Git y GitHub desde cero, paso a paso

Gustavo García

3 de junio de 2021.

# Resumen

Instrucciones paso a paso, desde cero, para trabajar localmente con Git y remotamente en GitHub.

# Instalar Git en nuestra computadora local

Ir a la pagina oficial de Git:

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

Descargar la versión de Git que corresponda a nuestra PC:

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

# Instalar Git en nuestra PC

Doble click en el instalador.

GNU General Public License. Next.

Carpeta donde se va a instalar. Next.

Select Components. Dejar los default. Next.

Select Start Menu Folder. Dejar el default. Next.

Choosing the default editor used by Git. Yo elegí Notepad++.

Adjusting the name of the initial branch in new repositories. Master está deprecado. Seleccionar Main. Es lo que se usa ahora. Next.

Seleccionar Use Git from Git Bash only. Next.

Use the OpenSSL library. Next.

Checkout Windows style. Next.

Next.

Next.

Next.

Next.

Experimental options NO.

Install.

Reboot.

# Archivos de configuración de Git

C:\Program Files\Git\etc\gitconfig (es el nombre, no tiene extensión).

C:\Users\usuario\_de\_Windows\.gitconfig (es así).

carpeta\_de\_repositorio\.git\config (el archivo config en la carpeta .git de cualquier repositorio)

# Configuración inicial de Git

En el menú de inicio arrancar Git Bash. Para copiar y pegar desde el portapapeles en Git Bash, se usa ***shift + Insert***.

En Git Bash:

# Lo que sigue al signo numeral es ignorado hasta el final de la línea.

## Tu identidad

Lo primero que debés hacer al instalar Git es configurar tu nombre de usuario y dirección de correo electrónico. Esto es importante porque cada commit de Git usa esta información, y se integra de manera inmutable en los commits que comenzás a crear. El nombre de usuario de Git no es el mismo que el nombre de usuario de GitHub. Se puede usar el mismo, por supuesto. Lo mismo vale para el email.

git config --global user.name "nombre\_de\_usuario"

git config --global user.email "algún\_email\_real"

git config --list # ver la configuración que tenemos

clear # limpiar la pantalla.

# Carpeta para los repositorios locales

Repositorio, proyecto, carpeta, son casi sinónimos. No del todo, pero casi.

Crear una carpeta en la PC local, para guardar en ella los repositorios. Esta carpeta no es un repositorio, sino simplemente el lugar donde vamos a guardar los repositorios. A mí me resulta más fácil usar el Windows Explorer. Creé la carpeta:

C:\git

Dentro de esta carpeta, creamos otra carpeta y la llamamos prueba. Esta sí va a ser un repositorio.

De nuevo en Git Bash:

ls # El comando list (ele y ese). Nos sirve para ver en qué directorio estamos y los archivos que contiene.

ls -la # lista todos los files, incluso los ocultos. En azul los directorios.

cd \\ # hay que duplicar la "\". Cambiar directorio al raíz

Estando en el directorio raíz del disco C, escribimos:

cd gi

y apretamos la tecla tab. Se autocompleta y queda:

cd git/ # Cambiamos al directorio de los repositorios que creamos antes.

cd pru y apretamos tab. Se autocompleta y queda:

cd prueba/

git init # inicializa la carpeta prueba. Ahora es un repositorio.

Git Bash nos informa lo que ocurrió:

Initialized empty Git repository in C:/git/prueba/.git/

Si lo miramos en Windows, dentro de la carpeta prueba ahora hay una carpeta llamada .git.

Un modo más rápido, desde cualquier directorio en el que estemos, es escribir:

cd \\git\\prueba

Cuando un path está precedido por \\ es relativo al directorio raíz del disco.

