

Tutorial de Git. Manual básico con ejemplos

por Diego C. Martín, extraído de <https://www.diegocmartin.com/tutorial-git/>

INDICE DE CONTENIDOS:

- 1 Conceptos básicos de Git
- 2 Workflow de Git
- 3 Ayuda de Git
- 4 Configuración de Git
- 5 Comenzar a trabajar con Git
 - 5.1 Trabajar en un nuevo proyecto con Git
 - 5.2 Trabajar en un proyecto existente con Git
 - 5.3 Primer commit
- 6 Trabajando con Git en local
 - 6.1 Histórico de operaciones en Git. El log
 - 6.2 Eliminar archivos del repositorio de Git
 - 6.3 Mover archivos en Git
 - 6.4 Ignorar archivos en Git
- 7 Trabajando con Git en Remoto
 - 7.1 Configuración de una clave para la conexión por SSH y autenticación en GitHub
 - 7.2 Trabajando con un repositorio remoto en GitHub
- 8 Recursos

Conceptos básicos de Git

Git se basa en snapshots (instantáneas) del código en un estado determinado, que viene dado por el autor y la fecha.

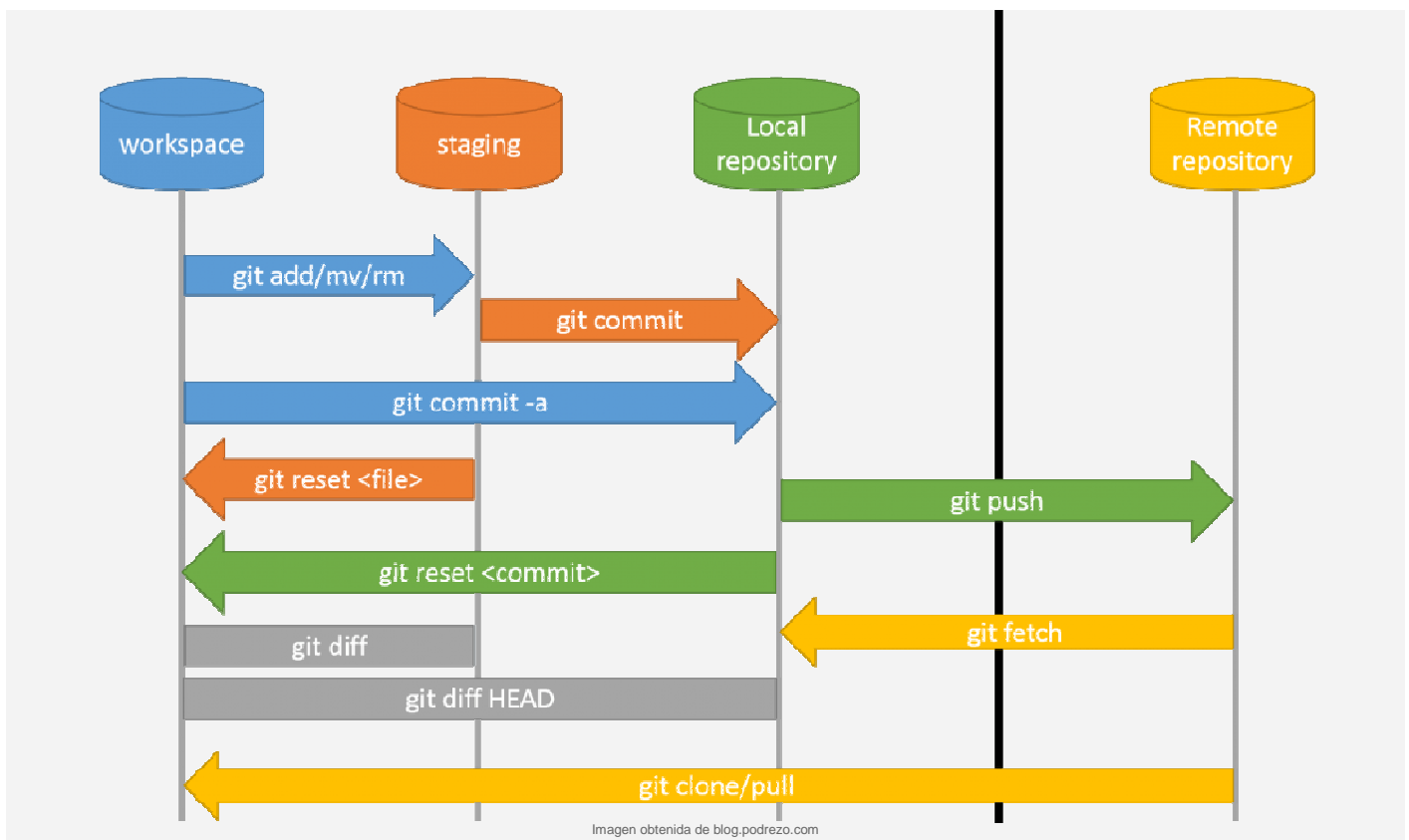
Un *Commit* es un conjunto de cambios guardados en el repositorio de *Git* y tiene un identificador *SHA1* único.

