# GitHub Superfácil

Contents

[GitHub Superfácil 1](#_Toc444521495)

[1 Que es github 2](#_Toc444521496)

[1.1 Primeros pasos. Crearse una cuenta 2](#_Toc444521497)

[1.2 Crear un repositorio remoto desde cero 2](#_Toc444521498)

[1.3 Crear un repositorio basado en uno existente (fork) 4](#_Toc444521499)

[1.4 Usar github sin instalar git 5](#_Toc444521500)

[1.5 Instalar git: podrás clonar 5](#_Toc444521501)

[1.6 Terminología importante 7](#_Toc444521502)

[2 Crear el proyecto en local 9](#_Toc444521503)

[2.1 Crear el proyecto local desde cero 9](#_Toc444521504)

[2.2 Crear el proyecto en local desde clone 10](#_Toc444521505)

[2.2.1 Clonar por HTTP 11](#_Toc444521506)

[2.2.2 Clonar por SSH 12](#_Toc444521507)

[3 Commit y push 14](#_Toc444521508)

[4 Acciones avanzadas: ramas 16](#_Toc444521509)

[4.1 Ramas en el repositorio remoto 17](#_Toc444521510)

[4.2 Has terminado: Merge 17](#_Toc444521511)

[5 Herramientas gráficas para Linux : gitg 18](#_Toc444521512)

[6 APÉNDICE: algunos comandos 18](#_Toc444521513)

# Que es github

Github es una herramienta para programar de forma colaborativa controlando las versiones del software. En realidad lo más importante de github es la herramienta git .

* Git es un sistema de control de versiones distribuido (scvd) escrito en C. aunque existen herramientas visuales para usarlo, lo mas recomendable es utilizar la línea de comando ya que hay muy pocas operaciones y están bien documentadas
* [www.github.com](http://www.github.com) es la página web donde nos damos de alta como usuarios, además de poder buscar y crear repositorios.

El logotipo de github es un “octocat”, la mezcla entre un pulpo y un gato. AL parecer las diferentes patas del pulpo “simbolizan” las ramificaciones (branches) que puede tener un proyecto

|  |  |
| --- | --- |
|  | Proyecto = repositorio |

A los proyectos se les llama “repositorios”. Al fin y al cabo git no solo sirve para proyectos de desarrollo software. Se podría usar (por ejemplo) para hacer un repositorio de fotos y del mismo modo que con los repositorios de software, se podrían crear ramas, variantes del repositorio que tienen fotos adicionales y/o fotos cambiadas.

## Primeros pasos. Crearse una cuenta

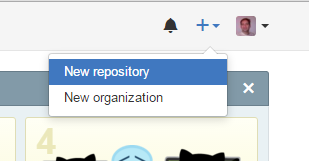
Lo primero es crear una cuenta entrando en [www.github.com](http://www.github.com)

## Crear un repositorio remoto desde cero

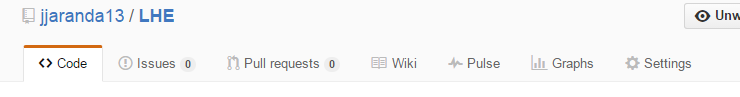
Para crear un proyecto (=repositorio) podemos crearlo desde la web. Esto lo haremos si vamos a crear un proyecto desde cero.

Si queremos crear un proyecto a partir de otro que ya existe en github y queremos trabajar aumentando sus funcionalidades, entonces lo que tenemos que hacer es un “fork”, lo vemos en el siguiente apartado

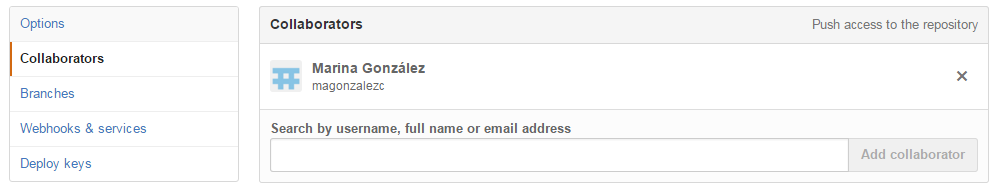
Lo primero es crear nuestro repositorio



Una vez creado el repositorio podemos crear y editar ficheros desde el interfaz web. De momento el proyecto solo está vivo en github. Nuestro PC no sabe nada de él. Después veremos cómo trabajar desde nuestro PC y subir los cambios que hagamos.

