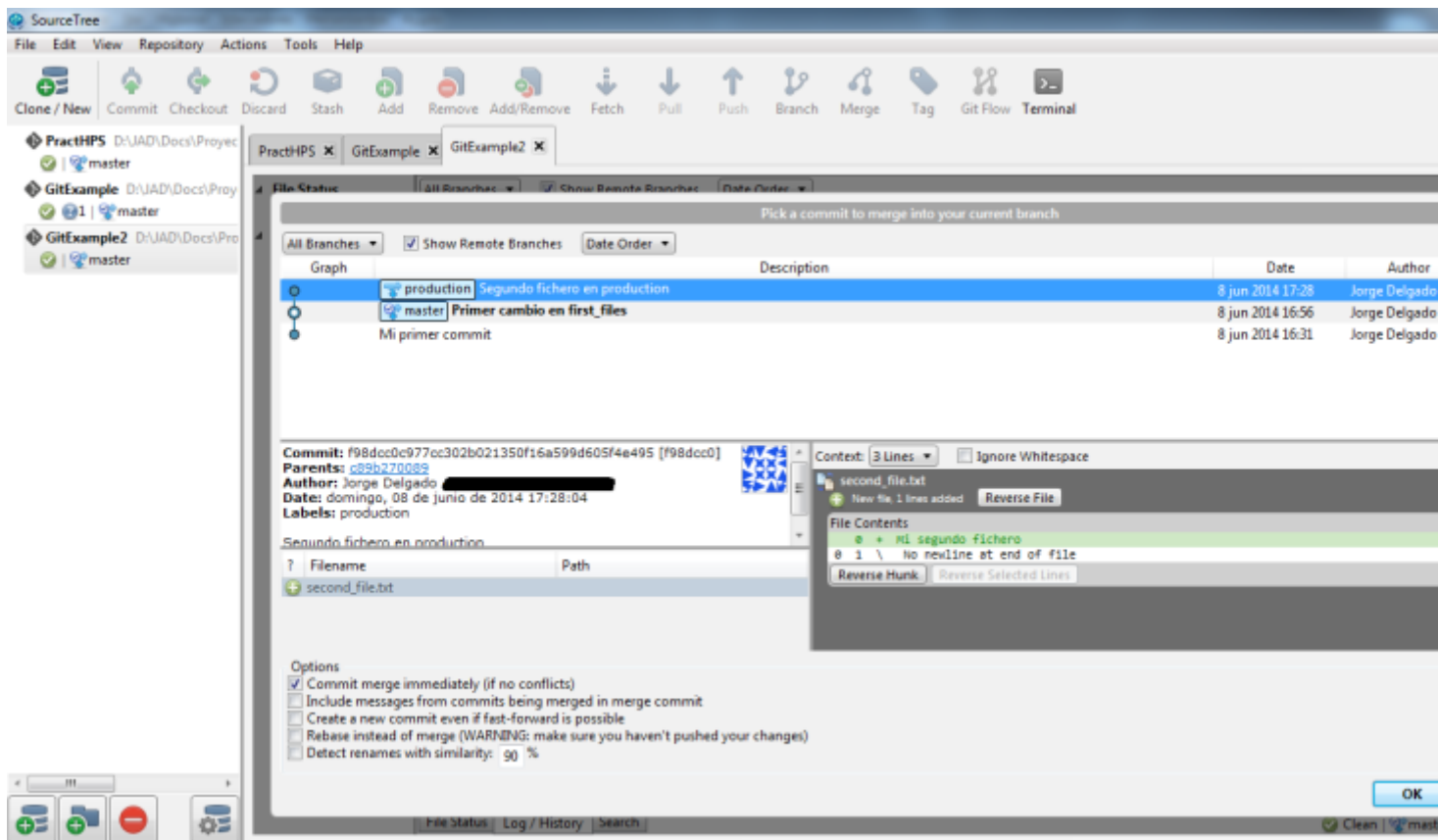


Hacemos el commit y al ir a Branches vemos que master está en el segundo commit y production en este último commit. Si hacemos un Merge a master pasará lo siguiente. Primero tendremos que posicionarnos en master, para ello hacemos 2 clicks sobre ella.

La rama master está marcada, nos situamos sobre production y pulsamos en Merge. Nos preguntará que commit queremos pasar a master, obviamente elegimos el commit que tiene apuntado production y como

siempre se nos muestra el ficheros involucrados.

