everis
an NTT DATA Company

GIT

BECA Java

El control de versiones con GIT

GIT

- Conceptos Básicos
- Software de control de versiones
- Workflow Básico con GIT
- Comandos GIT
- TortoiseGit
- Resolución de Conflictos

01 Conceptos Básicos

Ejercicio 1

¿Porqué necesito un control de versiones?

Para automatizar la información sobre los alumnos de la beca Java , vamos a crear el fichero alumnos.xml .

Cada alumno debe cumplimentar las etiquetas con su información personal y el resultado debe ser un único fichero que contenga la información personal de cada uno de los alumnos.

Se abrirá un brainStorming para que los alumnos propongan cómo organizar el trabajo para conseguir el resultado indicado en el menor tiempo posible.

everis
an NTT DATA Company

Sistemas de control de versiones

¿Qué es un control de versiones ?

Podemos pensar en un sistema de control de versiones o VCS como algo similar a una base de datos. Esta herramienta nos permitirá grabar una instantánea de todo nuestro proyecto en un momento determinado.

Posteriormente, podremos en el momento en que sea necesario comparar la última foto instantánea de nuestro proyecto con cualquier otra "versión" que hayamos grabado con anterioridad, comprobando así exactamente que diferencia una versión de otra.

everis
an NTT DATA Company

¿Porqué necesito un control de versiones ?

Un sistema de control de versiones aporta numerosas ventajas , enumeremos las más evidentes:

Colaboración

Sin un sistema de control de versiones , probablemente estés trabajando junto con otros compañeros en una carpeta compartida en red donde no es posible trabajar de manera colaborativa con un flujo de trabajo aceptable. Esta dinámica de trabajo es muy propensa a cometer errores, además es una mera cuestión de tiempo que alguien sobre escriba el trabajo de su compañero.

Con un SCV cualquier miembro del equipo puede trabajar con total libertad en cualquier fichero . Posteriormente podremos integrar todos estos cambios de manera simple y ordenada en una sola versión.

everis
an NTT DATA Company

Almacenar versiones (Correctamente)

Almacenar una versión de tu proyecto después de hacer cambios en él es un hábito más que saludable, sin embargo puede convertirse en una tarea tediosa y confusa rápidamente.

Pronto surgirán cuestiones sobre qué cambios deben almacenarse o cómo han de renombrarse las versiones sucesivas . Esto con el tiempo se degradará en una lista de nombre de fichero parecida a esta:

- Proyecto _acme
- Proyecto_Acme_v1
- Proyecto_Acme_revxx99B_sin_parametrizar
- proyecto Acme_revxx99B_sin_parametrizar_andres_ocutbre_2017_v4_o v_5_o unamezcla deyanimeacuerdo_3A)

Todo esto además de confuso y poco eficiente, hace cada vez más difícil responder a una cuestión esencial:

¿**Qué ha cambiado exactamente entre cada una de las versiones almacenadas** ?

everis
an NTT DATA Company

Recuperar versiones anteriores

Ser capaz de recuperar efectivamente versiones anteriores de un fichero o incluso de todo el proyecto, implica que no puedan cometerse errores irreversibles, de tal forma que cualquier fragmento de código erróneo que introduzcamos puede revertirse con unos cuantos clicks.

Comprender los cambios

Cada vez que subimos una nueva versión introducimos comentarios que ayudan a comprender los cambios que se han implementado. Además , si se trata código o un fichero en texto, podemos ver exactamente el cambio introducido.

02

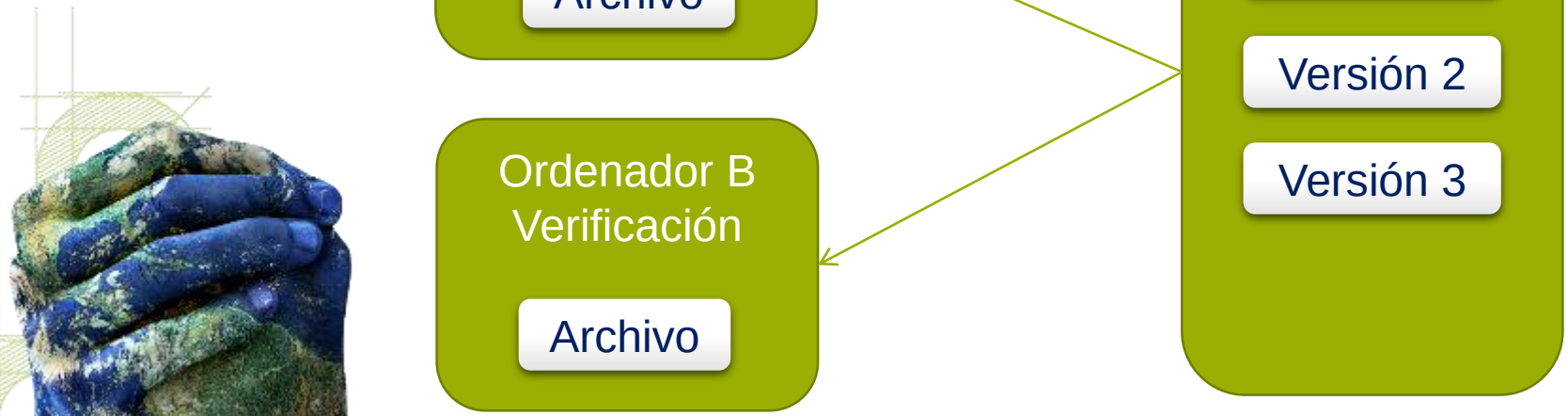
Software de control de versiones

everis
an NTT DATA Company

Sistemas centralizados y sistemas distribuidos

SCV Centralizados

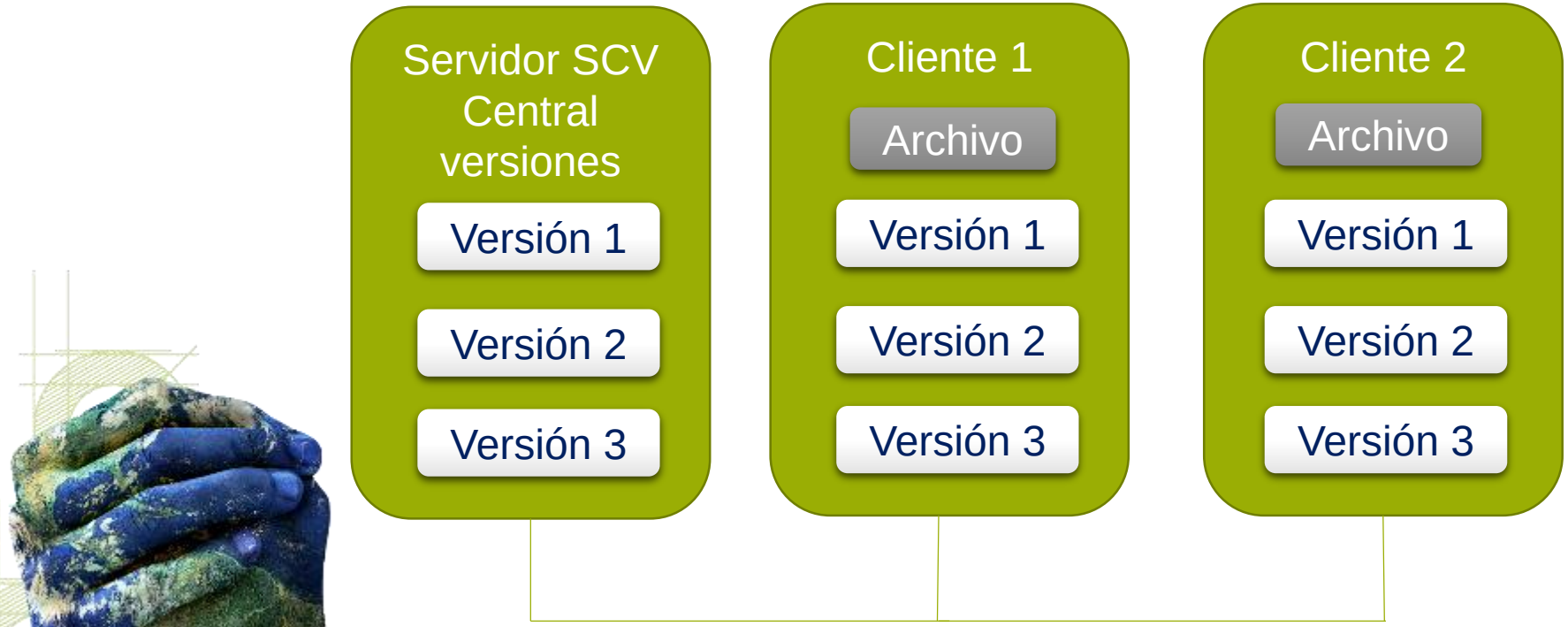
Estos sistemas tienen un único servidor que contiene todos los archivos versionados, y varios clientes que descargan los archivos desde ese lugar central. CVS, Subversion.