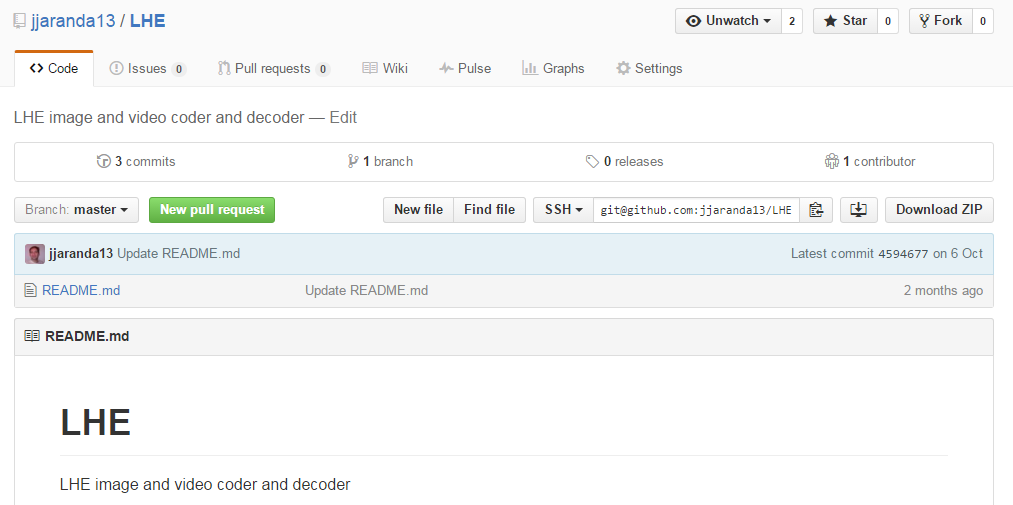


En el encabezado del repositorio podemos ver varias pestañas. La de <code> nos muestra todos los ficheros y directorios de que consta el proyecto. Una cosa que podemos hacer en este punto es entrar en “settings” y dar de alta colaboradores



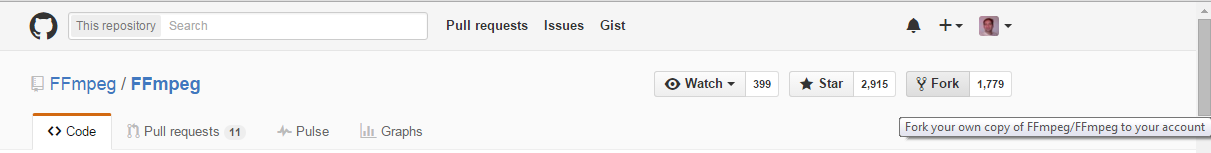
Un colaborador no es simplemente alguien que contribuye al proyecto, sino que es alguien que tiene ciertos derechos. Puede hacer un commit y modificar el repositorio sin pedir permiso. Sin embargo un “contributor” es alguien que hace una contribución a tu proyecto pidiéndote permiso.

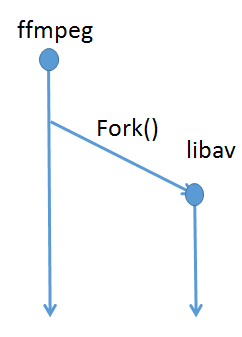
Dentro de la pestaña <code> vemos 4 elementos: commits, branch, release y contributors.



## Crear un repositorio basado en uno existente (fork)

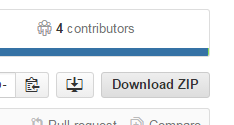
Supongamos que no queremos crear un proyecto desde cero sino implementar nuevas funcionalidades en un proyecto existente como ffmpeg. En ese caso buscamos el proyecto en github y haremos un “fork” pulsando el botón que aparece en el encabezado. EN el caso de ffmpeg vemos que entre toda la comunidad se han hecho nada menos que 1779 forks!!!



Un fork crea un proyecto a partir de otro, es decir, “copia” todo el repositorio desde el proyecto origen. Este vínculo permite que nuestras contribuciones al código puedan ser incorporadas en el repositorio original mediante una acción “pull request”, que deberá aceptar el creador del proyecto original si le parecen bien nuestras contribuciones. SI no aceptan nuestras contribuciones, quedaremos como un proyecto “independiente” y desconectado, como ocurrió con libav y ffmpeg, y cuanto más tiempo pase, más difícil será aceptar el pull request desde libav pues habrá más probabilidad de que los cambios ya no sean compatibles con modificaciones hechas sobre la versión actual de ffmpeg.