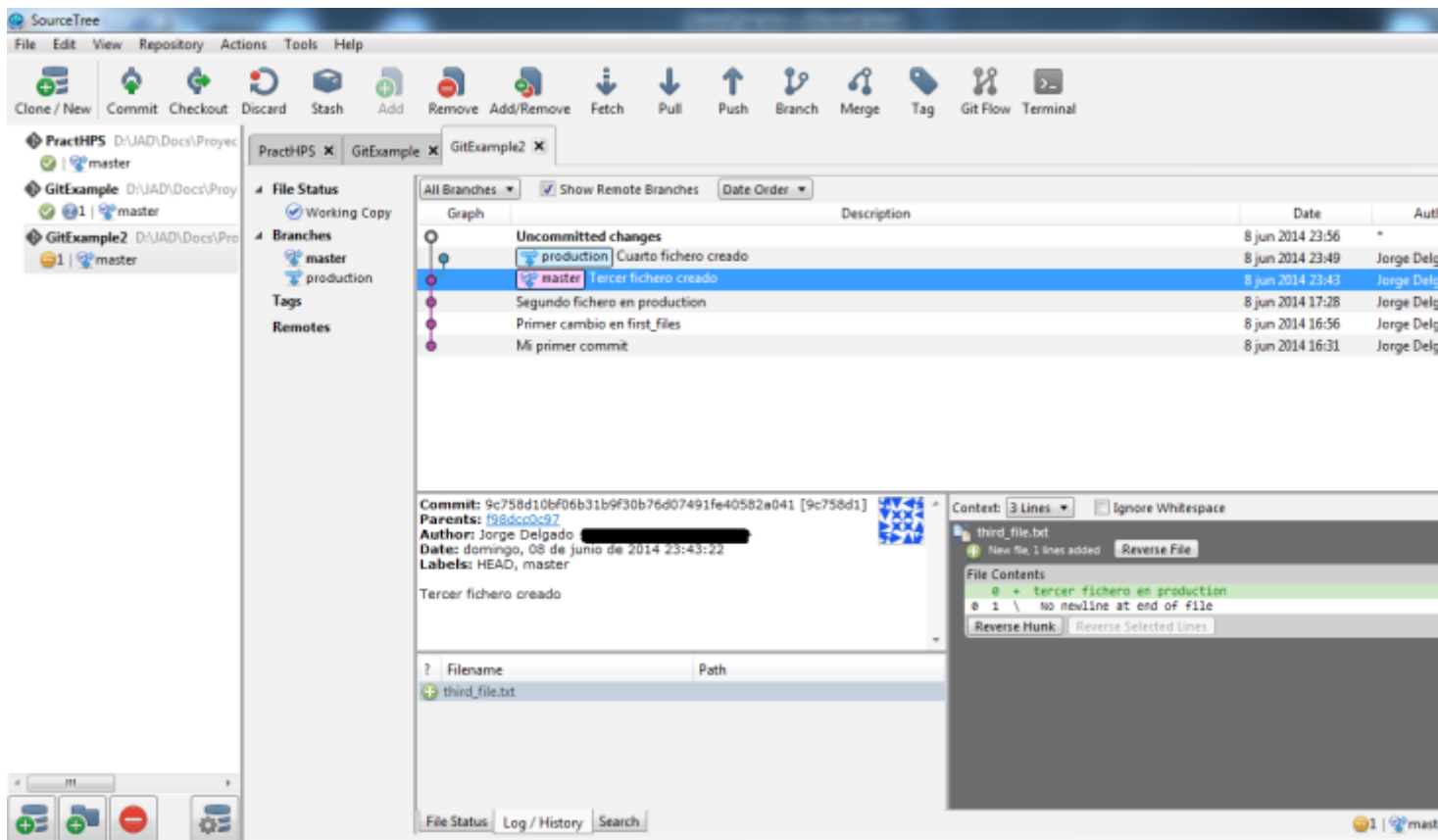




Le damos a OK y se producirá el Merge. Sabremos que está hecho el Merge porque master apuntará ahora al mis commit que production.

Ahora volvemos a la rama production (doble click sobre la rama) y creamos un nuevo fichero. Vamos a Working Copy para añadir el fichero y hacer un commit.

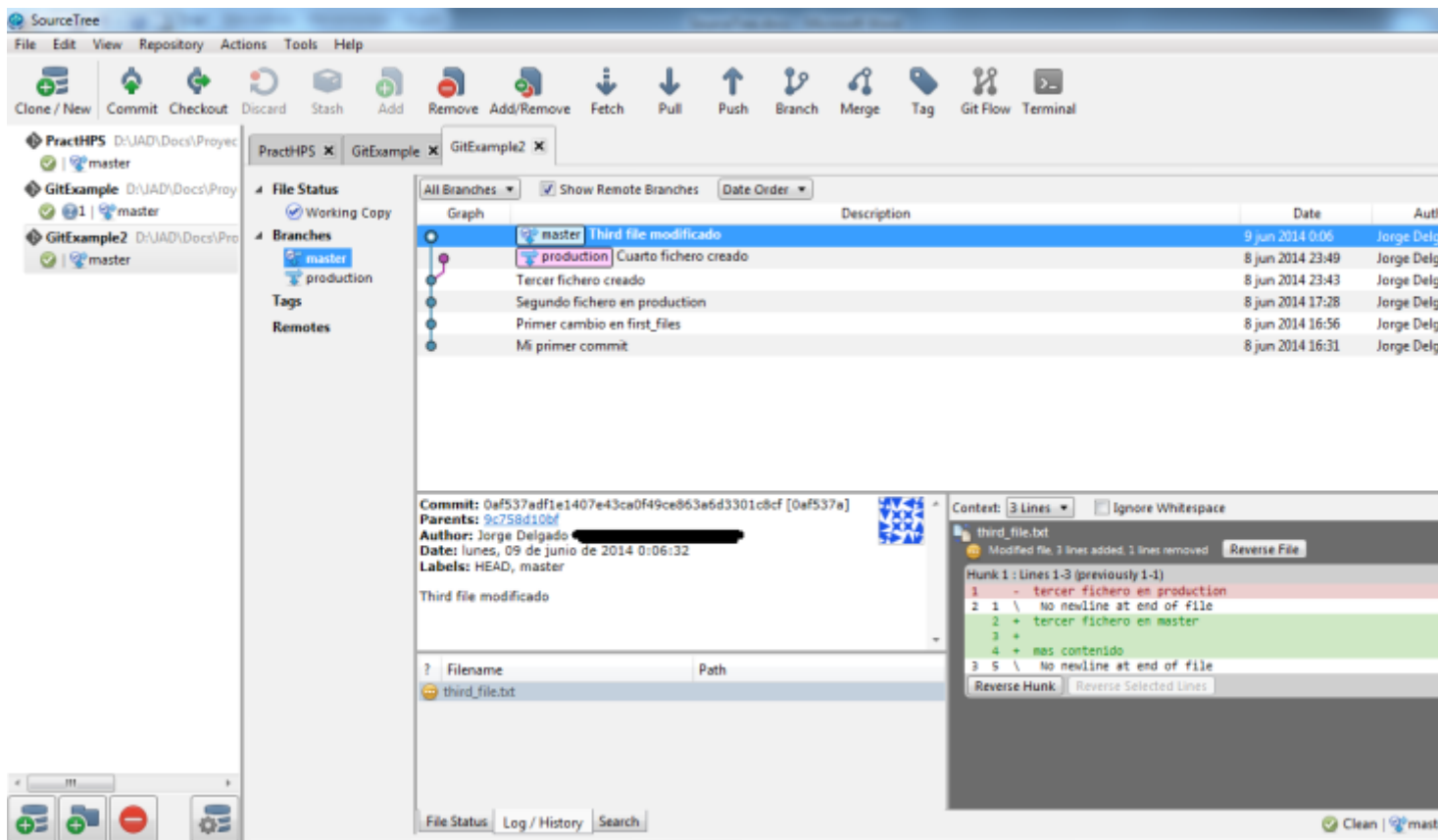
Ahora vamos a la rama master que seguirá apuntando al mismo commit que estaba y production estará apuntando al último commit creado. Si hago una modificación en alguno de los ficheros, al comprobar las ramas vemos que se bifurcan.



