

# Control de versiones y primeros pasos con Git

Luis Valencia Cabrera

Septiembre 2019

## Índice

1. Control de versiones	1
2. Servicios de gestión de repositorios en la nube	2
3. Trabajando con GitLab	2
3.1. Alta en GitLab . . . . .	3
3.2. Primer proyecto . . . . .	5
4. Primeros pasos con Git	7
4.1. Flujo de trabajo simplificado . . . . .	8

## 1. Control de versiones

Un aspecto crucial a la hora de empezar a trabajar en un proyecto, tanto de manera colaborativa como en solitario, es asegurarnos de que el trabajo que vamos desarrollando queda organizado de manera adecuada, no se pierde, se encuentra disponible por nosotros (convenientemente, de manera ubicua) y no se encuentra disponible para usuarios no autorizados.

Para ello, es más que aconsejable disponer de un sistema de control de versiones que asegure que podemos mantener la información que vamos generando, y que nos permita en un momento dado volver a una *foto anterior* del trabajo, por si hemos introducido cambios no deseados.

Ilustremos esto mediante el ejemplo que aparece en [esta excelente panorámica](#) (*overview*) acerca del **control de versiones**.

Hoy día, el sistema más extendido es **Git**, lo que hace muy probable que si trabajamos de manera colaborativa en grandes proyectos en empresas o instituciones debamos usar este sistema en nuestra labor diaria.

Se puede consultar mucha más información acerca del control de versiones y de Git en particular en este [libro gratuito](#), disponible online en español. Exploraremos los aspectos esenciales en apartados posteriores, pero por ir teniendo una idea podemos ver cómo usar **Git** en [este vídeo](#).

Como veremos, Git nos permite mantener el control de versiones de cualquier proyecto/carpeta de nuestro equipo. Sin embargo, lo más interesante es poder mantener este control de manera ubicua, independientemente de donde estemos trabajando cada día, y para eso es aconsejable hacer uso de servidores en la nube que nos proporcionen esos servicios. Dependiendo de la naturaleza educativa, comercial, personal, etc. del trabajo que vengamos desempeñando, necesitaremos hacer uso de repositorios privados, compartidos o públicos, como se detalla en el siguiente apartado.

## 2. Servicios de gestión de repositorios en la nube

Como decíamos, dependiendo del tipo de necesidad podremos disponer de recursos que nos permiten crear repositorios en la nube, bien privados, compartidos o directamente públicos.

Si queremos desarrollar contenidos para que queden automáticamente disponibles para cualquiera a través de Internet deberemos configurar un repositorio público, y ahí tradicionalmente la opción *ganadora* (digamos, la más popular) siempre ha sido GitHub.

Si por el contrario, lo que queremos es disponer de un número ilimitado de repositorios privados, la opción a escoger podría ser GitLab o Bitbucket, o al menos lo ha sido tradicionalmente.

Si junto con lo anterior queremos disponer de algunas funcionalidades básicas para la gestión de nuestros proyectos, así como una mayor disponibilidad de almacenamiento para cada proyecto, quizá la opción favorita sea GitLab.

Puede ver una útil comparativa de los servicios proporcionados por cada una en [este enlace](#).

## 3. Trabajando con GitLab

Se ha mencionado anteriormente Bitbucket. La cuenta en esta plataforma nos proporciona un servidor para gestionar nuestros repositorios y poder acceder a los mismos desde cualquier parte. No obstante, para otros aspectos relacionados como la gestión de tareas no nos ofrece gran cosa, debido a que otras aplicaciones de la misma compañía, como Jira o Confluence, cubren aspectos de gestión de proyectos y documentación.

Sin embargo, para el día a día de un desarrollo software, los productos anteriores conllevarían demasiada complicación. Como alternativa, para combinar el control de versiones basado en Git con una gestión de nuestras tareas pendientes para el desarrollo de nuestros proyectos disponemos de una alternativa sencilla pero que cubre éstas y otras muchas necesidades: GitLab. En la sección siguiente veremos más detalles sobre este aspecto, pero vamos a ver primero lo que nos ofrece desde el punto de vista del control de versiones.