## Usar github sin instalar git

Hay una cosa que se puede hacer sin instalar la herramienta git. Si tienes usuario en github y buscas un proyecto que te gusta y te lo quieres descargar a tu pc para probarlo puedes simplemente descargarte el repositorio en un zip y lo compilas o ejecutas en tu PC. Esto te convierte en un usuario final de un software desarrollado por alguien, pero no puedes hacer contribuciones ni nada



También puedes crear un repositorio o clonarlo, y hacer modificaciones desde la pagina web, haciendo clic en los ficheros y editándolos desde el browser. Después podrías hacer un pull request si clonaste el repositorio. Esto tiene sentido solo si el proyecto es pequeñito y/o quieres hacer una contribución sencilla.

Desde la web puedes incluso crear nuevas ramas (branches) del clone o repositorio que has creado. Sin embargo, hacerlo todo así es un rollo, porque necesitas compilar y probar tus cambios en tu PC y después copiar y pegar todos los cambios que hayas hecho en los ficheros del repositorio web usando el browser.

Si no instalas git, no podrás “clonar de verdad”, es decir, no crearas un vínculo entre el repositorio remoto y tu pc y tendrás que actualizar manualmente los cambios en la web. Si tus contribuciones son complejas y quieres crearte diferentes ramas o “branches” para probar diferentes cosas, lo mejor es instalarte git.

## Instalar git: podrás clonar

Clonar es copiar un repositorio en tu PC, pero estableciendo un vínculo con el repositorio remoto, de modo que le puedas pedir a git que suba (push) los cambios.

Trabajando con git, todos los programadores tienen una copia (un “clone”) del repositorio y de sus ramas en su PC, pudiendo hacer cambios en una rama local y después hacer commit para que la definición de la rama local lo tenga en cuenta antes de sincronizar (push) con un repositorio remoto. Los dueños de los repositorios pueden sincronizar los cambios via push (transfiere los cambios al repositorio remoto) o pull (obtiene los cambios desde un repositorio remoto).

* En Linux ubuntu

>sudo apt-get install git

* En Windows:

Descargar e instalar el ejecutable desde:

[http://msysgit.github.com](http://msysgit.github.com/)

La versión Windows viene con un interfaz de comandos y un interfaz grafico. Ambos podemos arrancarlos desde cualquier directorio

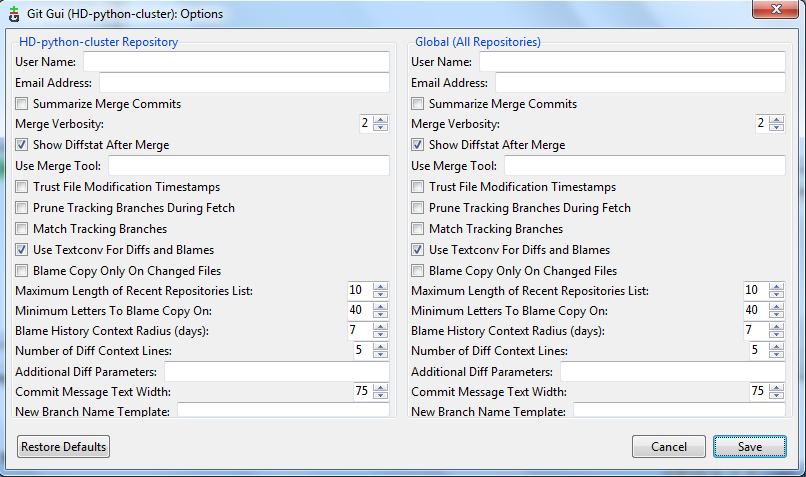
|  |  |
| --- | --- |
| *El menú contextual* | *El git GUI de windows* |

Una vez instalado podremos usarlo desde una línea de comando y lo primero que debemos hacer es decirle a git quienes somos, para que nos identifique en las operaciones que hagamos contra el servidor remoto. En Linux:

$ git config --global user.name "pepito"