Sistemas centralizados y sistemas distribuidos

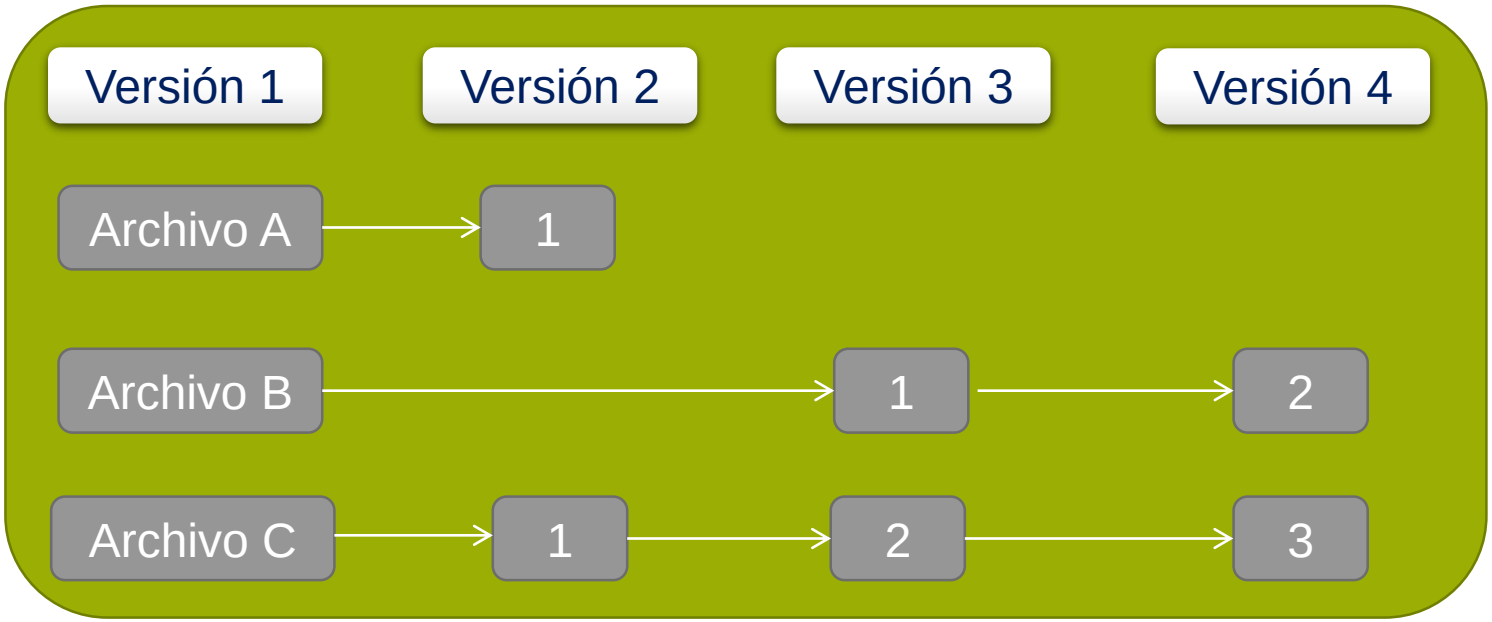
SCV Distribuidos

En un SCV, los clientes no sólo descargan la última instantánea de los archivos: replican completamente el repositorio así son independientes del servidor. GIT, Mercurial



everis
an NTT DATA Company

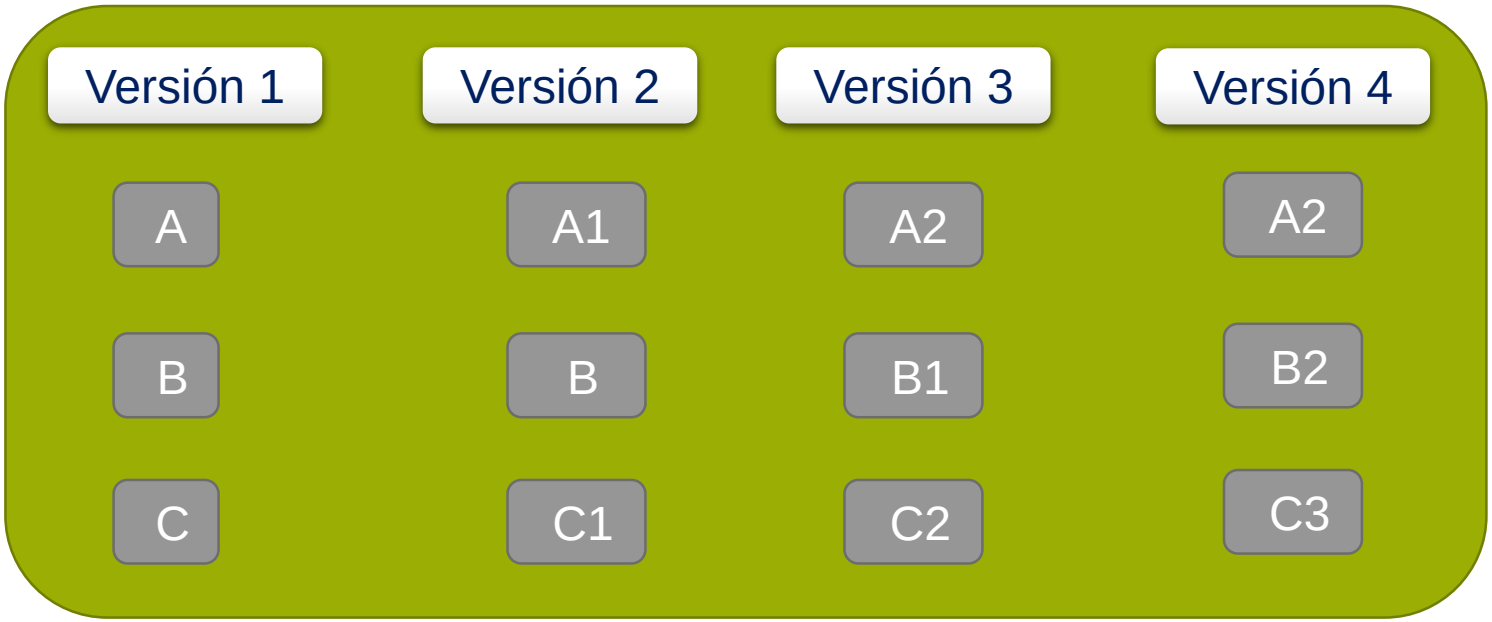
Versionado en subversión



Estos sistemas (CVS, Subversion, Perforce, Bazaar, etc.) modelan la información que almacenan como un conjunto de archivos y las modificaciones hechas sobre cada uno de ellos a lo largo del tiempo

everis
an NTT DATA Company

Versionado en GIT



Cada versión es un «back-up» de todo el contenido del repositorio en un momento concreto.

everis
an NTT DATA Company

Beneficios de GIT

Velocidad

Casi todas las operaciones que realiza GIT las ejecuta en el ordenador local. No necesita información de otro ordenador, por lo que la velocidad es mucho mayor que en sistemas centralizados.

Por ejemplo, para recuperar una versión anterior del proyecto no es necesario acudir al servidor porque nuestro ordenador local ya tiene una copia de todas las versiones, por lo que el acceso es casi instantáneo.

También podemos hacer «commit» de los cambios incluso cuando estamos «offline». Y cuando estemos conectados simplemente subir los cambios al servidor.

everis
an NTT DATA Company

Beneficios de GIT

Integridad

GIT realiza un verificación de la calidad de los datos al guardarlos, denominado «checksum» que es una forma de proteger la integridad de los datos.

Es imposible cambiar los contenidos de los archivos sin que GIT lo sepa.

No se puede perder información ni guardar archivos corrompidos sin que GIT lo detecte y lo impida.

GIT realiza el «checksum» utilizando un «hash» generado en SHA-1 y es la forma que tiene de identificar los archivos (no los identifica por su nombre).

everis
an NTT DATA Company

Beneficios de GIT

Tranquilidad

Las acciones en GIT siempre son modificables, es difícil hacer algo que provoque la perdida de datos o que sea inmodificable.

Después de hacer un «commit» es muy difícil perder datos puesto que se crean «snapshots» que se copian a todos los ordenadores que utilizan el proyecto.

Aunque el proyecto se trabaje sólo en un equipo se pueden utilizar sistemas como Github o Bitbucket para almacenar una copia del repositorio.

03 Workflow Básico con GIT

everis
an NTT DATA Company

El Workflow Básico en Control de versiones

El Repositorio

Antes de perdernos en comandos de GiT, será conveniente comprender cual es el flujo de trabajo básico. La punto de partida de todo esto es el Repositorio.

Pensad en un repositorio como una especie de Base de datos donde se almacenan todas la versiones y metadatos de tu proyecto. En GIT ese repositorio es una simple carpeta cuyo nombre es :

.git

A la hora de obtener ese repositorio pueden ocurrir dos cosas:

- Tenemos un proyecto que aún no está bajo control de versiones. En ese caso inicializaremos nuestro repositorio local para ese proyecto.
- Tenemos un proyecto en un servidor remoto, que deberemos clonar y descargar para contar con nuestro repositorio local. Para ello necesitaremos una URL a la que conectar, en nuestro caso es :

https://github.com/Ginxo/becajava07

everis
an NTT DATA Company

El Workflow Básico en Control de versiones

Commit

Podemos definir el commit como un conjunto de cambios específicos cada uno de estos conjuntos de cambios conformarán una versión diferente de nuestro proyecto. Por este motivo ,cada commit representa una foto instantánea de todo nuestro proyecto en un determinado momento.

