MANUAL GITHUB

INSTALACIÓN, CONFIGURACIÓN Y USO DEL SOFTWARE

Raúl Benítez Ordóñez.

1º DAM.

ÍNDICE

1. INTRODUCCIÓN.

Antes de empezar a instalar y configurar nuestro software, vamos a conocer mejor **QUÉ ES GIT** y **PORQUÉ ES TAN IMPORTANTE** en el mundo de la informática.

GIT nace de la necesidad de un sistema de control de versiones en nuestros proyectos.

Los sistemas de control de versiones son programas que tienen como objetivo controlar los cambios en el desarrollo de cualquier tipo de software, permitiendo conocer el estado actual de un proyecto, los cambios que se le han realizado a cualquiera de sus piezas, las personas que intervinieron en ellos, etc.

Para facilitarnos la vida existen estos sistemas como Git, que sirven para controlar las versiones de un software y que deberían ser una obligatoriedad en cualquier desarrollo. Nos ayudan en muchos ámbitos fundamentales, como podrían ser:

- Comparar el código de un archivo, de modo que podamos ver las diferencias entre versiones.
- Restaurar versiones antiguas.
- Fusionar cambios entre distintas versiones.
- Trabajar con distintas ramas de un proyecto, por ejemplo, la de producción y desarrollo.

Entonces, **Git** es un software más de control de versiones gratis y de código abierto creado por Linus Torvalds en 2005.

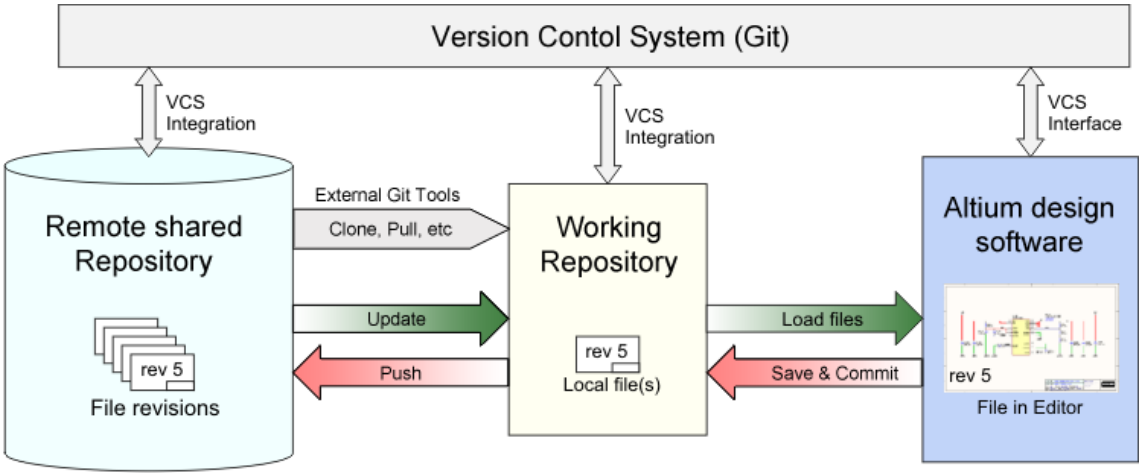
Esta herramienta fue inicialmente desarrollada para trabajar con varios desarrolladores en el núcleo de Linux.

Antes de empezar a trabajar con Git, necesitamos asignar un sitio donde podamos alojar nuestros repositorios de los proyectos.

Hay dos maneras en que puedes alojar tus repositorios. Uno es **en línea** (en la nube) y la segunda es **fuera de línea** (auto instalado en tu ordenador).