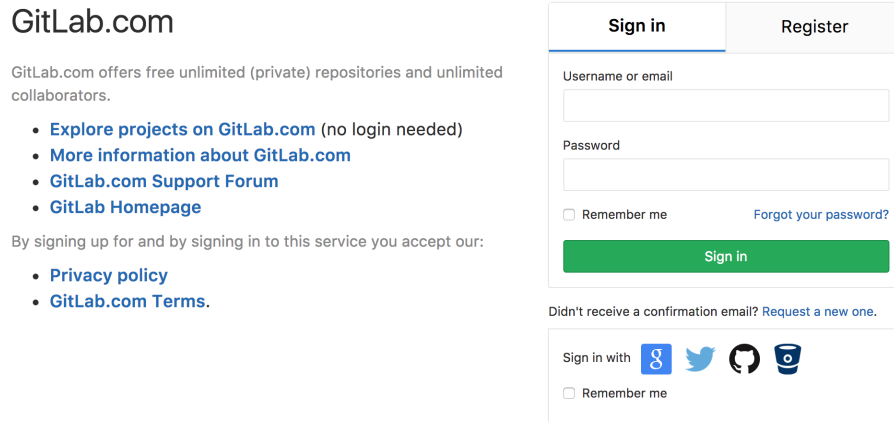
Las condiciones ofrecidas por la cuenta gratuita de GitLab son bastante ventajosas (ver [este enlace](#)): de acuerdo con su información, el plan gratuito de GitLab *has unlimited private repositories and unlimited collaborators on a project*.

Veamos cómo crearnos una cuenta con **GitLab**, y cómo crear un primer proyecto con su repositorio.

### 3.1. Alta en GitLab

Hay que seguir una serie de pasos:

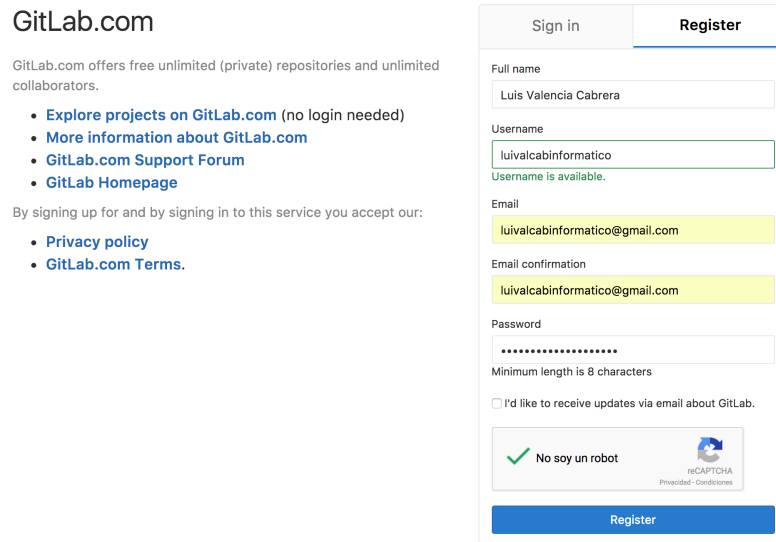
1. Acceder a **gitlab.com**:



The screenshot shows the GitLab.com homepage with a 'Sign in' and 'Register' tab. The 'Sign in' tab is active, showing fields for 'Username or email' and 'Password'. There are links for 'Remember me' and 'Forgot your password?'. A green 'Sign in' button is at the bottom. Below the button, there's a link to 'Request a new one.' if a confirmation email wasn't received. At the bottom, there are social login options for Google, Twitter, GitHub, and Docker, along with a 'Remember me' checkbox.

Figura 1: Pantalla de bienvenida

2. Registrar nuestro usuario:



The screenshot shows the GitLab.com registration form. The 'Register' tab is active. The form fields are: 'Full name' (Luis Valencia Cabrera), 'Username' (luivalcabinformatico, with a message 'Username is available.'), 'Email' (luivalcabinformatico@gmail.com), 'Email confirmation' (luivalcabinformatico@gmail.com), and 'Password' (masked with dots, with a note 'Minimum length is 8 characters'). There is a checkbox for 'I'd like to receive updates via email about GitLab.' and a reCAPTCHA widget with the text 'No soy un robot'. A blue 'Register' button is at the bottom.