Las ramas (*branches*) se pueden pensar como una línea de tiempo a partir de los *commit*. Hay siempre como mínimo una rama principal o predefinida llamada *Master*.

Head es el puntero al último commit en la rama activa.

Remote se refiere a sitios que hospedan repositorios remotos como GitHub.

Workflow de Git



Si trabajamos en local (comenzamos en la imagen por la izquierda), inicializamos el directorio de trabajo (working directory). Podemos trabajar (editar ficheros) en el directorio de trabajo.

Con el comando *Git add* enviamos los cambios a *staging*, que es un estado intermedio en el que se van almacenando los archivos a enviar en el *commit*. Finalmente con *commit* lo enviamos al repositorio local.

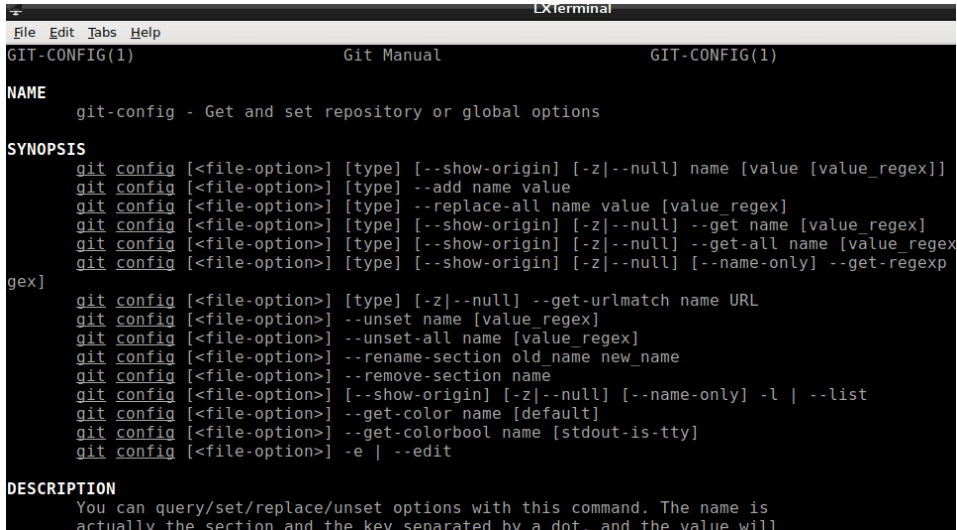
Si queremos colaborar con otros, con *push* subimos los archivos a un repo remoto y mediante *pull* podríamos traer los cambios realizados por otros en remoto hacia nuestro directorio de trabajo.

Si comenzamos trabajando en remoto, lo primero que hacemos es un clon de la información en el directorio local.