## Agregamos un archivo

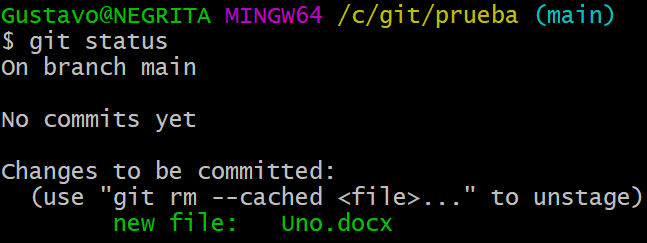
En la carpeta prueba (que es un repositorio) creamos un archivo de texto y le llamamos uno.txt. Dentro le escribimos, por ejemplo, la palabra "Uno", lo grabamos y lo cerramos.

### git add

The git add command adds a change in the working directory to the staging area. It tells Git that you want to include updates to a particular file in the next commit. However, git add doesn't really affect the repository in any significant way—changes are not actually recorded until you run [git commit](https://www.atlassian.com/git/tutorials/saving-changes/git-commit). In conjunction with these commands, you'll also need [git status](https://www.atlassian.com/git/tutorials/inspecting-a-repository) to view the state of the working directory and the staging area.

git add .

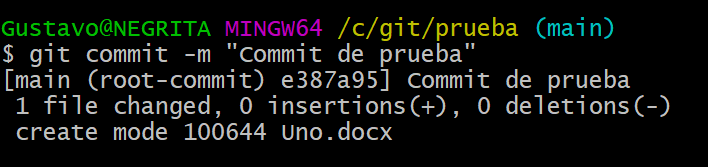
El punto (dot) significa "todo este directorio".

git status

Bash sabe que hay un archivo nuevo, y lo pone en verde. Está ahora en la staging area. Pero le falta el commit.

## Git commit

The git commit command captures a snapshot of the project's currently staged changes. Committed snapshots can be thought of as “safe” versions of a project—Git will never change them unless you explicitly ask it to. Prior to the execution of git commit, The [git add](https://www.atlassian.com/git/tutorials/saving-changes) command is used to promote or 'stage' changes to the project that will be stored in a commit. These two commands git commit and git add are two of the most frequently used.

git commit -m "Commit de prueba" # la opción m es por el mensaje, que va entre las comillas.

# SSH

Cuando yo quiero usar GitHub desde mi máquina local, las claves SSH son una forma de probarle a GitHub que yo soy el dueño de la cuenta.

## Claves SSH existentes

Si ya tengo claves SSH, estarán guardadas en el directorio:

C:\Users\nombre\_usuario\_Windows\.ssh

a menos que cuando la generé haya cambiado las opciones por defecto.

## Borrar claves SSH existentes

Si por alguna razón quiero borrar las clases SSH existentes, tengo que ir al directorio donde están guardadas las claves y borrar los archivos.

## Generar una clave SSH localmente

En Git Bash:

ssh-keygen -t rsa -b 4096 -C email\_que\_voy\_a\_usar\_para\_GitHub

rsa es el tipo de encriptado.

4096 es la cantidad de bits, o la fuerza de la encriptación.

El email que se ponga acá debe ser el mismo que se usa para loggearse en la cuenta de GitHub.

El archivo de la clave se guardará por defecto en el directorio:

C:\Users\usuario\_de\_Windows\.ssh\

y se llamará:

id\_rsa

de modo que todo quedaría:

C:\Users\ usuario\_de\_Windows\.ssh\id\_rsa

Esto lo podemos cambiar, si queremos. Si tenemos más de una clave, necesitaremos darles nombres distintos.

Nos va a pedir una clave o password. La podemos dejar en blanco, o darle alguna. Hay que anotarla.

$ ssh-keygen -t rsa -b 4096 -C email@gmail.com

Generating public/private rsa key pair.

