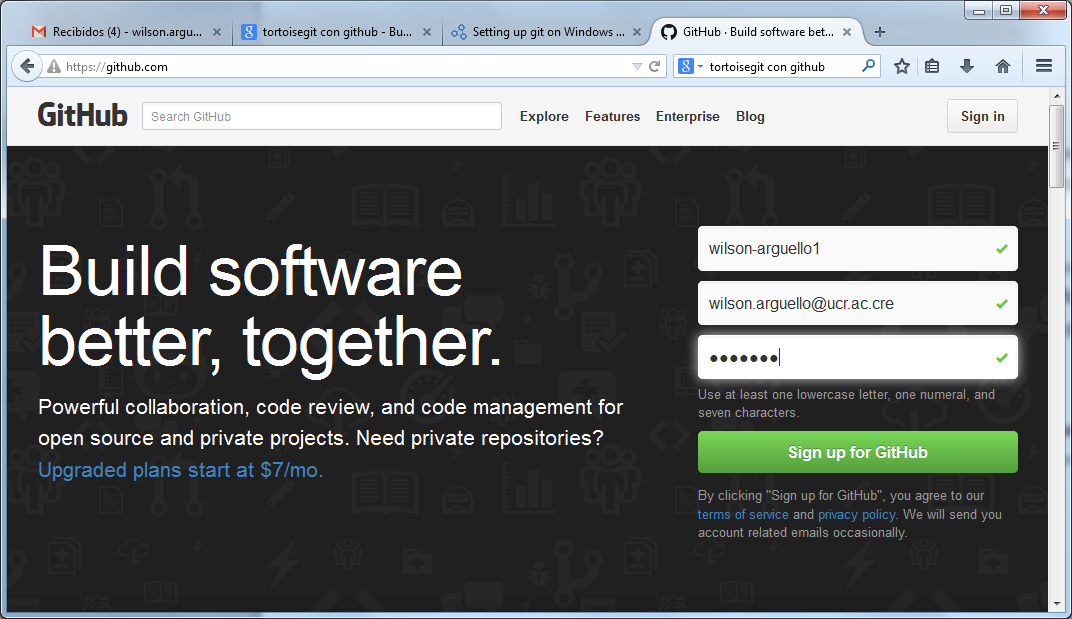
# Instalación de TortoiseGIT con GitHub

Para poder utilizar uno de los programas de control de versiones más popular en ambientes Windows y Mac el Tortoise, vamos a configurar tanto el cliente como una cuenta gratuita en el servidor de control de versiones GitHub.

Git es uno de los programas para gestionar el control de versiones más popular hoy en día, y sus desarrolladores crearon el servicio para control de versiones en la nube “GitHub”, el cual se puede utilizar de forma gratuita (siempre y cuando los archivos o código de desarrollo sean públicos y libres), o de pago.

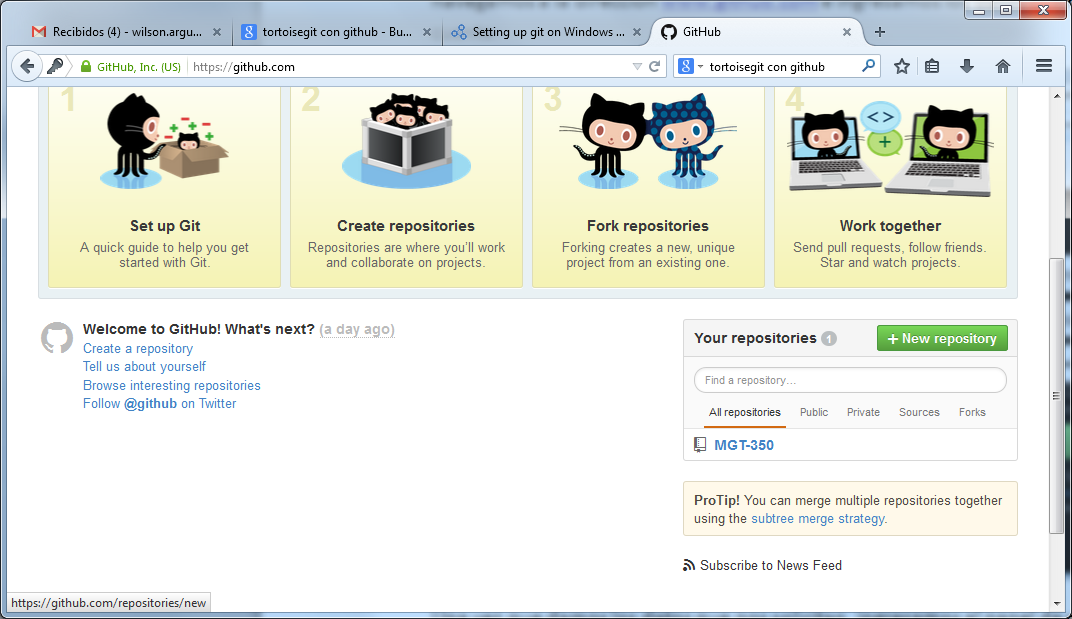
## Configuración de GitHub.