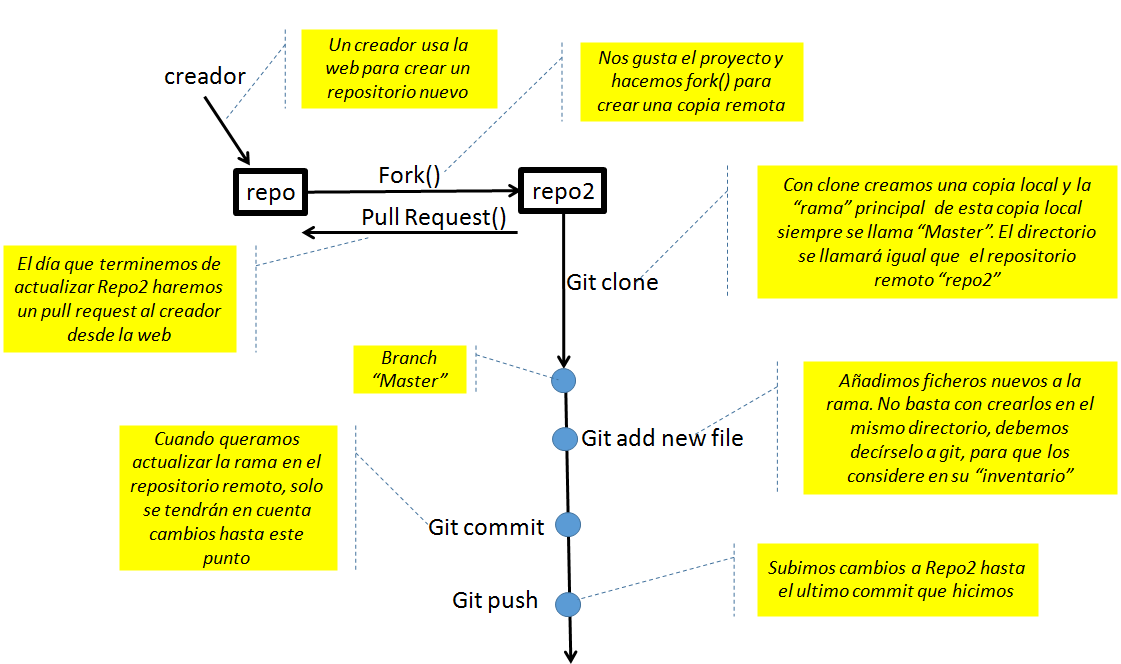
$ git config --global user.email pepito@example.com

En Windows para identificarte debes primero haber clonado algún repositorio y después vas a “options”. Verás un menú de opciones donde puedes rellenar el usuario global. El usuario con el que se trabaja en cada repositorio puede ser diferente pero lo normal es tener un único usuario para todo



## Terminología importante

Una imagen vale más que mil palabras: clonar, crear branches, commit, push.. este dibujo resume todo lo básico



|  |  |
| --- | --- |
| **Término** | **Definición** |
| Repositorio | Un repositorio contiene la historia, las diferentes versiones en el tiempo y todas los distintos branch y etiquetas. En Git, cada copia del repositorio es un repositorio completo. El repositoro te permite obtener revisiones en tu copia actual. |
| Branch | Un branch es una linea separada de código con su propia historia. Es posible crear un nuevo branch de a partir de uno existente y cambiar el código independientemente de otros branches. Uno de los branches es el original (generalmente llamado master). El usuario selecciona un branch y trabaja en ese branch seleccionado, el cual es llamado copia actual (working copy). Seleccionar un branch es llamado "obtener un branch" (checkout a branch) |
| Etiquetas (Tags) | Un tag (una etiqueta) apunta a un cierto punto en el tiempo en un branch específico. Con un tag, es posible tener un punto con un tiempo al cual siempre sea posible revertir el código, por ejemplo, ponerle el tag "testing" al código 25.01.2009 |
| Commit | Puedes hacer commit de los cambios a tu repositorio **LOCAL**. Esto crea una **Revisión** nueva (a la cual siempre podremos volver pues es como un punto de control). Cada commit contiene el autor del commit y un comentario. |
| URL | Una URL en Git determina la ubicación de un repositorio. La hay de dos tipos: HTTP y SSH. Para clonar necesitamos pasar como parámetro a git esta URL |
| Revisión | Cada commit genera una revisión (=una versión del código fuente). Git identifica revisiones con un id SHA1. Los id son de 160 bits de largo y son representados en hexadecimal. La última versión puede ser direccionada a través de "HEAD", la versión anterior mediante "HEAD-1" y así. |

# Crear el proyecto en local

Podemos hacerlo de dos formas:

* desde cero y creando el vínculo mediante el comando “git remote add”
* clonando con el comando “git clone” , que es mas sencillo y desde el momento que hacemos el clone ya hemos creado el vinculo

## Crear el proyecto local desde cero

Antes hemos visto como crear un repositorio pero no lo hemos “conectado” con nuestro PC.

