**Tutorial Git en Wollok**

**Versión 2.0**

**Agosto 2017**

**por Tobías Sandler,**

**Alfredo Sanzo**

**Versión 2.0**

**Agosto 2017**

**Índice**

[¿Qué es Git?](#_f5sbpnai9jkm)

[Instalar Plugin de Git en Wollok](#_at6w94vza7w)

[Clonar un proyecto usando Eclipse](#_wgc7jxuxttsk)

[¿Dónde está lo que quiero clonar?](#_bwh134iujt1n)

[Ahora sí, a clonar](#_h03t90995f0s)

[Hacer pull usando Eclipse](#_7ltbmslolgex)

[Hacer commit y push usando Eclipse](#_ygp8o34pbymc)

[¿Cada cuanto es conveniente hacer un commit?](#_m4a02195b8ll)

[Subiendo archivos al repositorio remoto](#_zdnky16vb16n)

[Deshacer mis cambios para hacer pull](#_grups7xhj4a1)

[Flujo de trabajo simple](#_hhfsefmkop0b)

[Resolución de conflictos usando Eclipse](#_lbru0k97q3lu)

[Flujo de trabajo completo](#_vq6kwxkgv5en)

[Etiquetar un commit (tag)](#_dlx1xyef3dwx)

[Significado de los símbolos](#_nm7jap5doa7h)

[Extra 1: Terminología completa de Git](#_urfyionqs8ns)

[Extra 2: Equivalencias con la línea de comandos](#_w42mfx5g59ho)

[Extra 3: ¿Cómo subo a Github cualquier proyecto mío?](#_8dl70cecms37)

# ¿Qué es Git?

Git es un *software de control de versiones* que facilita mucho el trabajo de programar en un equipo de personas. En particular resuelve varios problemas: los que tienen que ver con **compartir el código** para que varios desarrollen, y permitir tener **control sobre los cambios**. ¡Es como tener un Ctrl+Z con esteroides!

Entonces lo que hace git es *agregar dos lugares más donde va a estar tu código*, además del típico lugar en disco: esos lugares se llaman **repositorios**. Git básicamente trabaja con dos repositorios:

* **Repo Local**: almacena las modificaciones que fui haciendo dentro de mi máquina.
* **Repo Remoto**: almacena una versión estable del sistema que se está desarrollando. ¡Es donde va a estar el código accesible para todos!



*Estructura del sistema de versionado “Git”*

Una vez que los cambios realizados de forma local están funcionando correctamente, entonces se los sube al servidor remoto para que puedan ser compartidos, permitiendo que otras personas dentro del mismo grupo de desarrollo puedan acceder a esas modificaciones.

Cada usuario va trabajando dentro de su repositorio local (modificando, agregando o eliminando archivos), y una vez que se llega a un punto estable (no rompí nada de lo que había y lo que agregué funciona bien), entonces es un buen momento para subirlo al repositorio remoto. Para esto hay que seguir los siguientes pasos:

* Elegir qué cambios de los que hice a mis archivos quiero guardar en el repo local.
* Hacer un commit: esto guarda los cambios en el repo local.
* Hacer un push: esto pasa lo que hay guardado en el repo local al repo remoto. Este es el único paso que requiere conexión a internet ya que envía información, mientras que los 2 primeros pasos son locales.

Para usar git existen varios softwares, tanto de interfaz gráfica como por línea de comando. A continuación vamos a ver cómo compartir un proyecto simple en github. Para esto necesitan tener una cuenta creada en <https://github.com/> y pasarle el usuario al docente a cargo del curso para que les pueda dar permisos para acceder al repositorio.

# Instalar Plugin de Git en Wollok

A partir de la versión 1.6 Wollok trae el plugin eGit instalado por defecto, así que no es necesario ningún paso de instalación. Ahora vamos a traernos el proyecto remoto con el que queramos trabajar.

# Clonar un proyecto usando Eclipse



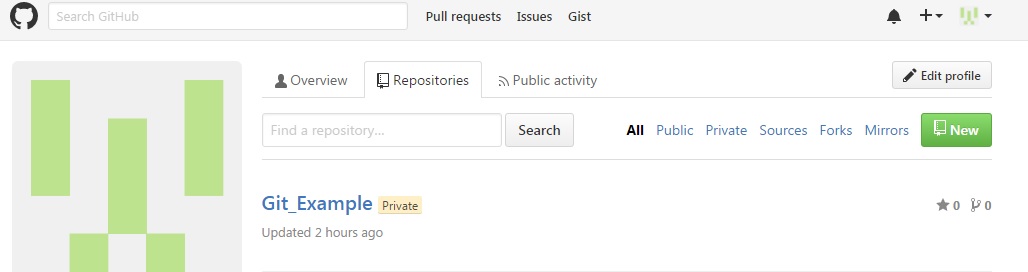
*Clonarse un proyecto*

## 

## **¿Dónde está lo que quiero clonar?**

Para poder clonar un repositorio primero hay que obtener el link asociado al mismo, para esto hay que hacer lo siguiente:

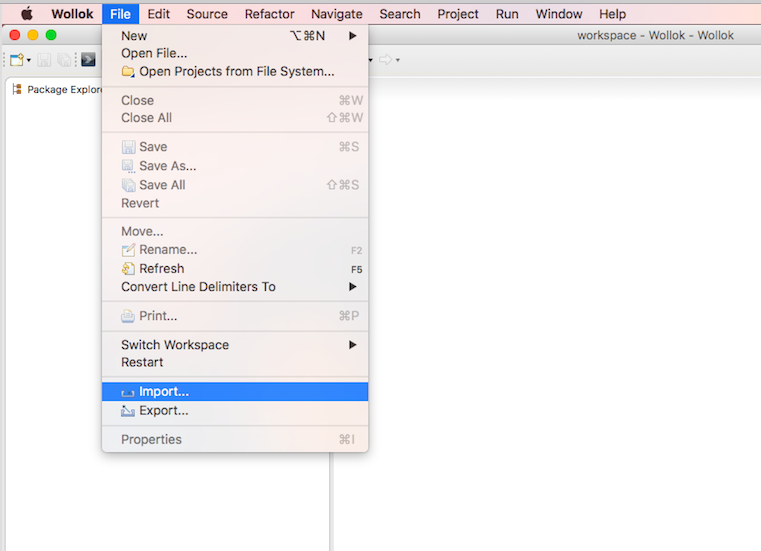
* Entrar a la página de github, ir al perfil de usuario y seleccionar el repositorio que se desee clonar.



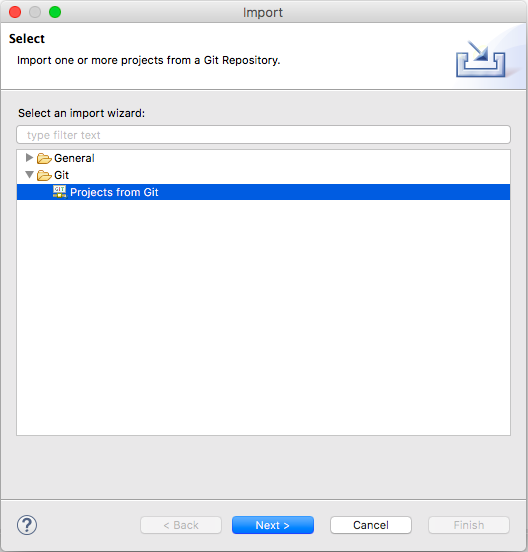
* Una vez dentro, seleccionamos “Clone or Download” y copiamos la ruta que nos aparece.