Enter file in which to save the key (/c/Users/Pirulo/.ssh/id\_rsa):

Enter passphrase (empty for no passphrase):

Enter same passphrase again:

Your identification has been saved in /c/Users/Pirulo/.ssh/id\_rsa

Your public key has been saved in /c/Users/Pirulo/.ssh/id\_rsa.pub

The key fingerprint is:

SHA256:qu2tGjn8KrWmSzwo8NDWkWTCCHT2dYa1Cbxq/RhWFX0 email@gmail.com

The key's randomart image is:

+---[RSA 4096]----+

|++.oo .oo+ .o |

|. ++...o+ o. . E |

| o. .o. . |

| . . . . . |

|o o . o S |

|.\* ..+ = |

|o =.\*.o + |

|....o\* o . |

| o=++=.. |

+----[SHA256]-----+

Yo dejé el directorio default, y le dí enter, o sea que no le puse password. Voy al directorio .ssh y veo que hay dos keys: id\_rsa y id\_rsa.pub.

La segunda, id\_rsa.pub es la clave pública, y es la que tengo que upload a mi interfaz de GitHub. Esta clave pública es para que la vean otras personas. La otra clave, la privada es solo para que la vea yo, no debo permitir que nadie la vea. La clave privada la mantengo segura en mi máquina local, y no la comparto con nadie.

Una vez que tengo configuradas las claves privada y pública, cada vez que me conecto a GitHub desde mi máquina local, uso mi clave privada para demostrarle a GitHub que yo soy quien digo ser. Es una prueba de tipo matemático, porque solo mi clave privada puede, usando el algoritmo y los parámetros especificados, generar esa clave pública.

cd \\Users\\user\_name\\.ssh

cat id\_rsa

Me muestra en la consola el hash de mi clave privada.

cat id\_rsa.pub

Me muestra en la consola el hash de mi clave pública.

# De la PC local a la nube

Hasta ahora hemos trabajado localmente en nuestra PC, creamos un repositorio local, conguramos Git e hicimos nuestro primer commit. Vamos a ver cómo trabajar en repositorios remotos con GitHub.

# Cuenta en GitHub

GitHub es la plataforma más popular para manejar repositorios remotos. Hoy por hoy es un estándar de la industria.

## Si tenés cuenta en GitHub

Si ya tenés una cuenta, ingresá:

<https://github.com/login>

## Si no tenés cuenta en GitHub

Necesitás crear una:

<https://github.com/join>

# Crear un repositorio público en GitHub

Clickear el icono de gato en la esquina superior izquierda, que lleva al Home de GitHub. Cerca del gato, abajo a la derecha, hay un botón verde New. Clickearlo. Si ya tenemos repositorios en GitHub, los veremos listados.

Vemos un botón verde que dice Create repository. Lo clickeamos. Se abre la página Crear un nuevo repositorio.

En el cuadro correspondiente, escribir el nombre del nuevo repositorio. Para los fines de esta prueba, le vamos a poner de nombre "prueba", sin las comillas. Será público, a menos que tengamos una cuenta paga. Seleccionar que agregue un archivo README. Notar que nos informa que la rama por defecto (default branch) se llamará main. Clickear el botón verde Crear repositorio, que está en la parte inferior de la página.

No es obligatorio que el repositorio remoto tenga el mismo nombre que el repositorio local. Pero si ambos pertenecen al mismo equipo y al mismo proyecto, es lógico que se llamen igual.

# La página de nuestro repositorio recién creado

Se abre la página del repositorio que acabamos de crear. Vemos que cerca del centro de la página, arriba y a la derecha, hay un botón verde que dice Code. Los deplegamos. La URL de nuestro repositorio será:

<https://github.com/NUESTRO_NOMBRE_DE_USUARIO/EL_NOMBRE_QUE_LE_PUSIMOS_AL_REPOSITORIO>

Por el momento lo dejamos así.

# Subir mi clave pública a GitHub

De regreso en mi PC local. Abro el file de mi clave pública (id\_rsa.pub) con Notepad++ y copio todo el contenido al clipboard. Voy a GitHub. A la derecha, busco mi cuenta. Despliego. Abajo busco Settings. Se abre un menú. Busco SSH y GPG keys en la barra de links, que corre verticalmente a la izquierda del panel principal. Click. Me muestra si tengo claves cargadas. Click New SSH key. Le pongo en Title un nombre que sea significativo para mi propia referencia. En el cuadro de key pego la clave que copié de mi PC. Click Add SSH key. Si me la pide, pongo mi clave de GitHub, no la password que le puse en mi PC a mi clave SSH. Ahora me muestra que tengo una clave asociada con mi cuenta.

