

Introducción al Sistema de Control de Versiones Distribuido Git

Antonio García Domínguez

Universidad de Cádiz



16 octubre 2014

Contenidos

- 1 Introducción
- 2 Trabajo local
- 3 Trabajo distribuido

Materiales en

<http://osl2.uca.es/wikiinformacion/index.php/Git> y

<http://gitorious.org/curso-git-osluca>.

Una versión más antigua de la presentación (para Linux) está disponible en <http://goo.gl/2sCoK6>.

Contenidos

- 1 **Introducción**
 - Antecedentes
 - Tipos de SCV
- 2 Trabajo local
- 3 Trabajo distribuido

Contenidos

- 1 **Introducción**
 - Antecedentes
 - Tipos de SCV
- 2 Trabajo local
- 3 Trabajo distribuido

Historia de los SCV

Sin red, un desarrollador

1972 Source Code Control System

1980 Revision Control System

Centralizados

1986 Concurrent Version System

1999 Subversion («CVS done right»)

Distribuidos

2001 Arch, monotone

2002 Darcs

2005 Git, Mercurial (hg), Bazaar (bzt)

Historia de Git

Antes de BitKeeper

Para desarrollar Linux, se usaban parches y tar.gz.

BitKeeper

02/2002 BitMover regala licencia BitKeeper (privativo)

04/2005 BitMover retira la licencia tras roces

Git

04/2005 Linus Torvalds presenta Git, que ya reúne ramas

06/2005 Git se usa para gestionar Linux

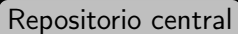
02/2007 Git 1.5.0 es utilizable por mortales

09/2014 Última versión: Git 2.1.2

Contenidos

- 1 **Introducción**
 - Antecedentes
 - Tipos de SCV
- 2 Trabajo local
- 3 Trabajo distribuido

SCV centralizados

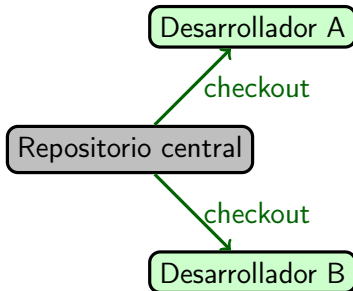


Repositorio central

A diagram showing a single rounded rectangle labeled "Repositorio central", representing a centralized repository.

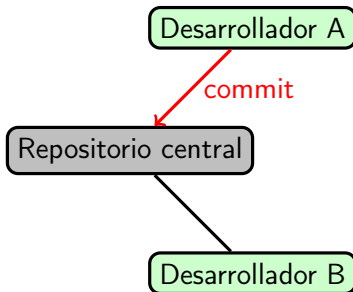
Tenemos nuestro repositorio central con todo dentro.

SCV centralizados



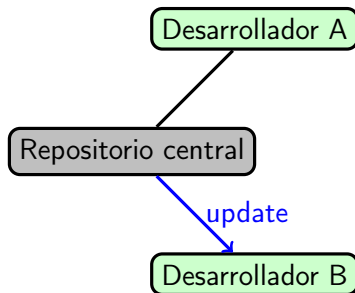
Los desarrolladores crean **copias de trabajo**.

SCV centralizados



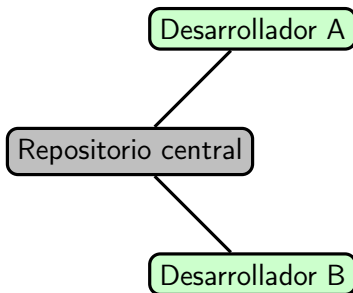
El desarrollador A manda sus cambios al servidor.

SCV centralizados



El desarrollador B los recibe.

SCV centralizados



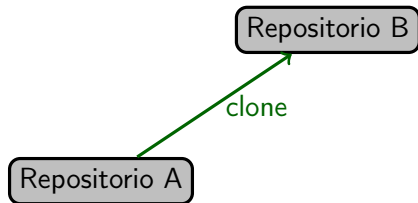
¿Y si se cae el servidor, o la red?

SCV distribuidos

Repositorio A

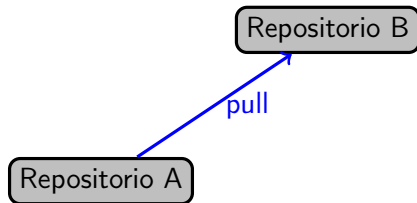
Tenemos nuestro repositorio.

SCV distribuidos



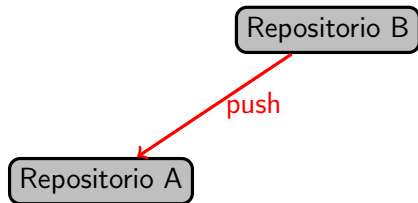
Alguien **clona** el repositorio.

SCV distribuidos



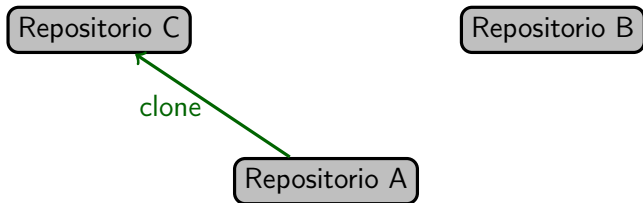
De vez en cuando se trae nuestros cambios recientes.

SCV distribuidos



De vez en cuando nos manda sus cambios.

SCV distribuidos



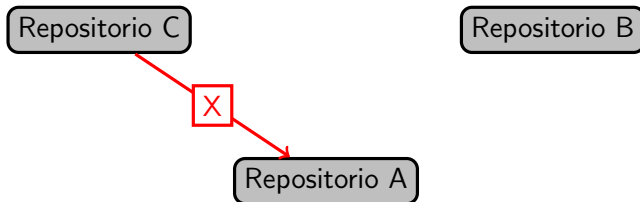
Viene otro desarrollador.

SCV distribuidos



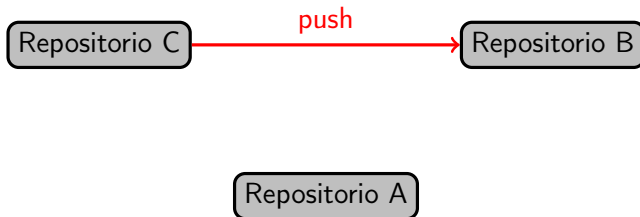
Intenta hacer sus cambios locales...

SCV distribuidos



Pero no le funciona, o no tiene permisos para ello.

SCV distribuidos



Se los pasa al otro desarrollador sin más.

SCV distribuidos



La diferencia entre los repositorios es *social*, no técnica.

Ventajas de un SCV distribuido (I)

Rapidez

- Todo se hace en local: el disco duro es más rápido que la red, y cuando esté todo en caché será más rápido aún
- Clonar un repositorio Git suele tardar *menos* que crear una copia de trabajo de SVN, y ocupa menos

Revisiones pequeñas y sin molestar

- Nadie ve nada nuestro hasta que lo mandamos
- Podemos ir haciendo revisiones pequeñas intermedias
- Sólo mandamos cuando compila y supera las pruebas
- Podemos hacer experimentos de usar y tirar

Ventajas de un SCV distribuido (I)

Rapidez

- Todo se hace en local: el disco duro es más rápido que la red, y cuando esté todo en caché será más rápido aún
- Clonar un repositorio Git suele tardar *menos* que crear una copia de trabajo de SVN, y ocupa menos

Revisiones pequeñas y sin molestar

- Nadie ve nada nuestro hasta que lo mandamos
- Podemos ir haciendo revisiones pequeñas intermedias
- Sólo mandamos cuando compila y supera las pruebas
- Podemos hacer experimentos de usar y tirar

Ventajas de un SCV distribuido (II)

Trabajo sin conexión

- En el tren, avión, autobús, etc.
- Aunque no tengamos permisos de escritura
- Aunque se caiga la red, se puede colaborar

Robustez

Falla el disco duro del repositorio bendito. ¿Qué hacer?

