Git y GitHub

Gustavo García

5 de agosto de 2022.

# Qué son

Git es una aplicación de escritorio.

GitHub es un sitio web, un portal en la nube.

Ambos son gestores de repositorios, y pueden trabajar en conjunto.

# Para qué sirven

* Actualizar mi repositorio local desde el repositorio remoto.
* Actualizar el repositorio remoto desde mi repositorio local.
* Información de cambios: quién, cuándo, qué.
* Control de versiones.
* Restaurar versiones anteriores.
* Trabajar con branches y forks.
* Control de accesos.

# Repositorio

Un repositorio es una carpeta que ha sido inicializada (o sea configurada) como repositorio. Se reconoce porque contiene una carpeta (oculta) llamada .git.

Un repositorio guarda uno o más proyectos. Adentro de esa carpeta y sus subcarpetas, guardo el código, las imágenes, los documentos de desarrollo y cualquier recurso.

Si el repositorio está físicamente en mi PC, es local. Si está físicamente en la nube, le llamo remoto.

# Control de versiones

El control de versiones es la gestión de los cambios en un sistema cualquiera, en el código que desarrollamos. Trabajo en mi PC en un sistema. De vez en cuando, por ejemplo cada noche, le digo a Git que me guarde una foto (snapshot) de mi proyecto. Git la guarda, y nunca más la pierde ni se la olvida, de modo que yo siempre puedo recuperar versiones anteriores de mi proyecto.

# Git

Uno de los sistemas más popular de control de versiones. Es libre y gratuito. Es la herramienta que en mi PC local rastrea todos los cambios de los proyectos que yo le dije que vigile.

Git corre siempre dentro de una shell Bash. Nosotros usamos Git Bash, y dentro de ella ejecutamos git.

# GitHub

Administrador de repositorios y herramienta de control de versiones en la nube, en github.com.

# Git Bash

Git Bash es una aplicación para entornos de Microsoft Windows que proporciona una capa de emulación para una experiencia de línea de comandos de Git. Bash es un acrónimo de Bourne Again Shell. Un shell es una aplicación de terminal que se utiliza para interactuar con un sistema operativo a través de comandos escritos.

* Git es el administrador local de repositorios.
* Git es distinto de Git Bash.
* Git Bash tiene un montón de comandos.
* Una de las cosas que puede hacer Git Bash es ejecutar el comando git, pasándole argumentos de línea de comando.

# Comandos de Git Bash

## cd

Change directory. OJO: dejar un espacio después de cd, y poner \\ en lugar de \.

## Ctrl + c

Interrumpe la ejecución de cualquier comando.

## Copiar de la consola al clipboard

Simplemente seleccionar el texto deseado en la consola. Listo. Ya está.

## env[[1]](#footnote-1)

**env** es un [comando](https://es.wikipedia.org/wiki/Comando_(inform%C3%A1tica)) del [sistema operativo](https://es.wikipedia.org/wiki/Sistema_operativo) [Unix](https://es.wikipedia.org/wiki/Unix) y los sistemas [Unix-like](https://es.wikipedia.org/wiki/Unix-like). Es usado para mostrar una lista de las [variables de entorno](https://es.wikipedia.org/wiki/Variables_de_entorno) y para ejecutar otro programa con las variables de entorno modificadas sin tener que modificar las variables de entorno del equipo. Usando el comando env se pueden añadir y eliminar variables, además de poder modificar el valor de las variables actuales.

## env|grep PATH

Muestra el contenido de todas las variables de sistema para el usuario actual que contienen "PATH".

## exit

Sale de Git Bash.

## grep[[2]](#footnote-2)

**grep** es una utilidad de la [línea de comandos](https://es.wikipedia.org/wiki/L%C3%ADnea_de_comandos) escrita originalmente para ser usada con el sistema operativo [Unix](https://es.wikipedia.org/wiki/Unix).

Usualmente, grep toma una [expresión regular](https://es.wikipedia.org/wiki/Expresi%C3%B3n_regular) de la línea de comandos, lee la [entrada estándar](https://es.wikipedia.org/wiki/Entrada_est%C3%A1ndar) o una lista de archivos, e imprime las líneas que contengan coincidencias para la expresión regular.

## help

Muestra la ayuda.

## Shift + Insert o simplemente Insert

Pega en la consola el contenido del clipboard de Windows.

## pwd

Print working diretory. Me dice en qué directorio estoy.