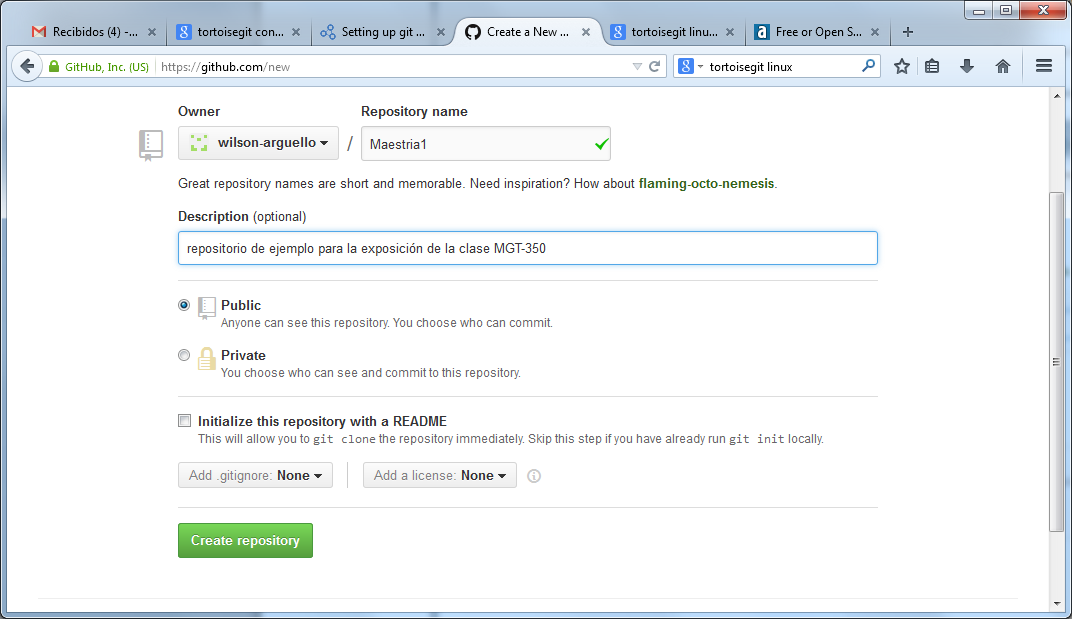
Lo primero que vamos a hacer es configurar una cuenta gratuita en el servidor GitHub, para ello navegamos a la dirección [www.github.com](http://www.github.com) e ingresamos los datos que nos solicitan para crear una cuenta:



Una vez que damos los datos que nos solicitan, ingresamos al panel de control en donde podemos empezar a gestionar nuestro repositorio, para ello le damos click al botón que dice crear nuevo repositorio.

Dentro de las opciones nos van a preguntar por un nombre para nuestro repositorio (para este ejemplo el cual vamos a llevar de inicio a fin, le vamos a poner Maestria1), y una descripción, el cual va a indicarnos cuál es el propósito del mismo, lenguaje de programación, desarrolladores, y cualquier otro dato que creamos importante.



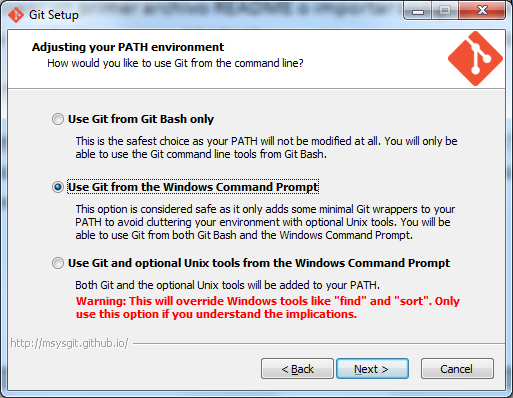


GitHub recomienda crear un primer archivo README o importar código inicial, para nuestro ejemplo de damos la opción de crear un README.txt en blanco y estamos listos por el momento.