- Centralizado: copias de seguridad
- Distribuido: copias de seguridad y/o colaborar por otros medios

Ventajas de un SCV distribuido (II)

Trabajo sin conexión

- En el tren, avión, autobús, etc.
- Aunque no tengamos permisos de escritura
- Aunque se caiga la red, se puede colaborar

Robustez

Falla el disco duro del repositorio bendito. ¿Qué hacer?

- Centralizado: copias de seguridad
- Distribuido: copias de seguridad y/o colaborar por otros medios

Cuándo NO usar Git

Git no escala ante muchos ficheros binarios

- No sirve para llevar las fotos
- Ni para almacenar vídeos

Git no guarda metadatos

- No guarda el dueño de los ficheros
- Sólo guarda si un fichero es ejecutable o no
- No sirve como sistema de copias de seguridad

Contenidos

1 Introducción

2 Trabajo local

- Primeras revisiones
- Conceptos
- Operaciones comunes

3 Trabajo distribuido

Contenidos

- 1 Introducción
- 2 Trabajo local
 - Primeras revisiones
 - Conceptos
 - Operaciones comunes
- 3 Trabajo distribuido

Instalación de Git en Windows

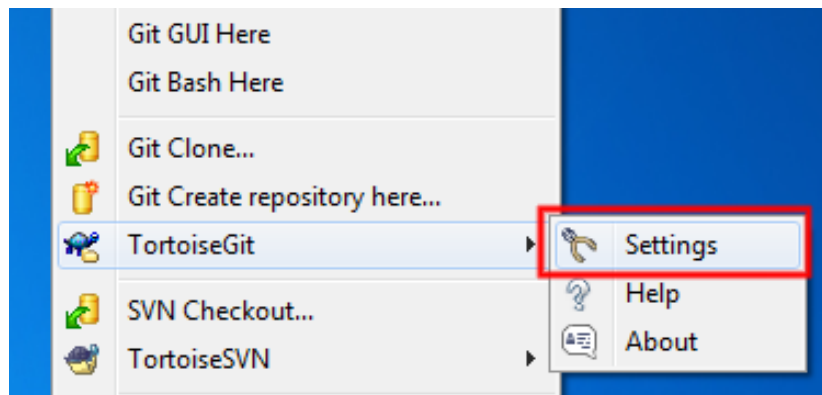
Clientes libres

- El original: [Git for Windows](http://msysgit.github.io/) (<http://msysgit.github.io/>)
 - “Git Bash” permite usar todas las órdenes típicas en Linux
 - “Git GUI” permite preparar y hacer revisiones de forma gráfica
- Como TortoiseSVN: [TortoiseGit](http://code.google.com/p/tortoisegit/) (<http://code.google.com/p/tortoisegit/>)
 - Para aprovechar toda la potencia de Git, hay que complementarlo con los de arriba

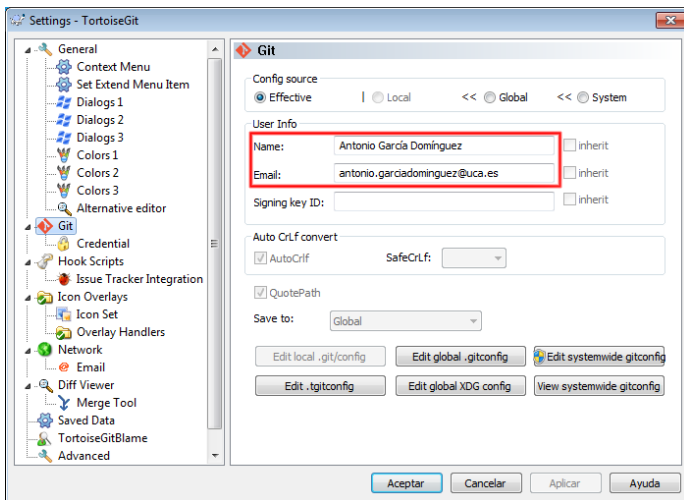
Clientes privativos

- \$79/persona: [SmartGit](http://www.syntevo.com/smartgit/) (<http://www.syntevo.com/smartgit/>)
- Gratis por ahora: [SourceTree](http://www.sourcetreeapp.com/) (<http://www.sourcetreeapp.com/>)

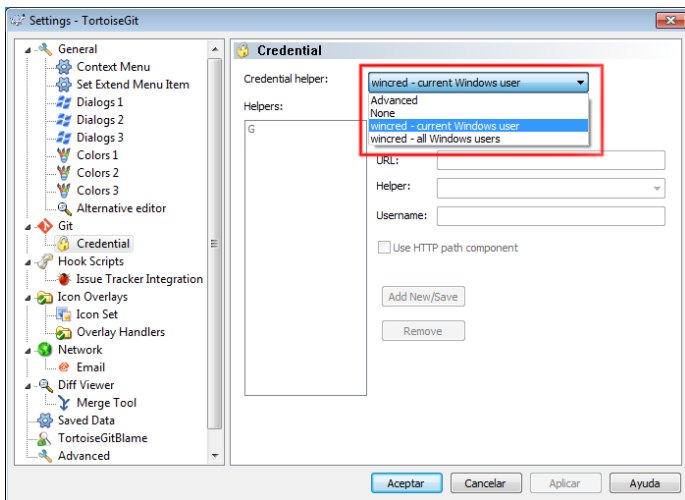
Configuración inicial: acceso a opciones



Configuración inicial: datos personales

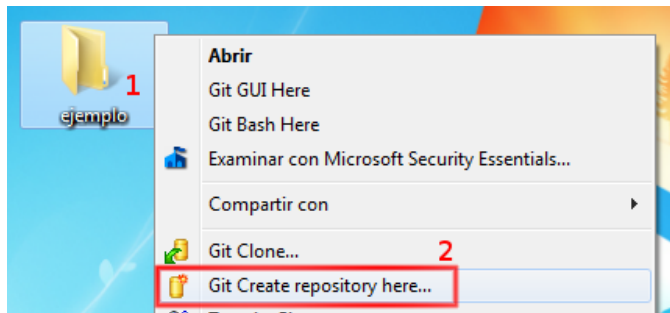


Configuración inicial: gestor de credenciales



Creación de un repositorio: orden inicial

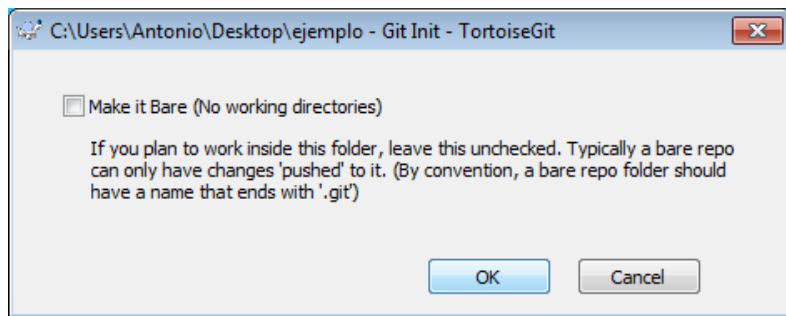
Sólo tenemos que crear un nuevo directorio y decirle a Git que cree un repositorio ahí.



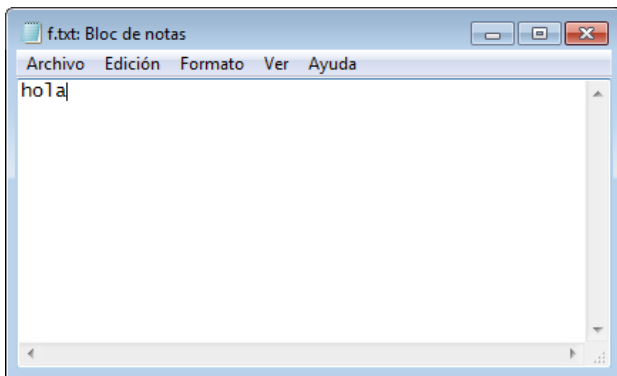
Creación de un repositorio: repositorios “pelados”

Un repositorio *bare* no tiene un directorio de trabajo asociado: sólo lleva el histórico. Normalmente se usan en servidores, como en `des-sinf.uca.es`.