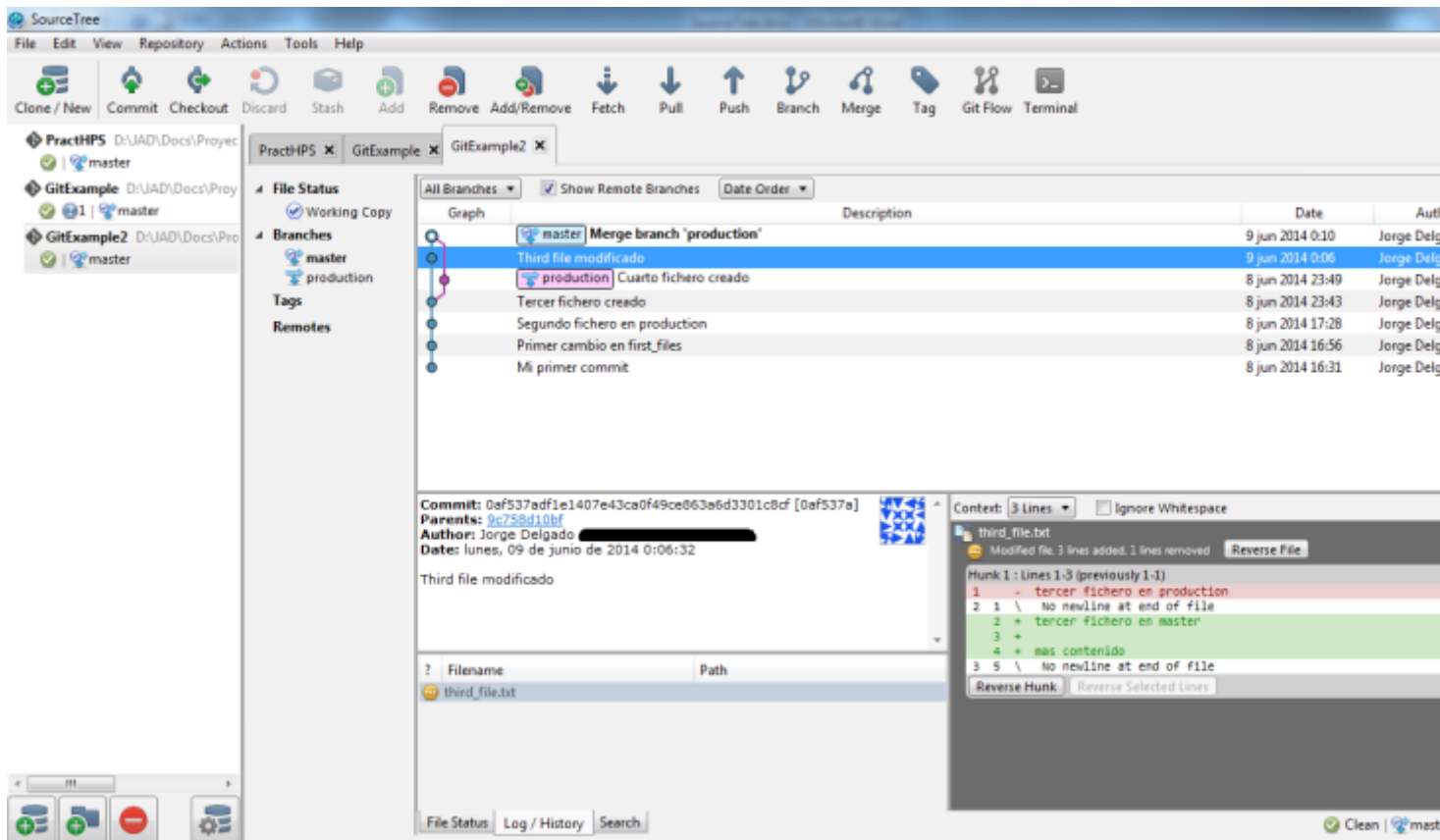
¿Por qué pasa esto?

En production tenemos los mismos ficheros que en master más otro más que se llegó a crear. En master tenemos los mismos fichero que en production pero si vamos al directorio vemos que están todos menos el último (eso es porque la rama master no tiene de momento el último commit que tiene añadido ese fichero). Al modificar uno de los ficheros, estando en master, ese fichero que tenían en común la rama master y la rama production es diferente ahora y por eso se produce la bifurcación.

Vamos a Working Copy, hacemos un Add, luego el commit y al volver a Branches para comprobar las ramas, y vemos algo como esto:



Como estamos en master ahora hacemos un Merge con la rama production y vemos que se vuelven a unir, pues el commit del fichero que teníamos en production lo tenemos ahora también en la master.