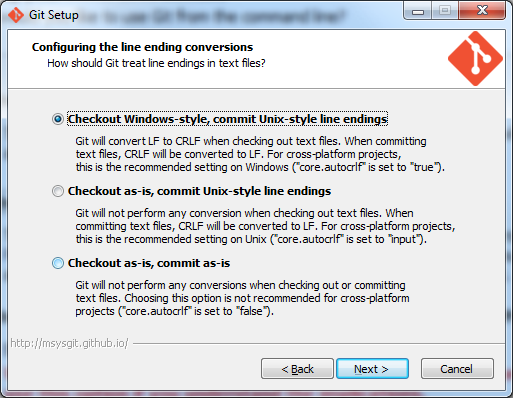
## Instalación y configuración de TortoiseGit.

Para configurar el TortoiseGit, vamos a instalar primero el complemento “Git for Windows”, el cual nos permite comunicarnos con el servidor GitHub. La instalación es muy básica, después de descargar el programa desde la página web <http://msysgit.github.io> y ejecutar el instalador, solo tenemos que prestar atención a un par de pasos:

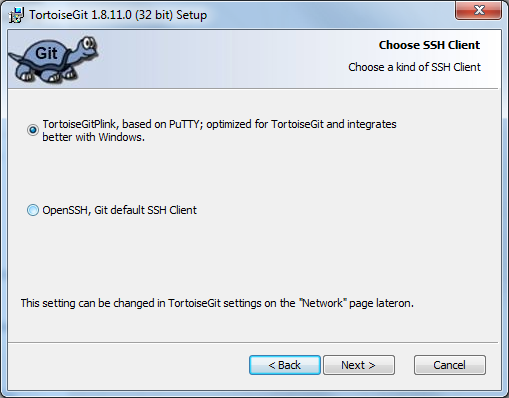
1. Cuando pregunte por el ajuste del “PATH environment”, vamos a escoger la opción que dice “Use Git from the Windows Command Prompt”.



1. Cuando pregunte por el “Line ending conversions”, escogemos la opción que dice “Checkout Windows-style, commit Unix-style line ending”



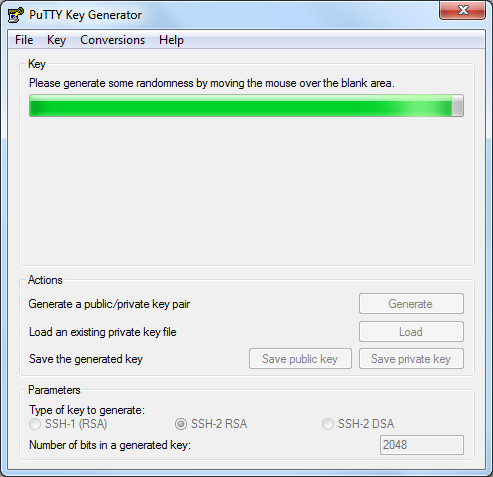
Ahora vamos a la página web del proyecto de TortoiseGit (<https://code.google.com/p/tortoisegit> ) y descargamos la aplicación para el sistema operativo correspondiente (en nuestro caso Windows 32 bits). Iniciamos el instalador y seguimos los pasos sugeridos, solamente cuando no pregunte por el cliente para SSH, escogemos la opción que dice “TortoiseGitPlink, base don Putty; optimized for TortoiseGit and integrates better with Windows”.