Ahora queremos uno normal, por lo que dejamos la caja sin marcar.

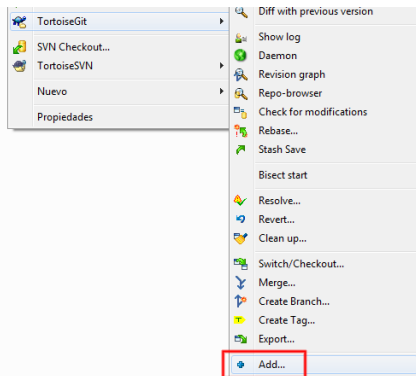


Nuestra primera revisión en «master»



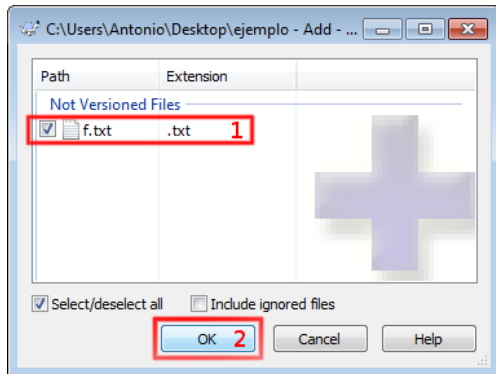
Creamos un fichero *f.txt* con “hola”.

Nuestra primera revisión en «master»



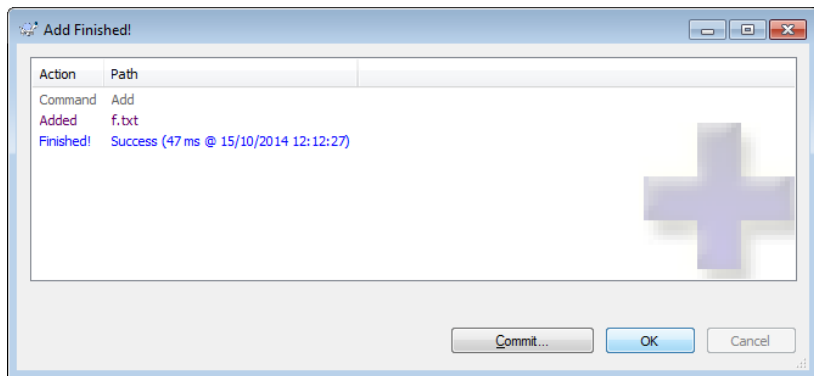
Añadimos el fichero a control de versiones mediante “Add”.

Nuestra primera revisión en «master»



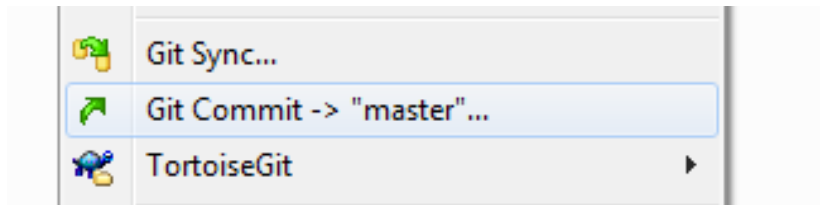
Marcamos el fichero y pulsamos en “OK”.

Nuestra primera revisión en «master»



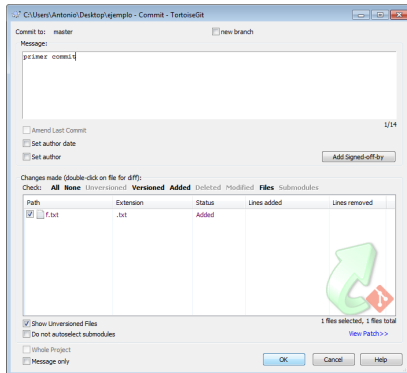
Se nos confirma el añadido.

Nuestra primera revisión en «master»



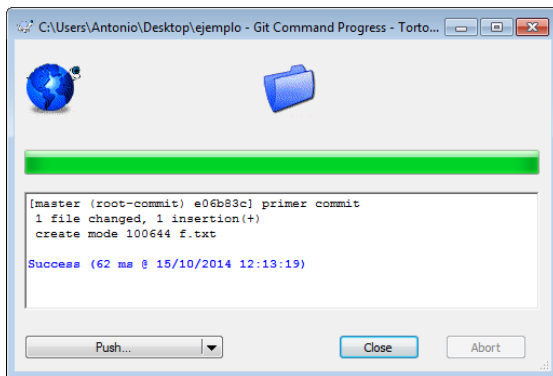
Usamos “Commit” para iniciar la creación de la revisión.

Nuestra primera revisión en «master»



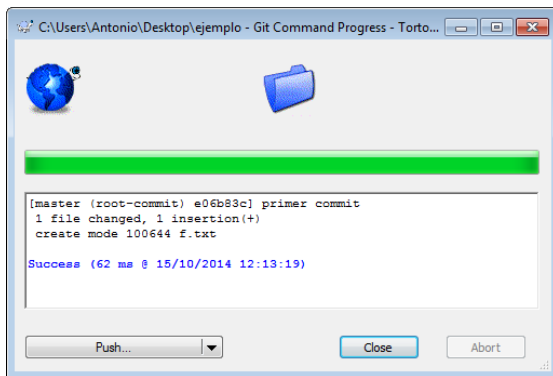
Introducimos el mensaje. En des-sinf.uca.es, no debemos olvidar poner refs #XYZ al final de la primera línea.

Nuestra primera revisión en «master»



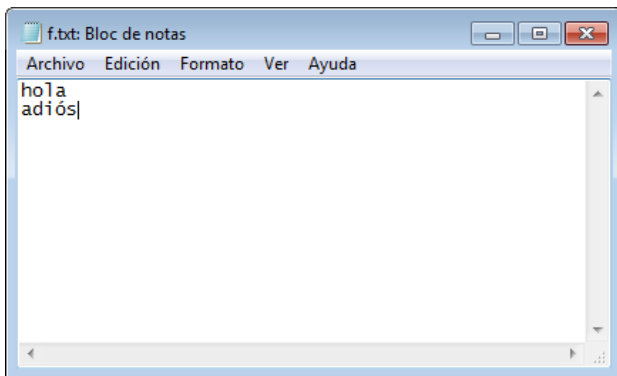
Se nos confirma el commit. **Ojo:** no se envía a ningún lado. Para eso existe "Push", que veremos después.

Nuestra primera revisión en «master»



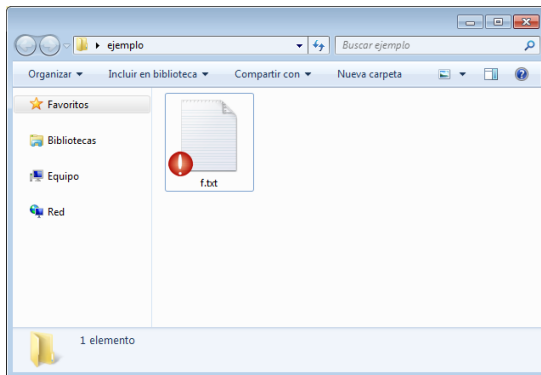
Se nos confirma el commit. **Ojo:** no se envía a ningún lado. Para eso existe “Push”, que veremos después.

Nuestra segunda revisión en «master»



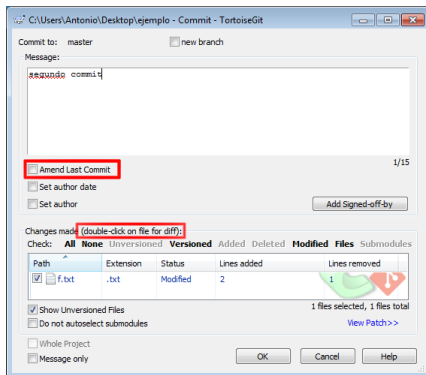
Añadimos una línea con “adiós” a *f.txt*.