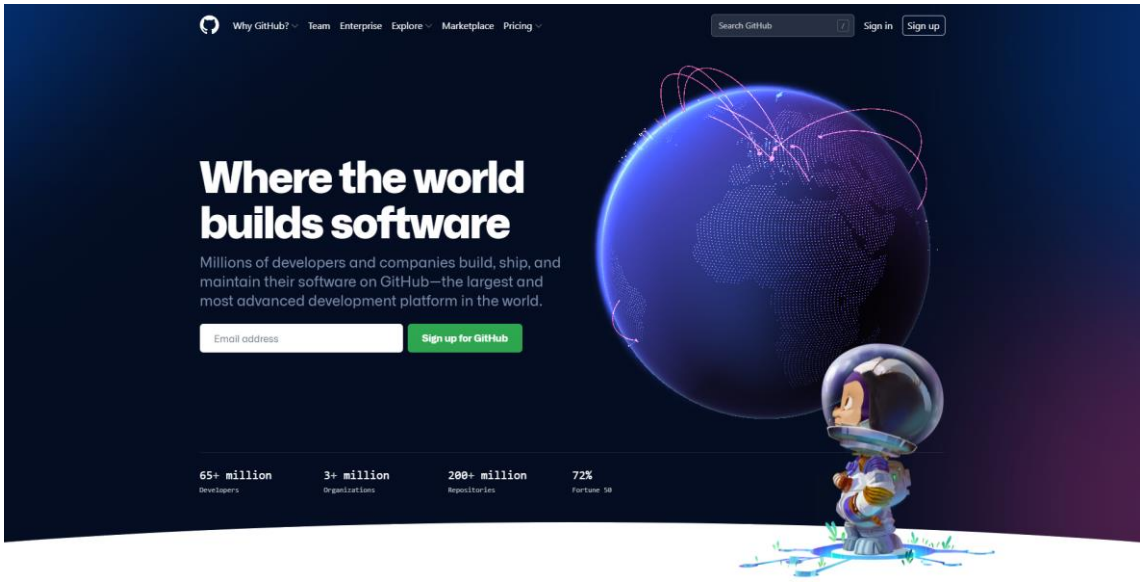
Este sitio en la nube es **GITHUB.**

GitHub es la plataforma de desarrollo colaborativo de Microsoft, o también llamada "la red social de los desarrolladores" donde se alojan dichos repositorios. El código en este programa se almacena de forma pública, es decir, cualquiera puede entrar a tu código y comentarte o ayudarte a desarrollarlo.

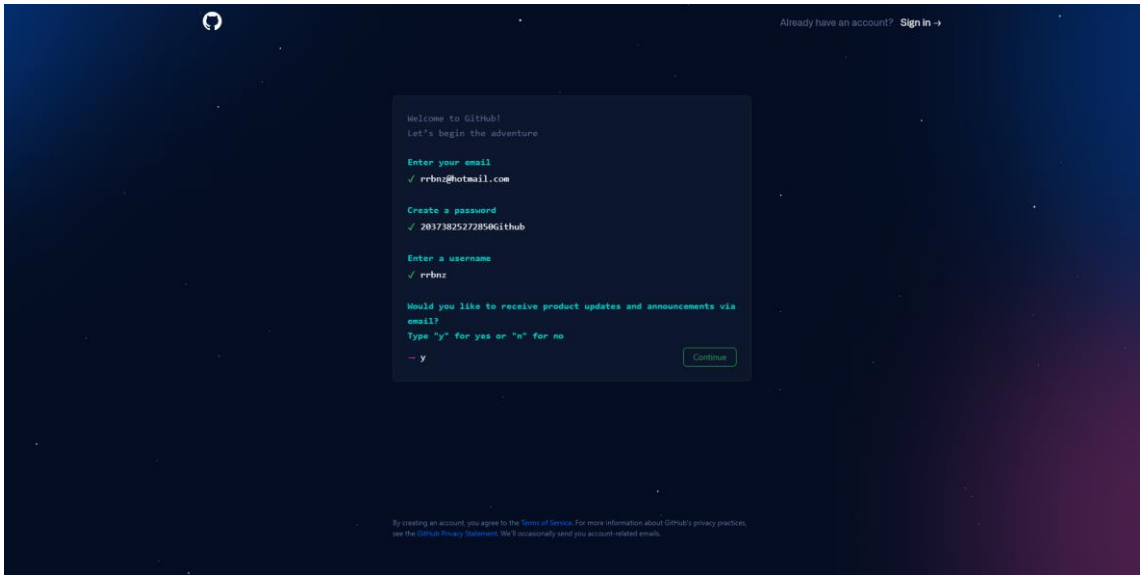


2. INSTALACIÓN.

Para empezar a usar Git y GitHub, vamos a crearnos primero una cuenta en github.com.