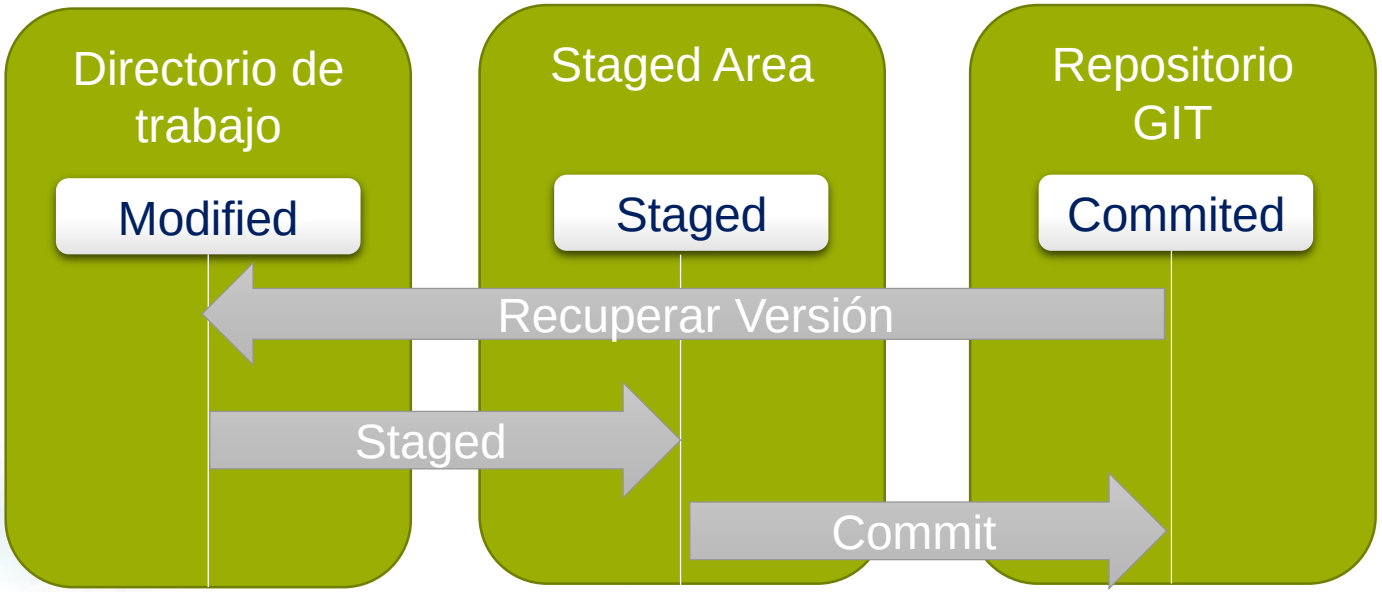
Staging Area

El mero hecho de modificar un archivo no implica necesariamente que esa modificación deba subir como nueva versión. Debemos indicar a GIT explícitamente que cambios queremos que formen parte de la próxima versión. Para ello añadiremos los ficheros a la llamada **Staging Area** .

Una vez hecho esto es el momento de realizar el commit, como buena práctica incluiremos siempre un breve comentario descriptivo del cambio, y lo subiremos al repositorio.

everis
an NTT DATA Company

Estados de los archivos en GIT



everis
an NTT DATA Company

04 Comandos GIT

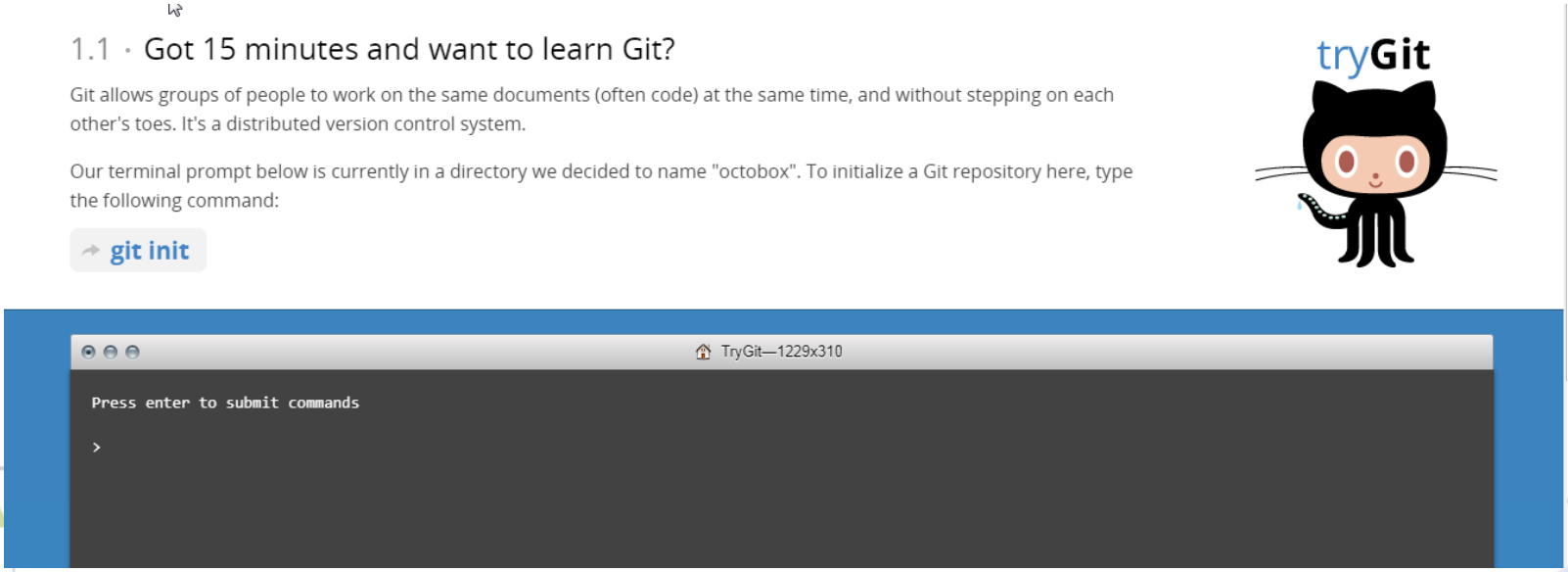
everis
an NTT DATA Company

Principales comandos GIT

Antes de instalar GIT y crear nuestro repositorio conoceremos los principales comandos que nos ayudarán a realizar las acciones más comunes .

Ejercicio 2. Tutorial de comandos de consola.

La URL https://try.github.io/levels/1/challenges/1 nos proporciona un emulador de consola GIT donde se nos guiará a través de las operaciones básicas de GIT a través de la línea de comandos.



everis
an NTT DATA Company

05

TortoiseGit

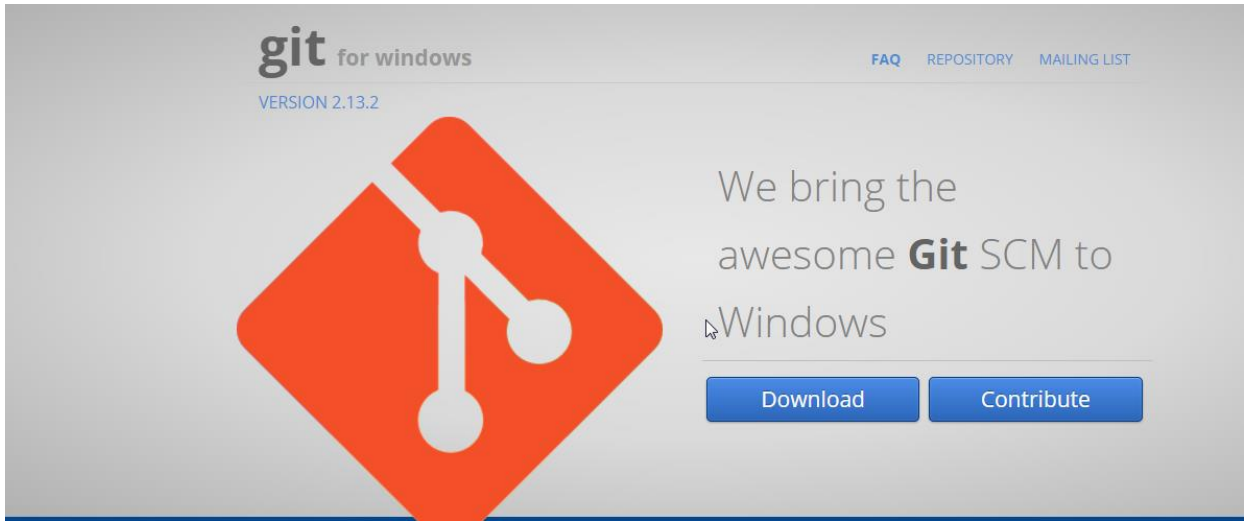
Instalación

Prerrequisitos

Antes de instalar Tortoisegit necesitaremos un cliente GIT en línea de comandos quien proveerá el ejecutable git.exe