Ayuda de Git

Con *git help* en el terminal obtenemos ayuda. Vamos a mirar la ayuda de configuración con

```
#git help config
```



```

NAME
    git-config - Get and set repository or global options

SYNOPSIS
    git config [<file-option>] [type] [--show-origin] [-z|--null] name [value [value_regex]]
    git config [<file-option>] [type] --add name value
    git config [<file-option>] [type] --replace-all name value [value_regex]
    git config [<file-option>] [type] [--show-origin] [-z|--null] --get name [value_regex]
    git config [<file-option>] [type] [--show-origin] [-z|--null] --get-all name [value_regex]
    git config [<file-option>] [type] [--show-origin] [-z|--null] [--name-only] --get-regexp
    git config [<file-option>] [type] [-z|--null] --get-urlmatch name URL
    git config [<file-option>] --unset name [value_regex]
    git config [<file-option>] --unset-all name [value_regex]
    git config [<file-option>] --rename-section old_name new_name
    git config [<file-option>] --remove-section name
    git config [<file-option>] [--show-origin] [-z|--null] [--name-only] -l | --list
    git config [<file-option>] --get-color name [default]
    git config [<file-option>] --get-colorbool name [stdout-is-atty]
    git config [<file-option>] -e | --edit

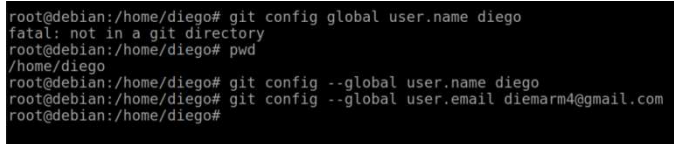
DESCRIPTION
    You can query/set/replace/unset options with this command. The name is
    actually the section and the key separated by a dot, and the value will

```

Configuración de Git

Como mínimo debemos configurar el nombre y el email en la aplicación con los siguientes comandos:

```
git config --global user.name "Tu nombre aquí"
git config --global user.email "tu_email_aquí@example.com"
```



```

root@debian:/home/diego# git config global user.name diego
fatal: not in a git directory
root@debian:/home/diego# pwd
/home/diego
root@debian:/home/diego# git config --global user.name diego
root@debian:/home/diego# git config --global user.email diemarm4@gmail.com
root@debian:/home/diego#
```

Para comprobar podemos usar:

```
git config --global -list
```

Lo que hace el sistema es crear un archivo de texto llamado *.gitconfig*, por lo que podemos mostrarlo también con *cat*:

```
cat ~/.gitconfig
```

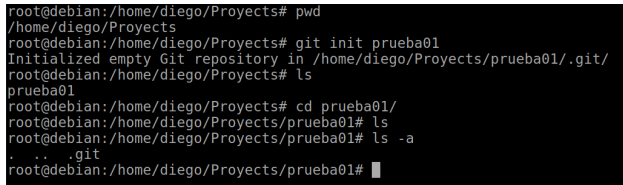
Comenzar a trabajar con Git.

Trabajar en un nuevo proyecto con Git

Nos situamos en la carpeta en la que queremos trabajar. Nos aseguramos con *pwd*, para saber dónde estamos.

Ahora con *git init* y el nombre del proyecto, creamos un nuevo proyecto:

```
git init prueba01
```



```

root@debian:/home/diego/Proyectos# pwd
/home/diego/Proyectos
root@debian:/home/diego/Proyectos# git init prueba01
Initialized empty Git repository in /home/diego/Proyectos/prueba01/.git/
root@debian:/home/diego/Proyectos# ls
prueba01
root@debian:/home/diego/Proyectos# cd prueba01/
root@debian:/home/diego/Proyectos/prueba01# ls
.  ..  .git
root@debian:/home/diego/Proyectos/prueba01#
```

Esto crea una carpeta con el mismo nombre y en su interior, podemos observar con *ls -a* que se crea una subcarpeta oculta *.git* para el control y gestión de la herramienta.

Trabajar en un proyecto existente con Git

Usamos [Initializr](#) para descargar un proyecto demo, lo ponemos en una carpeta, nos situamos en ella y ejecutamos simplemente *git init*.

```

root@debian:/home/diego/Projects# ls
prueba01 Web
root@debian:/home/diego/Projects# cd Web/
root@debian:/home/diego/Projects/Web# ls
404.html      crossdomain.xml  humans.txt      js              tile-wide.png
apple-touch-icon.png  css              img             robots.txt
browserconfig.xml    favicon.ico      index.html      tile.png
root@debian:/home/diego/Projects/Web# git init
Initialized empty Git repository in /home/diego/Projects/Web/.git/
root@debian:/home/diego/Projects/Web# ls -la
.                  browserconfig.xml  .git            index.html      tile-wide.png
..                 crossdomain.xml    .htaccess      js              robots.txt
404.html           favicon.ico         img             tile.png
apple-touch-icon.png  favicon.ico         img             tile.png