Una vez creado el repositorio, al que habremos llamado por ejemplo **Prueba**, entramos en nuestro terminal favorito, y tecleamos un par de comandos sencillos:

mkdir Prueba

cd Prueba

git init

echo "Este proyecto va a ser la leche!" > README

Esto que hemos hecho anteriormente sirve para crearnos el directorio de trabajo en nuestra máquina local, inicializar los ficheros necesarios para que GIT funcione y creamos un fichero README básico para que nuestro proyecto tenga algo dentro.

Una vez hecho esto, añadimos este fichero a GIT y hacemos nuestro primer commit. Estos dos pasos son importantes cada vez que queramos que nuestros cambios se envíen al servidor, cada commit que hagamos será como una copia de seguridad de nuestro trabajo, en cualquier momento que queramos podemos volver a una de estas copias y además nos sirven para ver el proceso de cambio de nuestro proyecto.

git add README

git commit -m 'Primer commit de mi gran proyecto'

Y ahora procedemos a decirle a nuestra copia local donde debe alojarse, es decir, cual es el repositorio remoto donde va a estar alojado y donde se guardarán los cambios, en este caso en el de GitHub. El comando push envía estos cambios al servidor, en este caso indicando el repositorio donde va a alojarse (más adelante podremos hacer push sin parámetros):

git remote add origin [git@github.com:tuusuario/Prueba.git](mailto:git@github.com:tuusuario/Prueba.git)

git push origin master

Con esto tendremos nuestro proyecto iniciado y disponible en GitHub.

Por último, vamos a hacer un par de cambios en varios ficheros, y volver a hacer commit y push. En este caso no es necesario indicar el alias con el que llamamos al proyecto, ya que esto queda almacenado en la copia local.

touch index.php

echo "<?php phpinfo(); ?>" > index.php

git status –s

?? index.php

git add index.php

git commit -m 'Añadido el fichero inicial de la aplicación'

git push

Y listo, ya tenemos un nuevo fichero añadido en nuestro repositorio, si nos fijamos, justo al crear y editar el fichero, hacemos git status -s para comprobar los ficheros que hemos cambiado y aparece el nuevo fichero junto con dos interrogantes que nos indican que git ha encontrado el fichero pero no lo tiene incluido aún en su sistema. Con git add index.php incluimos el nuevo fichero para que GIT lo tenga en cuenta en su sistema.

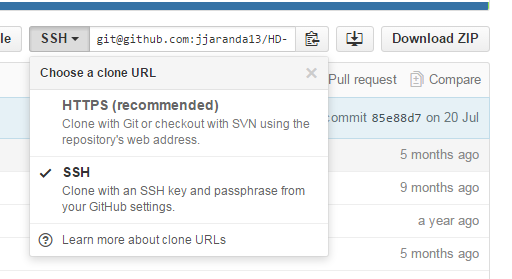
Y por ahora creo que está bien, hemos aprendido lo básico para trabajar con GIT.

## Crear el proyecto en local desde clone

Podemos clonar de dos modos diferentes. OJO elegimos desde la web de github el método que vamos a permitir para clonar

* Por HTTP: no hacen falta claves ni nada. Es lo más fácil si estas empezando, siempre y cuando no quieras hacer push de tus cambios locales.
* Por SSH: permite “proteger” el código para que nadie no autorizado haga un clone. Debemos crear claves en los ordenadores desde los que vamos a hacer clone y dar de alta esas claves en nuestro repositorio github, en la web. Este mecanismo es “seguro”, nadie puede clonar el repositorio si no le hemos dado permiso.

Aquí se muestra el cuadro que debemos seleccionar para permitir uno u otro método

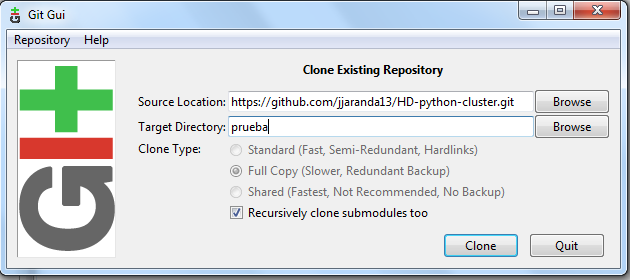


En cualquiera de los dos casos lo primero es crear un directorio con el nombre del proyecto y entrar en dicho directorio desde consola.