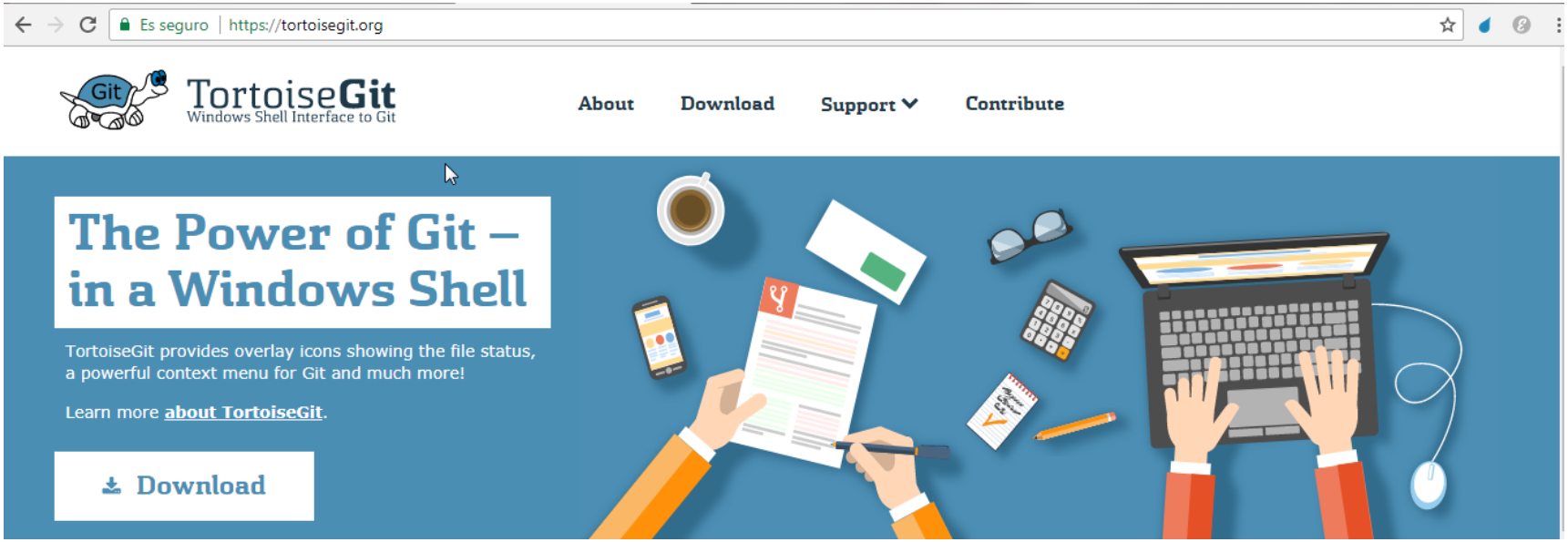
Git for Windows es el más habitual. Podemos descargarlo aquí:

https://git-for-windows.github.io/



everis
an NTT DATA Company

TortoiseGit



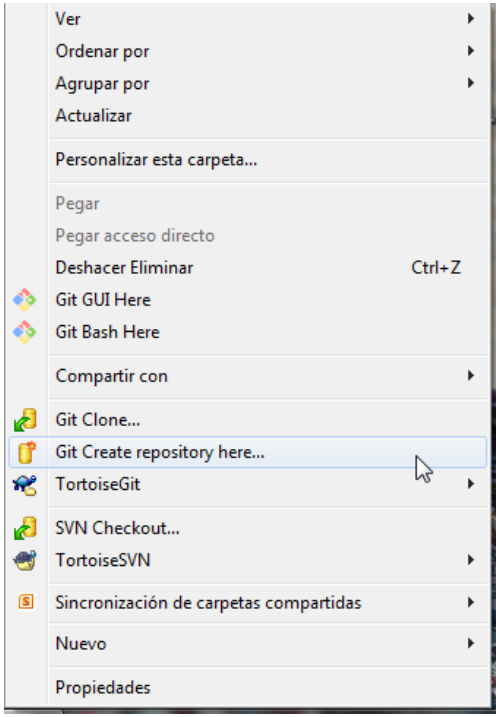
Tortoisegit es una extensión shell de Windows para git basada en **TortoiseSVN**. Como no es un pluggin de ningún IDE específico como los existentes para eclipse o visual Studio, puede ser utilizado de forma independiente y con cualquier tipo de fichero.

Como veremos , podemos acceder las operaciones más comunes de git tales como commit, push , mostrar logs o crear branches a través del menú contextual del explorador de Wndows.

https://tortoisegit.org/download/

Crear Repositorio Local.

Crear un repositorio local es muy sencillo. Simplemente hacemos click con el botón derecho del ratón y seleccionamos “crear repositorio aquí” desde el menú contextual.

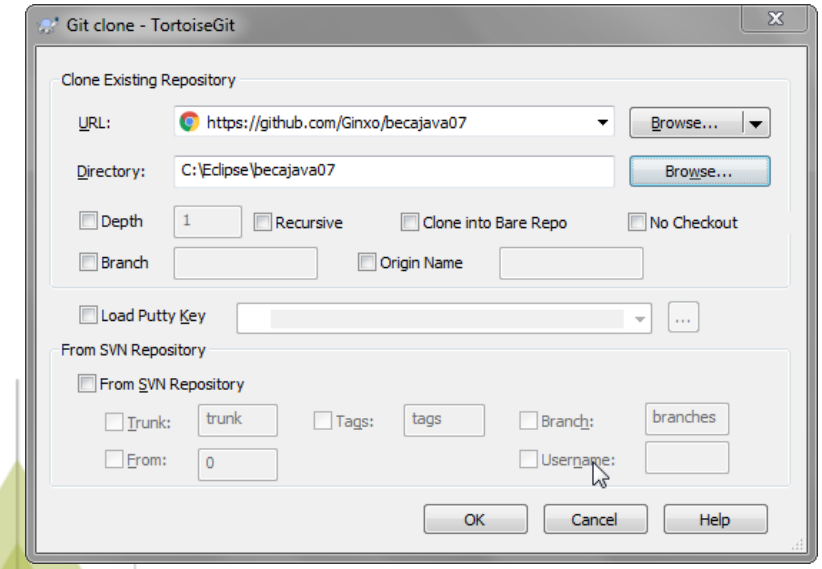


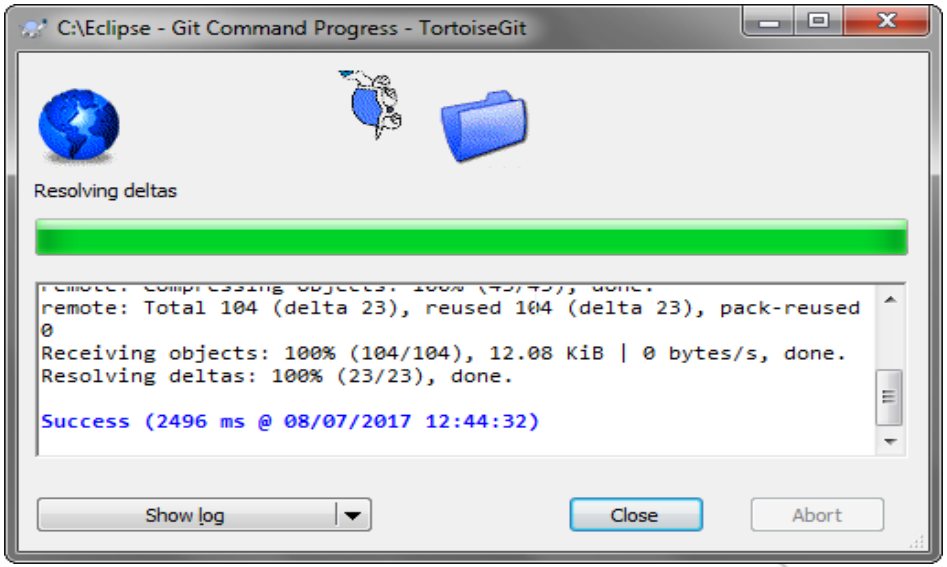
Clonar repositorio.

Con esta operación crearemos una copia completa de un repositorio remoto. Para esto tan sólo tendremos que situarnos en un directorio remoto y hacer clic sobre **git clone.**

En el diálogo emergente indicaremos la URL del repositorio de este curso:

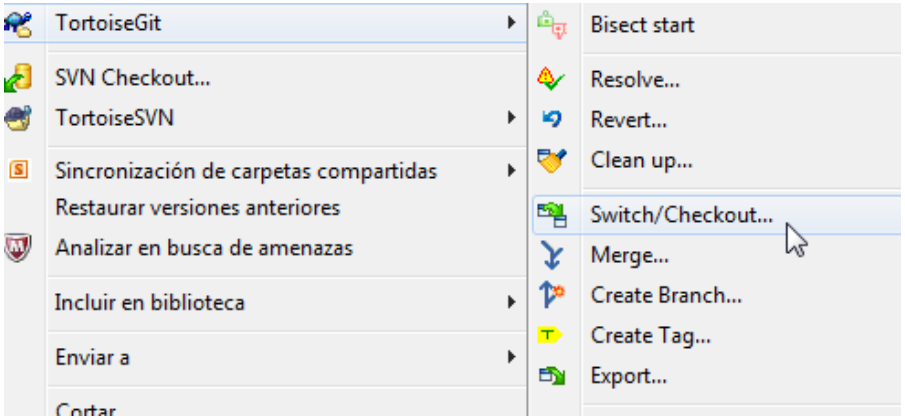
https://github.com/Ginxo/becajava07

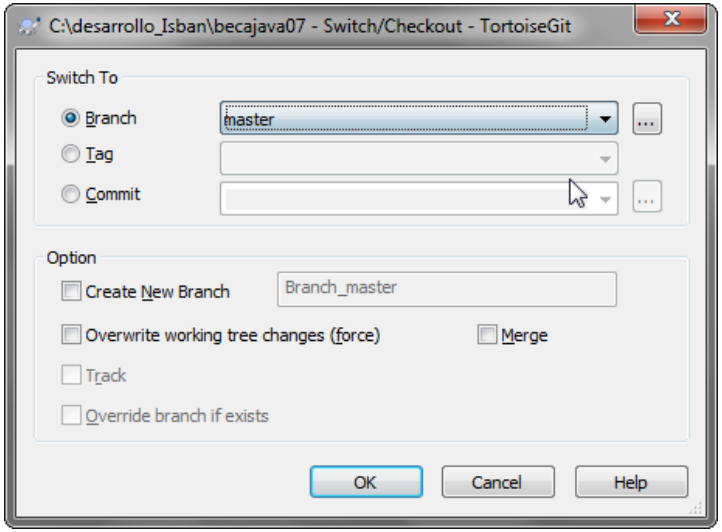




Checkout

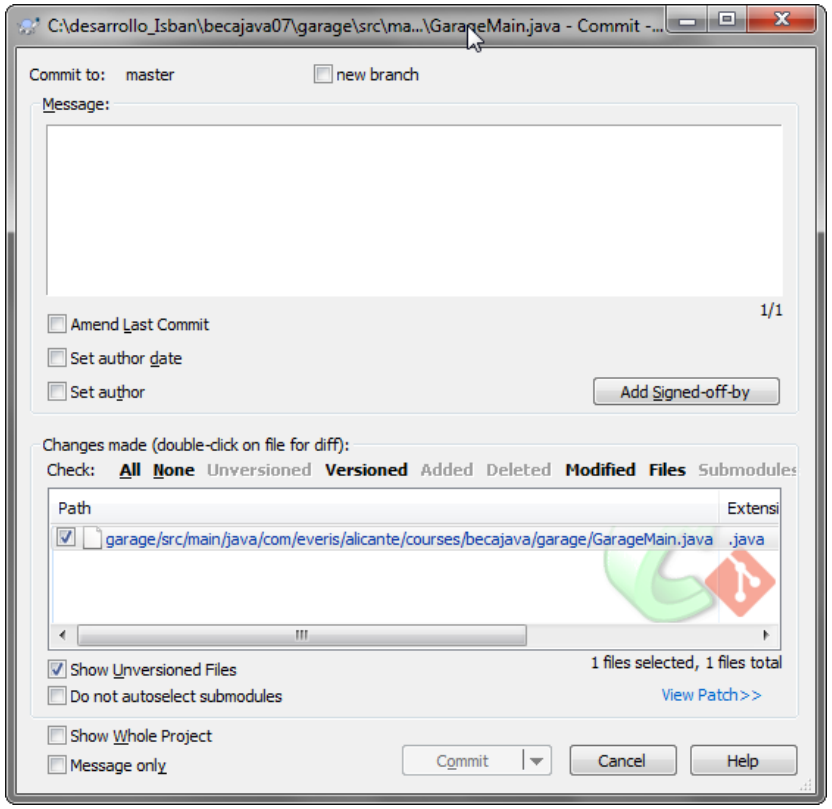
Desde esta opción podemos cambiar de rama “activa” , los commit se realizarán sobre esa rama. También podemos usar esta opción para crear directamente una nueva rama.





Commit

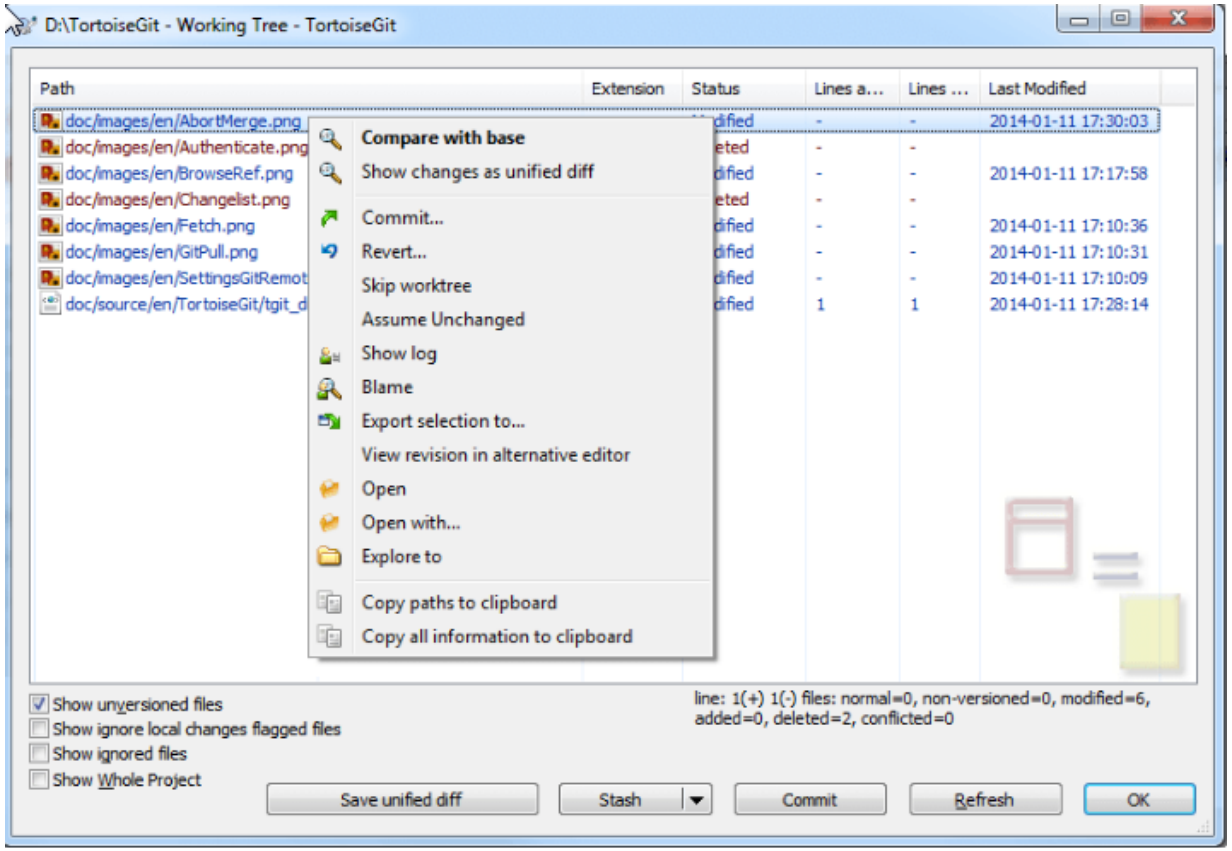
Si no hay conflictos (veremos después el concepto y cómo resolverlo) podemos hacer **commit** de nuestros cambios sobre la rama seleccionada.



everis
an NTT DATA Company

Check for modifications

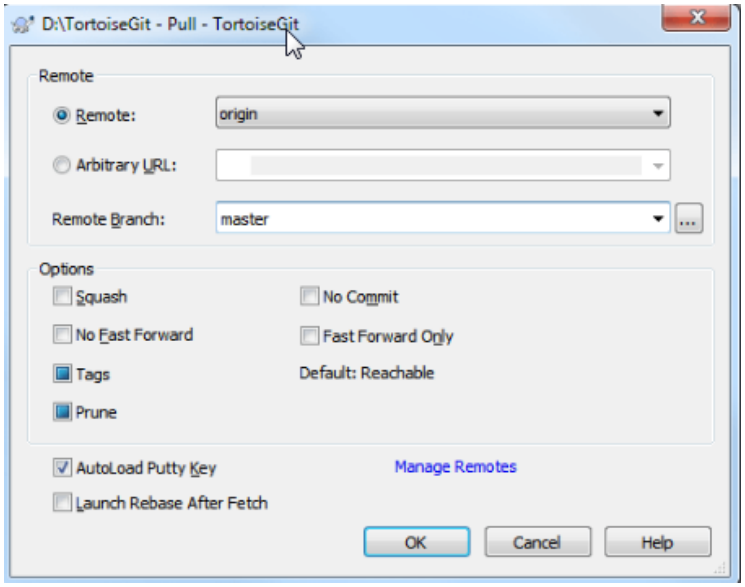
A menudo puede resultar útil qué ficheros han cambiado y qué ficheros han sido modificados por otros. Este diálogo mostrará todos los ficheros que han sido modificados en tu working tree . Así como los ficheros no versionados que se encuentren en él.