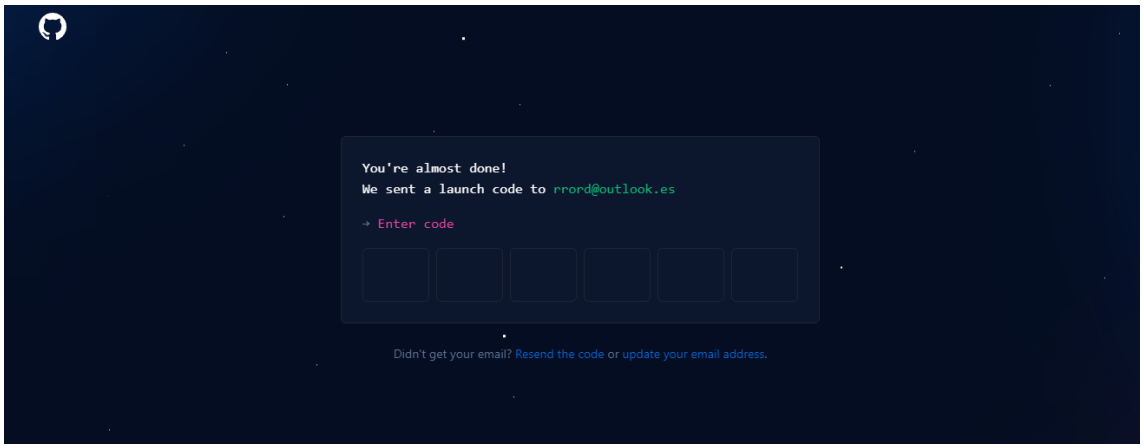
En la esquina superior derecha, clicamos donde pone “Sign Up”.



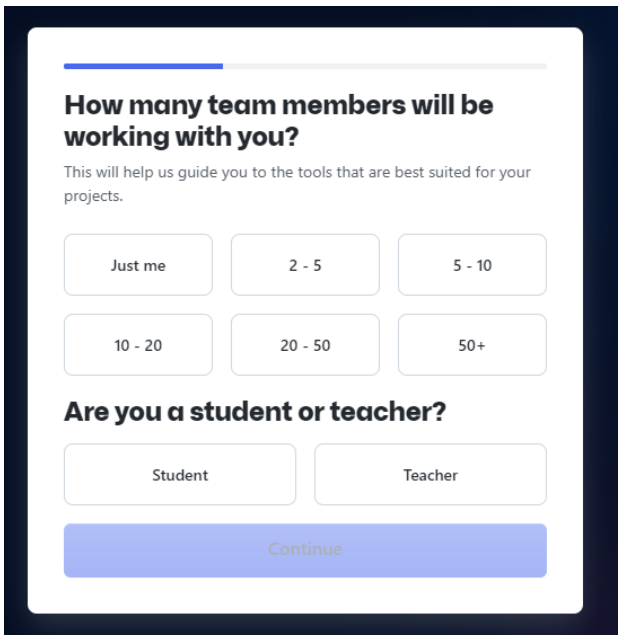
Nos van a pedir un email, contraseña y nombre de usuario para podemos crearnos la cuenta en GitHub.

Al final de todo nos viene una petición de Git para enviarnos actualizaciones de sus productos y noticias a nuestro email.

Si queremos aceptarlo, escribimos una “y” en el cuadro de texto y, si no queremos, una “n”.