Es muy importante acostumbrarse a trabajar de esta forma. Tenemos que tener claro que siempre tendremos la rama master cuando nos hemos creado el repositorio. A partir de ahí, es bueno crear y trabajar con otra rama diferente y cuando estemos seguros de que esos ficheros cumplen con lo que

queríamos, se hace un Merge de la rama master a esa otra rama para traernos los commit. Ya por último si esa rama no nos hace falta entonces podemos borrarla y quedarnos sólo con la master.

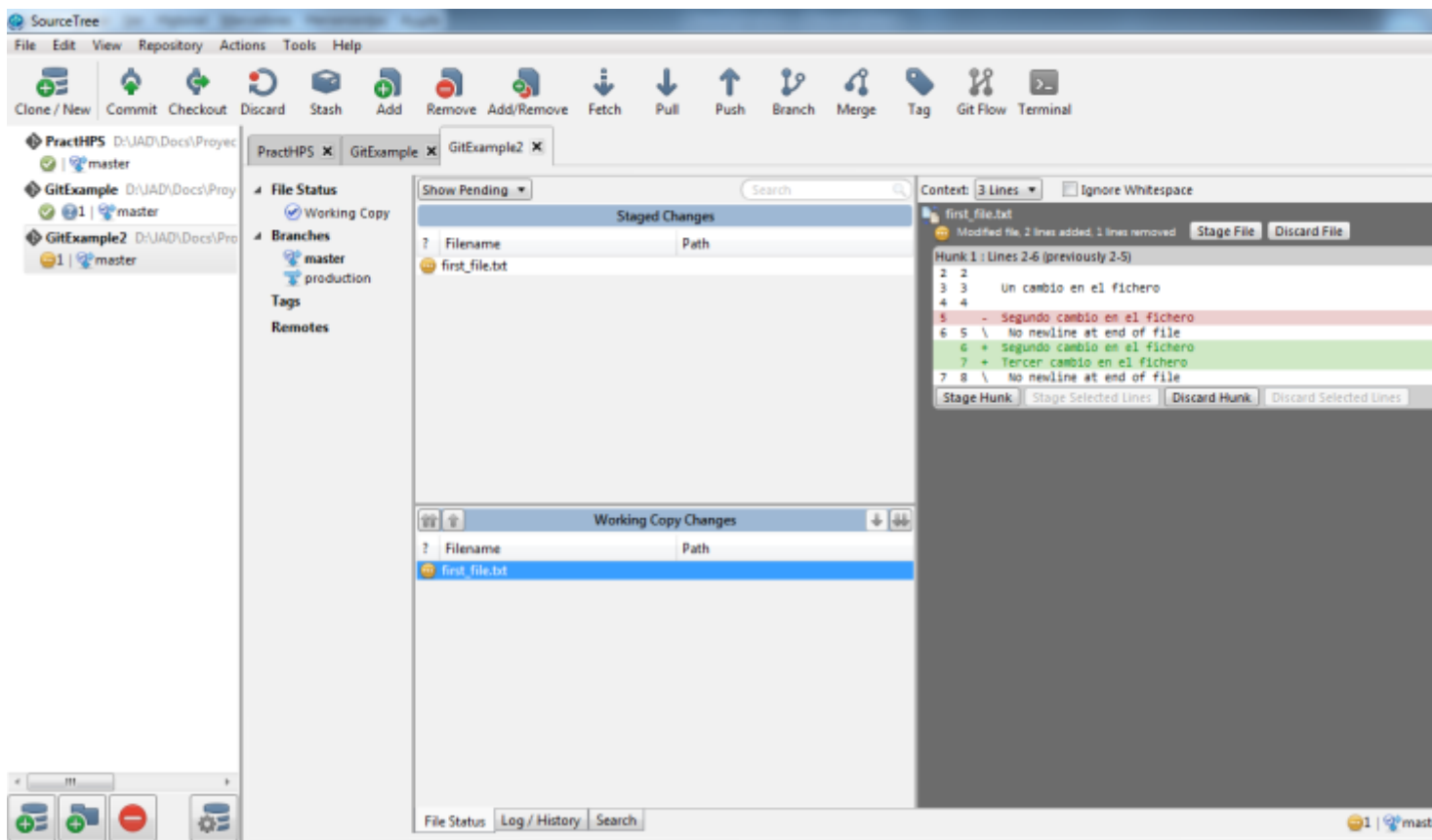
Vamos a modificar uno de los ficheros, da igual cual sea. Hacemos Add, después commit y en Branches vemos que master apunta a ese commit. Si nos arrepentimos de este commit lo que podemos hacer es situarnos en el commit anterior, damos click derecho y nos aparece la opción *Reset current branch to this commit* y vemos que tenemos tres opciones:

Soft para mantener todos los cambios en local

Mixed para mantener el working copy pero reseteando el índice

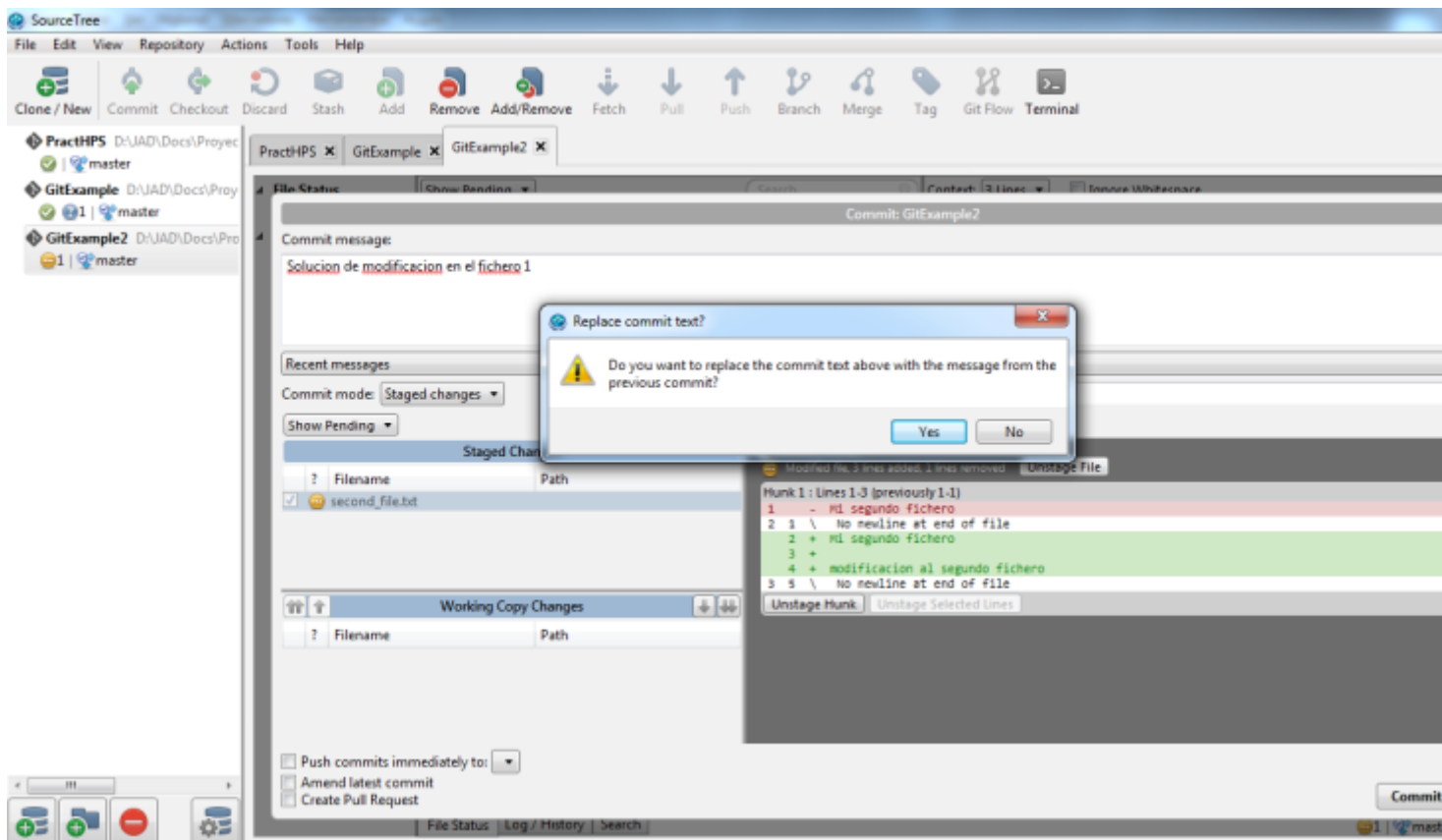
Hard elimina todo, como si no se hubiese hecho ni el commit ningún cambio en el fichero y tampoco se hubiese creado el fichero.

Elegimos soft, vemos que la rama master vuelve al commit anterior y si vamos a Working Copy vemos que seguimos teniendo el fichero en Staged Changes para volver hacer el commit. Antes de eso hacemos un nuevo cambio al fichero y vemos que tenemos tanto los cambios del fichero en Staged Changes como los nuevos cambios en Working Copy Changes.



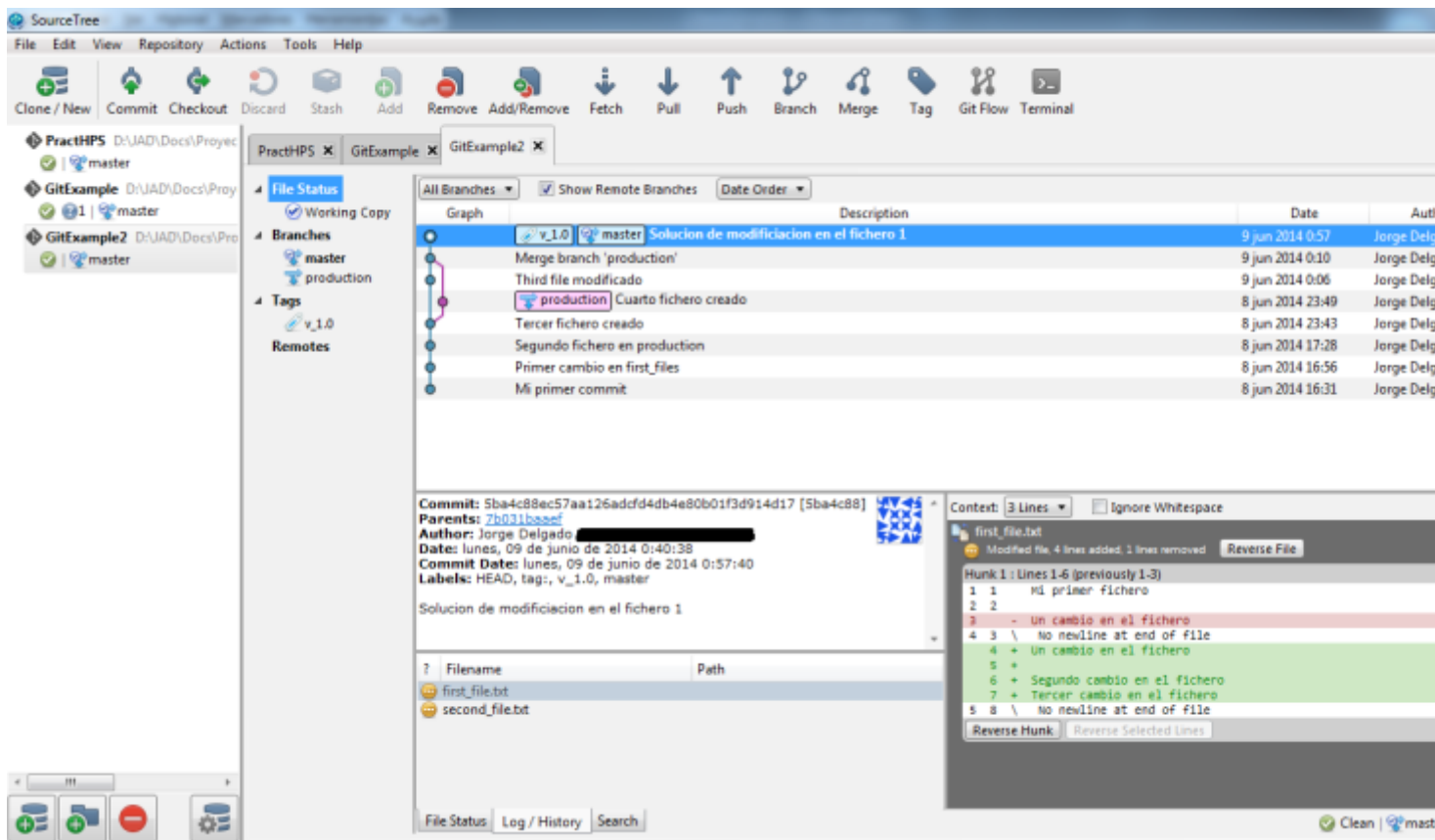
Hacemos Add y las nuevas modificaciones se unirán a las modificaciones de antes para hacerse el commit.