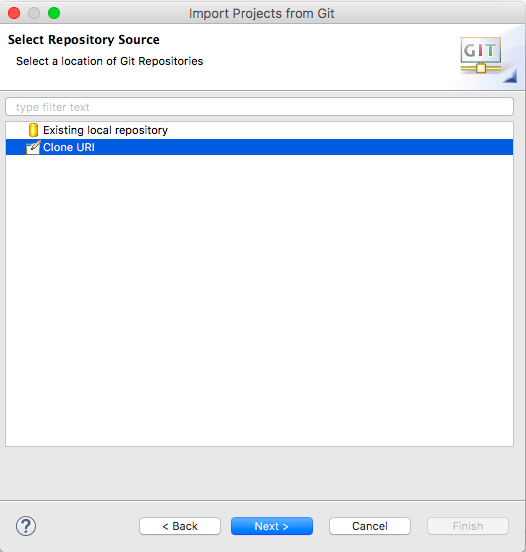
## **Ahora sí, a clonar**

* Para importarlo directo a nuestro workspace necesitamos importarlo directamente desde git. Para eso ir a File -> Import...

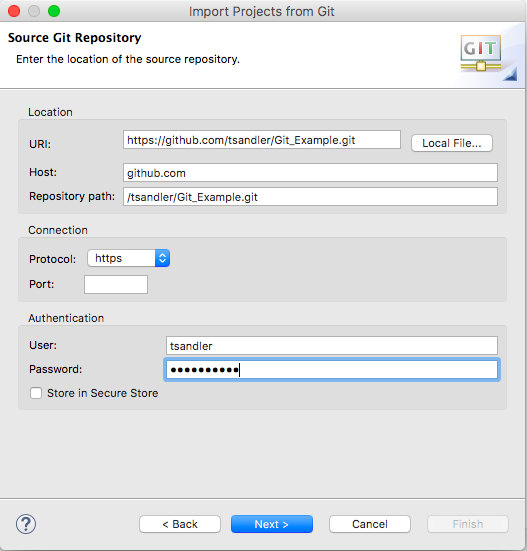


* Luego, seleccionar para importar el proyecto desde git, darle "Next" y elegir la opción "Clone URI"

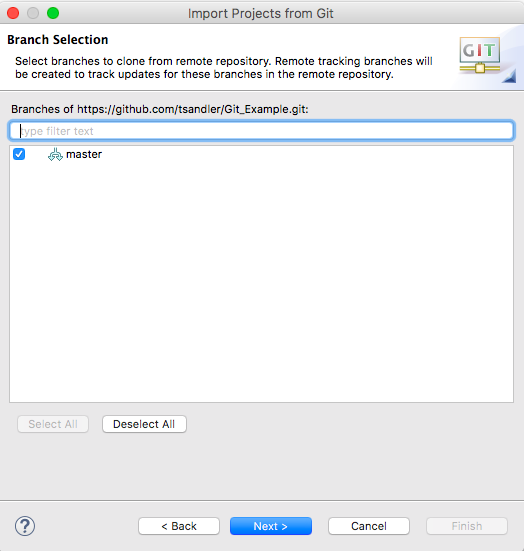




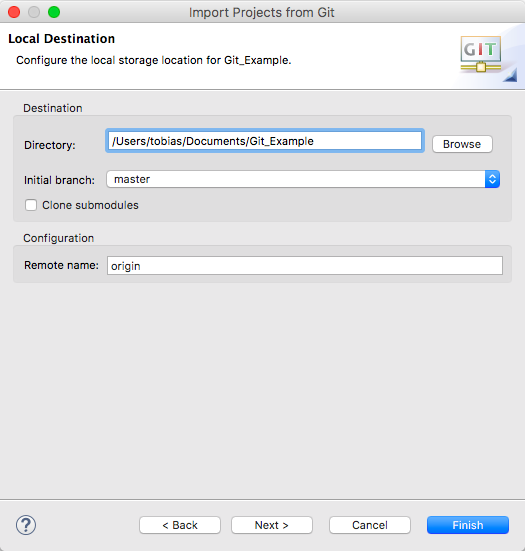
* Ahora, ingresar la dirección que se desea clonar en el campo “URI:” (los campos Host y Repository path son completados automáticamente), el usuario y la password en los campos correspondientes, y presionar “Next”.



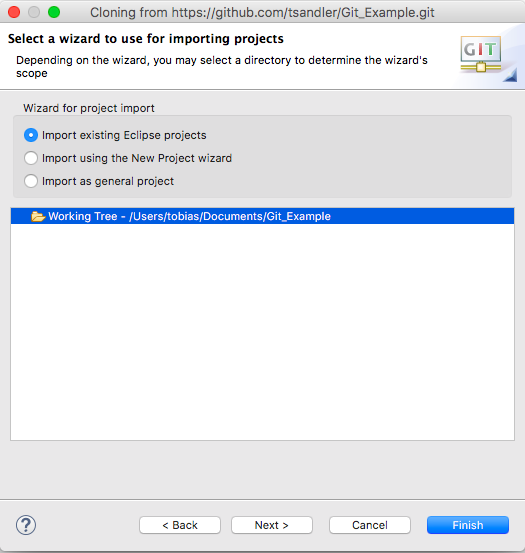
* Luego, nos aparece “Branch Selection”. En otro momento aprenderemos lo que son los branches de Git. Por ahora, con el default (master) estamos. Le damos "Next" nuevamente.



* Luego nos preguntará donde queremos clonar el proyecto. Seleccionan la ruta y le dan "Next".



* En el siguiente paso, primero tratará de autenticarlos con el repositorio y, en caso de haber ingresado bien los datos, descargará los archivos remotos. Ahora hay que seleccionar como queremos importar el proyecto a Eclipse, para eso en la siguiente ventana seleccionar "Import existing Eclipse proyect" y darle a "Finish".



Una vez hecho esto ya tenemos clonado el repositorio e importado a nuestro workspace.



Hasta este momento del documento, se muestran cosas que **sólo deben hacer la primera vez**. Una vez que tienen todo esto hecho, no hace falta repetir los pasos.

¡Ya tienen su propia copia del proyecto en su máquina!

A partir de ahora, todo lo que hay que hacer es **commit, pull** y **push**

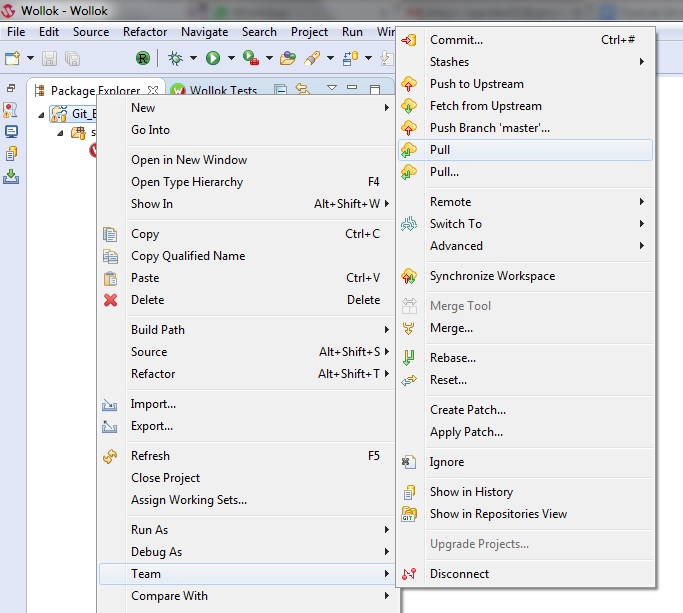
# Hacer pull usando Eclipse



*Qué es lo que hace un pull*

Hacer pull implica descargar los cambios que otro haya realizado en el repositorio remoto y traerlos a mi repositorio local para poder trabajar sobre ellos.

* Para esto hacemos click derecho en el proyecto -> Team -> Pull (seleccionar el primer pull que aparezca porque realiza la descarga por defecto, con el segundo hay que configurar algunas cosas que no nos interesan ahora).



* Ingresar usuario y contraseña y listo, ya van a tener los cambios realizados en el repositorio remoto.