# El comando git y sus argumentos

## git add

Cada vez que agrego, edito o borro algo en mi proyecto, le digo a Git que quiero que me vigile eso: un directorio o un file. Puedo poner add ., o sea el dot, este directorio. Si no uso add, Git no sabe que yo lo quiero trackear.

git add .

git add --all

## git clone URL\_remote\_repo

Le digo a Git que me copie en mi PC local un repositorio remoto, que está hosteado por ejemplo en GitHub. De este modo, yo trabajo en mi proyecto localmente, en mi PC local. Hay que poner la URL del repositorio remoto.

Hay que navegar hasta la carpeta donde queremos clonar el repositorio remoto. Este último quedará como una subcarpeta contenida en la carpeta en la que estamos.

## git commit -m "a" -m "some description"

Es como save o grabar, para que Git grabe los cambios que hice desde el último commit. Le digo a Git que me saque una foto (snapshot) de mi proyecto en este momento. La m es por message. Debe haber un mensaje, pero puede ser un carácter y no significar nada.

## git config –list

Me muestra la configuración global. El archivo es C:\Program Files\Git\etc\ gitconfig, es texto plano y se puede editar con Notepad++.

## git pull

Lo contrario de push. Descargo el proyecto desde el repositorio remoto a mi PC local.

## git push origin master (o main)

Upload los commits de Git (mi PC local) a un repositorio remoto, como por ejemplo GitHub.

***origin*** es la carpeta donde está el repositorio que quiero publicar.

***master*** (o main)es la rama que quiero publicar.

## git push -u origin branch

-u significa creame un ***upstream***. Graba los parámetros que le pongo.

***origin*** es el repositorio local.

***branch*** es la rama que quiero publicar.

Una vez hecho esto, uso git puch origin

## git remote add <shortname\_del\_repo\_remoto> <URL\_repositorio\_remoto>

Agrega una referencia de un repositorio local a un repositorio remoto. Para agregar un nuevo repositorio Git remoto como un nombre corto al que puede hacer referencia fácilmente, ejecute git remote add <shortname> <url>. La expresión "repositorio Git remoto" está en el manual oficial de Git, y yo pienso que puede resultar confusa. Los repositorios remotos "están físicamente hosteados" en GitHub. Git corre en mi PC local, y solo en ella. Todos los repositorios que Git maneja directamente son locales, están en mi PC. sin embargo, Git también puede operar sobre los repositorios remotos, que nativamente son de GitHub, SUPUESTO que le doy los privilegios suficientes.

## git remote --v

Me muestra todos los repositorios remotos que estén conectados al repositorio local en el cual estoy parado.

## git status

Me muestra todos los files que fueron creados, editados o deleteados. Ojo que solo los files que hayan sido added serán trackeados. Los files modificados que no hayan sido added, se mostrarán como untracked. Antes de poder commit ese cambio, lo tengo queadd que que Git lo trackee. Una vez que están added, entonces sí están listos para ser committed.

## git version

Muestra en Git Bash o la consola que uno use la versión instalada de Git.

# GitHub

Un sitio web para hostear mis repositorios en la nube. Lo que yo hago localmente en mi PC, y localmente Git lo mantiene vigilado porque yo le dije, GitHub lo sincroniza a la nube. Esto facilita el trabajo colaborativo remoto.

## Comandos de GitHub

### pull request

# README.md

La extensión md es por mark down. Es en principio el primer file que uno crea en un proyecto. Contiene texto que explica qué es proyecto, qué hace, y cualquier otra información relevante. Se puede crear on line o localmente.

# Git y GitHub desde cero, paso a paso

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

## Descargar Git

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.

## Crear un repositorio local

Un repositorio local es una carpeta que está configurada para ser sincronizada con un repositorio remoto. Probablemente, ambos repositorios tendrán el mismo nombre.

Crear una carpeta en la PC local. Esta carpeta no es todavía un repositorio. A mí me resulta más fácil usar el Windows Explorer. Vamos a la carpeta que queremos configurar, click derecho, click en Git Bash Here. Se abre una ventana de Git Bash, que ya está posicionada en ese directorio. 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.

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

Git Bash nos informa lo que ocurrió:

Initialized empty Git repository in C:/path\_a\_la\_carpeta/.git/

Si lo miramos en Windows, dentro del directorio ahora hay un subdirectorio llamado .git. En Bash:

ls -la

ahora nos muestra el nuevo subdirectorio.

