# **Git en el Servidor**

# Los Protocolos:

# Git puede usar cuatro protocolos principales para transferir datos: Local, HTTP, Secure Shell (SSH) y Git. Quer son

### **1.Local Protocol**

El más básico es el Protocolo Local, donde el repositorio remoto es simplemente otra carpeta en el disco.

Para clonar un repositorio como estos, o para añadirlo como remoto a un proyecto ya existente, usa el camino (path) del repositorio como su URL. Por ejemplo, para clonar un repositorio local, puedes usar algo como:

$ git clone /opt/git/project.git

O como:

$ git clone file:///opt/git/project.git

Para añadir un repositorio local a un proyecto Git existente, puedes usar algo como:

$ git remote add local\_proj /opt/git/project.g

Con lo que podrás enviar (push) y recibir (pull) desde dicho remoto exactamente de la misma forma a como lo harías a través de una red.

#### **Ventajas:**

Las ventajas de los repositorios basados en carpetas y archivos, son su simplicidad y el aprovechamiento de los permisos preexistentes de acceso. S

#### **Desventajas:**

La principal desventaja de los repositorios basados en carpetas y archivos es su dificultad de acceso desde distintas ubicaciones. Por ejemplo, si quieres enviar (push) desde tu portátil cuando estás en casa, primero tienes que montar el disco remoto; lo cual puede ser difícil y lento, en comparación con un acceso basado en red.

### **2.Protocolos HTTP**

ipuede utilizar el protocolo HTTP de dos maneras. Antes de la versión 1.6.6 de Git, solo había una forma de utilizar el protocolo HTTP y normalmente en sólo lectura. Con la llegada de la versión 1.6.6 se introdujo un nuevo protocolo más inteligente que involucra a Git para negociar la transferencia de datos de una manera similar a como se hace con SSH.

#### **HTTP Inteligente**

El protocolo HTTP “Inteligente” funciona de forma muy similar a los protocolos SSH y Git, pero se ejecuta sobre puertos estándar HTTP/S y puede utilizar los diferentes mecanismos de autenticación HTTP. Esto significa que puede resultar más fácil para los usuarios, puesto que se pueden identificar mediante usuario y contraseña (usando la autenticación básica de HTTP) en lugar de usar claves SSH.

#### **HTTP Tonto**

Este protocolo espera obtener el repositorio Git a través de un servidor web como si accediera a archivos normales. La simplicidad para configurarlo. Básicamente, todo lo que tenemos que hacer es poner el repositorio Git bajo el directorio raíz de documentos HTTP y especificar un punto de enganche (hook) de post-update, para permitir acceso de lectura con HTTP, debes hacer algo similar a lo siguiente:

$ cd /var/www/htdocs/

$ git clone --bare /path/to/git\_project gitproject.git

$ cd gitproject.git

$ mv hooks/post-update.sample hooks/post-update

$ chmod a+x hooks/post-update

Este comando se ejecuta cuando se envían cambios (push) al repositorio (mediante SSH, por ejemplo); luego, otras personas pueden clonar mediante algo como:

$ git clone https://example.com/gitproject.git

#### **Ventajas**

#### La simplicidad de tener una única URL para todos los tipos de acceso y que el servidor pida autenticación sólo cuando se necesite, hace las cosas muy fáciles para el usuario final. Permitir autenticar mediante usuario y contraseña es también una ventaja sobre SSH, ya que los usuarios no tendrán que generar sus claves SSH y subir la pública al servidor antes de comenzar a usarlo. También se pueden servir los repositorios en sólo lectura con HTTPS, lo que significa que se puede cifrar la transferencia de datos; incluso se puede identificar a los clientes haciéndoles usar certificados convenientemente firmado.

#### **Inconvenientes**

Git sobre HTTP/S puede ser un poco más complejo de configurar comparado con el SSH en algunos sitios. En otros casos, se adivina poca ventaja sobre el uso de otros protocolos.