# Decirle a Git que tengo una clave local

Ahora tengo que decirle al Git de mi PC local que tengo esta clave. Arranco el agente ssh en background. Pego en Git Bash:

eval "$(ssh-agent -s)"

y me dice:

Agent pid 1234

Tengo que ir al directorio donde tengo la clave:

cd \\Users\\user\_Windows\\.ssh

Ahora le digo al agente ssh de mi clave privada:

ssh-add id\_rsa

No le pongo la clave -K, porque si lo hago me pediría un footprint de mi clave, y no quiero. Me informa:

Identity added: id\_rsa (email@gmail.com)

# Crear una referencia entre un repositorio local y uno remoto

Necesito estar dentro del repositorio local que me interesa:

cd \\git\\prueba

git remote add prueba git@github.com:user\_GitHub/prueba.git

git remote -v

Me informa:

prueba git@github.com:user\_GitHub/prueba.git (fetch)

prueba git@github.com:user\_GitHub/prueba.git (push)

Es decir que cuando yo escriba prueba en un contexto en el que se requiere una referencia a un repositorio remoto, Git va a completar la URL que aparece a la derecha. En este caso ambas son iguales, tanto para fetch como para push. Ahora uno puede usar "prueba" en lugar de la URL.

Si cometimos un error en la URL, podemos corregirlo con el comando:

git remote set-url prueba URL\_correcta

Si queremos remover el shortname usamos:

git remote remove prueba

# Pulling un repositorio remoto a uno local

Para mergear toda la información que tiene el repositorio remoto con el repositorio local, usamos:

git pull -v --allow-unrelated-histories --no-commit prueba main

***-v*** significa verbose.

***--allow-unrelated-histories*** significa que queremos forzar un merge de historias no relacionadas. Lo que pasó es que nosotros creamos un repositorio local en nuestra PC y le pusimos un archivo. Luego creamos un repositorio remoto en GitHub y le pusimos un README.md. Cuando los queremos mergear, Git se da cuenta de que cada repositorio tiene su propia historia, yson distintas. O sea que nunca han sido sincronizados.

***--no-commit*** No hace un commit, para dar al usuario la oportunidad de revisar todo por sí mismo. Una vez que revisa y está feliz, debe hacer el commit explícitamente.

***prueba*** es al shortname del repositorios remoto.

***main*** es la rama que queremos bajar.

Si aparece el siguiente mensaje, significa que Git se quiere asegurar de que el server al que nos estamos conectando es realmente quien dice ser. Esto no es una cuestión menor. Se hace para prevenir un ataque de tipo MITM[[1]](#footnote-1) (Man in the Middle). El mensaje nos dice que el server remoto tiene una determinada finger print:

The authenticity of host 'github.com (140.82.113.3)' can't be established.

RSA key fingerprint is SHA256:nThbg6kXUpJWGl7E1IGOCspRomTxdCARLviKw6E5SY8.

This key is not known by any other names

Are you sure you want to continue connecting (yes/no/[fingerprint])? yes

Lo que tenemos que hacer es ir a la página oficial donde GitHub publica sus finger prints:

<https://docs.github.com/en/github/authenticating-to-github/keeping-your-account-and-data-secure/githubs-ssh-key-fingerprints>

Vemos que la RSA de GitHub coincide con la que muestra Git. Significa que el server es quien dice ser. Le damos yes, y listo. Aparece lo siguiente:

Warning: Permanently added 'github.com' (RSA) to the list of known hosts.