Vamos hacer una modificación a otro fichero, hacemos un add y nos damos cuenta que estos últimos cambios los queremos en el commit que master está apuntando y no en uno nuevo. Lo que hacemos hacer un nuevo commit con el mensaje del commit que apunta master y marcamos Amend latest commit.



Esto lo que hará es sobrescribir el último commit, añadiendo los cambios que iba tener el nuevo commit a el último commit. Le damos que si, luego a Commit y saldrá una nueva ventana advirtiendó que esto podría causar problemas si previamente has enviado los anteriores commit al repositorio remoto. Como no es el caso, no nos debemos de preocupar de hacer el Amend. Si revisamos los Branches vemos que tenemos los cambios de los 2 ficheros guardado en el commit actual.

Si suponemos que nuestro proyecto está en la versión 1.0 por ejemplo, podemos añadir una etiqueta. Vamos a Tag, ponemos un nombre a la etiqueta y podemos elegir a que commit queremos que se añada la etiqueta. Dejamos la opción que viene marcada para que sea en el último commit.



Ya es hora de subir estos commit a un repositorio remoto. Tal y como vimos en [Git en NetBeans – Parte 1](#) y [Git en NetBeans – Parte 2](#) para los repositorios usamos GitHub. Podemos crearnos nuestro repositorio remoto en GitHub, pero para este tutorial os quiero mencionar otro sitio donde poder crear tus repositorios remotos y es en Bitbucket <https://bitbucket.org/>

La razón por la que os menciono Bitbucket, es para que aprendan más posibilidades a las que podemos recurrir para gestionar nuestros repositorios remotos, de la misma forma que todo lo que hemos hecho en local ha sido desde Source Tree para gestionar el repositorio de forma local. Al final tú eliges que herramientas te gusta más usar.

Continuamos entonces para crear una cuenta en Bitbucket. Una vez creada vamos a Create para crear el repositorio y le damos un nombre. A diferencia de GitHub, podemos tener repositorios privados sin tener que pagar por lo que dejamos la casilla *This is a private repository* marcada, teniendo acceso sólo tú y aquellos que invites como colaboradores del proyecto. El tipo de repositorio es Git, por lo que dejamos también esa opción y podríamos indicar que lenguaje estamos usando (en este caso ninguno). Creamos el repositorio.

Create a new repository You can also [import a repository](#)

Name*

Description

Access level ☒ This is a private repository

Forking


Repository type ☒ Git ☐ Mercurial

Project management ☐ Issue tracking ☐ Wiki

Language


New to Bitbucket?

Learn the basics of using Git and Mercurial by exploring the Bitbucket 101.