Nuestra segunda revisión en «master»



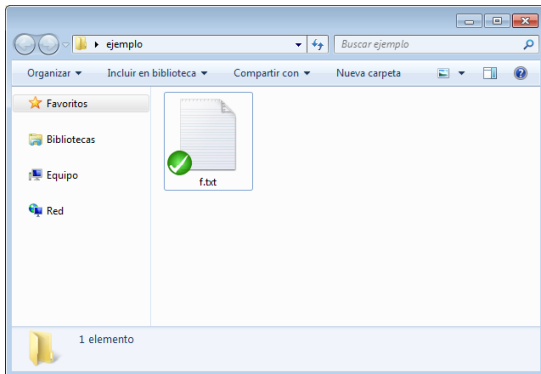
Se cambia el emblema de *f.txt*, indicando que su contenido no coincide con el de la revisión actual.

Nuestra segunda revisión en «master»



Creamos la segunda revisión con el mensaje “segundo commit”. Podemos arreglar la última revisión (siempre que no la hayamos subido) o ver qué cambios hemos introducido.

Nuestra segunda revisión en «master»



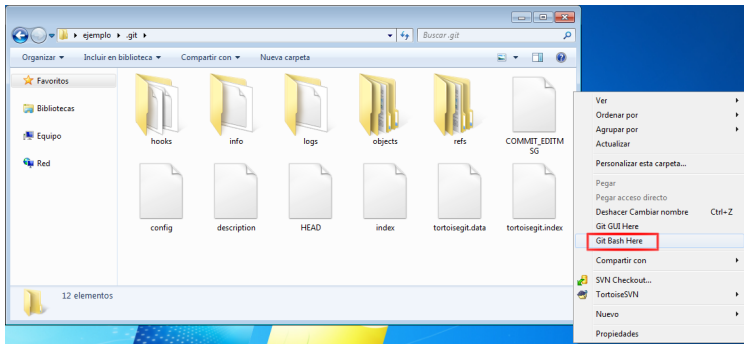
Se cambia el emblema de *f.txt*, indicando que su contenido sí coincide con el de la revisión actual.

Contenidos

- 1 Introducción
- 2 Trabajo local
 - Primeras revisiones
 - **Conceptos**
 - Operaciones comunes
- 3 Trabajo distribuido

Estructura física de un repositorio Git

- Directorio de trabajo: *ejemplo*
- Grafo de objetos: directorio oculto *ejemplo/.git*
- Área de preparación: *ejemplo/.git/index*



Modelo de datos de Git

Características

- Un repositorio es un grafo orientado acíclico de objetos
- Hay 4 tipos de objetos: *commit*, *tree*, *blob* y *tag*
- Los objetos son direccionables por contenido (resumen SHA1)

Consecuencias del diseño

- Los objetos son inmutables: al modificarse, cambia su SHA1
- Git *no* almacena información de ficheros movidos/copiados: los detecta automáticamente, así que no hay que mover/copiar de forma especial
- Git *nunca* guarda más de un objeto una vez en el DAG, aunque aparezca en muchos sitios

Modelo de datos de Git: revisiones (*commits*)

Contenido

- Fecha, hora, autoría, fuente y un mensaje
- Referencia a revisión padre y a un *tree*

```
$ git cat-file -p HEAD
tree 65de8c1fce51aedbc5b0c838d5d2be0883b3ab0e
parent 720ddc4362d8ebba86e8e6fccd409206fe50b2a7
author Antonio <a@b.com> 1413416387 +0200
committer Antonio <a@b.com> 1413416387 +0200
```

segundo commit

Modelo de datos de Git: árboles (*trees*)

Contenido

- Lista de *blobs* y *trees*
- Separa el nombre de un fichero/directorio de su contenido
- Sólo gestiona los bits de ejecución de los ficheros
- No se guardan directorios vacíos

```
$ git cat-file -p HEAD:  
100644 blob 9114647dde3052c36811e94668f951f623d8005d f.txt
```

Modelo de datos de Git: ficheros (*blobs*)

Contenido

Secuencias de bytes sin ningún significado particular.

```
$ git cat-file -p HEAD:f.txt  
hola  
adios
```

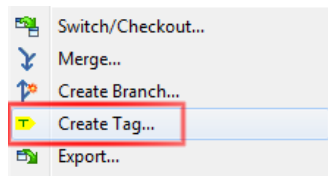
Modelo de datos de Git: etiquetas (*tags*)

Contenido

- Referencias simbólicas inmutables a otros objetos
- Normalmente apuntan a *commits*
- Pueden firmarse mediante GnuPG, protegiendo la integridad de todo el historial hasta entonces

```
$ git cat-file -p v1.0
object 54a05390adc0d59cea7bd4131f59be4648e91127
type commit
tag v1.0
tagger Antonio <a@b.com> 1413371045 +0200

version 1.0
```



Algunos de los ficheros en *.git*

config

Contiene la configuración local.

```
$ cat config
[core]
repositoryformatversion = 0
filemode = true
bare = false
logallrefupdates = true
```

Algunos de los ficheros en *.git*

HEAD

Referencia simbólica a la revisión sobre la que estamos trabajando.

```
$ cat HEAD  
ref: refs/heads/master
```

Algunos de los ficheros en *.git*

hooks

Manejadores de eventos. Ahora sólo tenemos ejemplos.

```
$ ls hooks | head -5  
applypatch-msg.sample  
commit-msg.sample  
post-update.sample  
pre-applypatch.sample  
pre-commit.sample  
prepare-commit-msg.sample  
pre-push.sample  
pre-rebase.sample  
update.sample
```


Algunos de los ficheros en *.git*

index

Contiene el área de preparación (la veremos después).

```
$ git ls-files -s  
100644 9114647dde3052c36811e94668f951f623d8005d 0 f.txt
```

Algunos de los ficheros en *.git*

info/exclude (también *.gitignore* y/o global)

Patrones de ficheros a ignorar.

```
$ cat info/exclude
# git ls-files --others --exclude-from=.git/info/exclude
# Lines that start with '#' are comments.
# For a project mostly in C, the following would be a good set of
# exclude patterns (uncomment them if you want to use them):
# *.lo
# *
```

Algunos de los ficheros en *.git*

logs

Historial de las referencias: medida de seguridad.

```
$ git reflog
9751be8 HEAD@{0}: commit: segundo commit
720ddc4 HEAD@{1}: commit: primer commit
c112dcf HEAD@{2}: commit: tercer commit
68b18b1 HEAD@{3}: commit: segundo commit
f1d301e HEAD@{4}: commit: primer commit
```

Algunos de los ficheros en *.git*

refs

Referencias simbólicas a puntas de cada rama y etiquetas.

```
$ ls -R refs  
ejemplo/.git/refs:  
heads  tags
```

```
ejemplo/.git/refs/heads:  
master
```

```
ejemplo/.git/refs/tags:
```

Algunos de los ficheros en *.git*

objects

Objetos, sueltos (gzip) o empaquetados (delta + gzip).

```
$ ls objects
72 97 c1 info pack
$ ls objects/pack
$ git gc
$ ls objects
info pack
$ ls objects/pack
pack-3ddc9842698016c5588dd536a9355b49f1303373.idx
pack-3ddc9842698016c5588dd536a9355b49f1303373.pack
```

Área de preparación, caché o índice

Concepto

Instantánea que vamos construyendo de la siguiente revisión.