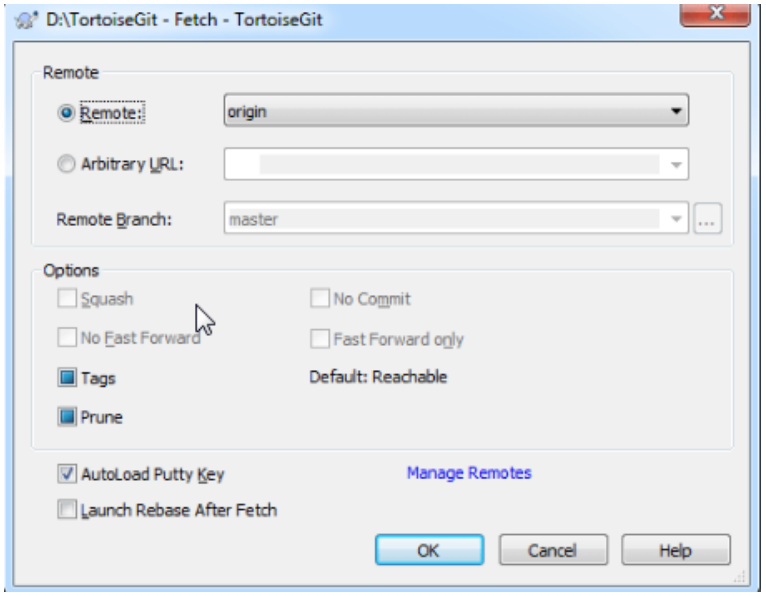


everis
an NTT DATA Company

Pull y Fetch

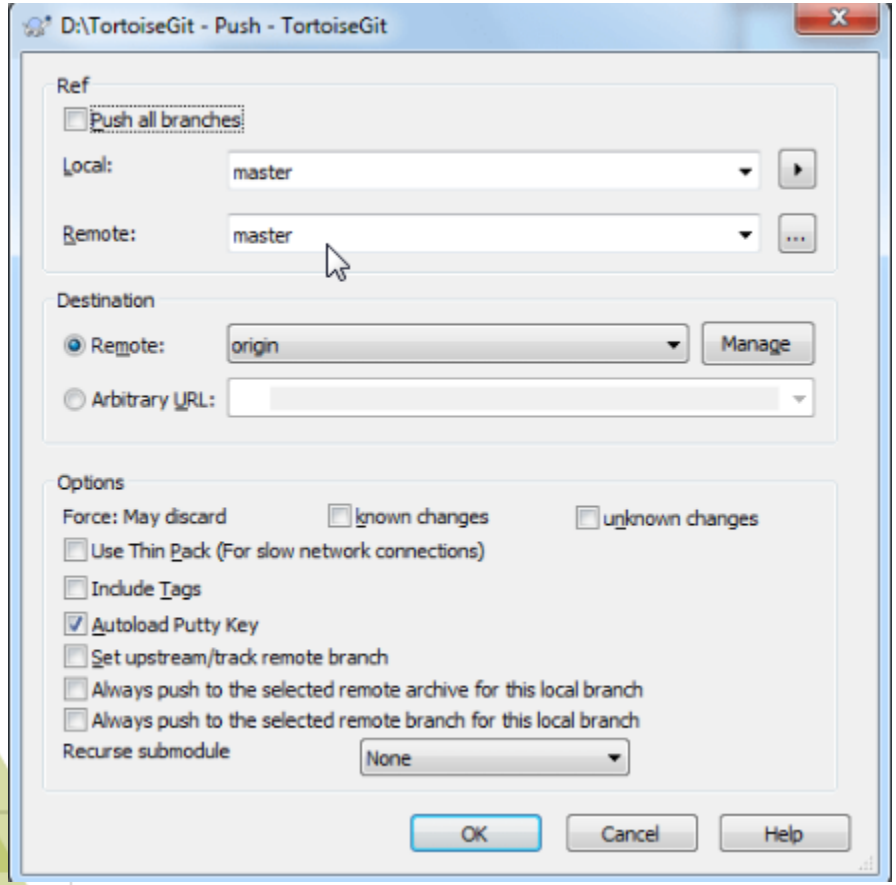
En ambos casos se trata de descargar modificaciones desde un repositorio remoto. La diferencia entre ambos radica en que el **Fetch**, solo descargará los cambios, mientras que **pull** hará además un "**merge**" de esos cambios con los de nuestra rama.





Push

Con el comando Push enviaremos nuestros cambios a un repositorio remoto.



Branch

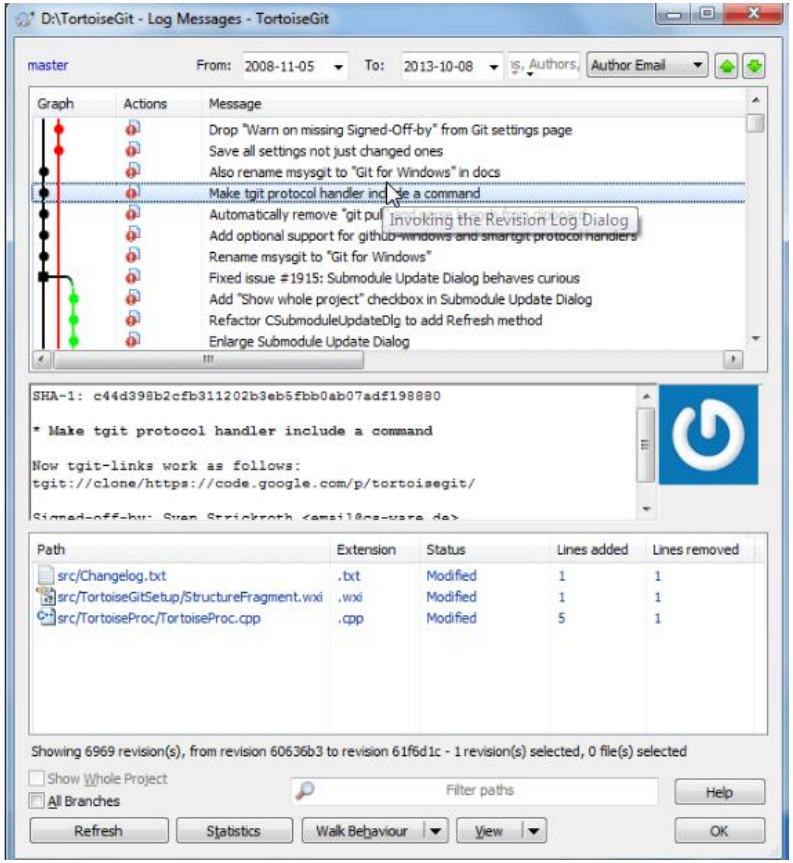
- **Local**: Rama que pasará como push al repositorio remoto.
- **Remote**: Rama del repositorio remoto.

Destination

- **Remote:** Selecciona un repositioro remoto.
- **Arbitary URL:** URL del repositorio remoto.

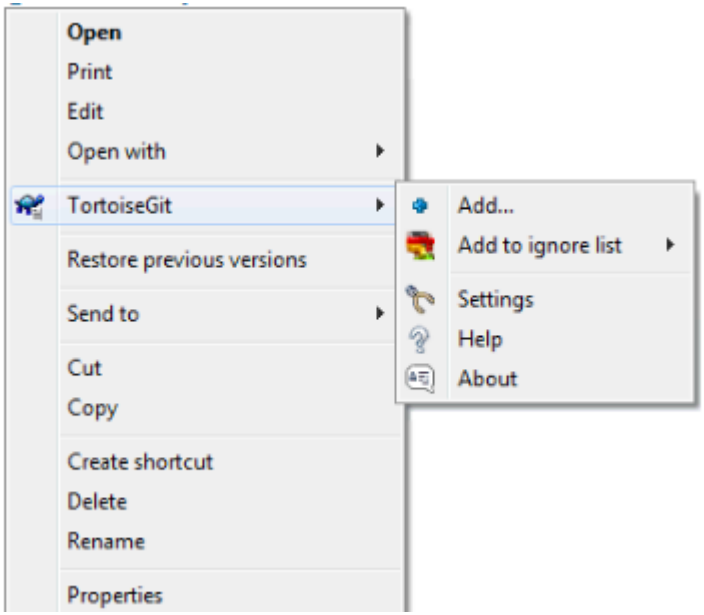
El diálogo de log.

Para cada cambio que se realiza y se comitea, se provee una breve descripción donde se indica que ha cambiado y porqué. El comando Log recuperará todos esos comentarios, mostrándolos de manera ordenada.



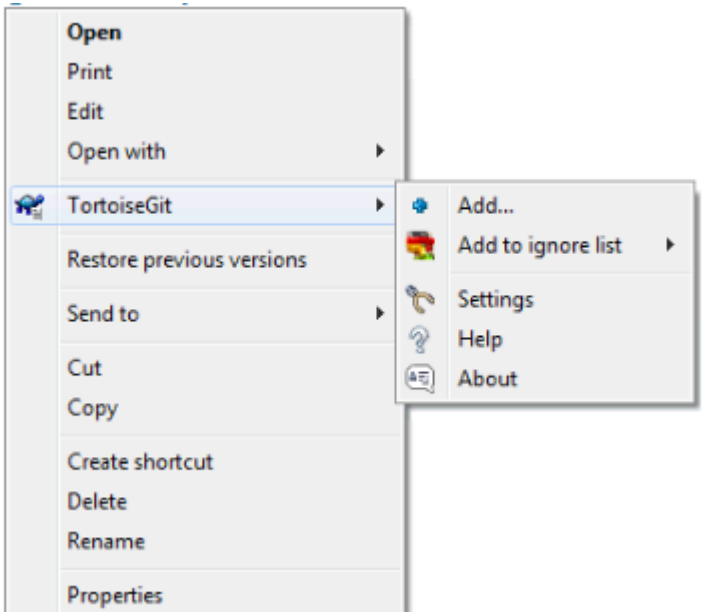
Añadiendo ficheros (Add).

Durante el desarrollo de nuestro proyecto añadiremos nuevos ficheros al repositorio. Para ello no basta sólo con que estén en nuestro workspace, hay que indicar a git explícitamente que queremos incorporar esos ficheros al repositorio.