```

Primer commit

Ahora vamos a crear algún archivo en el primero de los proyectos. Yo he creado un archivo README.md con el contenido: #Demo de Git. Ahora, en la carpeta del proyecto ejecutamos

```
git status
```

```

diego@debian:~/Projects$ cd prueba01/
diego@debian:~/Projects/prueba01$ ls
README.md
diego@debian:~/Projects/prueba01$ git status
On branch master

Initial commit

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        README.md

nothing added to commit but untracked files present (use "git add" to track)
diego@debian:~/Projects/prueba01$

```

Podemos ver los detalles. Estamos ubicados en la rama máster y que tenemos un archivo que no ha sido agregado aún. Lo primero que vamos a hacer es agregarlo al staging, también llamado Git Index, mediante el comando git add:

```
git add README.md
```

```

diego@debian:~/Projects/prueba01$ git add README.md
diego@debian:~/Projects/prueba01$ git status
On branch master

Initial commit

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

        new file:   README.md

```

Ahora vemos que el archivo ya está esperando a ser enviado al repo mediante commit.

Al ejecutar commit no especificamos qué archivos se enviarán. Se envían todos los que estén en staging. Vamos a agregar la opción -m para agregar un mensaje al commit, por ejemplo "commit inicial":

```
git commit -m "commit inicial"
```

```

1 file changed, 1 insertion(+)
create mode 100644 README.md

```

Veremos un mensaje similar a este:

Si hacemos un status ahora veremos que el área de staging está vacía y no hay ningún commit pendiente.

Trabajando con Git en local

Ahora vamos a agregar más contenido al archivo README y hacemos un status.

Veremos que ahora el sistema nos indica que hay cambios no enviados al staging:

```

diego@debian:~/Projects/prueba01$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   README.md

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        README.md~

no changes added to commit (use "git add" and/or "git commit -a")
diego@debian:~/Projects/prueba01$

```

Con el mismo procedimiento de antes, agregamos (add) y hacemos un nuevo commit:

```

diego@debian:~/Projects/prueba01$ git commit -m "2 commit agregando txt"
[master 8608d3f] 2 commit agregando txt
1 file changed, 1 insertion(+), 1 deletion(-)
diego@debian:~/Projects/prueba01$

```

También podemos hacer el add y el commit en un solo paso con:

```
git commit -a
```

Tal como nos recomienda el propio Git en la captura de arriba. Si queremos también agregar el mensaje quedaría

```
git commit -am "msj"
```

Podemos hacer algunos cambios más y agregar otro archivo, por ejemplo, un index.html al proyecto.

```
diego@debian:~/Proyectos/prueba01$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   README.md

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        README.md~
        index.html

no changes added to commit (use "git add" and/or "git commit -a")
```

En este caso vamos a agregar los archivos modificados o nuevos al staging de forma recursiva con `git add .`

```
diego@debian:~/Proyectos/prueba01$ git add .
diego@debian:~/Proyectos/prueba01$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   README.md
        new file:   README.md~
        new file:   index.html
```

Un nuevo commit:

```
diego@debian:~/Proyectos/prueba01$ git commit -m "nuevo archivo y algunos cambios en Readme"
[master 689e0e5] nuevo archivo y algunos cambios en Readme
 3 files changed, 94 insertions(+), 1 deletion(-)
 create mode 100644 README.md~
 create mode 100755 index.html
diego@debian:~/Proyectos/prueba01$
```

Vamos a volver a hacer algún cambio y lo agregamos al staging. Si nos fijamos en la captura de la anterior operación `add`, el sistema nos indica que podemos sacar un archivo del stage con `git reset HEAD` y el nombre del archivo. Lo probamos con `README`:

```
git reset HEAD README.md
```

```
diego@debian:~/Proyectos/prueba01$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   README.md
        modified:   README.md~

diego@debian:~/Proyectos/prueba01$ git reset HEAD README.md
Unstaged changes after reset:
M       README.md
```

Un nuevo status nos indicará que hay uno en el staging y otro por agregar:

```
diego@debian:~/Proyectos/prueba01$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   README.md~

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   README.md
```

Como podemos ver, también podemos descartar los cambios en el directorio de trabajo con `checkout`:

```
git checkout -- README.md
```

Esto devuelve nuestro archivo `readme` al estado anterior.