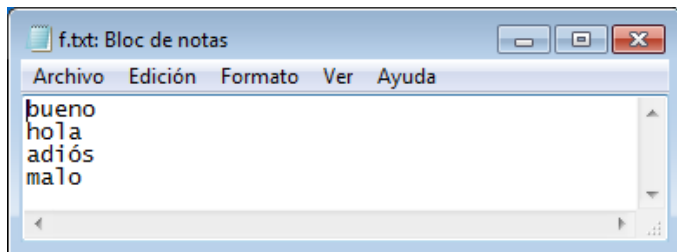
“Add” en herramientas para SVN y Git

- `svn add` = añadir fichero a control de versiones
 - TortoiseSVN y TortoiseGit hacen esto
- `git add` = añadir contenido a área de preparación
 - “Git Bash” y “Git GUI” hacen esto

Pros y contras

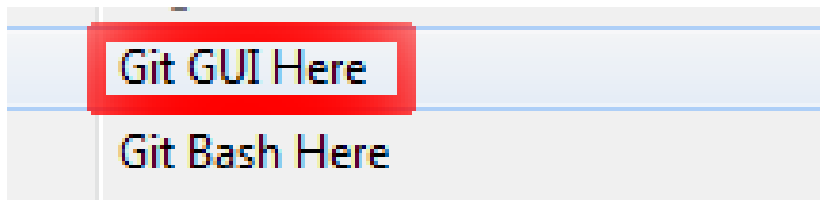
- Controlamos exactamente qué va (y qué **no**) en cada revisión
- Algo raro hasta acostumbrarse

Preparación de una revisión parcial



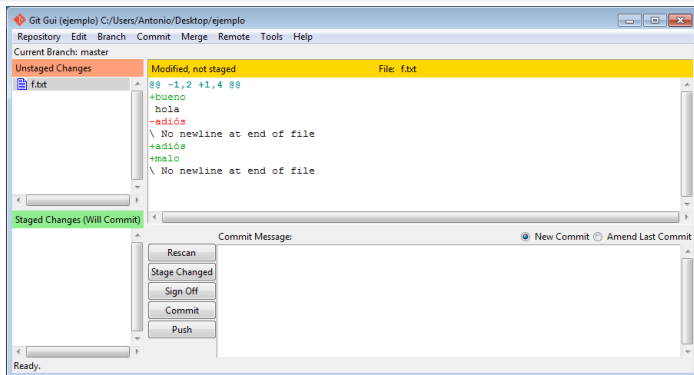
Añadimos una línea al principio y al final de *f.txt*.

Preparación de una revisión parcial



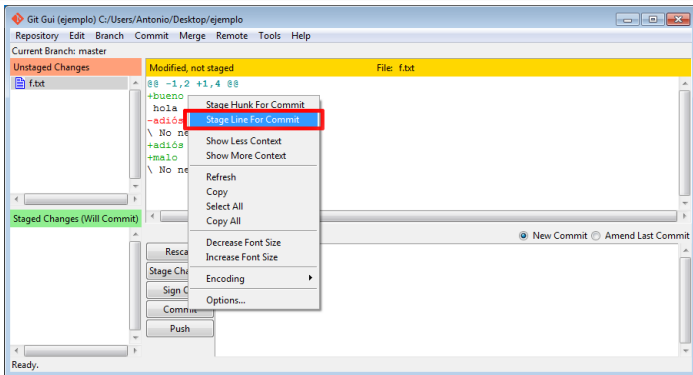
Lanzamos “Git GUI” mediante el menú contextual del directorio actual.

Preparación de una revisión parcial



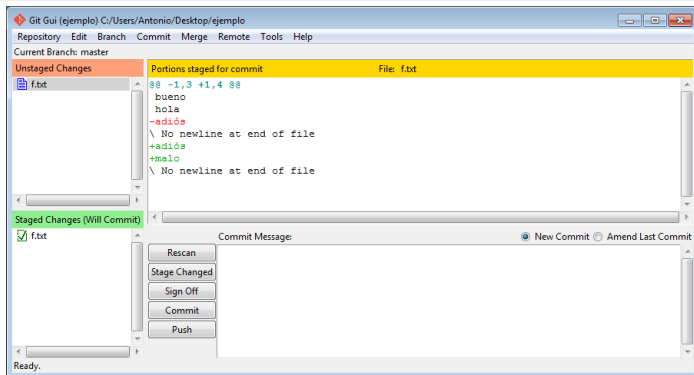
“Git GUI” nos muestra lo que no está aún en el área de preparación.

Preparación de una revisión parcial



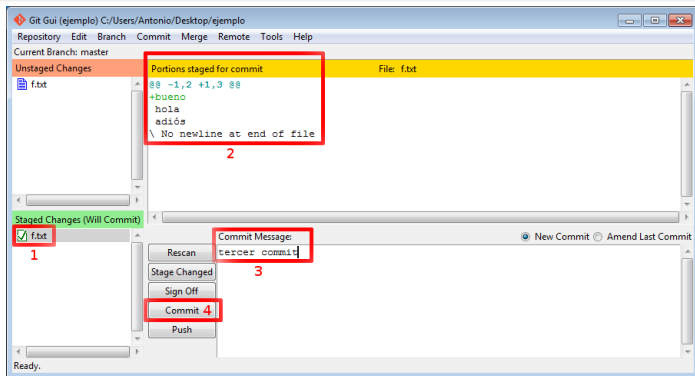
Introducimos la línea añadida al principio de *f.txt* en el área de preparación.

Preparación de una revisión parcial



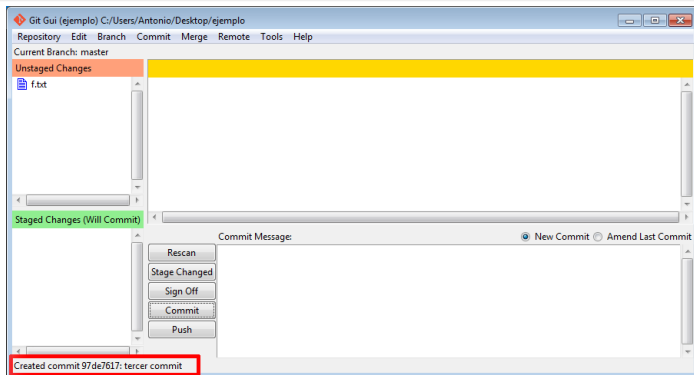
Ahora ya no sale esa línea como pendiente de añadir.

Preparación de una revisión parcial



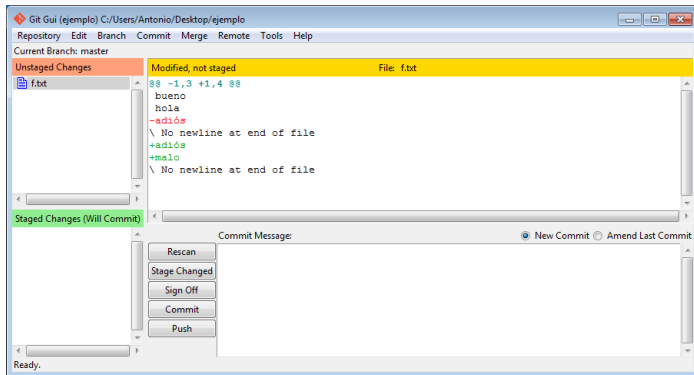
Por el contrario, sale en la sección de “preparado”. Creamos la revisión rellenando el mensaje y pulsando en “Commit”.

Preparación de una revisión parcial



“Git GUI” nos confirma que la nueva revisión ha sido creada con éxito.

Preparación de una revisión parcial

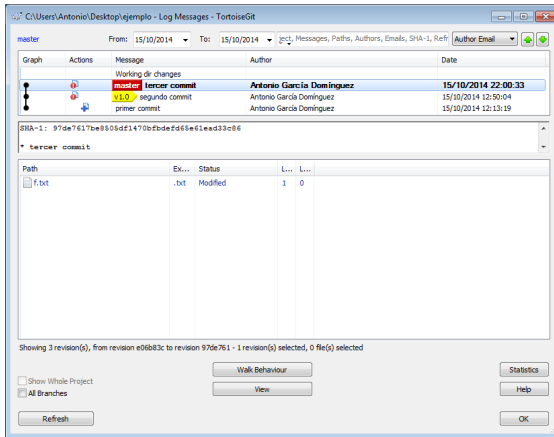


Podemos volver a lo que está pendiente: son las líneas que no se introdujeron en la revisión anterior.