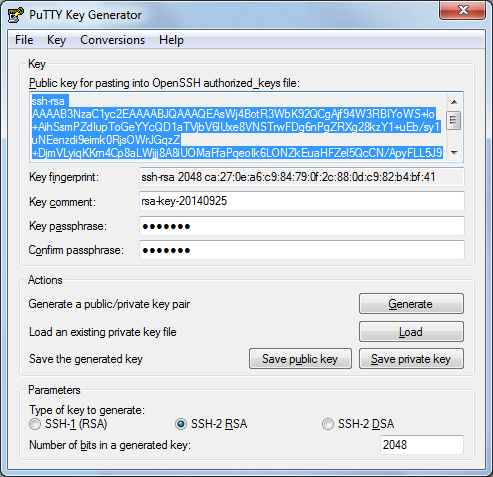
**Configurando la seguridad para comunicar el cliente con el servidor.**

Para poder usar el TortoiseGit integrado con el GitHub, necesitamos configurar los dos de tal forma que generen una relación de confianza en el equipo, siguiendo estos pasos:

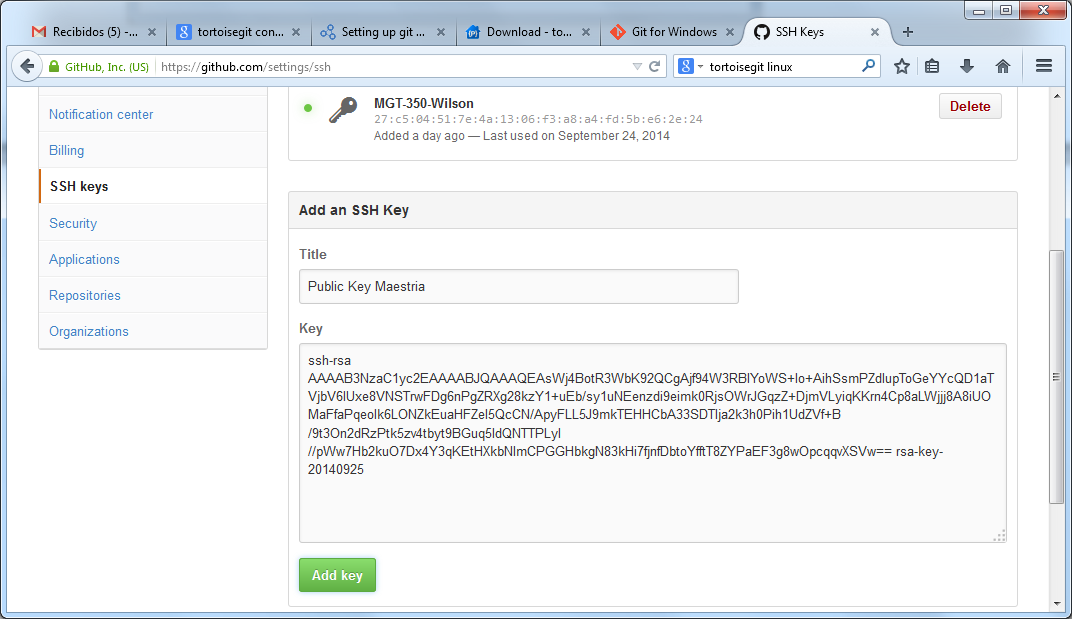
1. Nos vamos a la página de GitHub, en “Edit Profile” 🡪 “SSH key” y damos click donde dice “Add SSH Key”. (por el momento dejamos la página abierta, ya que la ocuparemos más tarde).
2. Entre los paquetes que instalamos previamente con el TortoiseGit, hay uno llamado PuttyGen, lo buscamos en los programas instalados y lo ejecutamos. Le damos click al botón “Generate” y movemos insistentemente el mouse en el espacio en blanco (según las indicaciones del programa).



1. Seleccionamos y copiamos (en el portapapeles) el código generado, le ponemos un password o Key passphrase, y le damo click a “Save private Key”, para guardar la llave privada en un lugar seguro (preferiblemente fuera del directorio de trabajo).



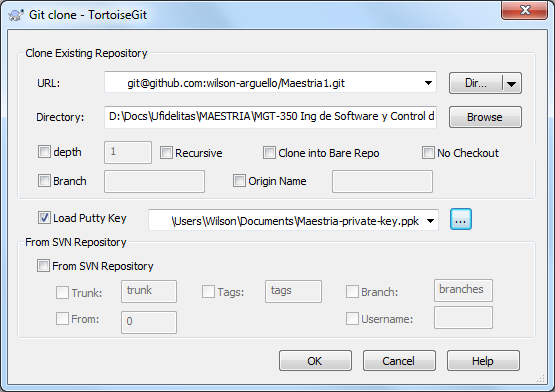
1. El código que copiamos (public Key), lo vamos a pegar en la página de GitHub en donde dice “Key” (la página que previamente dejamos abierta), le ponemos un título o nombre y le damos click a “Add Key”.