Recuerda que puedes consultar la ayuda para ver más opciones y comandos. [Aquí puedes algunos ejemplos adicionales.](#)

Histórico de operaciones en Git. El log

Con `git log` podemos ver el histórico de operaciones que hemos hecho:

```
git log
```

```

diego@debian:~/Proyectos/prueba01$ git log
commit 689e0e5b6a3e1ce21681b6255213543ffcffa2ce
Author: Diego <diemarm4@gmail.com>
Date:   Sun Dec 23 21:25:45 2018 +0100

    nuevo archivo y algunos cambios en Readme

commit 8608d3fe5fd44bc3939edfd1572aaefc45fb0eec
Author: Diego <diemarm4@gmail.com>
Date:   Sun Dec 23 21:06:20 2018 +0100

    2 commit agregando txt

commit d354b88dc51d08e7a40da5ef36636441701a57ce
Author: diego <diego@debian.localhost>
Date:   Sun Dec 23 21:01:46 2018 +0100

    modificando 2 commit

```

Podemos ver que los commit se han hecho en orden cronológico inverso con los distintos usuarios y que cada uno tiene asociado su clave y su fecha. Con git help log podemos ver las opciones que existen. Un ejemplo de una vista compacta y colorida sería:

```
git log --oneline --graph --decorate --color
```

```

diego@debian:~/Proyectos/prueba01$ git log --oneline --graph --decorate --color
* 689e0e5 (HEAD -> master) nuevo archivo y algunos cambios en Readme
* 8608d3f 2 commit agregando txt
* d354b88 modificando 2 commit
* 3130a6c commit inicial
diego@debian:~/Proyectos/prueba01$

```

Eliminar archivos del repositorio de Git

Imaginemos que uno de los archivos que ya están en el repositorio de git tras un commit no lo queremos allí. Podemos usar el comando rm de remove y el nombre del archivo:

```
git rm index.html
```

```

diego@debian:~/Proyectos/prueba01$ git rm index.html
rm 'index.html'
diego@debian:~/Proyectos/prueba01$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>.." to unstage)

        modified:   README.md~
        deleted:    index.html
diego@debian:~/Proyectos/prueba01$

```

Podemos observar que el status nos indica que el cambio ejecutado para el archivo a eliminar está en el staging. Para eliminarlo debemos hacer un nuevo commit:

```

diego@debian:~/Proyectos/prueba01$ git commit -m "eliminando index"
[master 2d2ebec] eliminando index
 2 files changed, 2 insertions(+), 89 deletions(-)
 delete mode 100755 index.html
diego@debian:~/Proyectos/prueba01$ git status
On branch master
nothing to commit, working tree clean
diego@debian:~/Proyectos/prueba01$

```

Ahora veremos que el archivo `index` ya no está en el directorio de trabajo.

Ahora vamos a agregar un nuevo archivo, lo agregamos al staging y hacemos un commit. Vamos a eliminarlo manualmente desde el sistema de archivos del SO, no desde Git.

```

diego@debian:~/Proyectos/prueba01$ git add .
diego@debian:~/Proyectos/prueba01$ git commit -m "agregando humans"
[master cae33a2] agregando humans
 1 file changed, 15 insertions(+)
 create mode 100755 humans.txt
diego@debian:~/Proyectos/prueba01$

```

Ahora elimino el archivo `humans.txt` y hacemos un status:

```

diego@debian:~/Proyectos/prueba01$ rm humans.txt
diego@debian:~/Proyectos/prueba01$ git status
On branch master
Changes not staged for commit:
  (use "git add/rm <file>.." to update what will be committed)
  (use "git checkout -- <file>.." to discard changes in working directory)

        deleted:    humans.txt

no changes added to commit (use "git add" and/or "git commit -a")
diego@debian:~/Proyectos/prueba01$

```

Como la misma pantalla indica, podemos usar add o rm para que definitivamente los cambios se agregan al staging.

```
diego@debian:~/Proyectos/prueba01$ git add .
diego@debian:~/Proyectos/prueba01$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        deleted:    humans.txt

diego@debian:~/Proyectos/prueba01$ git commit -m "eliminar humans"
[master b805ded] eliminar humans
1 file changed, 15 deletions(-)
delete mode 100755 humans.txt
diego@debian:~/Proyectos/prueba01$
```

En este caso se ha usado un *add* y tras un *commit* se ha ejecutado el borrado en el repo.