### Creamos 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.

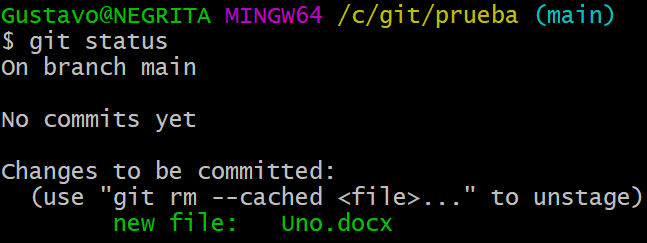
### git add

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.

Si sale un mensaje de error

$ git add -A

fatal: in unpopulated submodule 'algun\_nombre'

Probar lo siguiente:

$ git rm --cached tutorial-8 -f

fatal: pathspec 'tutorial-8' did not match any files

Si eso no funciona, como me pasó a mí, probar lo siguiente:

$ git rm --cached . -rf

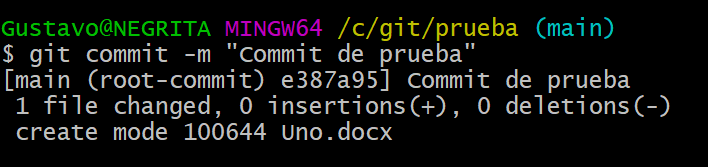
rm 'tutorial-8'

Ahora sí me funcionó.

## 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.



## 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>

## C:\Users\Gustavo\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\3CD83CDB.tmpCrear 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.

Clickeamos el botón New. Se abre la página Create a new repository, o Crear un nuevo repositorio.

En el cuadro correspondiente, escribir el nombre del nuevo repositorio. 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.

## 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.

## Para saber si está corriento el servicio SSH

Para saber si está corriendo el servicio SSH, arrancar el Task Manager con Ctrl + Shift+ Esc. Con la ventana del Task Manger en el tope, teclear en rápida sucesión ssh, es decir las tres letras "s", "s", y "h". Si el servicio está corriendo, lo veremos inmediatamente. Si más de una instancia corriendo, dejar solo una.

## Arrancar el servicio SSH[[3]](#footnote-3)

Pego en Git Bash (no en la consola de Windows):

eval "$(ssh-agent -s)"

y me dice:

Agent pid 1234

Alternativamente, en una ventana de PowerShell (que también es la terminal de VS Code), pegar el siguiente comando de Git:

start-ssh-agent

Esto es lo que muestra la terminal:

PS C:\GIT\20-que-es-angular\tutorial-8> start-ssh-agent

Removing old ssh-agent sockets

Starting ssh-agent: done

Identity added: /c/Users/Gustavo/.ssh/id\_rsa (ggarcia@gmail.com)

Microsoft Windows [Version 10.0.22000.675]

(c) Microsoft Corporation. All rights reserved.

C:\GIT\20-que-es-angular\tutorial-8>start-ssh-agent

Found ssh-agent at 12504

Found ssh-agent socket at /tmp/ssh-0PlKV8k2iFU4/agent.541

Notar que carga automáticamente la identidad.

Cuidado: este comando deja en la terminal de VS Code corriendo la consola de Windows. Para salir de la consola de Windows y volver a PowerShell, usar el comando:

exit

## Decirle a Git que tengo una clave local

Ahora tengo que decirle al Git de mi PC local que tengo esta clave. 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

o bien todo junto:

eval "$(ssh-agent -s)"

ssh-add \\Users\\Gustavo\\.ssh\\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](mailto:email@gmail.com))

Esta ventana de Git Bash no hay que cerrarla, porque eso detendría el servicio SSH.

## 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.

## Crear una referencia entre un repositorio local y uno remoto

Necesito estar dentro del repositorio local que me interesa:

cd [\\git\\prueba](file:///\\git\\prueba)

git remote # para ver qué remotos tengo declarados.

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

## Traer los cambios desde un repositorio remoto a uno local[[4]](#footnote-4)

### Opciones para obtener cambios

Hay varios comandos útiles para interactuar con un repositorio remoto. clone y fetch descargan código de un repositorio remoto a su computadora local, merge se usa para fusionar el trabajo de diferentes personas con el suyo, y pull es una combinación de fetch y merge.

### clone

Para obtener una copia local completa de un repositorio remoto:

git clone [https://github.com/*USERNAME*/*REPOSITORY*.git](https://github.com/USERNAME/REPOSITORY.git)

En este comando no se puede usar el nombre (o alias) de remoto, porque esos remotos, como ya sabemos, están grabados en un archivo de la carpeta .git. El repositorio que queremos clonar todavía no lo tenemos localmente, por lo que tampoco tenemos esos alias. No hay más remedio que poner la URL.