### **3.El Protocolo SSH**

SSH es un protocolo muy habitual para alojar repositorios Git en hostings privados. Esto es así porque el acceso SSH viene habilitado de forma predeterminada en la mayoría de los servidores, y si no es así, es fácil habilitarlo. Además, SSH es un protocolo de red autenticado sencillo de utilizar.

Para clonar un repositorio a través de SSH, puedes indicar una URL ssh:// tal como:

$ git clone ssh://user@server/project.git

También puedes usar la sintaxis estilo scp del protocolo SSH:

$ git clone user@server:project.git

Pudiendo asimismo prescindir del usuario; en cuyo caso Git asume el usuario con el que estés conectado en ese momento.

#### **Ventajas**

El uso de SSH tiene múltiples ventajas. En primer lugar, SSH es relativamente fácil de configurar: los “demonios” (daemons) SSH son de uso común, muchos administradores de red tienen experiencia con ellos y muchas distribuciones del SO los traen predefinidos o tienen herramientas para gestionarlos. Además, el acceso a través de SSH es seguro, estando todas las transferencias encriptadas y autentificadas.

#### **Desventajas**

El aspecto negativo de SSH es su imposibilidad para dar acceso anónimo al repositorio. Todos han de tener configurado un acceso SSH al servidor, incluso aunque sea con permisos de solo lectura; lo que no lo hace recomendable para soportar proyectos abiertos. Si lo usas únicamente dentro de tu red corporativa, posiblemente sea SSH el único protocolo que tengas que emplear.

### **4.El protocolo Git**

El protocolo Git es un “demonio” (daemon) especial, que viene incorporado con Git. Escucha por un puerto dedicado (9418) y nos da un servicio similar al del protocolo SSH; pero sin ningún tipo de autentificación.

#### **Ventajas**

El protocolo Git es el más rápido de todos los disponibles. Si has de servir mucho tráfico de un proyecto público o servir un proyecto muy grande, que no requiera autentificación para leer de él, un “demonio” Git es la respuesta. Utiliza los mismos mecanismos de transmisión de datos que el protocolo SSH, pero sin la sobrecarga de la encriptación ni de la autentificación.

#### **Desventajas**

El principal problema del protocolo Git, es su falta de autentificación. No es recomendable tenerlo como único protocolo de acceso a tus proyectos. Habitualmente, lo combinarás con un acceso SSH o HTTPS para los pocos desarrolladores con acceso de escritura que envíen (push) material, dejando el protocolo **git://** para los accesos solo-lectura del resto de personas.

## **Configurando Git en un servidor**

Aquí demostraremos los comandos y pasos necesarios para hacer las instalaciones básicas simplificadas en un servidor basado en Linux, aunque también es posible ejecutar estos servicios en los servidores Mac o Windows.

Para configurar por primera vez un servidor de Git, hay que exportar un repositorio existente en un nuevo repositorio vacío - un repositorio que no contiene un directorio de trabajo.

$ git clone --bare my\_project my\_project.git

Cloning into bare repository 'my\_project.git'...

done.

Deberías tener ahora una copia de los datos del directorio Git en tu directorio my\_project.git. Esto es más o menos equivalente a algo así:

$ cp -Rf my\_project/.git my\_project.git

### **Colocando un Repositorio Vacío en un Servidor**

### Ahora que tienes una copia vacía de tú repositorio, todo lo que necesitas hacer es ponerlo en un servidor y establecer sus protocolos.

$ scp -r my\_project.git user@git.example.com:/opt/git

En este punto, otros usuarios con acceso SSH al mismo servidor que tiene permisos de lectura-acceso al directorio / opt / git pueden clonar tu repositorio mediante el comando

$ git clone user@git.example.com:/opt/git/my\_project.git

Si un usuario accede por medio de SSH a un servidor y tiene permisos de escritura en el directorio git my\_project.git/ opt / /, automáticamente también tendrá acceso push.