Mover archivos en Git

Vamos a crear una nueva carpeta "**style**" para alojar un nuevo archivo de estilos css que ahora se encuentra en la carpeta raíz del proyecto y ya en el repositorio. Probamos primero a mover manualmente el archivo:

```
diego@debian:~/Proyectos/prueba01$ mkdir style
diego@debian:~/Proyectos/prueba01$ mv main.css ./style/
diego@debian:~/Proyectos/prueba01$ ls
README.md  README.md~  style
diego@debian:~/Proyectos/prueba01$ cd style/
diego@debian:~/Proyectos/prueba01/style$ ls
main.css
diego@debian:~/Proyectos/prueba01/style$ cd ..
diego@debian:~/Proyectos/prueba01$ git status
On branch master
Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        deleted:    main.css

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        style/
```

En este caso vamos a ejecutar **git commit -a** para agregar al staging y hacer el commit con una sola operación:

```
diego@debian:~/Proyectos/prueba01$ git commit -a
[master c1bffd3] Mover estilos a su carpeta
1 file changed, 373 deletions(-)
delete mode 100755 main.css
diego@debian:~/Proyectos/prueba01$
```

Ojo! Si volvemos a hacer un *status* veremos que la carpeta *styles* no se ha actualizado correctamente. Tendremos que hacerlo recursivo con **git add .** y luego el **commit**.

Ignorar archivos en Git

Ahora disponemos de un archivo que no queremos que se agregue nunca al repo, por ejemplo un archivo de log.

```
diego@debian:~/Proyectos/prueba01$ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)

        app.log

nothing added to commit but untracked files present (use "git add" to track)
```

En Linux Debian, con el comando *nano* podemos editar un archivo en un editor de textos en el terminal. Vamos a editar el archivo *.gitignore*, en el que especificamos los archivos a ignorar en líneas independientes.

```
nano .gitignore
```

Y en el archivo, que aparecerá vacío, agregamos en la primera línea **.log*. Guardamos y salimos.

```
GNU nano 2.7.4      File: .gitignore

*.log
```

Ahora hacemos un *status* y veremos el archivo **.gitignore** para agregar a *staging*:

```
diego@debian:~/Proyectos/prueba01$ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)

        .gitignore

nothing added to commit but untracked files present (use "git add" to track)
diego@debian:~/Proyectos/prueba01$ git add .gitignore
diego@debian:~/Proyectos/prueba01$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        new file:   .gitignore

diego@debian:~/Proyectos/prueba01$ git commit -m "Agregar archivos ignorados"
[master 0a91a51] Agregar archivos ignorados
 1 file changed, 1 insertion(+)
 create mode 100644 .gitignore
diego@debian:~/Proyectos/prueba01$
```

También hemos hecho un nuevo *commit*.

Trabajando con Git en Remoto

Configuración de una clave para la conexión por SSH y autenticación en GitHub

Lo primero es crear la autenticación SSH. Nos dirigimos a la carpeta raíz de usuario, creamos una carpeta llamada `.ssh`. Nos movemos a ella y mediante el comando `ssh-keygen` creamos una clave asociada a la cuenta de correo. Vamos a emplear una clave tipo RSA, de modo que el comando quedaría:

```
ssh-keygen -t rsa -C "correo@dominio.com"
```

Nos pregunta que dónde queremos guardar el fichero con la clave. Podemos seleccionar la opción predefinida.

```
diego@debian:~$ pwd
/home/diego
diego@debian:~$ mkdir .ssh
diego@debian:~$ cd .ssh
diego@debian:~/.ssh$ ssh-keygen -t rsa -C "diemarm4@gmail.com"
Generating public/private rsa key pair.
Enter file in which to save the key (/home/diego/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/diego/.ssh/id_rsa.
Your public key has been saved in /home/diego/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:DYc/DDwsWD/ra4Wqt1DmPldeJlrHKd0neUXhX16dvfY diemarm4@gmail.com
The key's randomart image is:
+---[RSA 2048]---+
|
|  o + . .
| . . X o .
| . @ o .
| o S . B + o
| * O . . = + .
| ... = B = . . .
| oo + . . . o o
| . oo . oo . E |
+---[SHA256]---+
diego@debian:~/.ssh$
```

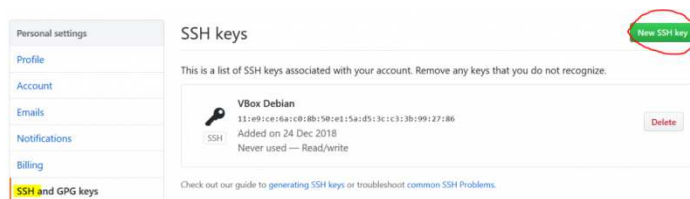
Con `ls -al` podemos ver en detalle los nuevos archivos generados en la operación.