Sis vamos a la carpeta .ssh, donde se guardan las claves, y vemos que apareción un archivo llamado known\_hosts. Lo abrimos con Notepad++ y vemos que contiene la inforamación del server github.com. A continuación comienza a trabajar en el fetch que le habíamos pedido:

remote: Enumerating objects: 3, done.

remote: Counting objects: 100% (3/3), done.

remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0

Unpacking objects: 100% (3/3), 586 bytes | 65.00 KiB/s, done.

From github.com:gsga/prueba

\* [new branch] main -> prueba/main

## Pushing

Para compartir una rama, hay que pushearla a un repositorio remoto en el que uno tenga privilegios de escritura. Las ramas locales no se sincronizan automáticamente con las remotas; hay que pushear explícitamente las ramas que se desea compartir. De esa manera, uno puede usar las ramas privadas para el trabajo que no desea compartir y pushear al repositorio remoto solo las ramas puntuales que desea compartir.

# Hacemos cambios en el repositorio local

Editamos uno.txt y le agregamos otra línas, que dice "dos". Agregamos todo el repositorio al tracking:

git add .

Examinamos el resutado:

git status

Nos informa que tenemos en la rama main cambios para ser commited (o sea, están staged). Cometemos:

git commit -m "agregamos dos"

Nos informa lo que pasó.

git log # para ver la historia

Vemos que nos muestra la historia de todo lo que hicimos: el commit inicial en el remoto, el pull cuando bajamos todo del remoto al local, y el push cuando modificamos uno.txt en el local y subimos los cambios al remoto.

# Cambiar de branch

El comando checkout cambia entre branches o restaura directorios. El comando restore específicamente restaura directorios.

git restore -- "nombre-del.archivo" # para deshacer los cambios que no han sido commited.

git diff -- "nombre-del.archivo" # para ver los cambios que se hicieron en el archivo.

# Cada vez que cambio algo

Cada vez que cambio un archivo tengo que volver a darle add, y luego commit. Si le doy restore antes de haberle dado commit, Git no va a poder rehacer el cambio que antes deshice.

Creo un archivo index.html, que contiene Texto HTML.

git add index.html

git commit -m "c"

git status

Edito index.html y le reemplazo el contenido por zzzzzzz. Grabo.

git status

Vuelvo a cambiar index.html como antes, grabo.

git status ahora me dice que index.html ha sido modificado. Para ver las diferencias, o sea lo que cambió, le doy

git diff index.html.

$ git diff index.html

**diff --git a/TUTORIALES/index.html b/TUTORIALES/index.html**

**index d8e0f82..0e3f705 100644**

**--- a/TUTORIALES/index.html**

**+++ b/TUTORIALES/index.html**

@@ -1 +1 @@

-Texto HTML

\ No newline at end of file

+zzzzzz

\ No newline at end of file

Gustavo@NEGRITA MINGW64 /c/RepositoriosGit/TUTORIALES (master)

La línea que empieza con – se quitó, y la que empieza con + se agregó.

Crear en la carpeta raíz del proyecto un archivo de texto que se llame .gitignore. Es uno para cada proyecto. Tiene los nombres de los archivos y las carpetas que Git debe ignorar.

git add .gitignore

En Windows Explorer creo carpeta

C:\RepositoriosGit\junit5-samples

pwd

Gustavo@NEGRITA MINGW64 /c/RepositoriosGit/TUTORIALES (master)

cd ..

Gustavo@NEGRITA MINGW64 /c/RepositoriosGit (master)

pwd

Gustavo@NEGRITA MINGW64 /c/RepositoriosGit (master)

git add \junit5-samples

git remote add junit5-samples https://github.com/gsga/tutorialGitHub.git

git pull –all https://github.com/gsga/junit5-samples

# Script resumido

Arrancar Git Bash.

Copiar y pegar con shift + insert lo siguiente:

cd \\

cd git

git config --global user.name gsga

git config --global user.email ggarciaa@gmail.com

git config --list

1. <https://en.wikipedia.org/wiki/Man-in-the-middle_attack> [↑](#footnote-ref-1)