# Hacer commit y push usando Eclipse



*Qué significa hacer commit*



*Qué significa hacer push*

Hacer push implica subir los cambios realizados localmente al servidor remoto.

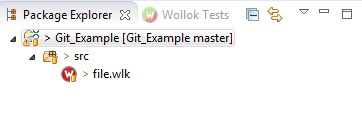
## **¿Cada cuánto es conveniente hacer un commit?**

Esto depende del gusto de cada uno. Lo ideal es que los commits sean chicos y frecuentes, bien enfocados en una funcionalidad que se está desarrollando.

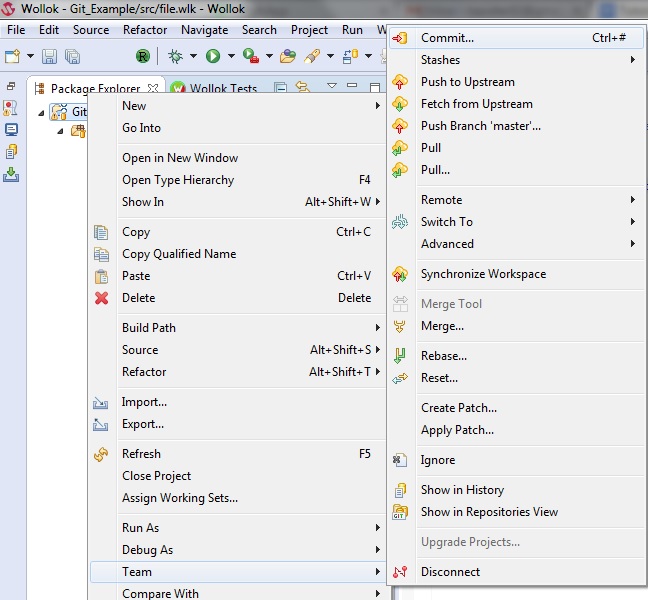
No es conveniente realizar commits constantemente, por cada cambio insignificante, pero tampoco es bueno hacer uno cada 3 meses, esto produciría que posiblemente los archivos que se estuvieron modificando hayan sido editados por otra persona del grupo y se complicaría en la hora de subir los archivos. Este tipo de problemas git trata de arreglarlos haciendo un auto-merge, que en el mejor de los casos lo hace bien, pero en caso de que las líneas modificadas sean similares git no puede resolver el conflicto y hay que arreglarlo a mano.

## **Subiendo archivos al repositorio remoto**

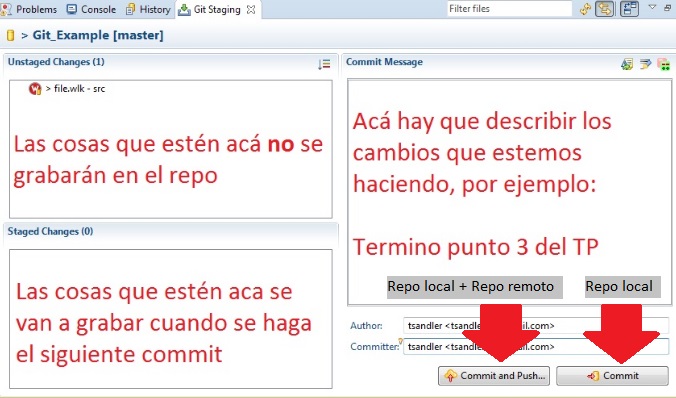
* Inicialmente van a ver que una vez que hayan modificado archivos, los mismos van a aparecer con un “>” adelante del nombre, que indica que no han sido subidos aún.



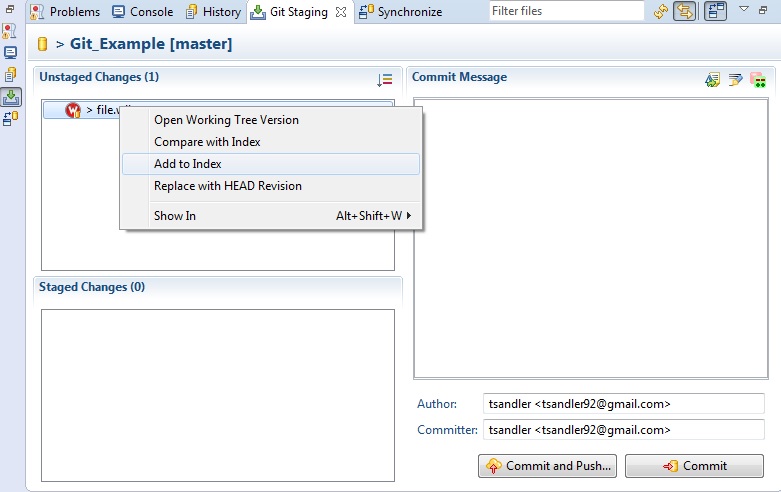
* Para subirlos hacemos click derecho en el proyecto -> Team -> Commit



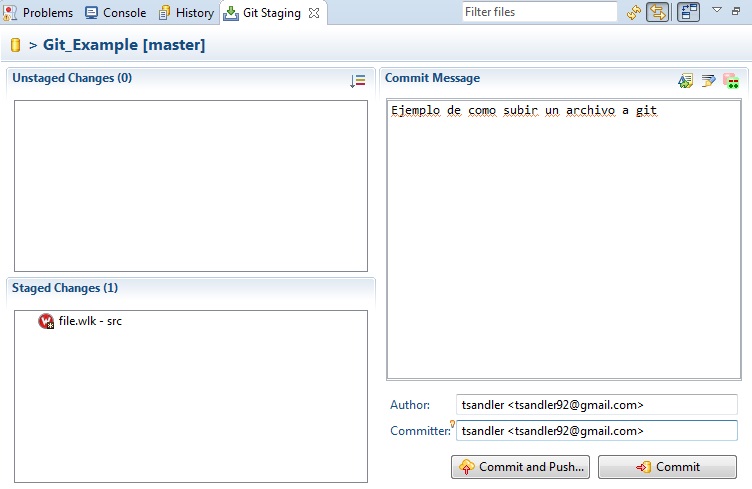
* Una vez hecho esto, verán que aparece una pantalla como la siguiente.



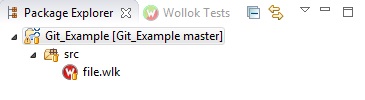
* **Unstaged Changes**: acá se van a listar los archivos que hayan modificados y no hayan sido subidos aún, los archivos presentes acá **no** van a ser enviados en el próximo commit.
* **Staged Changes**: acá se van a listar los cambios que van a ser guardados en el repo, si este listado está vacío entonces no podremos realizar ni commit ni push ya que no hay cambios para guardar.
* **Commit Message**: es un mensaje con el que se van a guardar los cambios en el repo. El mensaje es obligatorio y debería ser lo suficientemente expresivo como para que se entienda qué cambio se está realizando.
* **Commit**: guarda los cambios solo en el repositorio local para ser subidos más adelante. Puede que los cambios no puedan ser enviados al repositorio remoto en el momento del push por diversas razones, como por ejemplo no tener conexión a internet.
* **Commit and Push**: guarda los cambios en el repositorio local y en el repositorio remoto
* **Author y committer**: acá pongan su nombre de usuario (o nombre completo) y mail. Estos son los datos que muestra git por cada commit, sino no puede identificar quién lo hizo.
* Para agregar un archivo a la zona de Staged Changes solo hay que hacerle click derecho -> Add to Index



* Verán que el archivo aparece en el listado de Staged Changes, ahora solo se ingresa un mensaje para el commit, se selecciona “Commit and Push…” ó “Commit”, ingresan usuario y contraseña y listo.