## Usando TortoiseGit

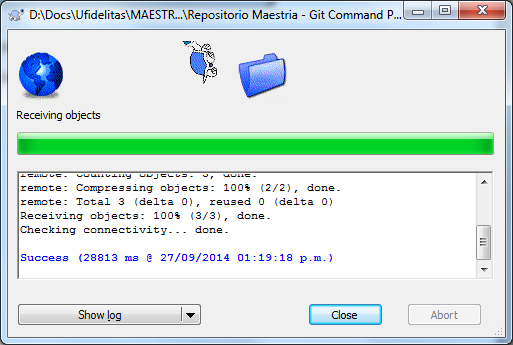
Ahora que ya tenemos todo configurado, vamos a usar nuestro cliente para crear un proyecto local y sincronizarlo con nuestro repositorio en GitHub.

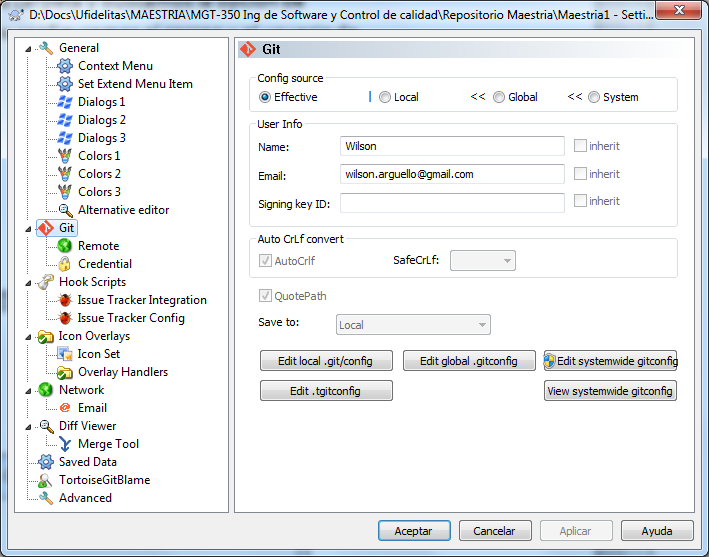
Lo primero que hacemos es crear nuestra carpeta local, en donde vamos a sincronizar nuestro repositorio, desde ahí le podemos dar la opción de crear un repositorio en nuestro GitHub, o en nuestro caso para este ejemplo, clonar un repositorio que ya existe. Le damos click derecho en la carpeta creada y buscamos la opción “Git Clone”. En donde nos solicitan el URL, ingresamos el que nos indica la página del repositorio del GitHub de Maestria1 (lo buscamos en la sección que dice SSH clone URL), le indicamos el directorio donde queremos que quede guardada la copia, donde nos solicita un “Putty Key”, seleccionamos la llave privada que guardamos previamente en los pasos anteriores.



Le damos OK, y nos solicita un passfrase, la cual es la misma que hemos venido utilizando, le damos ok otra vez y listo, nos pide validar el certificado de la llave (le damos SI) y al final nos indica que se realizó con éxito. Si revisamos la carpeta local de referencia, veremos los archivos del repositorio copiados ahí.

Ahora le damos click con el botón derecho a nuestra carpeta 🡪TortoiseGit🡪Setting y buscamos la opción de GIT, donde configuramos el correo y un nombre para las modificaciones que queramos configurar.