### fetch

Usar fetch para descargar solo el nuevo trabajo (o sea los cambios) hechos en el repositorio remoto. Notar que si en el remoto hay ramas nuevas, fetch descarga esas ramas y todo lo que contienen, pero no hace un merge con las ramas locales.

git fetch remotename

### merge

merge combina los cambios locales con los cambios remotos, sean hechos por otros, o por uno mismo tal vez trabajando desde otra PC.

git merge remotename/branchname

### pull

pull es un comando abreviado que hace un fetch y un merge.

git pull -v --allow-unrelated-histories origin main

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

***-v*** significa verbose.

***--allow-unrelated-histories*** significa que queremos forzar un merge de historias no relacionadas. Historias no relacionadas significa que tanto en el remoto como en el local se hicieron cambios, y esos cambios no se han sincronizado.

***--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.

***origin*** es al shortname del repositorio 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[[5]](#footnote-5) (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

## 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.

## 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.

## .gitignore

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. En estas materias estamos usando lo siguiente:

.angular/

node\_modules/

.mvn/

target/

## 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.

git add --all

git commit -m "algún mensaje"

git remote -v

git push origin main

## 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

## Trabajar con branches

### See What Branch You're On

git status

### List All Branches

The current local branch will be marked with an asterisk (\*).

To see local branches, run this command:

git branch

To see remote branches, run this command:

git branch -r

To see all local and remote branches, run this command:

git branch -a

### Cambiar de branch

El comando checkout cambia entre branches o restaura directorios.

git checkout branch-name

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.

### Merge a branch

git status

git checkout main

git merge la-rama-a-mergear

# Tutoriales de Git y GitHub

## ¿Qué es Git y GitHub? - Repositorios, ramas y mucho más

122,264 views •Premiered May 20, 2019 - 2:30

<https://www.youtube.com/watch?v=DinilgacaWs&ab_channel=Platzi>

### Preguntas

Qué hace Git.

Qué hace GitHub.

Diferencia entre Git y GitHub.

Qué tipos de repositorios manejan Git y GitHub y cuál es la diferencia.

## Como utilizar GitHub y Git para novatos| Primeros pasos

35,322 views - •Mar 30, 2017 - 43854SHARESAVE

[Fisherman Site](https://www.youtube.com/channel/UCdtQg5_rvO9aQdiMC0CFnEg) - 7.51K subscribers

<https://www.youtube.com/watch?v=UWovblxkARQ&ab_channel=FISHERMANSITE>

### Preguntas

Qué es GitHUb.

## Git and Github | Practical Course from Scratch

742,231 views •Jan 2, 2018 21K240SHARESAVE [Fazt](https://www.youtube.com/channel/UCX9NJ471o7Wie1DQe94RVIg) 327K subscribers 50:47

<https://www.youtube.com/watch?v=HiXLkL42tMU&ab_channel=Fazt>

### Preguntas

Qué es GitHUb.

Quién creó Git, y para qué lo hizo.

Qué es el working directory.

Qué es la staging area.

Qué es una snapshot.

Para qué se usa el comando git init.

Para qué se usa el comando git add.

Para qué se usa el comando git status.

Para qué se usa el comando git commit.

Para qué se usa el comando git push.

Para qué se usa el comando git pull.

Para qué se usa el comando git clone.

Para qué sirve y cómo se usa el archivo .gitignore.

Para qué se usa el comando git branch.

Para qué sirve el archivo README.md.

Qué es el formato Markdown.

Para qué se usa el comando git clone.

## Git tutorial en español 2020. Que es el control de versiones? Por qué usarlo? Git y Sourcetree.

2,485 views -- •Apr 6, 2020 - 752SHARESAVE - [Guillermo Gerard](https://www.youtube.com/channel/UCp7w1Gx-8u5nxuWzF3n7BKA)

<https://www.youtube.com/watch?v=NwlZZfcv400&ab_channel=GuillermoGerard>

Este chico es un capo. El ejemplo inicial seguramente te hará reir. O llorar.

### Preguntas

Qué es el control de versiones.

La palabra ***"estadío"*** no existe en español.

1. <https://es.wikipedia.org/wiki/Env> [↑](#footnote-ref-1)
2. <https://es.wikipedia.org/wiki/Grep> [↑](#footnote-ref-2)
3. <https://stackoverflow.com/questions/18683092/how-to-run-ssh-add-on-windows> [↑](#footnote-ref-3)
4. <https://docs.github.com/en/get-started/using-git/getting-changes-from-a-remote-repository> [↑](#footnote-ref-4)
5. <https://en.wikipedia.org/wiki/Man-in-the-middle_attack> [↑](#footnote-ref-5)