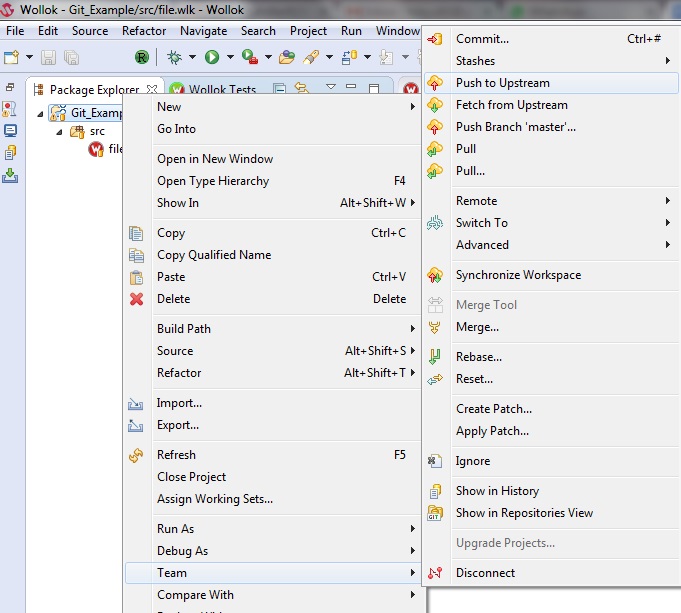


* Una vez hecho esto podemos verificar que los cambios se guardaron en el repo ya que el “>” que aparecía delante del nombre de los archivos debió desaparecer.



* En caso de haber seleccionado “Commit and Push” los cambios ya fueron enviados al repositorio remoto, ahora si solo seleccionaron “Commit”, todavía falta enviarlos. Pueden hacer la cantidad de commits que prefieran y enviar todos juntos con un solo push, no hay un máximo posible (no es recomendable hacer demasiados porque puede pasar que en el medio pude subir algo y por no hacerlo otro tocó mi mismo archivo, generando un conflicto que voy a tener que resolver).

Para subir lo que está en el repositorio local al remoto hay que darle click derecho al proyecto -> Team -> Push to Upstream, ingresan usuario y contraseña y listo.



# Deshacer mis cambios para hacer pull

¿Qué pasa si toco uno o más archivos produciendo cambios que posteriormente no me interesa conservar?

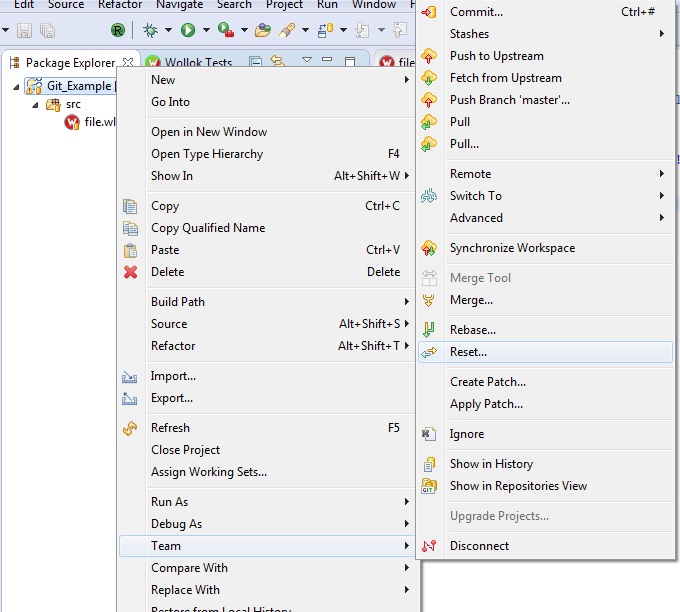
Este es un problema bastante común, porque puede pasar que yo realice cambios que posteriormente no funcionan o simplemente no quiero mantener. Con decir “esto no me interesa, hago pull y fue” no alcanza, porque git detecta que hubo cambios en tu workspace, por lo que no va a poder descargar los cambios del repo normalmente. La solución **no** es borrar todo y volver a clonar, eso implicaría volver a preparar todo el entorno de trabajo. ¿Tiene sentido hacer esto? Claramente no.

Para solucionar este problema hay dos alternativas:

* Hacer un stash: esto sería prácticamente como hacer un Ctrl + X de mis cambios, que puedo posteriormente volverlos a agregar en el código actualizado (entre otras cosas).
* Hacer un checkout del estado del repo en el último commit que tengo: esto implica **borrar** los cambios hechos localmente y **no**  hay vuelta atrás, llevándome al último commit hecho, permitiéndome así bajar los nuevos cambios en caso de que haya.

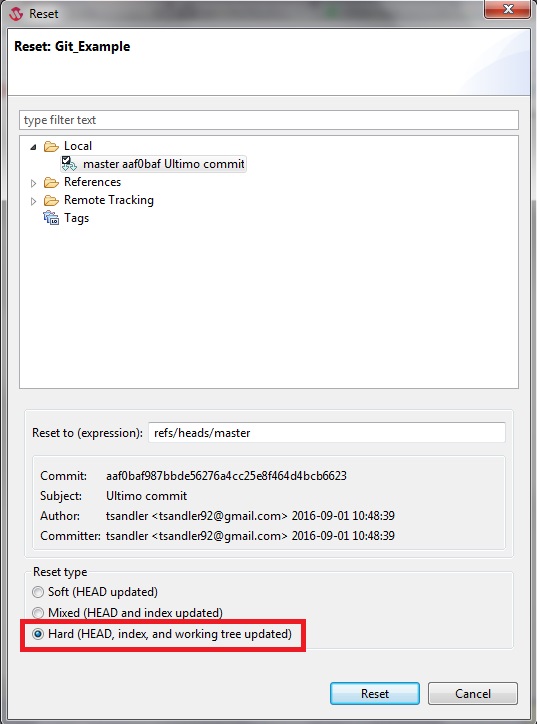
Por el objetivo que tiene este apunte, vamos a ir por la segunda opción ya que la primera tiene más complejidad:

* Primero hay que darle click derecho al proyecto -> Team -> Reset…



# 

* En la ventana que aparece, seleccionar “Hard (HEAD, index and working tree updated)”, darle a “Reset” y aceptar la ventana emergente que sigue.



Como se puede ver, nos indica a qué commit estaríamos retrocediendo, mostrando el código del commit, el mensaje con el que se hizo y quién lo subió.

# Flujo de trabajo simple

Como resumen, te recordamos que se recomienda siempre el siguiente flujo de trabajo:

1. Bajo cambios (git **pull**)
2. Corro **tests**, si están en rojo hay que avisar al equipo que hay un commit que hizo fallar. Medialunas para el equipo a cargo del que subió el commit.
3. ¡A **trabajar**! Hago ajustes necesarios.
4. Cuando terminé mis modificaciones... ¡¡volver a correr los **tests**!!
5. Ahora sí, hago un **push**

Y ya puedo irme a dormir tranquilo.

Resumen del resumen:

Me **bajo** lo último

corro **tests**

hago mis **cambios**

corro **tests**

**subo** mis cambios

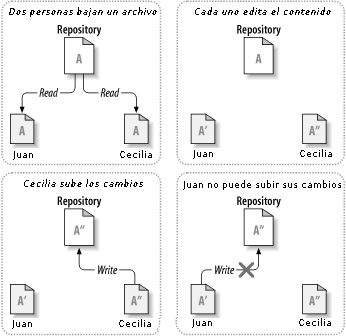
Pueden proceder a tatuarse esto.

Ah, una salvedad: el último paso puede no ser feliz. La siguiente sección aborda la pregunta:

¿Qué pasa si alguien estuvo trabajando **al mismo tiempo** que yo?