Git automáticamente agrega permisos de grupo para la escritura en un repositorio apropiadamente si se ejecuta el comando git init con la opción` --shared`.

$ ssh user@git.example.com

$ cd /opt/git/my\_project.git

$ git init --bare –shared

### **Pequeñas configuraciones**

Si tienes un pequeño equipo o acabas de probar Git en tu organización y tienes sólo unos pocos desarrolladores, las cosas pueden ser simples para ti.

#### **Acceso SSH**

Si tienes un servidor al que todos los desarrolladores ya tienen acceso SSH, es generalmente más fácil de configurar el primer repositorio allí, porque no hay que hacer casi ningún trabajo.

## **Generando tu clave pública SSH**

Tal y como se ha comentado, muchos servidores Git utilizan la autentificación a través de claves públicas SSH. Y, para ello, cada usuario del sistema ha de generarse una, si es que ya no la tiene.

$ cd ~/.ssh

$ ls

authorized\_keys2 id\_dsa known\_hosts

config id\_dsa.pub

El archivo terminado en**.pub** es tu clave pública, y el otro archivo es tu clave privada. Si no tienes esos archivos (o no tienes ni siquiera la carpeta **.ssh**), has de crearlos; utilizando un programa llamado **ssh-keygen**, que viene incluido en el paquete SSH de los sistemas Linux/Mac o en el paquete MSysGit en los sistemas Windows:

$ ssh-keygen

Generating public/private rsa key pair.

Enter file in which to save the key (/home/schacon/.ssh/id\_rsa):

Created directory '/home/schacon/.ssh'.

Enter passphrase (empty for no passphrase):

Enter same passphrase again:

Your identification has been saved in /home/schacon/.ssh/id\_rsa.

Your public key has been saved in /home/schacon/.ssh/id\_rsa.pub.

The key fingerprint is:

d0:82:24:8e:d7:f1:bb:9b:33:53:96:93:49:da:9b:e3 [schacon@mylaptop.local](mailto:schacon@mylaptop.local)

Tras generarla, cada usuario ha de encargarse de enviar su clave pública a quienquiera que administre el servidor Git (en el caso de que éste esté configurado con SSH y así lo requiera). Esto se puede realizar simplemente copiando los contenidos del archivo terminado en **.pub** y enviándoselos por correo electrónico. La clave pública será una serie de números, letras y signos, algo así como esto:

$ cat ~/.ssh/id\_rsa.pub

ssh-rsa AAAAB3NzaC1yc2EAAAABIwAAAQEAklOUpkDHrfHY17SbrmTIpNLTGK9Tjom/BWDSU

GPl+nafzlHDTYW7hdI4yZ5ew18JH4JW9jbhUFrviQzM7xlELEVf4h9lFX5QVkbPppSwg0cda3

Pbv7kOdJ/MTyBlWXFCR+HAo3FXRitBqxiX1nKhXpHAZsMciLq8V6RjsNAQwdsdMFvSlVK/7XA

t3FaoJoAsncM1Q9x5+3V0Ww68/eIFmb1zuUFljQJKprrX88XypNDvjYNby6vw/Pb0rwert/En

mZ+AW4OZPnTPI89ZPmVMLuayrD2cE86Z/il8b+gw3r3+1nKatmIkjn2so1d01QraTlMqVSsbx

NrRFi9wrf+M7Q== schacon@mylaptop.local

## **Configurando el servidor**

amos a avanzar en los ajustes de los accesos SSH en el lado del servidor. En este ejemplo, usarás el método de las authorized\_keys(claves autorizadas) para autentificara tus usuarios.

$ sudo adduser git

$ su git

$ cd

$ mkdir .ssh && chmod 700 .ssh

$ touch .ssh/authorized\_keys && chmod 600 .ssh/authorized\_keys

Y a continuación añades las claves públicas de los desarrolladores al archivo authorized\_keys del usuario git que has creado.