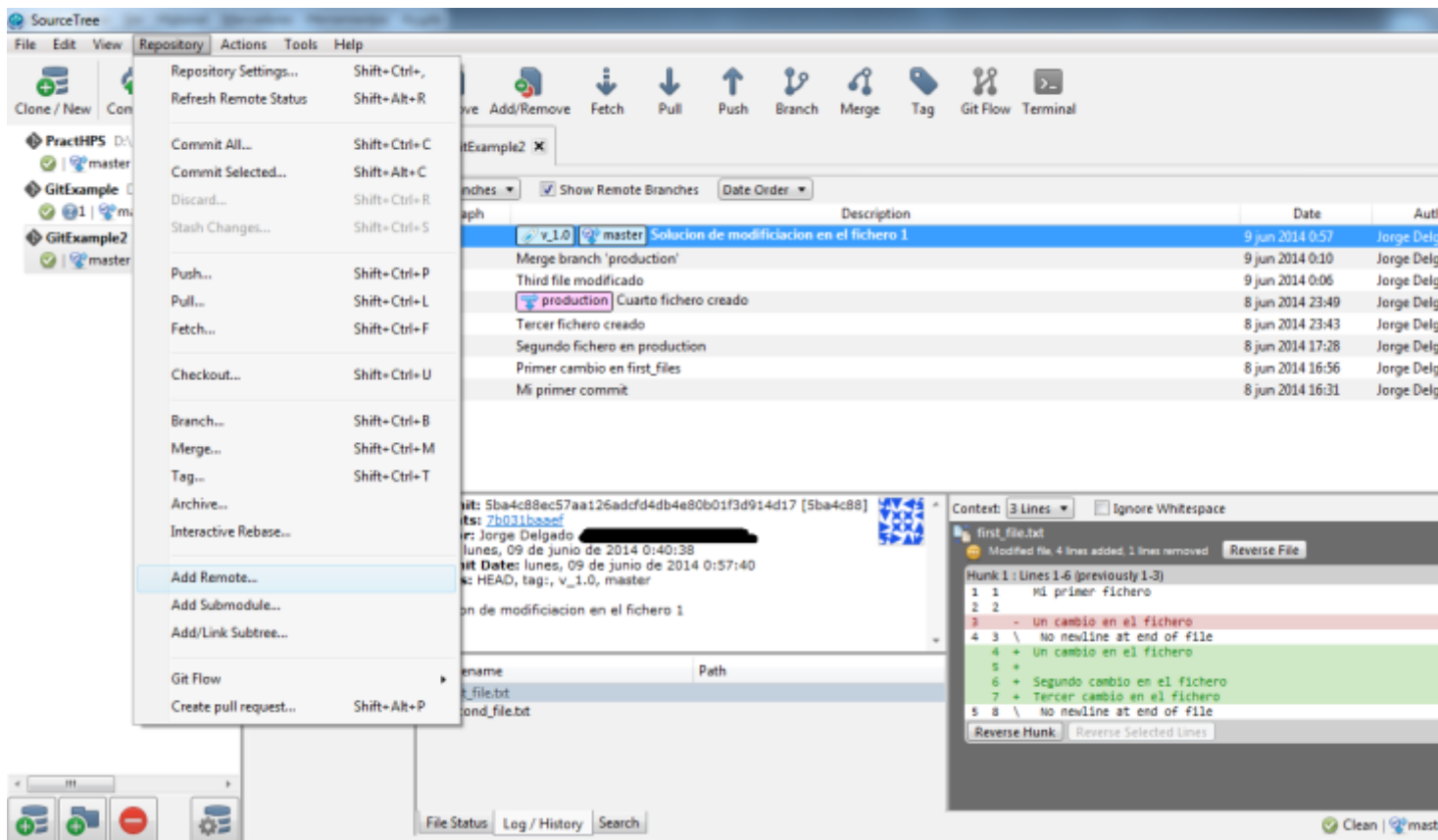
Working in a team?

Create a team account to consolidate your repos and organize your team's work.