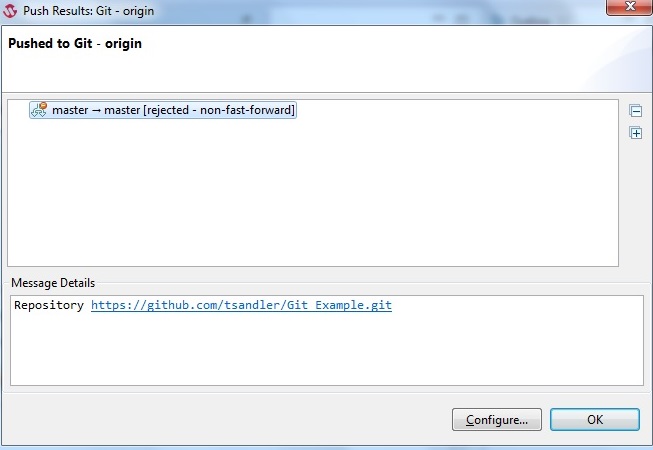
# Resolución de conflictos usando Eclipse

¿Qué ocurre si dos personas tocan simultáneamente el mismo archivo?



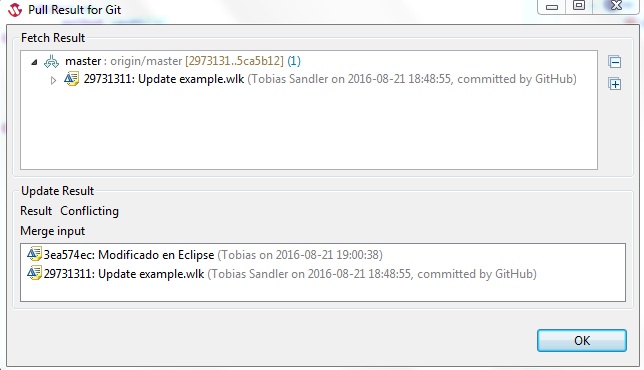
En este ejemplo se puede ver que tanto Juan como Cecilia descargaron el archivo A y estuvieron trabajando sobre él paralelamente. A su vez, Cecilia trabajó más rápido y subió sus cambios al repositorio. Entonces Juan al querer subir su versión se entera de que hay una nueva en git, por lo que no va a poder subir sus cambios con un simple commit.

Al querer subirlo posiblemente aparezca un cartel como el siguiente:



Este cartel nos indica que hubo un problema al querer subir los cambios al repositorio remoto.

Para solucionar este problema lo que debemos hacer es tener la última versión del archivo, para eso primero hay que hacer pull normalmente. Una vez hecho eso, van a ver el siguiente cartel:



Aquí se puede observar que hubo un problema de merge, y lo que git hace automáticamente es “romper” el archivo para que sea arreglado manualmente, quedando así:



Lo que podemos ver a simple vista es que el programa **no compila**, y eso es con el único fin de que podamos hacer un merge manual del archivo.

Lo que viene luego de “<<<<<<< HEAD” es el código que tenemos en nuestro repositorio local, mientras que lo que viene luego de “=======” es lo que está en el repositorio remoto. Ahora debemos unificar los archivos, modificando o borrando lo que sea conveniente, dejando al archivo nuevamente sin errores, sin problemas de compilación y, claramente, con los tests correspondientes funcionando.

También podemos ver que la estructura del proyecto en el Package Explorer cambió, indicando los archivos que tuvieron conflictos, mostrándose así:

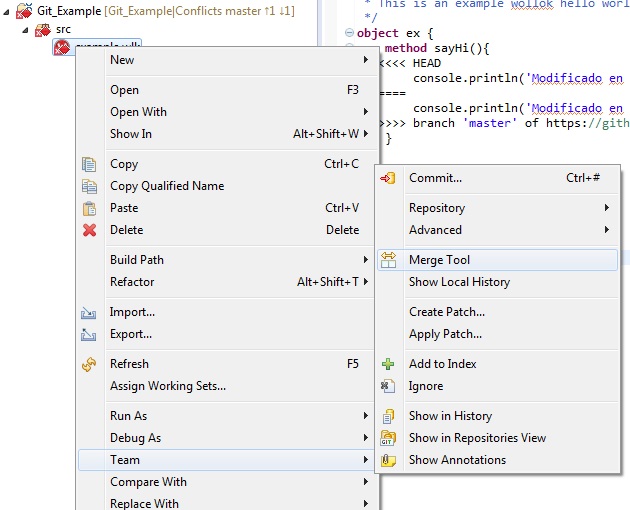


Una vez solucionado este problema se procede a realizar un commit y push normal hacia el repositorio y problema solucionado.

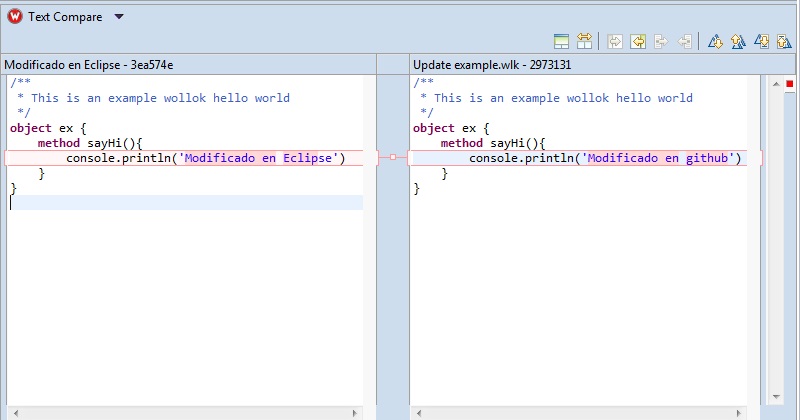
**Opción alternativa**:

Otra alternativa para solucionar este tipo de conflictos es usar una herramienta que provee Eclipse para tal fin.

Para eso hay que hacer click derecho en el archivo -> Team -> Merge Tool.



Luego de hacer eso veremos que se nos abre una ventana como la siguiente:



Acá podemos ver a simple vista los cambios que están en nuestro repositorio local (archivo de la izquierda) y los que están en el repositorio remoto (archivo de la derecha). El objetivo con esta vista es dejar el archivo de la **izquierda** tal y como queremos que sea el archivo final luego del merge.

Una vez hecho esto, le dan click derecho -> Save y les va a guardar los cambios en el archivo correspondiente.

Y ahora que está terminado el merge se puede subir al archivo con un commit y push normal.

# Flujo de trabajo completo

Un flujo de trabajo completo sería:

1. Hacer pull.
2. Correr tests.
3. Hacer mis cambios.
4. Correr tests.
5. Hacer un **push**,
   1. Si está todo bien ¡Terminamos!.
   2. Si no me deja pushear, entonces hacer lo siguiente:

* Hago un pull para traerme los cambios nuevos.
* Corro tests
  + Si está todo bien entonces no hubo conflictos.
  + Ahora, si hay conflictos, ups, hay que resolverlos a mano y correr los tests nuevamente
* Commitear y pushear una vez que los marcaron como resueltos

# Etiquetar un commit (tag)

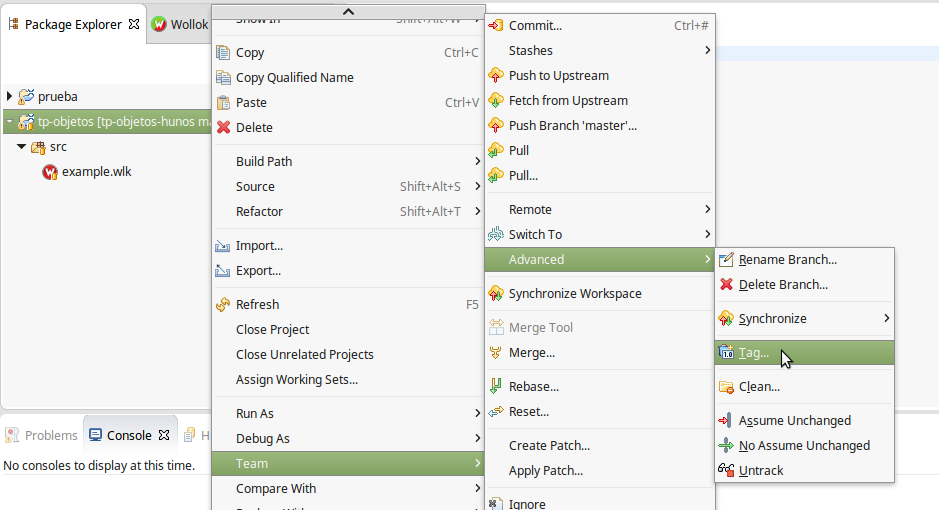
Cuando realizamos un commit, que es el último de una versión entregable de nuestro programa, nos interesa recordarlo. Pero guardar el nombre del commit es: poco expresivo, complejo -debería llevar en algún otro lugar una tabla que diga *entrega -> commit-*.