Figura 2: Registro de usuario

### 3. Confirmar en nuestro correo:

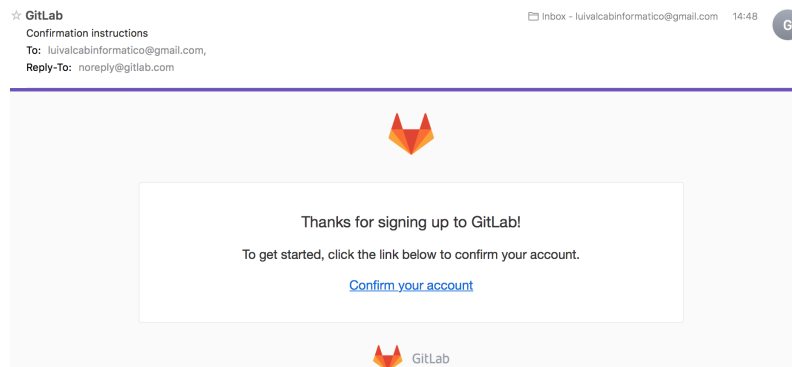


Figura 3: Verificación correo

### 4. Identificarnos (*login/sign up*) en el sistema:

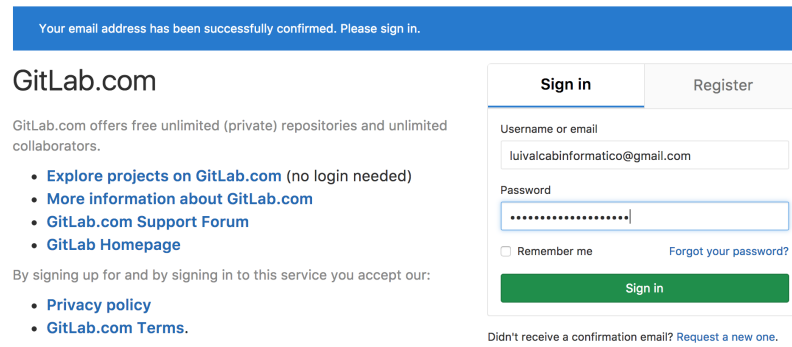


Figura 4: Sign up

Hemos llegado a la pantalla inicial:

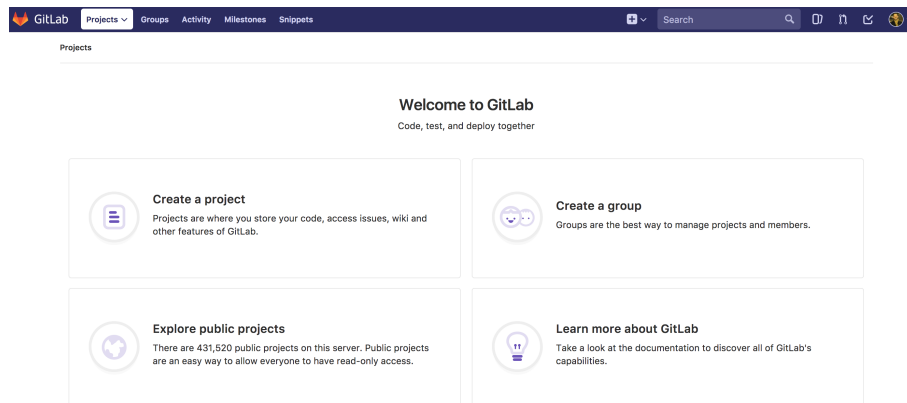


Figura 5: Menú principal

### 3.2. Primer proyecto

Desde la pantalla inicial una vez dentro de GitLab, debemos:

1. Hacer click en *Create a project*
2. Dar un nombre al proyecto:

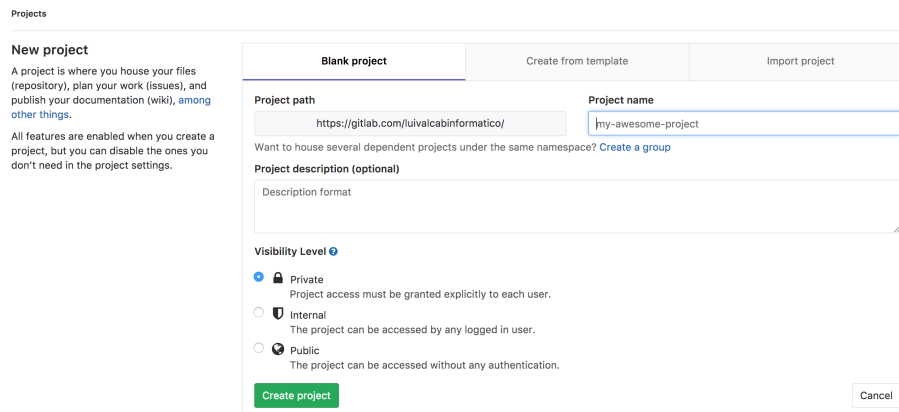


Figura 6: Nuevo proyecto

Esto nos lleva a la pantalla inicial del proyecto, donde se nos indica que se creó con éxito el mismo, vemos la dirección del repositorio, y unas instrucciones abajo indicando que el proyecto esté vacío, que debemos seguir las instrucciones de abajo o bien ir creando *online* archivo como **README.md**, que proporcionará la información general del proyecto.

3. Creación inicial de README.md:

El fichero queda guardado correctamente, como se nos muestra en la ventana de la figura 9.

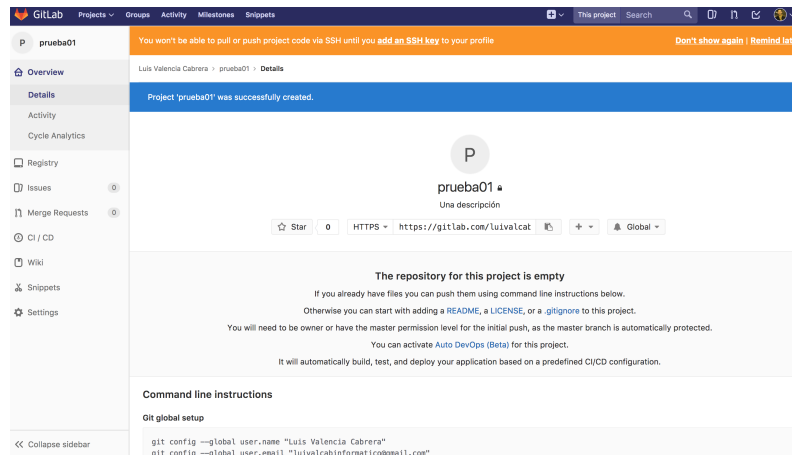


Figura 7: Creación de proyecto exitosa y pantalla inicial

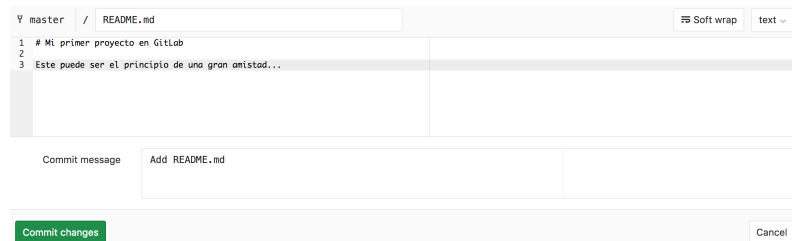


Figura 8: README.md

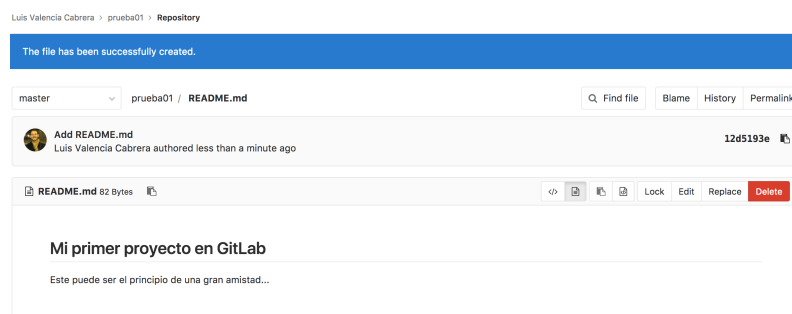


Figura 9: Fichero README.md listo

Ya podemos volver a la pantalla inicial del repositorio, seleccionar la opción HTTPS en el desplegable y copiar la dirección. Esta url deberemos usarla allá donde deseemos *clonar* el repositorio, por lo que podemos copiar en el icono señalado en la figura 10, y usar la url como se explicará a continuación.