$ cat /tmp/id\_rsa.john.pub

ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQCB007n/ww+ouN4gSLKssMxXnBOvf9LGt4L

ojG6rs6hPB09j9R/T17/x4lhJA0F3FR1rP6kYBRsWj2aThGw6HXLm9/5zytK6Ztg3RPKK+4k

Yjh6541NYsnEAZuXz0jTTyAUfrtU3Z5E003C4oxOj6H0rfIF1kKI9MAQLMdpGW1GYEIgS9Ez

Sdfd8AcCIicTDWbqLAcU4UpkaX8KyGlLwsNuuGztobF8m72ALC/nLF6JLtPofwFBlgc+myiv

O7TCUSBdLQlgMVOFq1I2uPWQOkOWQAHukEOmfjy2jctxSDBQ220ymjaNsHT4kgtZg2AYYgPq

dAv8JggJICUvax2T9va5 gsg-keypair

No tienes más que añadirlas al archivo authorized\_keys dentro del directorio .ssh:

$ cat /tmp/id\_rsa.john.pub >> ~/.ssh/authorized\_keys

$ cat /tmp/id\_rsa.josie.pub >> ~/.ssh/authorized\_keys

$ cat /tmp/id\_rsa.jessica.pub >> ~/.ssh/authorized\_keys

Tras esto, puedes preparar un repositorio básico vacío para ellos, usando el comando git init con la opción--barepara inicializar el repositorio sin carpeta de trabajo:

$ cd /opt/git

$ mkdir project.git

$ cd project.git

$ git init --bare

Initialized empty Git repository in /opt/git/project.git/

Para una mayor protección, puedes restringir fácilmente el usuario git a realizar solamente actividades relacionadas con Git, utilizando un shell limitado llamado git-shell que viene incluido en Git.

$ cat /etc/shells # mirar si `git-shell` ya está aquí. Si no...

$ which git-shell # buscar `git-shell` en nuestro sistema

$ sudo vim /etc/shells # y añadirlo al final de este archivo con el camino (path) completo

Ahora ya puedes cambiar la shell del usuario utilizando chsh <username>:

$ sudo chsh git # poner aquí la nueva shell, normalmente será: /usr/bin/git-shell

De esta forma dejamos al usuario **git** limitado a utilizar la conexión SSH solamente para enviar (push) y recibir (pull) repositorios, sin posibilidad de iniciar una sesión normal en el servidor. Si pruebas a hacerlo, recibirás un rechazo de inicio de sesión:

$ ssh git@gitserver

fatal: Interactive git shell is not enabled.

hint: ~/git-shell-commands should exist and have read and execute access.

Connection to gitserver closed.

Los comandos remotos de Git funcionarán con normalidad, pero los usuarios no podrán obtener un intérprete de comandos del sistema. Tal como nos avisa, también se puede establecer un directorio llamado git-shell-commands en la cuenta del usuario git para personalizar un poco el git-shell.

## **El demonio Git**

Ahora vamos a configurar un “demonio” sirviendo repositorios mediante el protocolo “Git”. Es la forma más común para dar acceso anónimo, pero rápido, a los repositorios.

El protocolo Git es relativamente fácil de configurar. Básicamente, necesitas ejecutar el comando con la variante “demonio” (daemon):

git daemon --reuseaddr --base-path=/opt/git/ /opt/git/

Este proceso se puede demonizar de diferentes maneras, dependiendo del sistema operativo con el que trabajas. En una máquina Ubuntu, puedes usar un script de arranque. Poniendo en el siguiente archivo:

/etc/event.d/local-git-daemon

un script tal como:

start on startup

stop on shutdown

exec /usr/bin/git daemon \

--user=git --group=git \

--reuseaddr \

--base-path=/opt/git/ \

/opt/git/

respawn

Tras reiniciar tu máquina, el “demonio” Git arrancará automáticamente y se reiniciará cuando se caiga. Para arrancarlo sin necesidad de reiniciar la máquina, puedes utilizar el comando:

initctl start local-git-daemon

A continuación, has de indicar a Git a cuales de tus repositorios ha de permitir acceso sin autentificar. Lo puedes hacer creando en cada repositorio un archivo llamado git-daemon-export-ok.