### Clonar por HTTP

Es lo más sencillo pero es posible que algunas funcionalidades estén limitadas. Por ejemplo, la mayoría de respositorios no permiten “hacer push” de los commits utilizando http. Es cierto que es posible realizar el “clone” por http y, posteriormente, configurar nuestro proyecto local para que, en lugar de utilizar http, utilice ssh a la hora de realizar un push. Sin embargo, si se clona el proyecto utilizando ssh desde el principio quedará ya configurado para todo y no nos tendremos que preocupar. Para clonar por http hacemos:

Desde Windows puedes usar el gui. Incluso te permite darle un directorio para que lo cree



Se coge la url que indica en la web del repositorio y se escribe en source location y listo. También lo puedes hacer desde línea de comando, en cuyo caso se hace igual que en linux

Desde Linux simplemente entramos en el directorio y

git clone URL

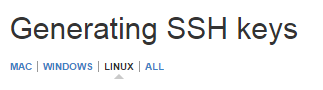
### Clonar por SSH

Necesitas seleccionar la URL SSH del repositorio pero antes debes crear una clave SSH siguiendo al pie de la letra estas instrucciones

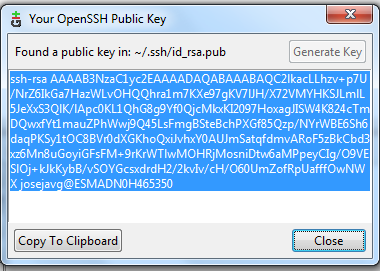
<https://help.github.com/articles/generating-ssh-keys/>

e instalarla en los settings de github, como veremos mas adelante.

Fíjate bien porque hay instrucciones para cada sistema operativo en esa web

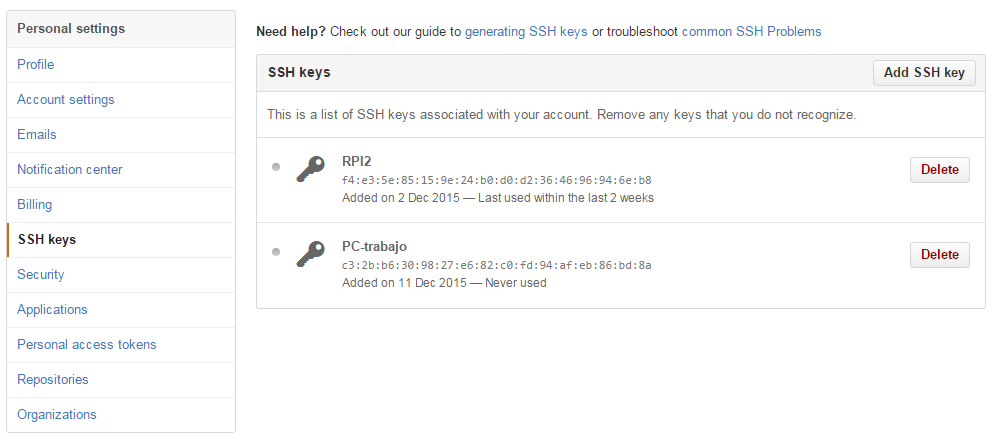


En el caso de Windows la clave se puede generar usando el GUI->help->generate key

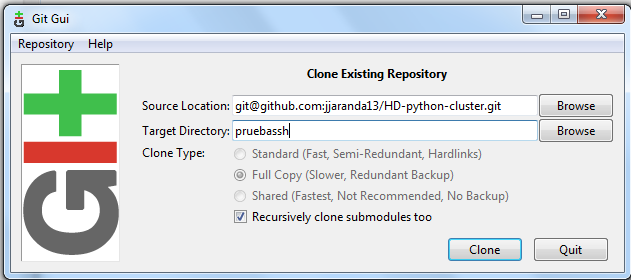


Una vez creada la clave ssh debe ser copiada en la web de github, ya que el servidor remoto debe conocer la clave publica que acabamos de crear, para ello vamos a la web de github y en nuestro perfil->settings podemos dar de alta tantas claves ssh como ordendadores desde los que vayamos a acceder.

Yo he creado dos, una que pertenece a un ordenador Linux Ubuntu “Raspberry Pi 2” (RPI2) y otra que pertenece al ordenador Windows del trabajo (PC-trabajo)



Y ahora ya si, podemos hacer clone



# Commit y push

En git todos los cambios que sufren los distintos ficheros de código fuente deben quedar registrados en un índice local (local a cada usuario, que se guarda en tu PC) llamado “staging index”. Si modificamos un fichero pero no añadimos ese cambio a la lista de cambios, no formará parte de los cambios realizados a la hora de subir los cambios a un repositorio central.