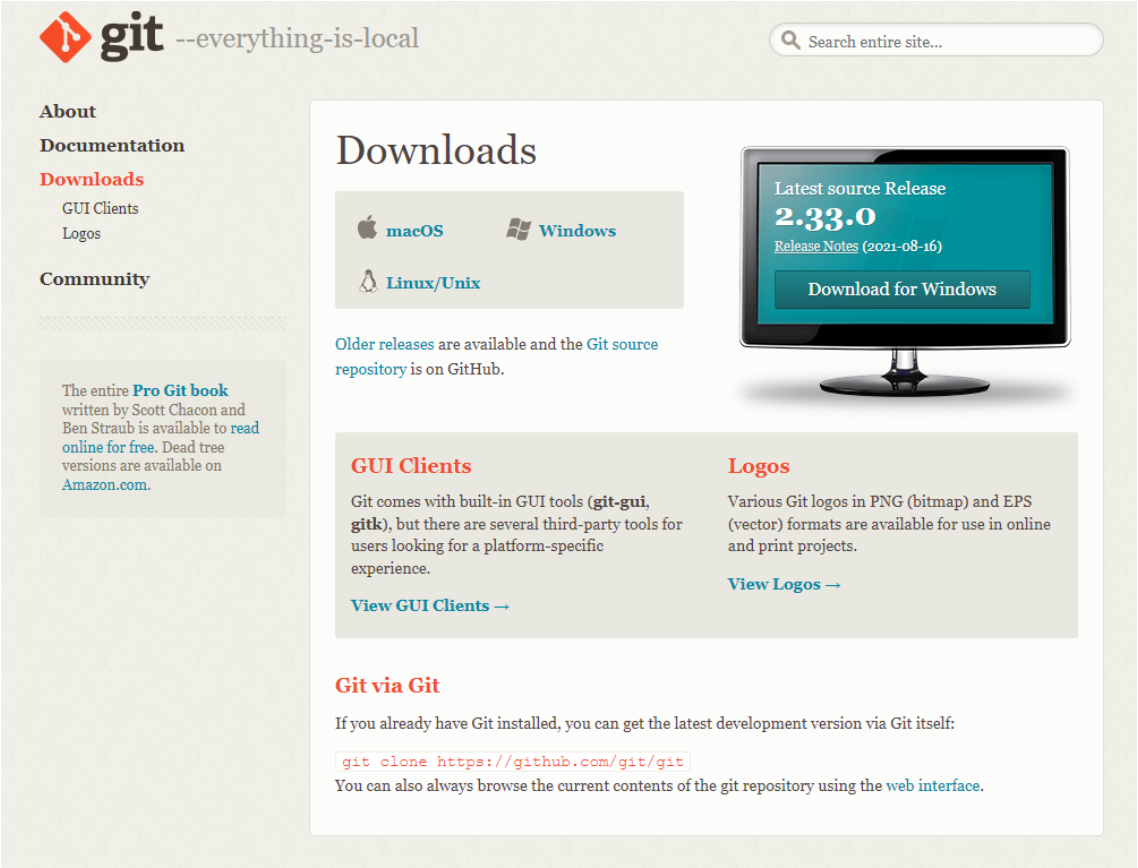
Automáticamente, nos van a redirigir a una página donde nos van a enviar un email con un código de 6 números desde GitHub para verificar la cuenta de correo electrónico. Solamente tenemos que escribir ese código en la página y ya tendríamos la cuenta creada.



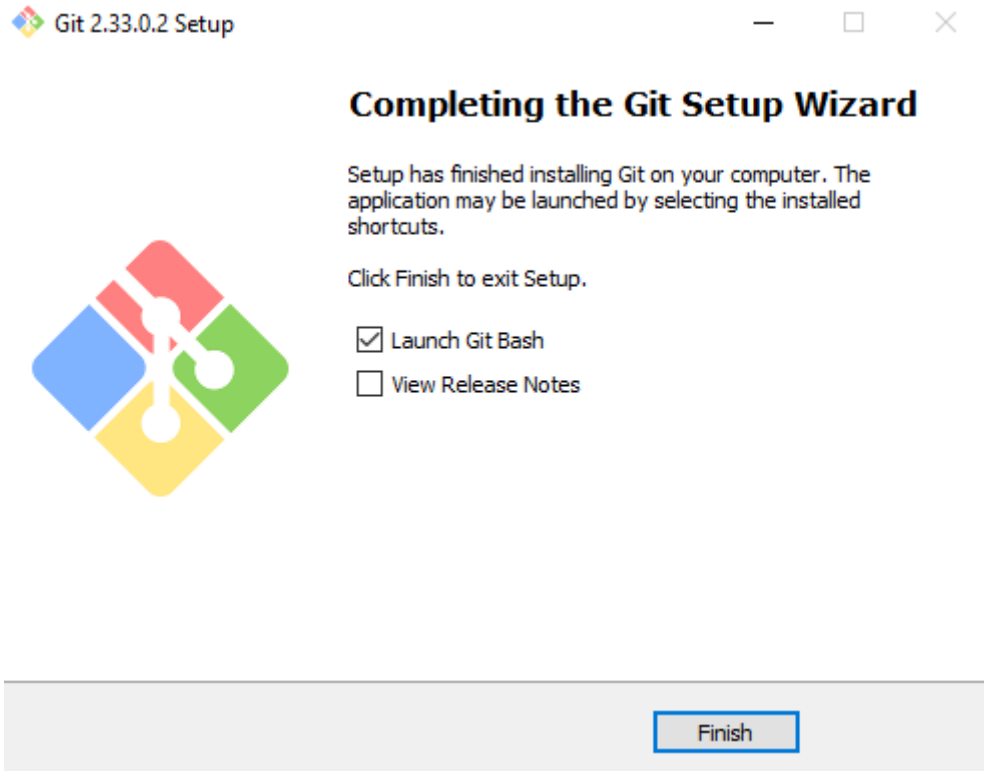
Justo después, ya para terminar la creación de la cuenta, nos van a hacer unas preguntas para adaptarse mejor a lo que queremos hacer en GitHub, seleccionamos “*Free Account*” y ya tendríamos la cuenta creada.