Para esto GIT nos da la funcionalidad de etiquetar (dar un nombre) a un commit particular.

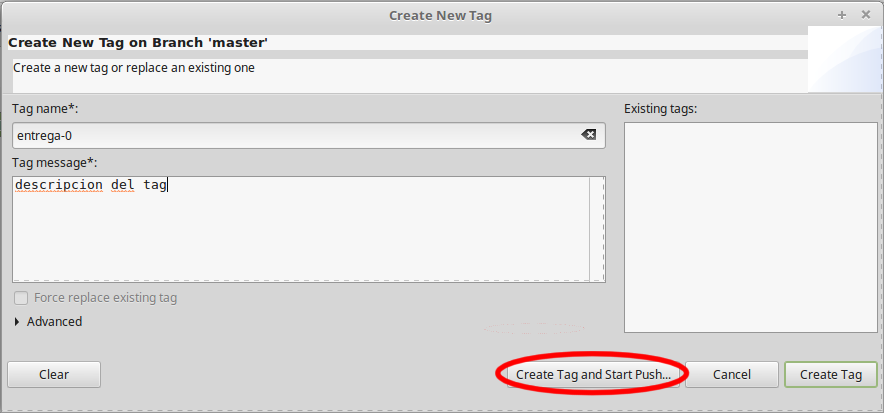
Se realiza de la siguiente manera:

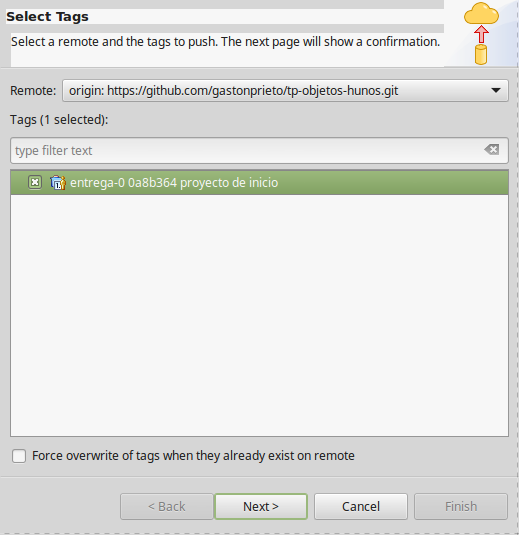
* **Crear el tag:** *Botón derecho sobre el proyecto -> Team -> Advanced -> Tag*

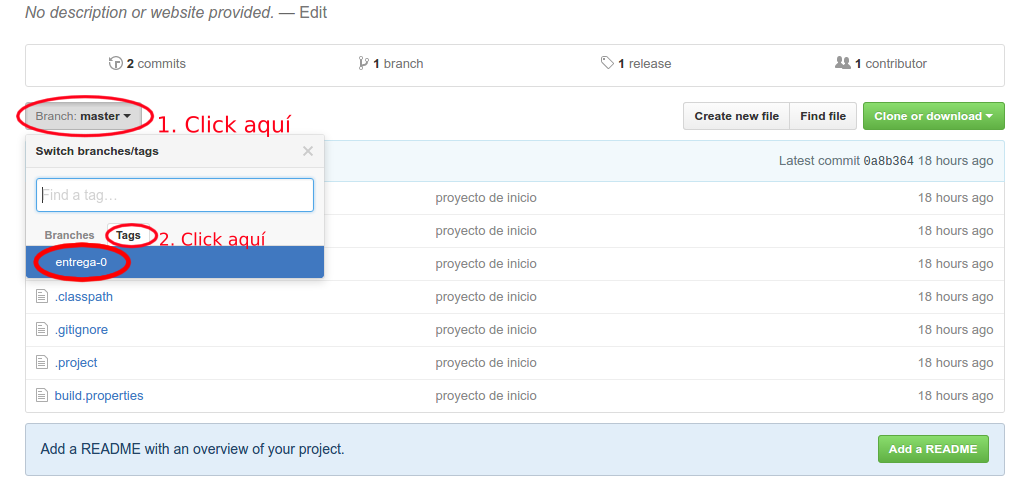
*Nota: Siempre hacer el tag después de haber hechos todos los commits necesarios y cuando se está seguro que se va a entregar esa versión (no genera commits nuevos).*



* Agregamos la información referente al tag y luego presionamos *“Create Tag and Start push”*.

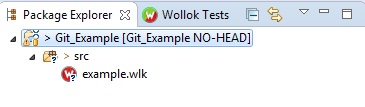




* Esto en GitHub se visualizará de la siguiente forma.