Git requiere que los cambios sean marcados explícitamente para el próximo commit. Por ejemplo, si haces un cambio en un archivo y quieres que ese cambio sea relevante para el próximo commit, es necesario que agregues el archivo a lo que se llama comúnmente "staging index" mediante el comando

Ejemplo:

git add archivo

git commit -m "comentario"

git push

Si solo hubieses hecho el commit, habrías creado una versión en tu PC pero si no la subes con “push” nadie puede obtenerla. Ojo: si haces push sin haber hecho commit, no subiras nada!!!

Cuando haces “commit”, estás protegiendo el software ante posibles modificaciones que finalmente quieras descartar. Siempre puedes revertir todos los cambios de un fichero concreto hasta el último commit así :

git checkout (esto te dice que ficheros han cambiado)

git checkout -- <filename> (esto revierte el fichero a su estado anterior (al del momento del ultimo commit)

Si no tienes claro que ficheros has cambiado (incluso cuales has borrado) puedes verlo con

git status

si has creado varios nuevos y quieres añadirlos todos a la vez, simplemente haz

git add \*

para eliminar un fichero no solo debes borrarlo. También debes hacer

git rm fichero

y además ir a github , navegar al fichero hacer clic en el y al visualizarlo, en la parte superior veras un icono de un cubo de basura. Deberás pulsarlo

por último, antes de hacer push, si tu mismo u otro usuario ha hecho cambios en el repositorio remoto, el push te fallará por no tener tu repositorio local coherente con el remoto.

Para que sea coherente, debes bajarte los cambios con

Git pull

Si te da error o te pide que especifiques el remote, puedes especificárselo asi (por ejemplo):

Git pull https://github.com/jjaranda13/8BP.git

Y después ya podrás subir con push tus nuevos cambios. Si al hacer pull machacas tus cambios pues mala suerte, lo que va a ocurrir es que git te va a mostrar un error y no te va a dejar hacer pull, a menos que borres los ficheros que has cambiado y que van a ser “machacados”.

Si borras en local una carpeta y después haces pull, la carpeta del repositorio origen no se te va a copiar, porque parece que git respeta estas cosas. Lo que hay que hacer es “git stash clear” para que git no “recuerde” que has borrado la carpeta y después “git pull” para que se la baje. Antes de hacer el git stash clear puedes hacer un git stash list para ver que la operación de borrado de la carpeta está ahí