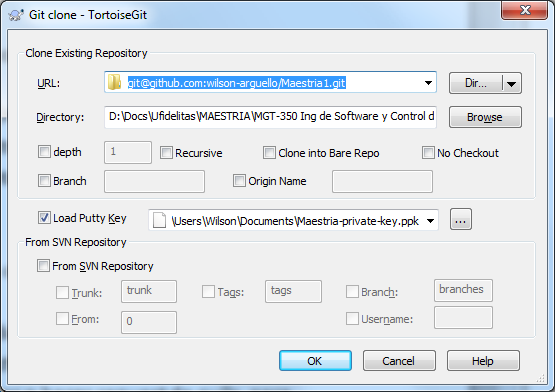


## Colaborando en el desarrollo de un proyecto con GitHub y TortoiseGit.

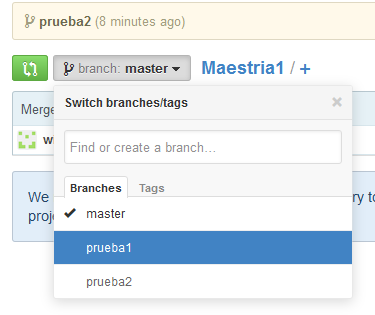
Uno de los puntos que tiene que quedar claro, a la hora de trabajar colaborativamente con un programa de control de versiones, es que para que los proyectos tengan éxito, debe existir un proyecto principal (Trunk) y cada uno de los colaboradores tener una o más copias del proyecto, para trabajar (Branch). Con esta metodología, una o más personas se encargan de valorar los cambios realizados por los colaboradores y hacen la aceptación de los cambios en el proyecto principal (Commits).

Para continuar con nuestro ejemplo vamos a crear dos carpetas de trabajos (prueba1 y prueba2), las cuales vamos a configurar y sincronizar cada una con una rama de nuestro proyecto principal, y utilizaremos los comandos básicos de esta herramienta para hacer modificaciones, sincronizar, borrar contenido, entre otros.

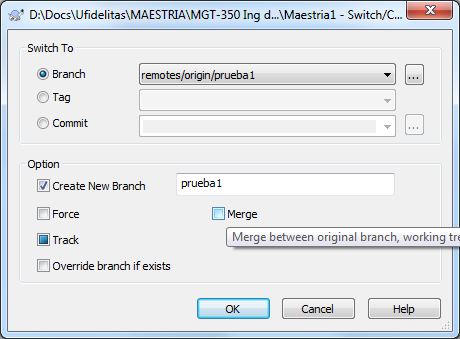
1. Vamos a crear dos carpetas Prueba1 y Prueba2, y las vamos a sincronizar con nuestro repositorio en GitHub.
   1. Creamos las carpetas y le damos botón derecho 🡪Git Clone
   2. En el campo URL ingresamos la dirección de nuestro repositorio.
   3. En el campo URL dejamos la dirección de la carpeta.
   4. Le damos OK



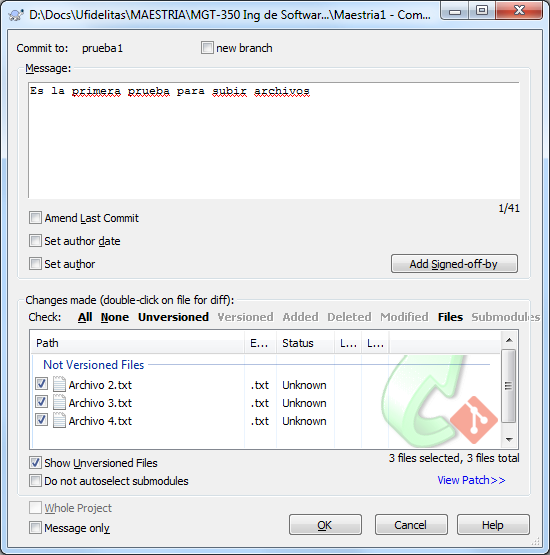
1. Desde la página principal de nuestro repositorio, vamos a crear dos branchs (uno para prueba1 y el otro para prueba2).
   1. Vamos a nuestro repositorio y buscamos donde dice Branch, ahí escogemos crear un branch nuevo y le ponemos el nombre prueba1
   2. Repetimos el paso anterior y le ponemos prueba2 como nombre.