El archivo `.pub` es la clave pública, que enviaremos a GitHub. Para ello debemos abrir el documento y copiar su contenido.

El editor Nano puede ser un poco complejo de usar para estas operaciones. Aquí tienes otras alternativas para Linux: [How to open a particular file from a terminal?](#). Yo usaré

`xdg-open nombre_archivo`

Ahora iniciamos sesión en GitHub y nos vamos a Settings à SSH Keys y creamos una nueva, indicando un nombre descriptivo, recuerda que esto se asocia al equipo desde el que trabajamos, y la clave pública que hemos copiado previamente.



Ahora ejecutamos

```
ssh -T git@github.com
```

Y deberíamos autenticarnos correctamente:

```
diego@debian:~/.ssh$ ssh -T git@github.com
The authenticity of host 'github.com (140.82.118.4)' can't be established.
RSA key fingerprint is SHA256:nThbg6kXUpJWGl7E1IGOCspRomTxdCARLviKw6E5SY8.
Are you sure you want to continue connecting (yes/no)? y
Please type 'yes' or 'no': yes
Warning: Permanently added 'github.com,140.82.118.4' (RSA) to the list of known hosts.
Hi diego@martin! You've successfully authenticated, but GitHub does not provide shell access.
diego@debian:~/.ssh$
```


Trabajando con un repositorio remoto en GitHub

Primero vamos a crear un nuevo repositorio desde la aplicación web de GitHub con la sesión iniciada.

Create a new repository

A repository contains all the files for your project, including the revision history.

Owner

 diegocmartin

/


Repository name


demo-git

Great repository names are short and memorable. Need inspiration? How about [machine](#).

Description (optional)

Se trata de una demo para probar a conectar desde Git

☒  **Public**
Anyone can see this repository. You choose who can commit.

☐  **Private**
You choose who can see and commit to this repository.

☐ **Initialize this repository with a README**
This will let you immediately clone the repository to your computer. Skip this step if you're


Add .gitignore: **None**

Add a license: **None** ⓘ

Create repository

Una vez creado, en la siguiente pantalla hacemos clic en SSH en la parte superior para ver las instrucciones de envío mediante línea de comandos.

Quick setup — if you've done this kind of thing before

 Set up in Desktop

 or

[HTTPS](#)

SSH

git@github.com:diegocmartin/demo-git.git

Get started by [creating a new file](#) or [uploading an existing file](#). We recommend every r

...or create a new repository on the command line

```
echo "# demo-git" >> README.md
git init
git add README.md
git commit -m "first commit"
git remote add origin git@github.com:diegocmartin/demo-git.git
git push -u origin master
```

...or push an existing repository from the command line

```
git remote add origin git@github.com:diegocmartin/demo-git.git
git push -u origin master
```

Ahora nos movemos al proyecto con el que queremos trabajar. Vemos que el status no hay nada pendiente y ejecutamos el primer comando. Usando después...

```
git remote -v
```

El comando `remote` sirve para ver los repositorios remotos asociados. con la opción `-v` vemos la [URL](#):

```
diego@debian:~/Proyectos/prueba01$ pwd
/home/diego/Proyectos/prueba01
diego@debian:~/Proyectos/prueba01$ git status
On branch master
nothing to commit, working tree clean
diego@debian:~/Proyectos/prueba01$ git remote add origin git@github.com:diegocmar
tin/demo-git.git
diego@debian:~/Proyectos/prueba01$ git remote -v
origin  git@github.com:diegocmartin/demo-git.git (fetch)
origin  git@github.com:diegocmartin/demo-git.git (push)
diego@debian:~/Proyectos/prueba01$
```

Las URLs que vemos en la captura anterior son de tipo SSH, indicativo de que podríamos enviar. También podemos agregar repositorios de otros usuarios para recibir con el comando `git remote add [nombre] [url]`.