$ cd /path/to/project.git

$ touch git-daemon-export-ok

## **HTTP Inteligente**

Ahora ya tenemos acceso autentificado mediante SSH y anónimo mediante git://, pero hay también otro protocolo que permite tener ambos accesos a la vez.

Revisemos una configuración básica. Pondremos Apache como servidor de CGI. Si no tienes Apache configurado, lo puedes instalar en un Linux con un comando similar a este:

$ sudo apt-get install apache2 apache2-utils

$ a2enmod cgi alias env

A continuación tenemos que añadir algunas cosas a la configuración de Apache para que se utilice git-http-backend para cualquier cosa que haya bajo la carpeta virtual /git.

SetEnv GIT\_PROJECT\_ROOT /opt/git

SetEnv GIT\_HTTP\_EXPORT\_ALL

ScriptAlias /git/ /usr/libexec/git-core/git-http-backend/

Ahora tienes que decirle a Apache que acepte peticiones en esta ruta con algo similar a esto:

<Directory "/usr/lib/git-core\*">

Options ExecCGI Indexes

Order allow,deny

Allow from all

Require all granted

</Directory>

Finalmente, si quieres que los clientes autentificados tengan acceso de escritura, tendrás que crear un bloque Auth similar a este:

<LocationMatch "^/git/.\*/git-receive-pack$">

AuthType Basic

AuthName "Git Access"

AuthUserFile /opt/git/.htpasswd

Require valid-user

</LocationMatch>

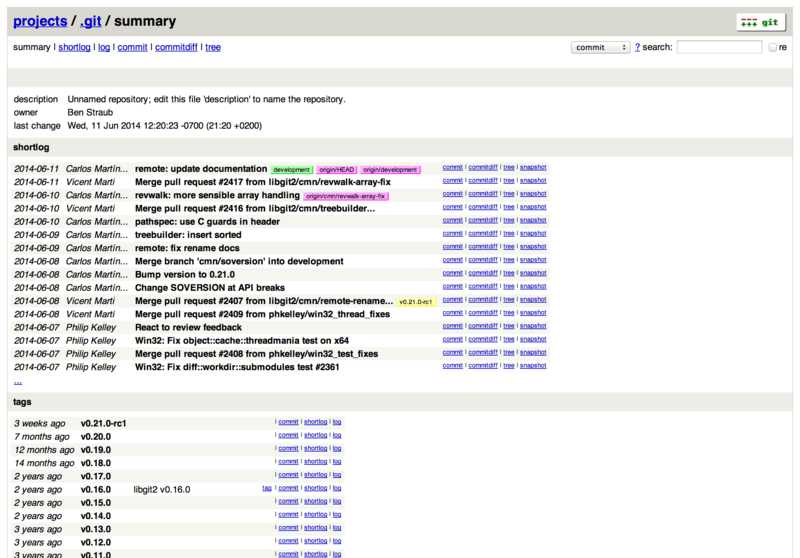
Esto requiere que hagas un archivo .htaccess que contenga las contraseñas cifradas de todos los usuarios válidos. Por ejemplo, para añadir el usuario “schacon” a este archivo:

$ htdigest -c /opt/git/.htpasswd "Git Access" schacon

No queremos profundizar en los detalles de la configuración de Apache, ya que puedes tener diferentes necesidades de autentificación o querer utilizar un servidor diferente.