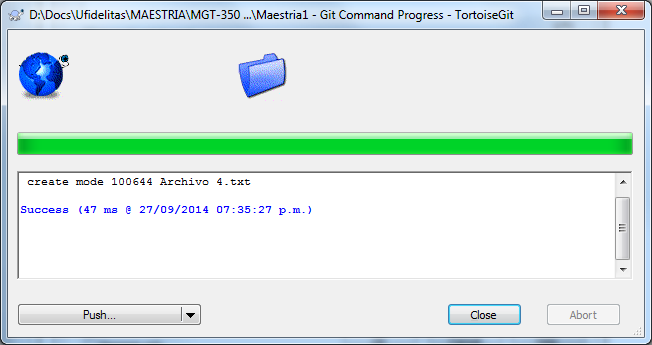
1. Vamos a sincronizar nuestras dos carpetas locales con cada uno de los branchs creados, para asegurarnos de tener los mismos archivos del proyecto.
   1. Volvemos a las carpetas creadas, dentro vamos a encontrar una carpeta nueva que se llama Maestria1. Le damos botón derecho🡪 TortoiseGit🡪Pull, para jalar toda la información del repositorio, incluido los nuevos branchs creados.
   2. Ahora vamos a decirle a nuestras carpetas que se van sincronizar con los branch respectivos, para esto le damos botón derecho🡪TortoiseGit🡪Switch/CheckOut, escogemos como branch el respectivo a cada carpeta (prueba1 y prueba2).
   3. Escogemos que cree un nuevo Branch local (prueba1 o Prueba2) y le damos ok.



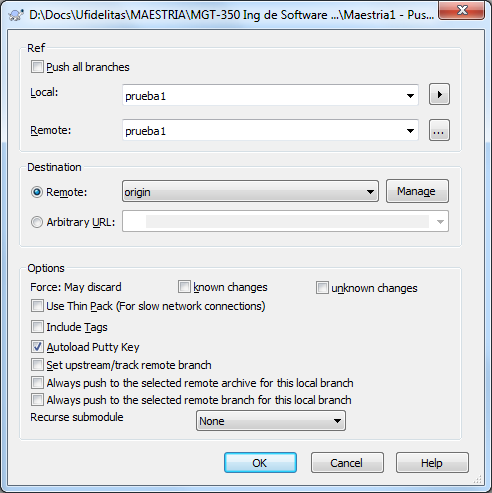
1. Vamos a hacer modificaciones en los proyectos, y a solicitar o hacer request de pulls, para la rama principal.
   1. Creamos directamente el “archivo 1” en el branch principal, hacemos un Pull desde nuestras carpetas para confirmar que las mismas siguen vacías, esto es porque ya están apuntando a un branch diferente.
   2. Copiamos los archivos: archivo 2, archivo 3 y archivo 4 en la carpeta respectiva a prueba1, y lo archivos: archivo 5, archivo 6 y archivo 7 en la carpeta prueba2.
   3. Para las dos carpetas, le damos click derecho🡪Git Commit, para que los cambios se reflejen en el repositorio local.

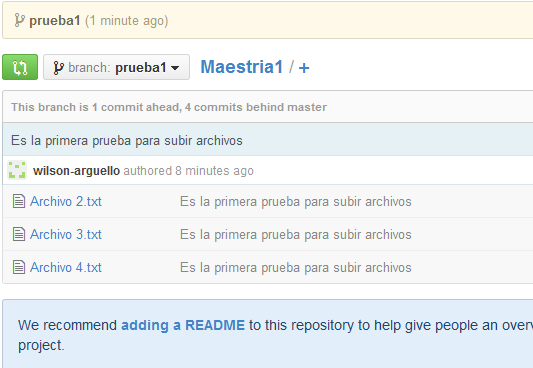


* 1. Escribimos un mensaje del Commit, marcamos los archivos que queremos que se actualicen y damos click a ok.
  2. Nos aparece otra ventana que nos indica que el commit fue exitoso, ahí le damos click a Push (para subir los cambios a GitHub).

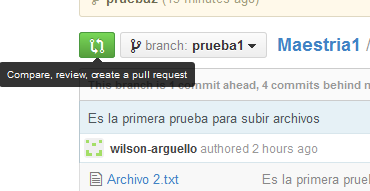


* 1. Le damos Ok (asegurándonos que tanto el local como remote apunten al branch correcto), y podemos revisar nuestro branch en GitHub para ver los cambios realizados.