Ejemplo: `$ git remote add pb git://github.com/paulboone/ticgit.git`

Para recibir los ficheros de uno de estos repositorios de otro usuario usamos el comando *fetch* seguido del nombre que hemos puesto en el comando anterior.
Ejemplo: **git fetch pb**

Ahora vamos a enviar los archivos en local al repo remoto con push y la opción -u para establecer un enlace entre ellos, especificando también el repositorio remoto (origin) y la rama de trabajo (master), quedando:

```
git push -u origin master
```

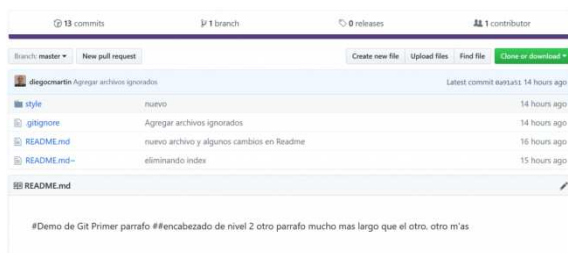
```
diego@debian:~/Proyectos/prueba01$ git push -u origin master
Warning: Permanently added the RSA host key for IP address '140.82.118.3' to the
list of known hosts.
Counting objects: 33, done.
Compressing objects: 100% (27/27), done.
Writing objects: 100% (33/33), 5.78 KiB | 0 bytes/s, done.
Total 33 (delta 8), reused 0 (delta 0)
remote: Resolving deltas: 100% (8/8), done.
To github.com:diegomartin/demo-git.git
 * [new branch]      master -> master
Branch master set up to track remote branch master from origin.
diego@debian:~/Proyectos/prueba01$
```

Una vez establecido el enlace no hará falta usar la opción -u, quedando la instrucción para actualizar los cambios en el repositorio remoto de la siguiente manera:

```
git push origin master
```

```
diego@debian:~/Proyectos/prueba01$ git push origin master
Everything up-to-date
diego@debian:~/Proyectos/prueba01$
```

Ahora podemos ver el repositorio actualizado en GitHub:



Ahora vamos a modificar o agregar algún archivo al repo local y actualizar de nuevo el remoto.

```
diego@debian:~/Proyectos/prueba01$ git add .
diego@debian:~/Proyectos/prueba01$ git commit -m "Agregar index.html"
[master e030932] Agregar index.html
1 file changed, 88 insertions(+)
create mode 100755 index.html
diego@debian:~/Proyectos/prueba01$
```

Si vemos el status ahora, este nos indica que hay cambios pendientes de sincronizar con el remoto:

```
diego@debian:~/Proyectos/prueba01$ git status
On branch master
Your branch is ahead of 'origin/master' by 1 commit.
  (use "git push" to publish your local commits)
nothing to commit, working tree clean
diego@debian:~/Proyectos/prueba01$
```

Aunque ahora estoy trabajando sólo, lo lógico es solicitar los últimos cambios mediante un pull, antes de enviar los nuestros con push, para evitar conflictos con otros miembros del equipo. De modo que ejecutamos:

```
git pull origin master
git push origin master
```

```
diego@debian:~/Proyectos/prueba01$ git pull origin master
From github.com:diegomartin/demo-git
 * branch      master    -> FETCH_HEAD
Already up-to-date.
diego@debian:~/Proyectos/prueba01$ git push origin master
Counting objects: 3, done.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 1.60 KiB | 0 bytes/s, done.
Total 3 (delta 1), reused 0 (delta 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To github.com:diegomartin/demo-git.git
 0a91a51..e030932  master -> master
diego@debian:~/Proyectos/prueba01$
```

Ya podemos ver los nuevos cambios en el repositorio remoto.

Al hacer *pull*, el sistema recupera y trata de unir la rama remota con la local, mientras que con el comando *fetch* que veíamos antes no. Más info en la [documentación oficial](#).

Recursos

Página oficial

<https://git-scm.com/>

Descripción en Wikipedia:

<https://es.wikipedia.org/wiki/Git>

English:

<https://en.wikipedia.org/wiki/Git>

Manual Desarrolloweb.com

<https://desarrolloweb.com/manuales/manual-de-git.html#capitulos185>

Para poder disponer de las funcionalidades en la máquina virtual es necesario instalar las Guest Additions de Virtual Box:

[How to Install VirtualBox Guest Additions in Ubuntu](#)

Para ejecutar el ejecutable de las guest additions en Debian podemos usar la instrucción

```
sh VBoxLinuxAdditions.run
```

Desde un terminal con privilegios.