Ignorar ficheros

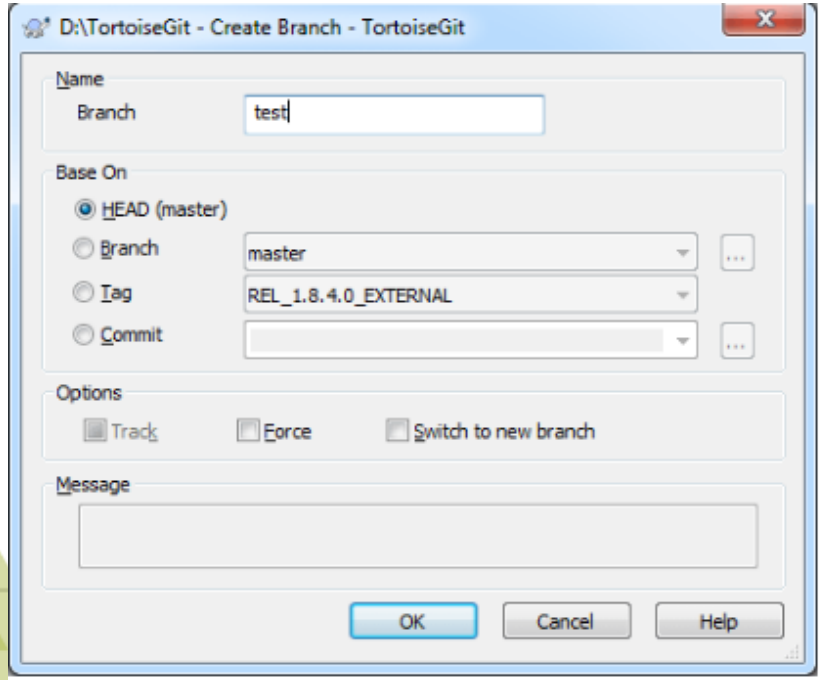
Del mismo modo , en ocasiones necesitaremos que ficheros o directorios no suban al repositorio , e incluso que no aparezcan en la lista de ficheros modificados. Para ello usaremos el comando **add to ignore list**

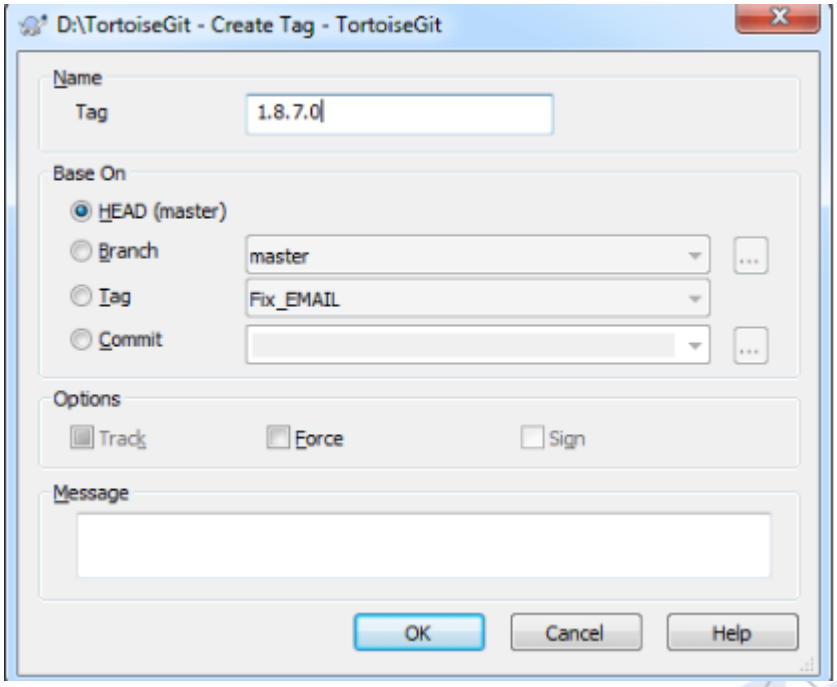


Branches y tags

Una de las características de cualquier SCV es la habilidad de aislar cambios en una línea separada de desarrollo Esta línea de desarrollo separada es lo que conocemos como Branch o Rama.

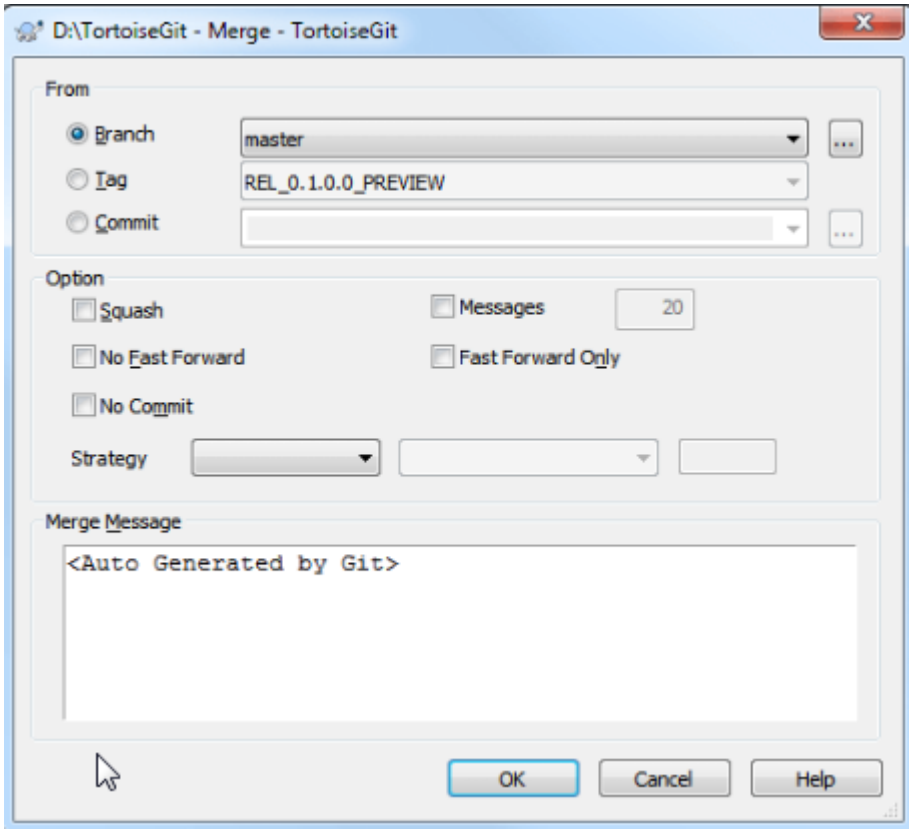
Otra de estas características es la capacidad para marcar una revisión particular, de modo que se pueda recrear, un entorno en un momento determinado del desarrollo. Este “marcado” es lo que se conoce como Tag.





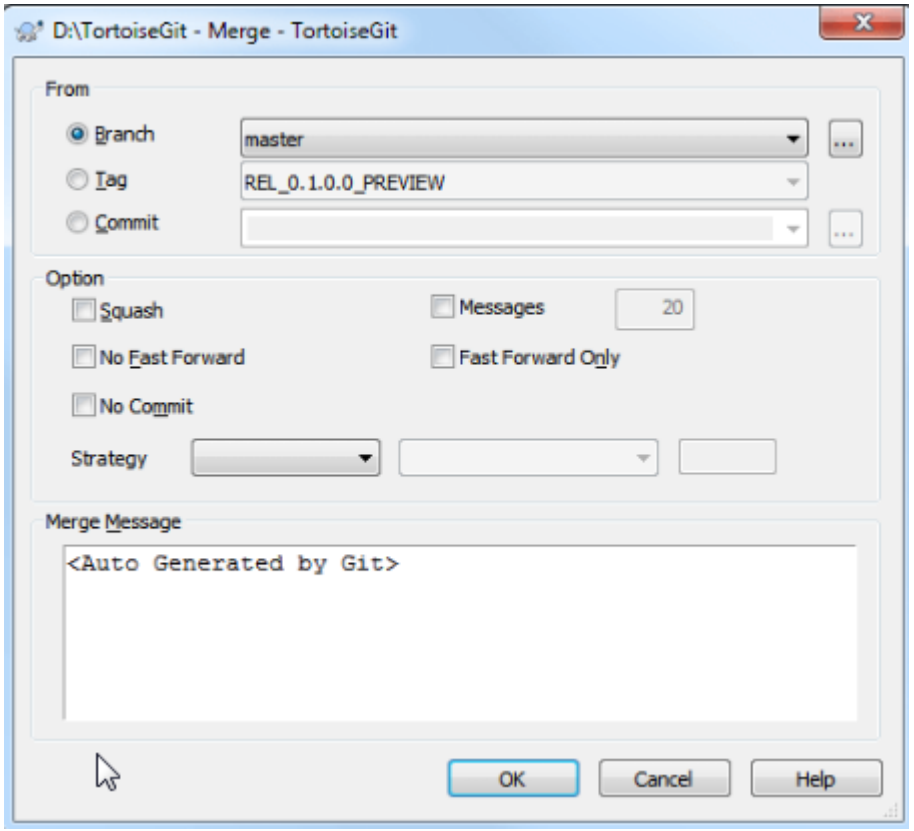
Merge

Mantener tus commits aislados en una rama separada puede ser de gran ayuda, pero tarde o temprano llegará un momento en el que tengamos que integrar nuestros cambios desde una rama a otra. Por ejemplo cuando terminemos nuestro desarrollo deberemos integrarlo dentro de una rama de producción. Para esto usaremos el comando **Merge.**