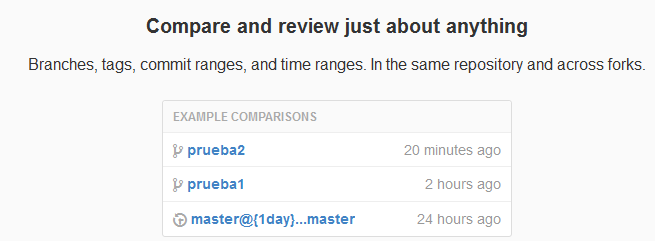


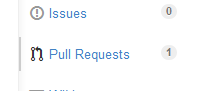


1. Desde el GitHub, vamos a aprobar o rechazar los cambios solicitados.
   1. Desde el panel principal de nuestro branch principal, podemos buscar un ícono que nos indica que podemos comparar, revisar o crear las solicitudes de Pull “Pull Request”, de damos click para poder ver las opciones que nos da. Otra opción es que nos llegue por correo una notificación de solicitud de “Pull Request” desde un colaborador, pero para nuestro ejemplo lo vamos a realizar de forma directa desde nuestro proyecto.

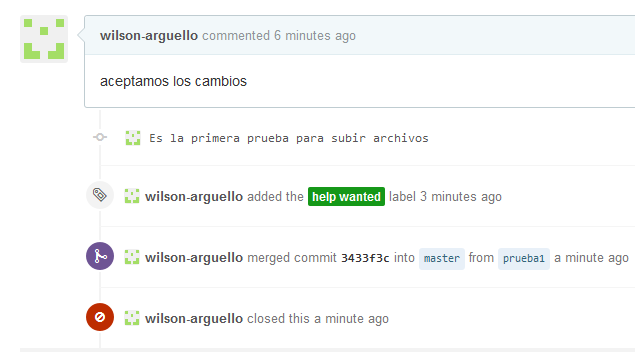


* 1. Escogemos el branch con el que queremos hacer el Pull, agregamos los comentarios que queramos al respecto y le damos “Create Pull Request”. Estos cambios aún no han sido aceptados para hacer Merge del branch principal con la rama.





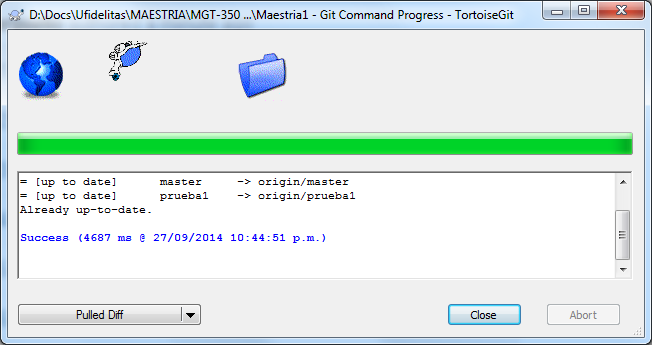
* 1. Cuando revisamos el “Pull Requests” vemos que tenemos una solicitud pendiente para aceptar o rechazar. La escogemos y podemos etiquetarla, asignarle un usuario, marcarla y finalmente aceptar el Merge.



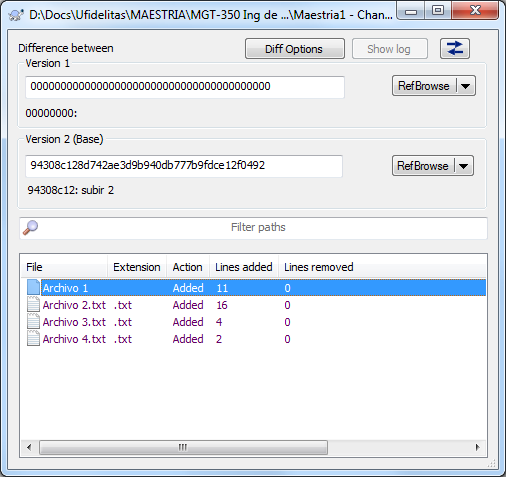
* 1. Ahora revisamos el Branch principal y podemos ver el resultado. Además podemos hacer un Pull Request para hacer un Merge de los brach prueba1 y prueba2 con respecto al principal, esto para tener los 3 brach sincronizados.



1. Aquí podemos volver a sincronizar nuestras dos carpetas para revisar los cambios realizados, además de modificar texto en los archivos para ver el manejo de las diferencias dentro de los mismos.
   1. Botón derecho en las carpetas locales 🡪TortoiseGit🡪Pull y le damos OK.



* 1. Podemos darle click a Pulled Diff para revisar los cambios realizados. Al darle click a cada uno de los archivos, el sistema nos muestra los cambios dentro del archivo, desde la versión anterior a la nueva.



* 1. Al revisar nuestra carpeta, podemos ver que los archivos ahora están actualizados.