Contenidos

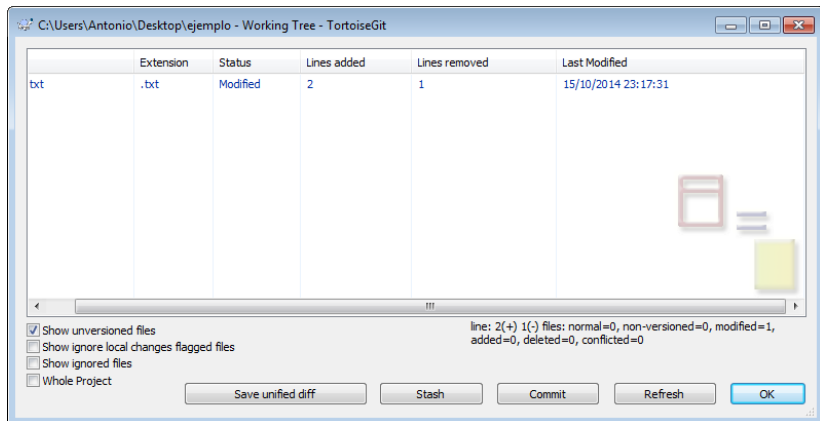
- 1 Introducción
- 2 Trabajo local
 - Primeras revisiones
 - Conceptos
 - Operaciones comunes
- 3 Trabajo distribuido

Historial: “Show Log”



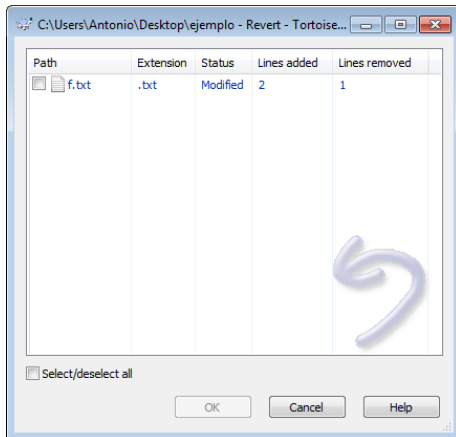
Permite visualizar el histórico, buscar sobre él, obtener diferencias y operar sobre revisiones.

Ver cambios desde la revisión actual: "Diff..."



Permite ver todos los cambios locales respecto a la revisión actual.

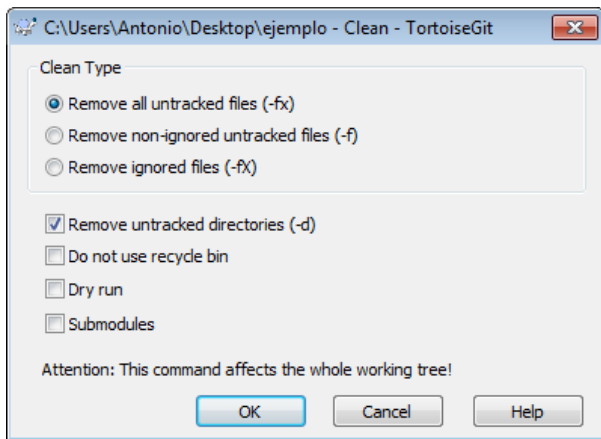
Deshacer cambios: “Revert...”



Permite deshacer algunos o todos los cambios locales respecto a la revisión actual.

Nota: para deshacer revisiones anteriores se usa “Revert changes by

Retirar ficheros inútiles: “Clean Up...”



Permite eliminar automáticamente los ficheros y/o directorios que no están bajo control de versiones.

Contenidos

1 Introducción

2 Trabajo local

3 Trabajo distribuido

- Manejo de ramas
- Interacción con repositorios remotos

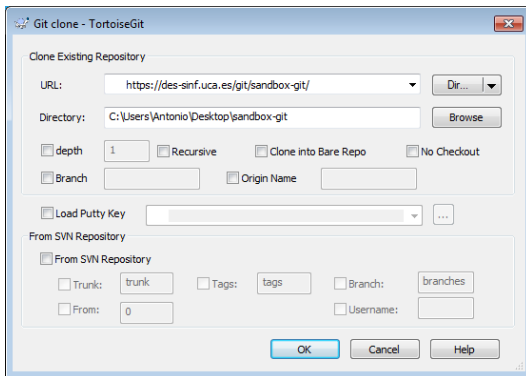
Contenidos

- 1 Introducción
- 2 Trabajo local
- 3 Trabajo distribuido
 - Manejo de ramas
 - Interacción con repositorios remotos

Clonar un repositorio

Métodos de acceso

- Git permite `git://`, SSH, HTTP(S) y *rsync*
- Nosotros: HTTPS con Apache + Redmine.pm + CGI de Git



Ramas en Git

Concepto: líneas de desarrollo

`master` Rama principal, equivalente a `trunk`

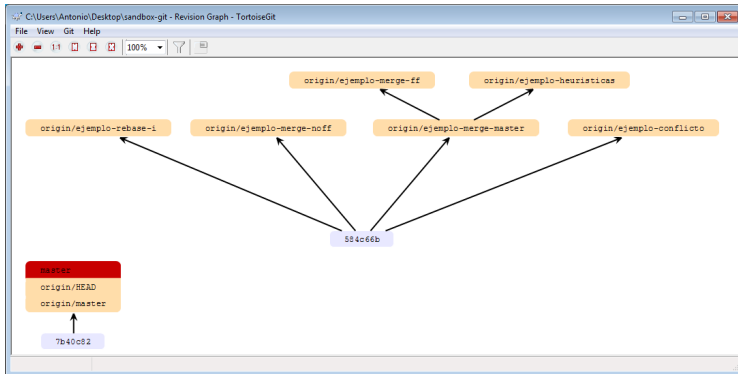
`develop` Rama de desarrollo

`nueva-cosa` Rama para añadir algo concreto («feature branch»)

Diferencias con otros SCV

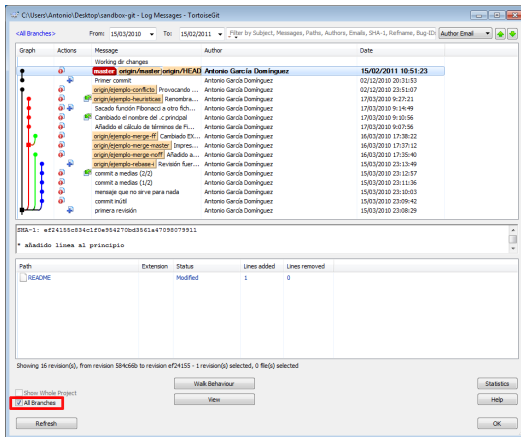
- No son apaños con directorios, sino parte del modelo de datos
- Rama en Git: referencia mutable y compartible a una revisión
- Etiqueta en Git: referencia *inmutable* a un objeto

Ver ramas disponibles: “Revision Graph”



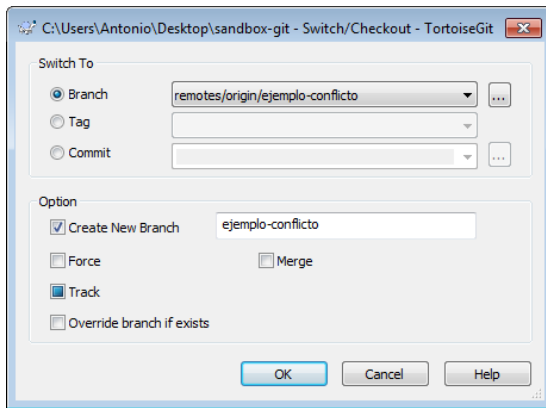
Muy útil para ver qué ramas hay y cómo se relacionan. Las que empiezan por “origin” son *remotas*: representan al repositorio original y son de sólo lectura.