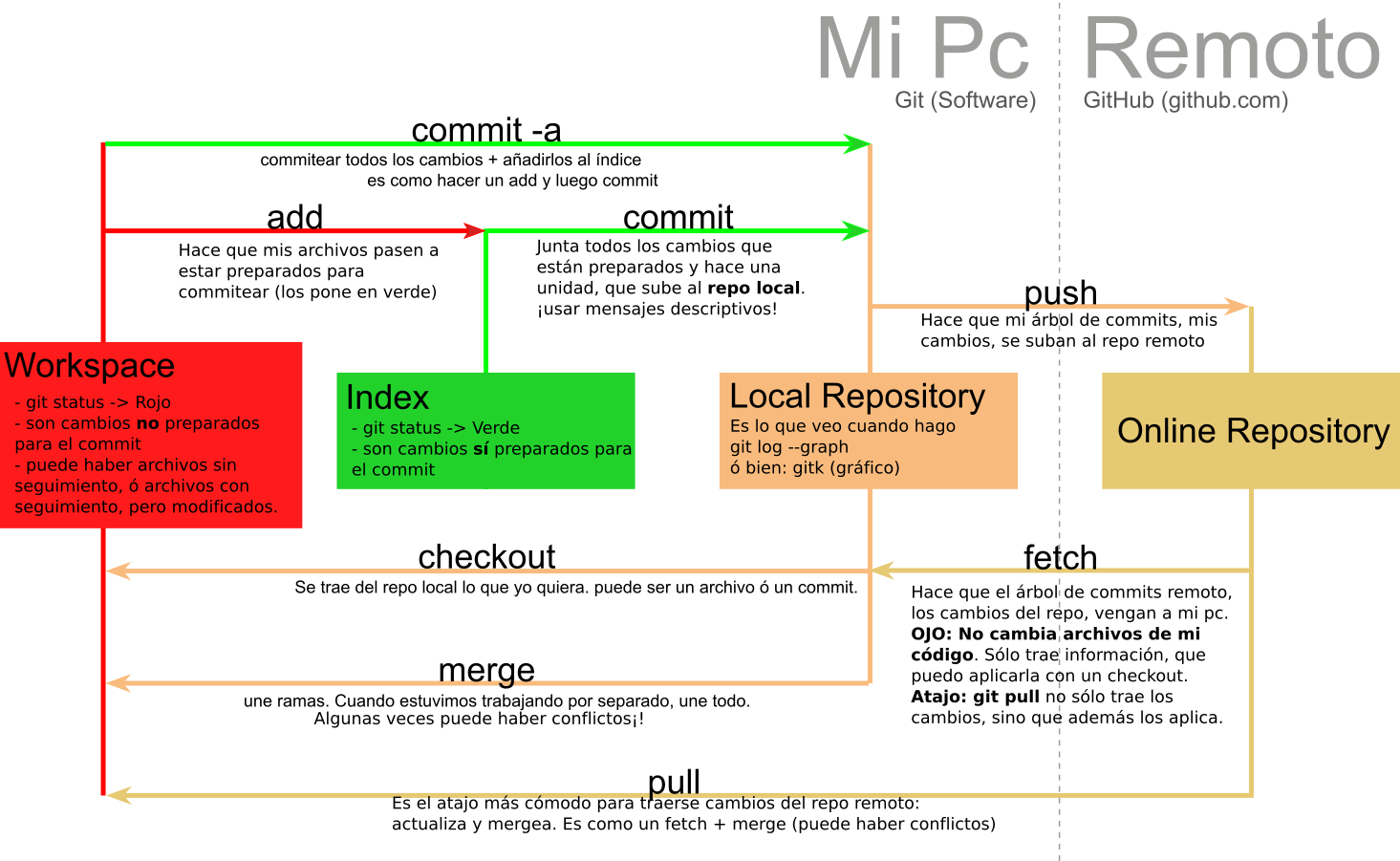
# Significado de los símbolos

En el project pueden aparecer cosas gráficas a medida de que vayan desarrollando, como por ejemplo algo asi:



* El “>” que aparece antes del nombre del proyecto significa que los cambios locales difieren de los remotos (o sea, que tenemos cosas para **pushear** ó para **pullear** ).
* Lo que aparece entre corchetes indica la rama en la que se trabaja (no entraremos en detalle sobre esto, en la cursada se va a usar la principal solamente).
* Los archivos o carpetas que tengan un “?” quiere decir que no existen en el repositorio remoto.
* El cilindro naranja ubicado abajo a la derecha indica que está vinculado con un repositorio remoto.

# Extra 1: Terminología completa de Git



# Extra 2: Equivalencias con la línea de comandos

¡Git se puede usar por línea de comandos!

De hecho, la mayoría de los desarrolladores lo hacen. Para eso se necesita tener una consola que entienda los comandos:

* En Ubuntu directamente desde la consola hay que correr “sudo apt-get install git” y debería descargar todo lo necesario para que funcione correctamente.
* En Windows y Mac recomendamos descargar una consola que se llama [git bash](https://git-scm.com/downloads).

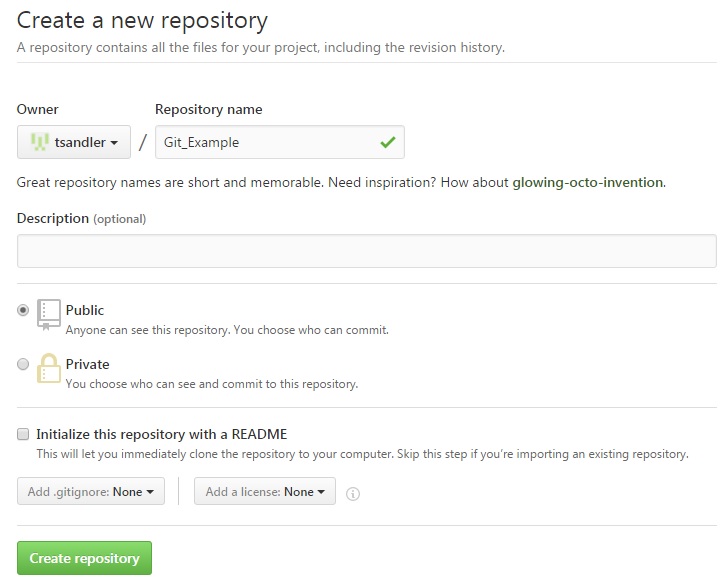
Aquí un pequeño resumen de los comandos de git comparados con el plugin de Eclipse EGit.

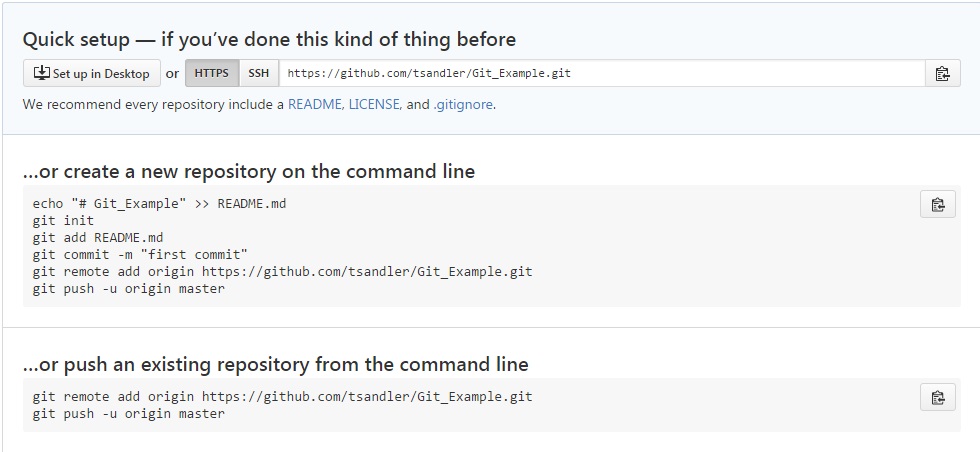
|  |  |  |
| --- | --- | --- |
| **Descripción** | **EGit** | **Línea de comandos** |
| **Clonar** el proyecto  (bajar el proyecto por primera vez) |  | git clone <ruta del repo> |
| **Cambiar** usuario y mail para todos los commits que se realicen en la computadora (para cualquier repositorio). | Esto solo lo realiza localmente, para hacerlo así borrar el “--global” | git config --global user.name “tsandler”  git config --global user.mail “tsandler92@gmail.com” |
| Hacer **pull**  (bajar los cambios) |  | git pull |
| **Preparar** los cambios para el commit.  (Pasar de Unstaged a Staged = pasar al Index) |  | git add <nombreArchivo>  o si no está en el directorio raíz:  git add <ruta/nombreArchivo> |
| Hacer **commit**  (guardar en mi repo **local**) |  | git commit -m “descripcion” |
| Hacer **push**  (subir los cambios commiteados) |  | git push |
| Hacer **checkout** y eliminar mis cambios para ir al último commit. |  | git checkout . |
| Conocer el estado actual de las cosas que cambié respecto del repo | Mirando los símbolos    También al hacer git commit aparecen los unstaged y staged. | git status  (aparecen unstaged en rojos y staged/preparados para el commit en verde) |

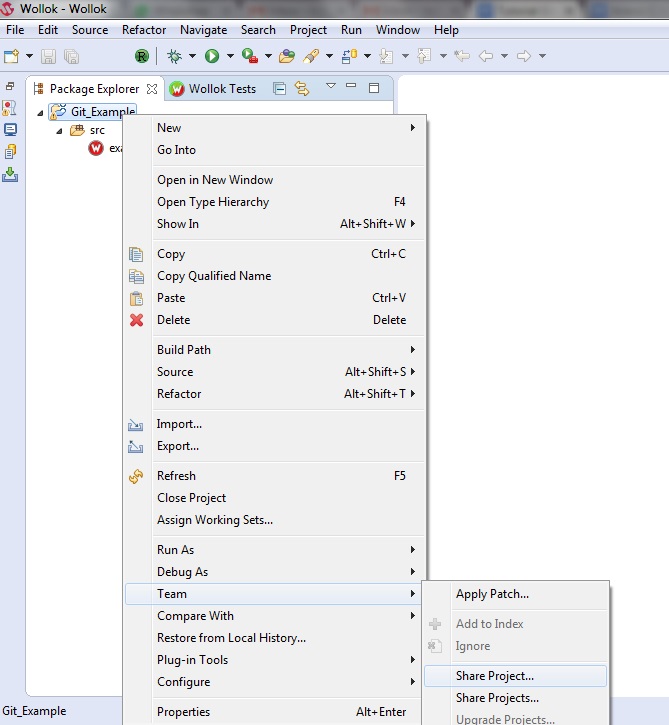
# Extra 3: ¿Cómo subo a Github cualquier proyecto mío?