Figura 10: URL del repositorio por HTTPS

## 4. Primeros pasos con Git

Para traer nuestro proyecto a nuestro equipo (o a cualquier otro) por primera vez, debemos hacer uso de la URL anterior de nuestro proyecto en el servidor, y en algún terminal como `Git Bash` (o una interfaz que nos dote de alguna facilidad visual adicional, aunque haciendo lo mismo) hace el enlace correspondiente mediante un primer comando de `Git`: `git clone`. La sintaxis será: `git clone URL`, donde URL será la que hemos copiado de GitLab. Si es la primera operación que hacemos sobre nuestra cuenta de GitLab en el equipo que estamos usando, nos solicitará nuestras credenciales de la plataforma (usuario y contraseña).

Para el resto de operaciones para el control de las sucesivas versiones de nuestros archivos, escribiremos el comando `git`, seguido del tipo de operación que corresponda, como iremos explicando a continuación.

Conforme vayamos trabajando en el proyecto, nuestros archivos estarán siendo modificados con respecto a la versión que había en el servidor (que, recordemos, únicamente tiene la versión inicial del fichero `README.md`). Una vez decidamos empezar a consolidar nuestros archivos mediante el control de versiones, conviene que tengamos presente el modelo de almacenamiento de `Git`, que vemos en la figura 11.

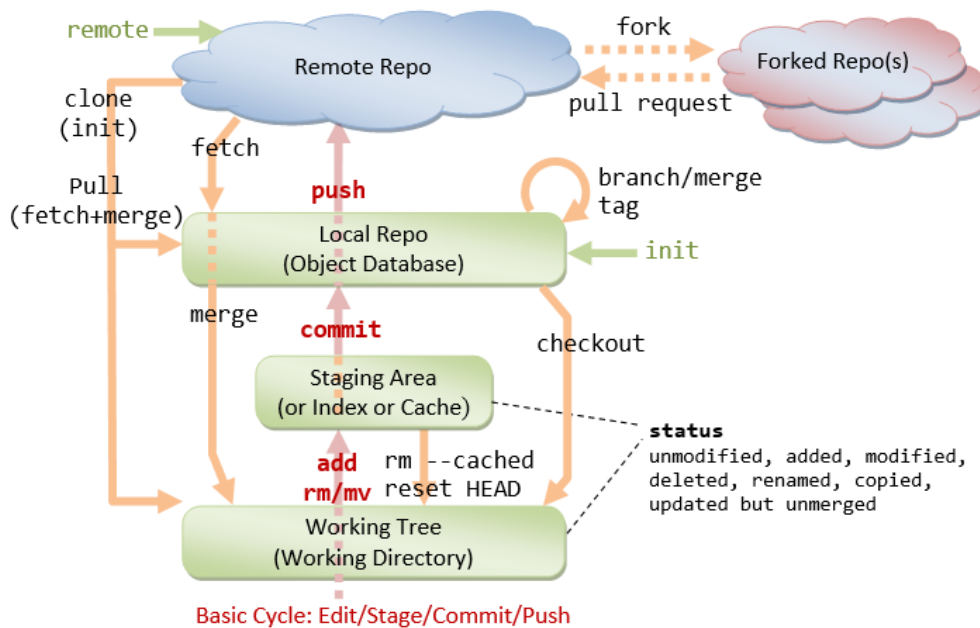


Figura 11: Modelo de almacenamiento de Git

En primer lugar, prestemos atención a los distintos niveles o áreas en que podemos encontrar los archivos:

- Área de trabajo: archivos de nuestro proyecto en los que estamos trabajando, que vamos editando, etc., tanto si nunca se han consolidado en el repositorio Git (*untracked*), como si ya existe una versión en el repositorio (*tracked*). Estos archivos podrían estar modificados (*modified*) o bien permanecer idénticos a la versión disponible en el repositorio (*unmodified*). Para pasar un archivo del

área de trabajo a la siguiente, se debe emplear el comando **git add** (el camino inverso se emprendería mediante **git rm**).