## **GitWeb**

Ahora que ya tienes acceso básico de lectura/escritura y de solo-lectura a tu proyecto, puedes querer instalar un visualizador web. Git trae un script CGI, denominado GitWeb, que es el que usaremos para este propósito.



Si quieres comprobar cómo podría quedar GitWeb con tu proyecto, Git dispone de un comando para activar una instancia temporal, si en tu sistema tienes un servidor web ligero, como por ejemplo lighttpd o webrick.

$ git instaweb --httpd=webrick

[2009-02-21 10:02:21] INFO WEBrick 1.3.1

[2009-02-21 10:02:21] INFO ruby 1.8.6 (2008-03-03) [universal-darwin9.0]

Esto arranca un servidor HTTPD en el puerto 1234, y luego arranca un navegador que abre esa página. Es realmente sencillo. Cuando ya hayas terminado y quieras apagar el servidor, puedes lanzar el mismo comando con la opción --stop:

$ git instaweb --httpd=webrick –stop

Enseguida vamos a revisar el proceso de instalar GitWeb manualmente. Primero, necesitas el código fuente de Git, que viene con GitWeb, para generar un script CGI personalizado:

$ git clone git://git.kernel.org/pub/scm/git/git.git

$ cd git/

$ make GITWEB\_PROJECTROOT="/opt/git" prefix=/usr gitweb

SUBDIR gitweb

SUBDIR ../

make[2]: `GIT-VERSION-FILE' is up to date.

GEN gitweb.cgi

GEN static/gitweb.js

$ sudo cp -Rf gitweb /var/www/

A continuación, tienes que preparar Apache para que utilice dicho script. Para ello, puedes añadir un VirtualHost:

<VirtualHost \*:80>

ServerName gitserver

DocumentRoot /var/www/gitweb

<Directory /var/www/gitweb>

Options ExecCGI +FollowSymLinks +SymLinksIfOwnerMatch

AllowOverride All

order allow,deny

Allow from all

AddHandler cgi-script cgi

DirectoryIndex gitweb.cgi

</Directory>

</VirtualHost>

## **GitLab**

GitWeb es muy simple. Si buscas un servidor Git más moderno, con todas las funciones, tienes algunas soluciones de código abierto que puedes utilizar en su lugar. Puesto que GitLab es una de las más populares, vamos a ver aquí cómo se instala y se usa, a modo de ejemplo. Es algo más complejo que GitWeb y requiere algo más de mantenimiento, pero es una opción con muchas más funciones.

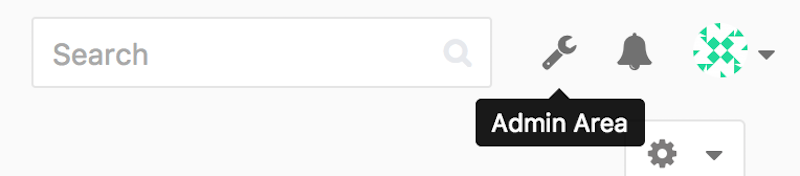
### **Instalación**

GitLab es una aplicación web con base de datos, por lo que su instalación es algo más complicada. Por suerte, es un proceso muy bien documentado y soportado.