También puede pasar que tengamos un proyecto y querramos subirlo a github, o mismo que necesitemos subir varios archivos al repo desde el inicio, entonces es más fácil crearlo localmente y subirlo en lugar de crearlo remotamente, clonarlo, copiar los archivos, cambiar el directorio dentro del workspace (para poder seguir usándolo desde Eclipse) y subirlo.

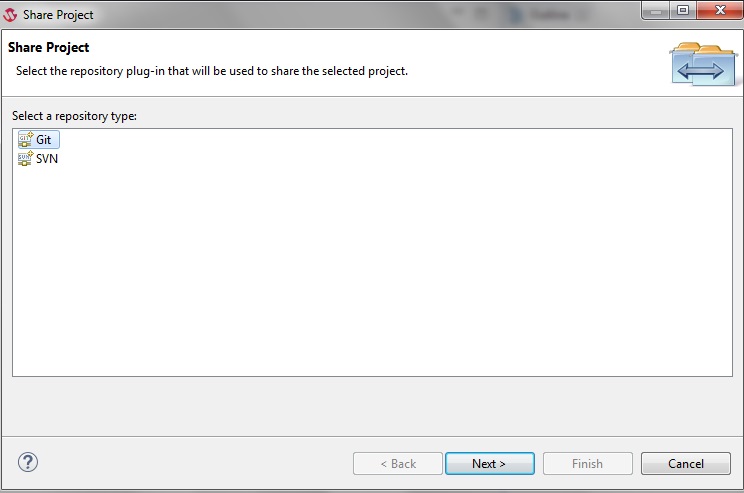
* Para esto inicialmente necesitamos crear un repositorio **vacío** en github, para eso hay que ir a la página de github y hacerlo.



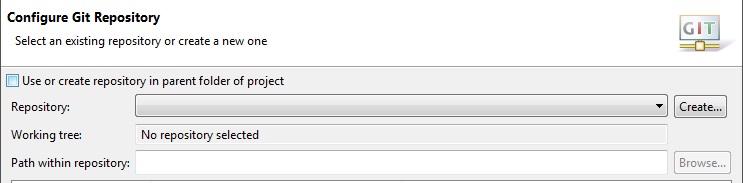
* Primero indicar el nombre del repositorio y el usuario en el que va a estar.
* Indicar la privacidad del repositorio (público o privado)
* Es **importante** dejar **desmarcada** la opción “Initialize this repository with a README”, esto hace que el repositorio se cree vacío.
* Al crear el repositorio vacío quedará así, siendo la ruta que aparece en https la que tendremos que usar para subir al repositorio. 
* Luego hay que crear un proyecto normalmente y crear un repositorio local para el mismo, para eso hay que hacer click derecho en el proyecto ->Team->Share Project…



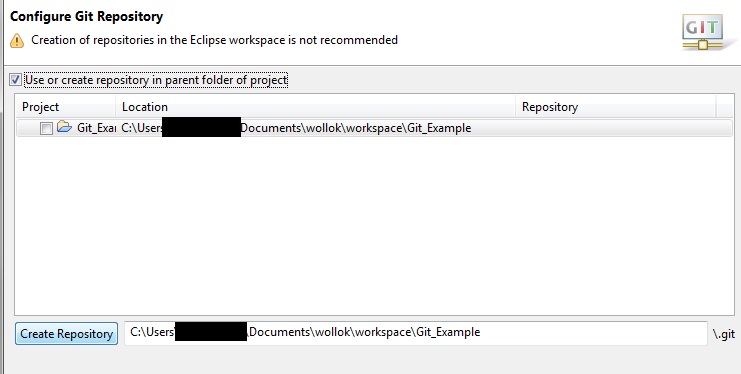
* Seleccionar Git y darle next (este paso no es requerido si solamente tenemos Git instalado, algo que ocurre por defecto desde Wollok 1.6.0)



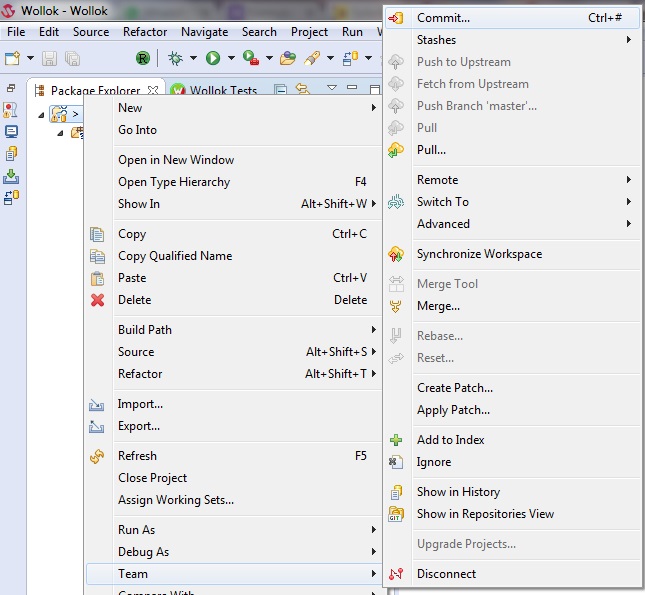
* Tildar el checkbox “Use or create repository in parent folder of project”.



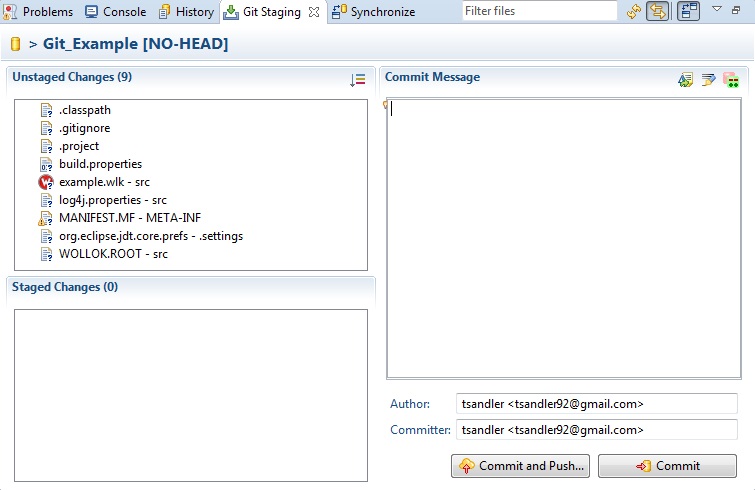
* Seleccionar el proyecto, , ignorar la advertencia, luego el botón “Create repository” y clickear Finish.



* ¡Listo! Ahora tu proyecto tiene un **repo local**.
* Para sincronizarlo con un repositorio remoto hay que darle click derecho al proyecto->Team->Commit



* Nos aparecerá esta ventana nuevamente, seleccionamos los archivos que queremos subir, ingresamos un mensaje (en general el primer commit se llama “Initial commit”) y le damos Commit and Push.



* Una vez hecho esto nos aparecerá esta ventana, donde ingresamos la ruta del repositorio, usuario y contraseña y le damos next. Seguimos avanzando, nos logueamos una vez más y listo, el repositorio ya fue inicializado con los archivos del repositorio local que hayamos seleccionado al hacer el commit.