- Área de faseado (*staging area*): archivos que hemos decidido *pre-consolidar* en un estado intermedio. La próxima acción de consolidación (**git commit**) incorporará todos los archivos de este área a la versión generada. Esta acción generará una nueva versión del proyecto en nuestro **repositorio local**.
- Repositorio local: contiene todas las versiones ya consolidadas en nuestro equipo local. Si estamos conectados a un servidor remoto, el repositorio local tendrá todas las versiones de los archivos del servidor anteriores a las instrucciones de tipo **git clone**, **git fetch** o **git pull** que hayamos ejecutado en nuestro equipo (**git fetch** trae del servidor todo el contenido del repositorio, a un espacio oculto, y **git pull** hace lo mismo, pero una vez descargados esos datos los consolida en el repositorio local mediante **git merge**). Para trasladar nuestras adiciones y modificaciones locales al servidor remoto (aquellas que se encuentran consolidadas mediante *commits* en el servidor local) se emplea el comando **git push**.
- Servidor remoto: contiene todas las versiones de todos los archivos que han sido recibidos (mediante **git push**) de los servidores locales de los distintos usuarios autorizados. Téngase presente que cualquier repositorio local de dichos usuarios contendrá, asumiendo que van haciendo los correspondientes *pulls*, la misma información que el servidor remoto.

Como comentamos al inicio del capítulo, podemos ver mucha más información sobre Git con todo detalle en este [libro gratuito](#).

Un interesante resumen a modo de *crash course* sobre Git se puede encontrar en [este enlace](#).

## 4.1. Flujo de trabajo simplificado

La potencia de Git a la hora de trabajar en proyectos colaborativos es más que considerable, pero esto conlleva también un crecimiento en su complejidad. No obstante, para el caso que nos ocupa lo único que necesitamos para trabajar con Git en RStudio es asimilar un flujo de trabajo sencillo en su caso general, para el trabajo diario:

- Nos traemos la versión disponible en el repositorio remoto (**git clone** la primera vez, **git pull** las restantes).
- Editamos todo lo que necesitemos en nuestro equipo, ajenos a Git.
- Añadimos al *staging area* los ficheros nuevos o modificados que queramos consolidar, mediante **git add**.



- Consolidamos en el repositorio local mediante **git commit**. Si no hemos trabajado anteriormente en el equipo actual, antes de poder consolidar tenemos que indicar quiénes somos, mediante:

- **git config --global user.name "mi nombre"**
- **git config --global user.email "micorreo@miservidor.ext"**

Una explicación más detallada de las opciones que permite esta configuración previa podemos encontrarla en [este enlace](#).

- Actualizamos la última versión consolidada en el servidor, mediante **git push**.

Sirva la figura 12 para mostrar de forma también simplificada el flujo de los ficheros locales.

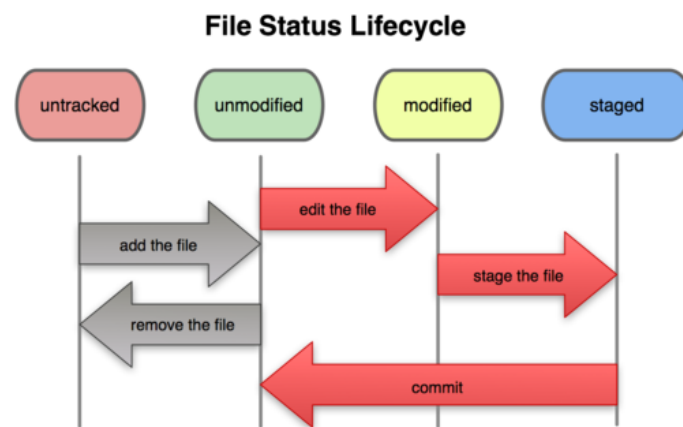


Figura 12: Estados ficheros Git simplificado

## Bibliografía

Se lista a continuación una serie de recursos bibliográficos de interés:

- [Git en 2 minutos](#), codigofacilito (video de Youtube).
- *Pro Git, el libro oficial de Git*, Scott Chacon (traducido por Juan Murua, Alejandro Fernández, Javier Eguiluz), libro online gratuito **en español**, muy completo.
- *Version Control with Git*, curso online abierto, con prácticas guiadas. Incluye sesiones cortas interesantes como:
  - [Automated Version Control](#), explicando la idea general de Git.
  - [Setting Up Git](#), sobre opciones de configuración de Git.
- [GitHub vs. Bitbucket vs. GitLab vs. Coding](#), entrada de blog.
- [Condiciones de las cuentas GitLab](#).