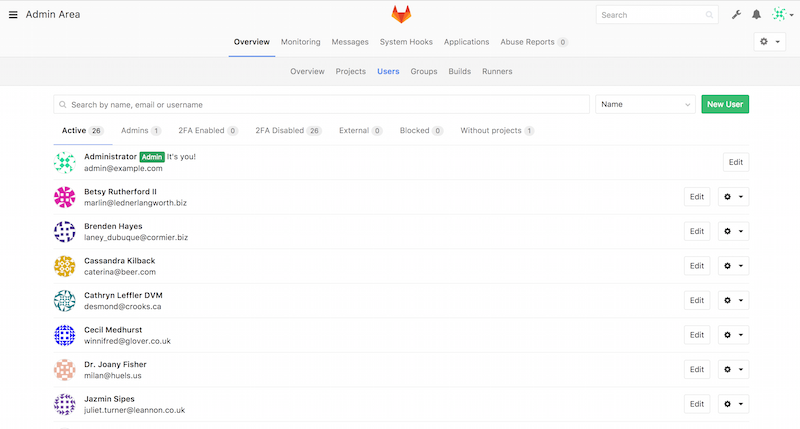
### Administración

La interfaz de administración de GitLab se accede mediante la web. Simplemente abre en tu navegador la IP o el nombre de máquina donde has instalado Gitlab, y entra con el usuario administrador.



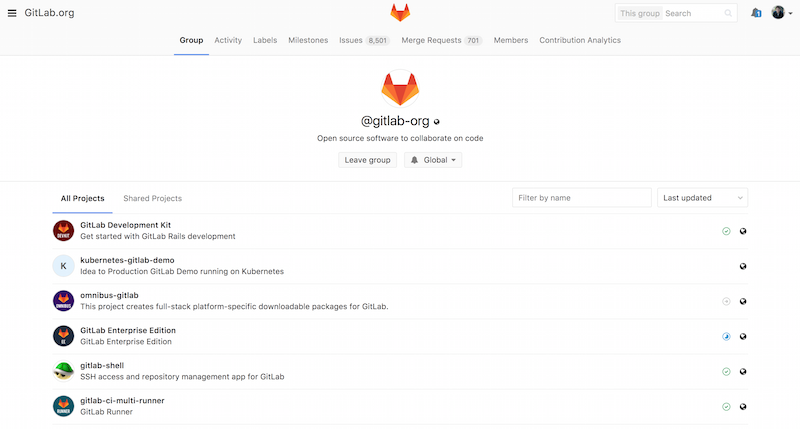
#### **Usuarios**

Los usuarios en Gitlab son las cuentas que abre la gente. Las cuentas de usuario no tienen ninguna complicación: viene a ser una colección de información personal unida a la información de login.



#### **Grupos**

Un grupo de GitLab es un conjunto de proyectos, junto con los datos acerca de los usuarios que tienen acceso. Cada grupo tiene también un espacio de nombres específico.



#### **Proyectos**

Un proyecto en GitLab corresponde con un repositorio Git. Cada proyecto pertenece a un espacio de nombres, bien sea de usuario o de grupo.

#### **Enganches (hooks)**

GitLab tiene soporte para los enganches (hooks), tanto a nivel de proyecto como del sistema. Para cualquiera de ellos, el servidor GitLab realizará una petición HTTP POST con determinados datos JSON cuando ocurran ciertos eventos.

### **Uso básico**

Lo primero que tienes que hacer en GitLab es crear un nuevo proyecto. Esto lo consigues pulsando el icono “+” en la barra superior. Te preguntará por el nombre del proyecto, el espacio de nombres al que pertenece y qué nivel de visibilidad debe tener.

Para un repositorio local existente, puedes crear un remoto llamado gitlab del siguiente modo:

$ git remote add gitlab https://server/namespace/project.git

Si no tienes copia local del repositorio, puedes hacer esto:

$ git clone <https://server/namespace/project.git>

### **Trabajando con GitLab**

Para trabajar en un proyecto GitLab lo más simple es tener acceso de escritura (push) sobre el repositorio git.

## **Git en un alojamiento externo**

Si no quieres realizar todo el trabajo que implica poner en marcha tu propio servidor Git, tienes varias opciones para alojar tus proyectos Git en un sitio externo dedicado. Esto tiene varias ventajas: normalmente en los alojamientos externos es fácil configurar y comenzar proyectos sin preocuparse del mantenimiento del servidor o de su monitorización. Aunque pongas en marcha tu propio servidor internamente, probablemente quieras usar un sitio público para tu código abierto. Será más fácil que la comunidad de software libre encuentre tu proyecto y colabore.