Ver ramas disponibles: “Show Log”



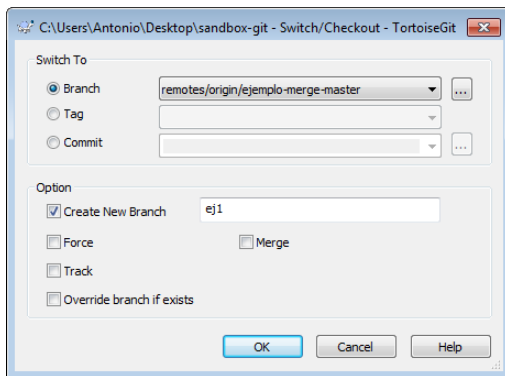
Para ver todas las ramas, basta con marcar la caja “All Branches” en la esquina inferior izquierda.

Cambiando entre ramas: “Switch/Checkout”



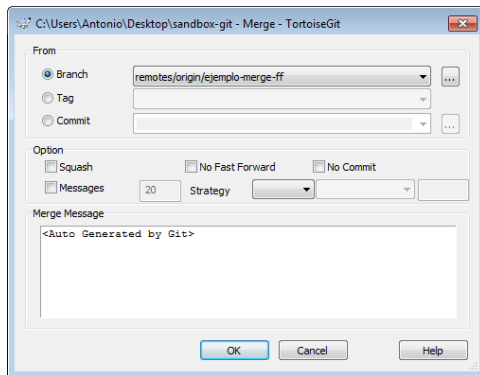
Podemos usar esta orden para cambiar a ramas locales o remotas. Normalmente, al cambiar a una rama remota es buena idea crear una rama local que la “siga”, para poder trabajar nosotros en ella.

Reuniendo ramas: «fast-forward»



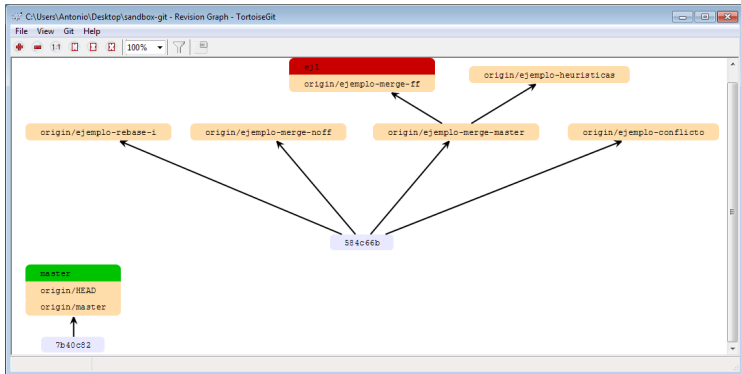
Creamos la rama `ej1` a partir de `origin/ejemplo-merge-master`.

Reuniendo ramas: «fast-forward»



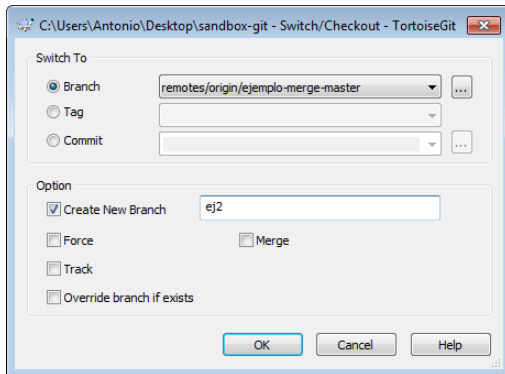
Reunimos la rama actual con `origin/ejemplo-merge-ff`.

Reuniendo ramas: «fast-forward»



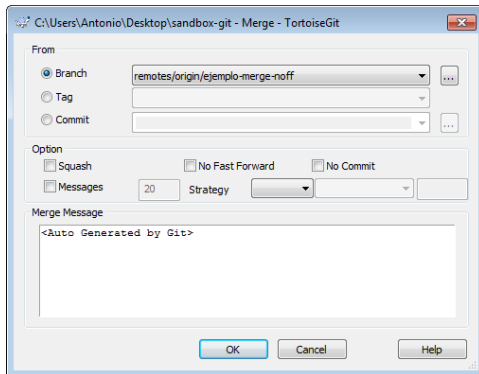
No se ha creado ninguna revisión nueva: sólo hemos “adelantado” la punta de la rama.

Reuniendo ramas: normal



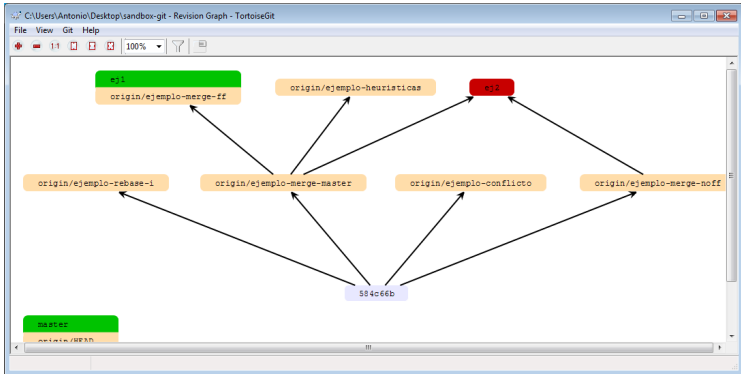
Creamos la rama `ej2` a partir de `origin/ejemplo-merge-master`.

Reuniendo ramas: normal



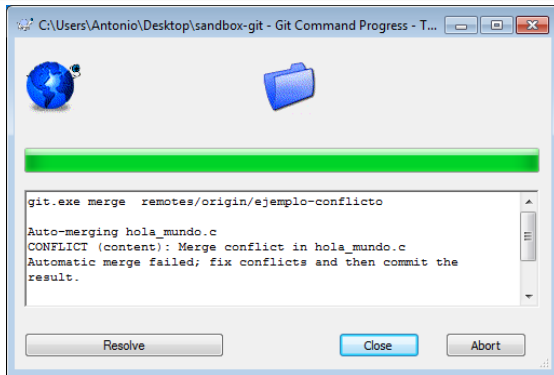
Reunimos la rama actual con `origin/ejemplo-merge-noff`.

Reuniendo ramas: normal



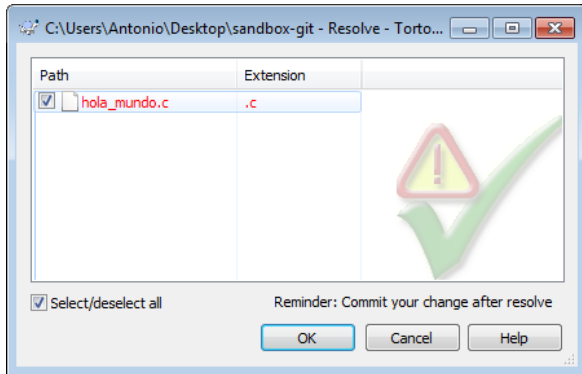
Se ha creado ninguna revisión nueva con dos revisiones padre.

Reuniendo ramas: conflictos



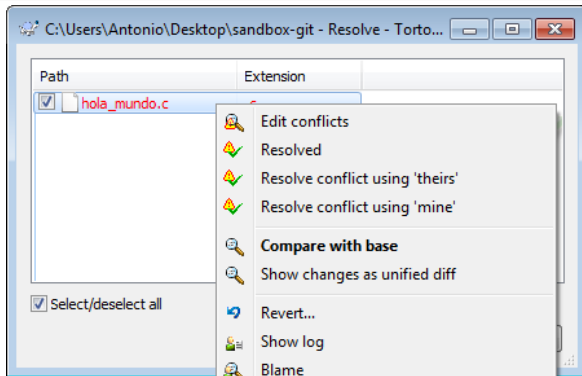
Tratamos de reunir `ej3` (también sacada de `origin/ejemplo-merge-master`) con `origin/ejemplo-conflicto`, y nos da este error.