Merge

Mantener tus commits aislados en una rama separada puede ser de gran ayuda, pero tarde o temprano llegará un momento en el que tengamos que integrar nuestros cambios desde una rama a otra. Por ejemplo cuando terminemos nuestro desarrollo deberemos integrarlo dentro de una rama de producción. Para esto usaremos el comando **Merge.**

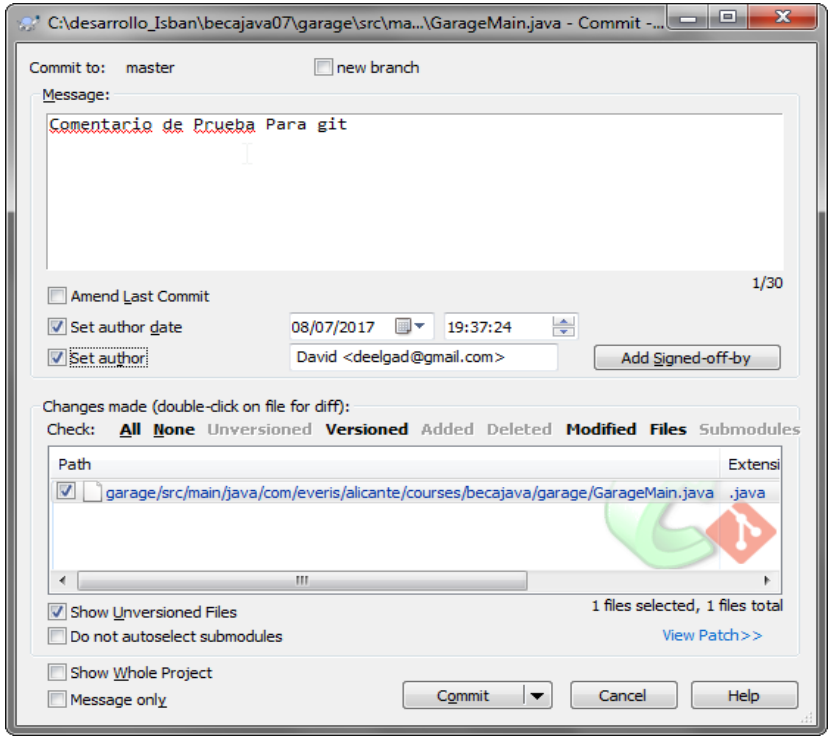


06

Resolución de Conflictos

Resolución de conflictos

En algunas situaciones mergeando no conseguiremos el resultado esperado, en ese supuesto Git nos notificará que se han producido conflictos porque no se han podido combinar correctamente los cambios. Por ejemplo porque alguien ha modificado la misma línea de código de dos maneras diferentes. En esos casos será el usuario quien tenga que decidir qué contenido debe prevalecer.

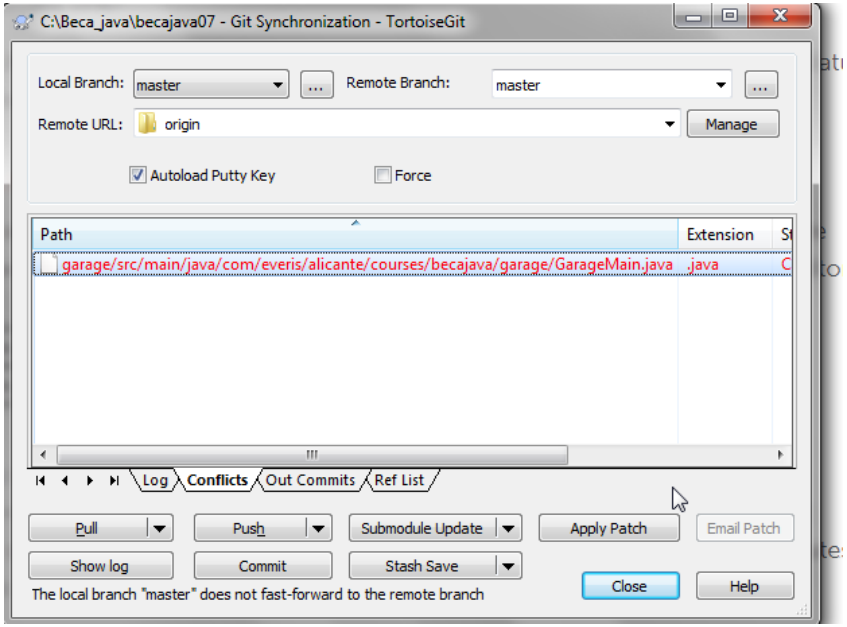


everis
an NTT DATA Company

Cómo resolver conflictos

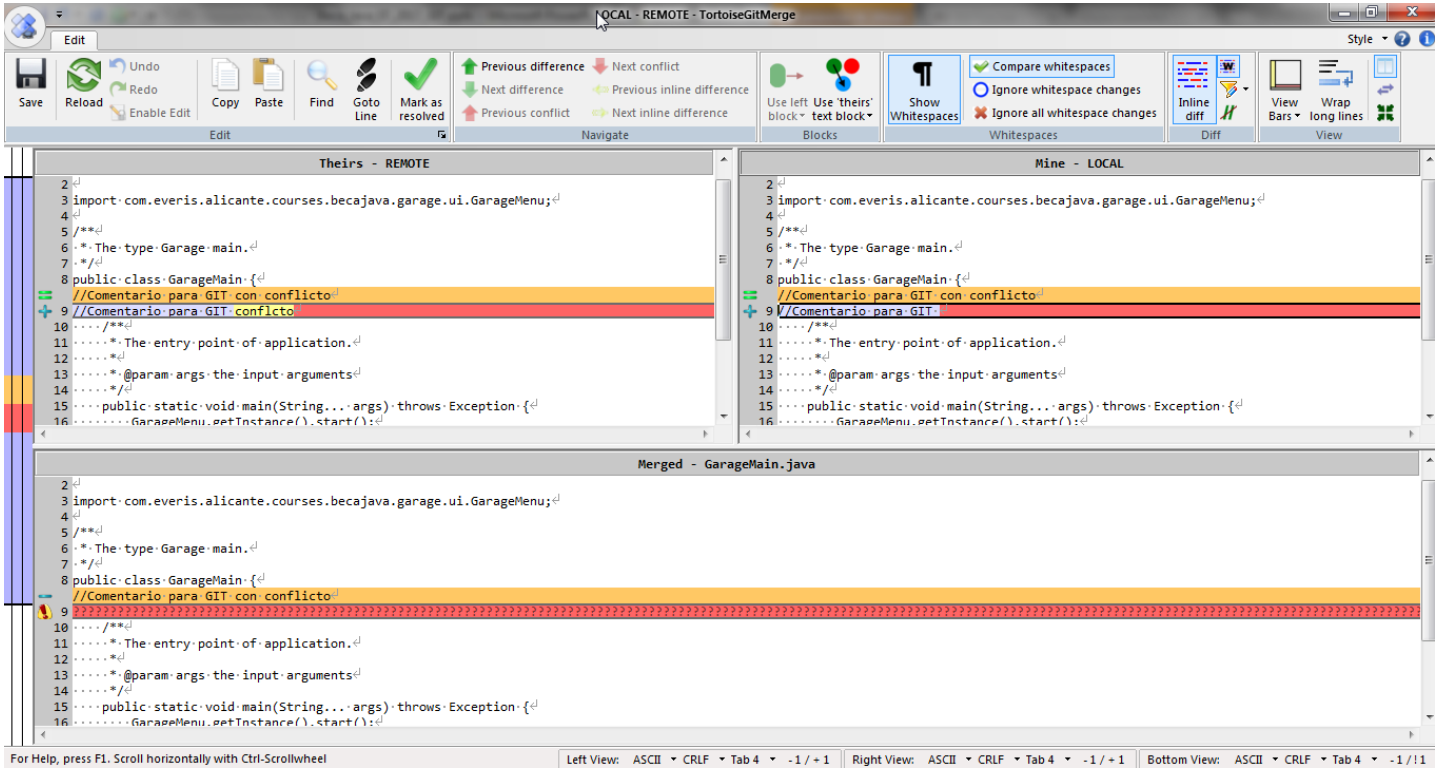
Cuando nos enfrentamos a un conflicto de Merge, el primer paso debe ser siempre comprender qué ha pasado. Los supuestos más comunes son :

- Alguien editó la misma línea del mismo fichero
- Alguien borró el mismo fichero que has editado.
- Alguien añadió un fichero con el mismo nombre que el que tu has añadido.



Comparando ficheros

Haciendo clic sobre el fichero en conflicto abriremos el comparador de ficheros (por defecto tortoiseGitMerge). Podremos seleccionar entonces que fichero, línea o parte del cambio debe prevalecer.



everis
an NTT DATA Company

Ejercicio Final

Para poner en práctica todo lo aprendido, seguiremos los siguientes pasos:

- Crearemos nuestro usuario de (gitHubhttps://github.com/)
- Crearemos un fork desde el repositorio principal
- Clonaremos nuestro Fork al repositorio local
- Editaremos el fichero del ejercicio inicial alumnos.xml
- Resolveremos los conflictos que se presenten.