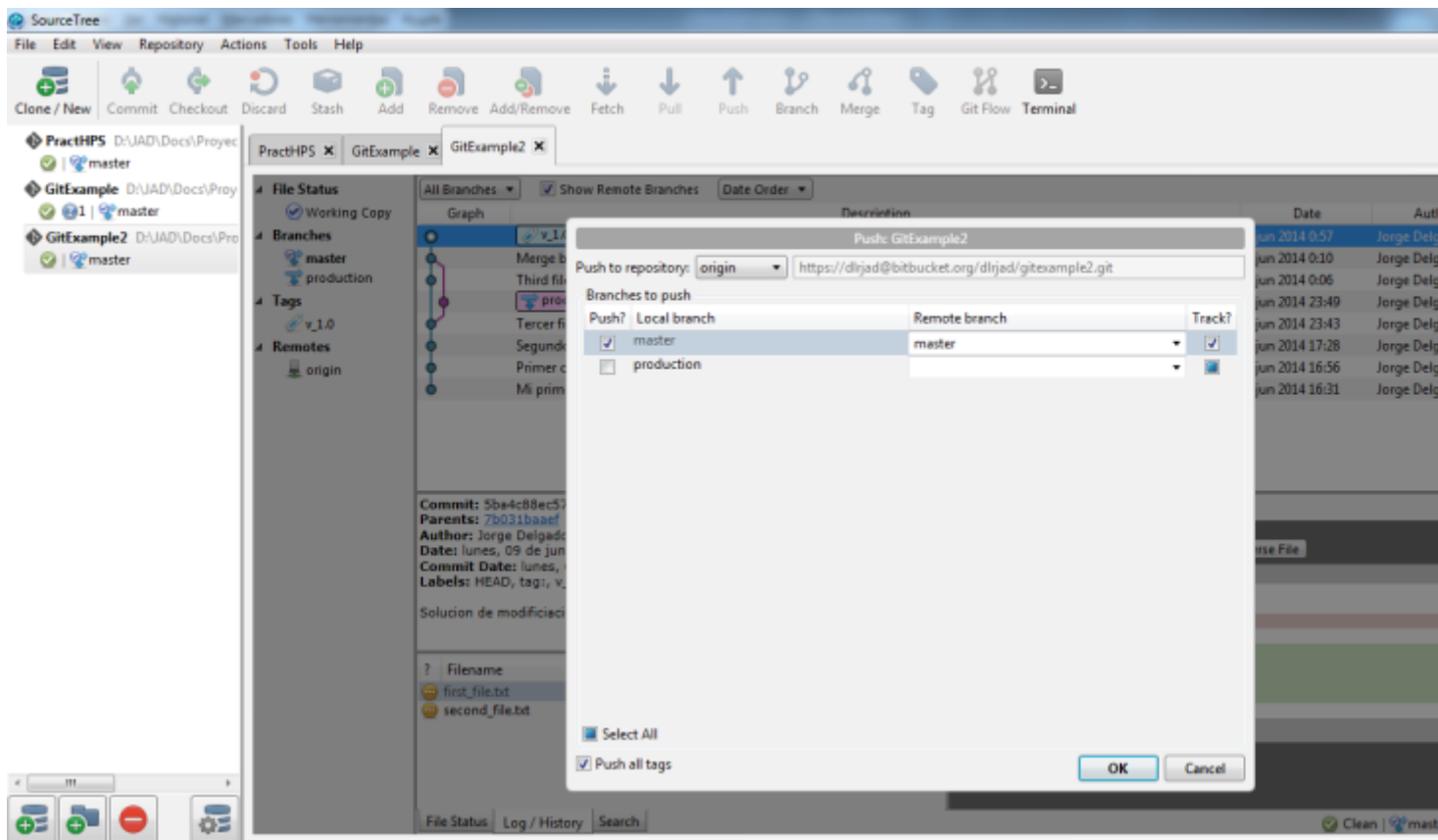


<https://confluence.atlassian.com/x/cgozDQ>

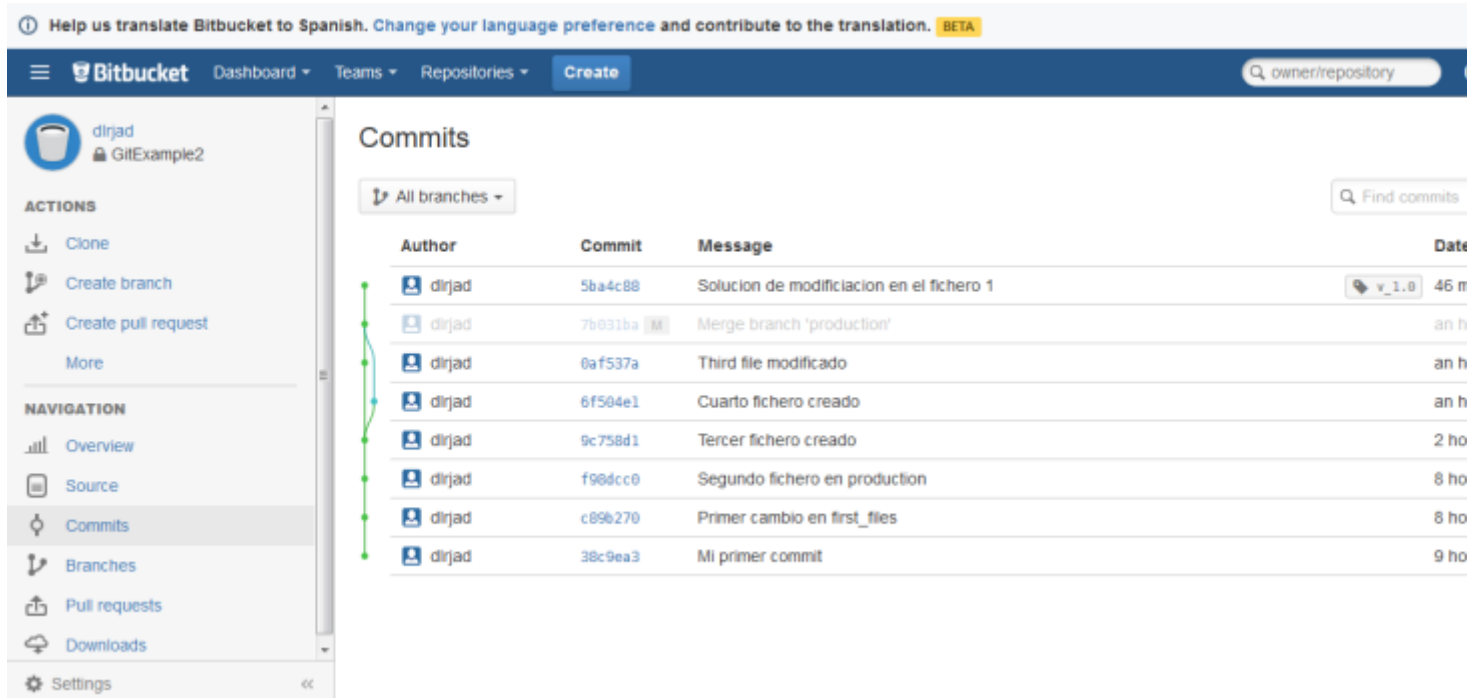
Como lo que queremos es enviar al repositorio remoto que acabamos de crear, donde pone Command line elegimos *I have an existing project* y obtendremos la url del repositorio, lo copiamos y regresamos a Source Tree para ir a Repository / Add Remote...



Le damos a Add para añadir repositorio remoto, le ponemos un nombre y la del repositorio que creamos en Bitbucket. Le damos a OK, vemos que ya tenemos el repositorio remoto añadido, damos nuevamente a OK y ahora en Remotes nos aparecerá el nombre del repositorio remoto que acabamos de añadir. Ya lo tenemos preparado para enviarlo a Bitbucket, le damos a Push y seleccionamos la rama master pero tal y como vemos podemos elegir otra rama local que no sea la master y lo mismo con el repositorio remoto.



Le damos a OK y todo se habrá podido subir al repositorio remoto.



Ya por último, cada vez que hagamos un nuevo commit en el repositorio local, Source Tree nos indicará con un número que aparecerá donde está Push indicando el número de commit pendientes por subir al repositorio remoto.

No vamos hacer un Pull porque viene siendo idéntico a lo que vimos en [Git en NetBeans – Parte 2](#).

Desde Terminal, podemos hacer todos los pasos que acabamos de hacer con líneas de comando por si uno está más acostumbrado a trabajar de esa forma.

Obviamente no hemos visto todas las opciones de Source Tree, pero con lo aprendido en este tutorial es más que suficiente para controlar las versiones de nuestro proyecto.