Reuniendo ramas: conflictos



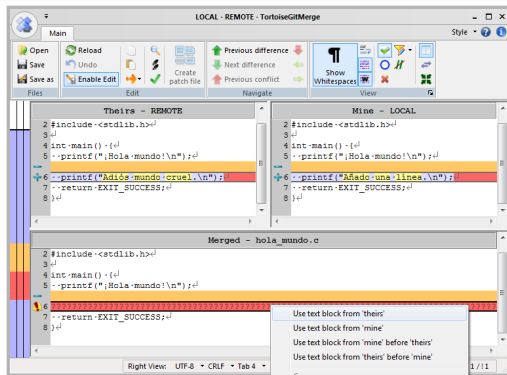
Usamos “Resolve” para que nos indique los conflictos a resolver.

Reuniendo ramas: conflictos



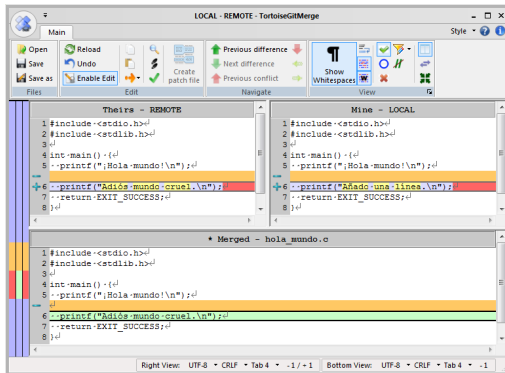
El menú contextual sobre un fichero nos da las opciones disponibles: seleccionamos “Edit conflicts”.

Reuniendo ramas: conflictos



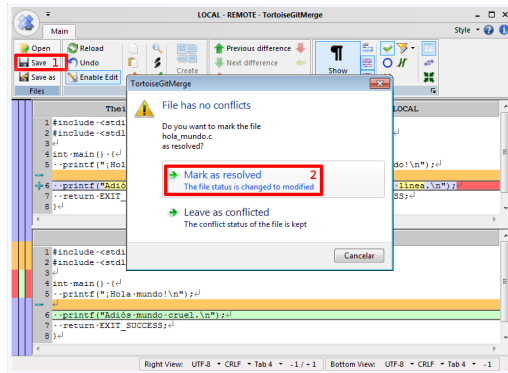
El editor integrado nos muestra nuestra versión (“Mine”) y la última en el repositorio (“Theirs”), con los conflictos en rojo. El menú contextual del conflicto nos deja decidir: elegimos usar “Theirs”.

Reuniendo ramas: conflictos



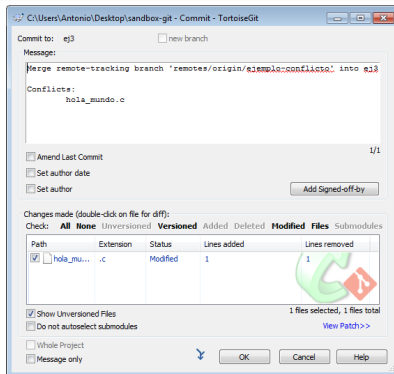
Así es como queda el fichero.

Reuniendo ramas: conflictos



Pulsamos “Save” y se nos avisa de que no quedan más conflictos: marcamos el fichero como resuelto.

Reuniendo ramas: conflictos

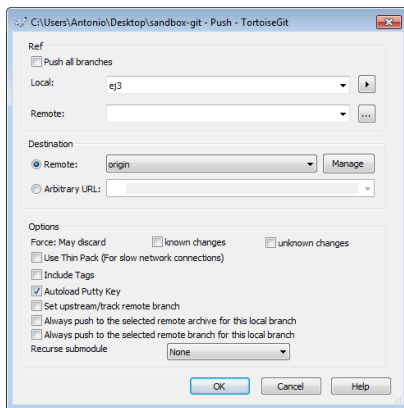


Ya podemos crear la nueva revisión. Es buena idea conservar el mensaje tal y como está, incluyendo el hecho de que hubo que resolver un conflicto.

Contenidos

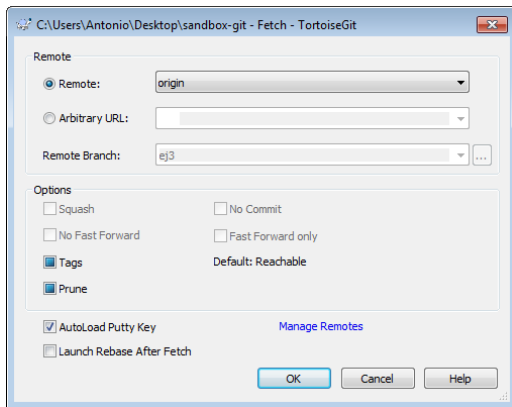
- 1 Introducción
- 2 Trabajo local
- 3 Trabajo distribuido**
 - Manejo de ramas
 - Interacción con repositorios remotos**

Envío de objetos: “Push”



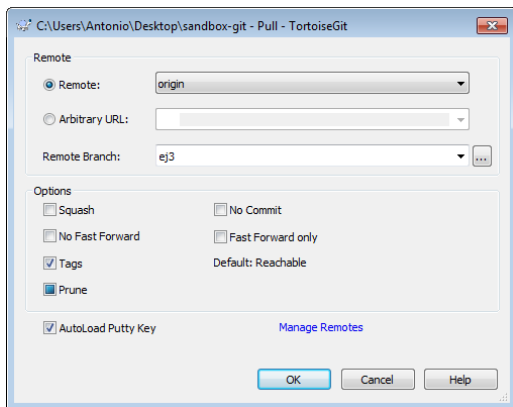
Actualizamos una rama en un repositorio remoto (“remote”) a partir de una rama local. Puede subir todas las ramas de una vez si es necesario, o subir etiquetas (normalmente no se hace).

Recepción de objetos: “Fetch”



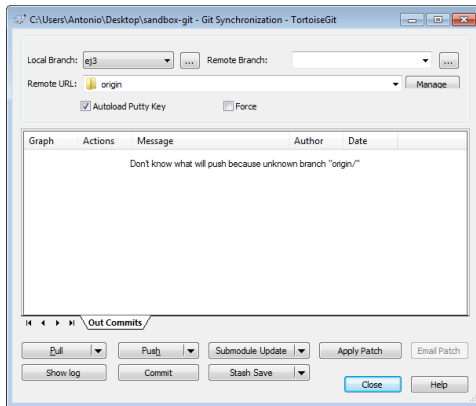
Actualizamos las ramas remotas (y posiblemente las etiquetas) a partir de los contenidos de un repositorio remoto.

Recepción y reunión: “Pull”



Atajo de conveniencia: hace un “Fetch” y luego un “Merge”.

Sincronización: “Sync”



Específico de TortoiseGit: permite comparar una rama local con una remota y ver qué podemos mandar (“Out Commits”) o recibir (“In Commits”). Tiene atajos para lanzar los verbos anteriores.

Flujo de trabajo centralizado

Secuencia típica: muy similar a SVN

- Creamos un clon del repositorio *dorado*
- Nos actualizamos con `git pull`
- Si hay conflictos, los resolvemos
- Hacemos nuestros cambios sin preocuparnos mucho de Git
- Los convertimos en revisiones cohesivas y pequeñas
- Los enviamos con `git push`

Flujos de trabajo distribuidos: variantes

Un integrador

- Cada desarrollador tiene rep. privado y rep. público
- Los desarrolladores colaboran entre sí
- El integrador accede a sus rep. públicos y actualiza el repositorio oficial
- Del repositorio oficial salen los binarios
- Los desarrolladores se actualizan periódicamente al oficial

Director y tenientes

- El integrador (dictador) es un cuello de botella
- Se ponen intermediarios dedicados a un subsistema (teniente)

¡Gracias por su atención!

antonio.garciadominguez@uca.es

@antoniogado_es