Git stash list

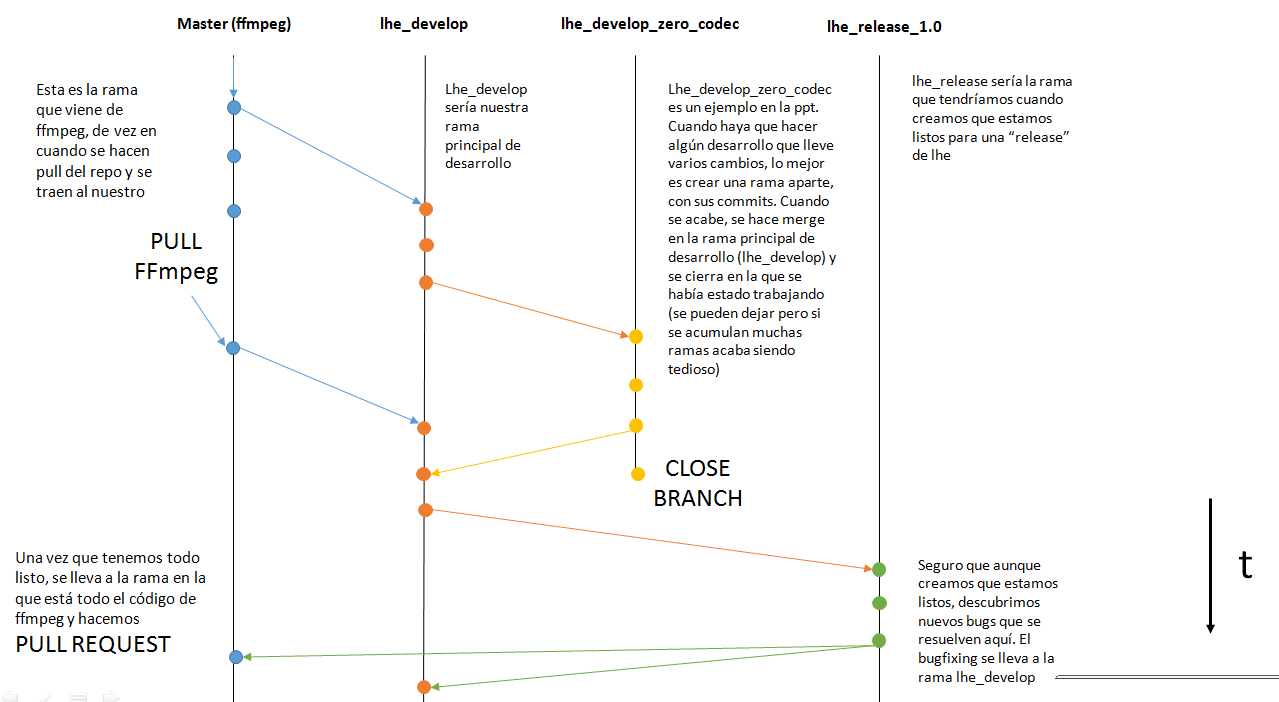
Git stash clear

Git pull

# Acciones avanzadas: ramas

Las ramas son un aspecto importante del modelo de programación en github. Puedes prescindir de ellas, aunque en ese caso estarás simplemente trabajando con la rama “master”.

Una rama es una “variante” del desarrollo. Las ramas a su vez pueden tener ramas



Cuando haces una nueva rama, git es muy eficiente y no copia todos los archivos en un nuevo directorio ni nada parecido. La rama es un concepto “lógico”. Git simplemente toma nota de qué ficheros han cambiado en esa rama, y de este modo almacena únicamente lo necesario.

Para saber en qué rama estás (asi como ver todas las ramas que hay):

git branch

git branch –a (Lista todas las ramas, incluidas las remotas)

Para crear y cambiar a una rama nueva:

git branch rama1

git checkout

O bien, simplemente:

git checkout -b rama1

Para cambiarse a una rama existente

git checkout rama1

## Ramas en el repositorio remoto

Ahora supongamos que has clonado un proyecto y quieres trabajar sobre una de sus ramas. Debes hacer un checkout pero debes especificar que quieres trabajar sobre una rama que esta creada en el repositorio remoto, que no ha sido creada en local

git checkout origin/rama1

Esto permite cambiarse a la rama origin/rama1 sin "bajárnosla" a nuestro ordenador. Siempre que lleve origin/ significa que es algo del remoto.

Posteriormente, cuando hagas un commit y aún no lo tengas subido a github, te dirá que estás "1  commit ahead origin/lhe\_develop\_sequential " lo que significa que tu rama local está 1 commit por delante de la rama remota. O sea, que tienes que subir los cambios

Si queremos trabajar en local lo lógico es crear una rama local basada en la rama remota

git checkout -b rama1 origin/rama1

Te crea una rama local llamada rama1 que "apunta" a la rama remota origin/rama1. Con esto se te bajaría ya la rama y trabajarías en ficheros locales que nadie podrá ver hasta que no hagas commit y push

## Has terminado: Merge

Supongamos que tu trabajo con la rama1 está ya completo y listo para fusionarlo (merge) con la rama master. Te sitúas en la rama master y haces el merge de la rama1. De este modo incorporas todos los cambios de la rama1 en la rama maestra

git checkout master

git merge rama1

El mismo procedimiento es aplicable a cualquier rama de una rama que no sea master

# Herramientas gráficas para Linux : gitg

Hay algunas herramientas que hacen uso de ella, como “gitg”, mostrando una intefaz visual e invocando masivamente a git para mostrar listas de cosas como veremos más adelante.

# APÉNDICE: algunos comandos

git help <comando>

Te muestra ayuda

git status

Lo primero que te muestra este comando es la rama en la que te encuentras.

Te muestra que ficheros nuevos han aparecido y cuales han cambiado desde el ultimo commit. Si quieres hacer un nuevo commit y quieres tener en cuenta esos cambios, tendras que añadir los ficheros nuevos que te diga status

git add

Añade nuevos ficheros para el commit

git branch

Te permite crear una rama

git checkout

Permite identificar que ficheros han cambiado desde el último commit.

Con la opción “--“ te permite revertir cambios en un fichero que quieres dejar como estaba justo en el anterior commit, descartando los cambios posteriores a dicho commit

Ejemplo:

Git checkout -- unfichero.c

Si pones un nombre de rama después del checkout, conmutas a dicha rama

Git checkout unarama

Git stash

Es como un git chechout -- de todos los ficheros que hayan cambiado pero permite recuperar todos los cambios realizados. Digamos que se guarda todos tus cambios y deja la rama como estaba desde el último commit. Posteriormente puedes hacer:

Git stash clear: borra la lista de cambios y los pierdes definitivamente

Git stash list: muestra todos los cambios

Git stash apply stash@{}

git merge

Para fusionar rama hija con su rama madre