Ahora, para instalar GIT, nos tenemos que ir al siguiente enlace:

*** git-scm.com/downloads ***



Dentro de esta página, seleccionamos “Download for Windows” y automáticamente se nos va a descargar un ejecutable. Una vez descargado, abrimos el ejecutable y en la instalación, le damos a todo “Next” hasta que empiece a instalarse el programa.



Nos aparecerá algo como esto una vez finalizada la instalación.

Git Bash es la línea de comandos de Git para Windows, que además te permite lanzar comandos de Linux básicos. Es la opción más común para usar Git en Windows.

Sin embargo, nosotros vamos a descartar esta opción de GitBash ya que vamos a trabajar con comandos en Windows Powershell.

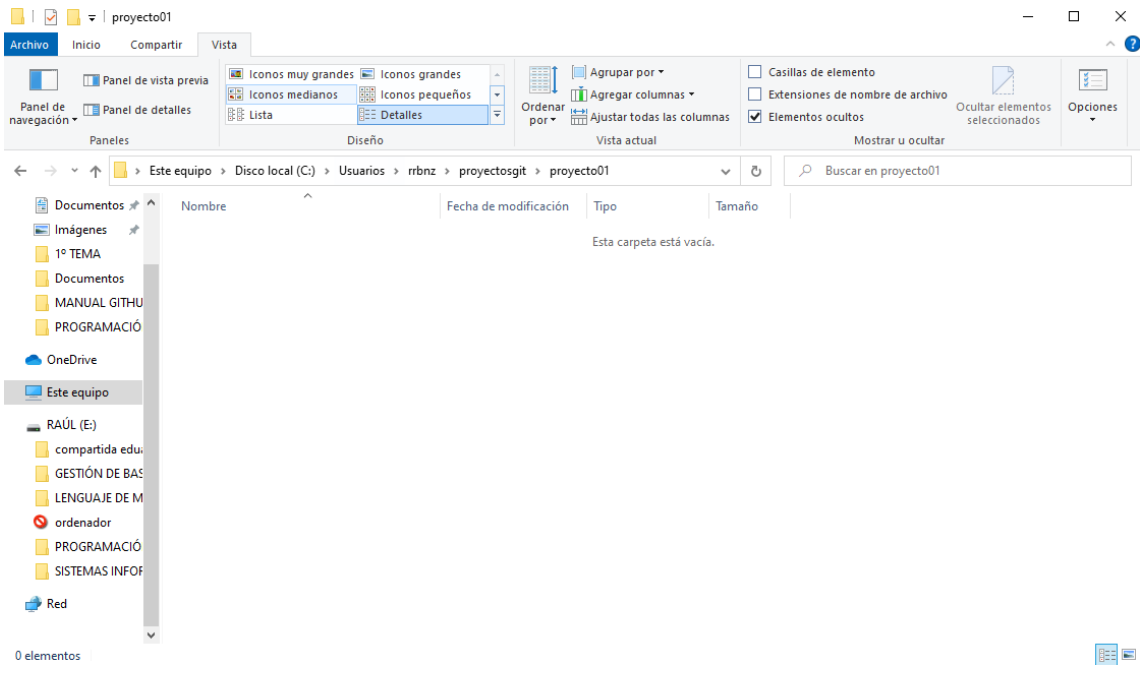
(En realidad, da igual en que líneas de comandos trabajemos, así que decidid el que más os guste).

3. CONFIGURACIÓN.

Antes de empezar a trabajar con nuestros repositorios y los comandos Git, hay algunos aspectos de la configuración que tenemos que aclarar.

Necesitamos aclarar donde vamos a guardar localmente toda nuestra información Git.

Yo, por ejemplo, voy a crear en algún directorio bastante accesible, una carpeta donde guardar los repositorios, la información de la configuración y todo lo relacionado.



****Muy importante evitar los espacios en las carpetas****

(Se corrige más adelante)

Abrimos Windows Powershell y ejecutamos:

cd C:\Users\rrbnz\proyectosgit\proyecto01

Para posicionarme en esa carpeta dentro de la línea de comandos.

.

.

.

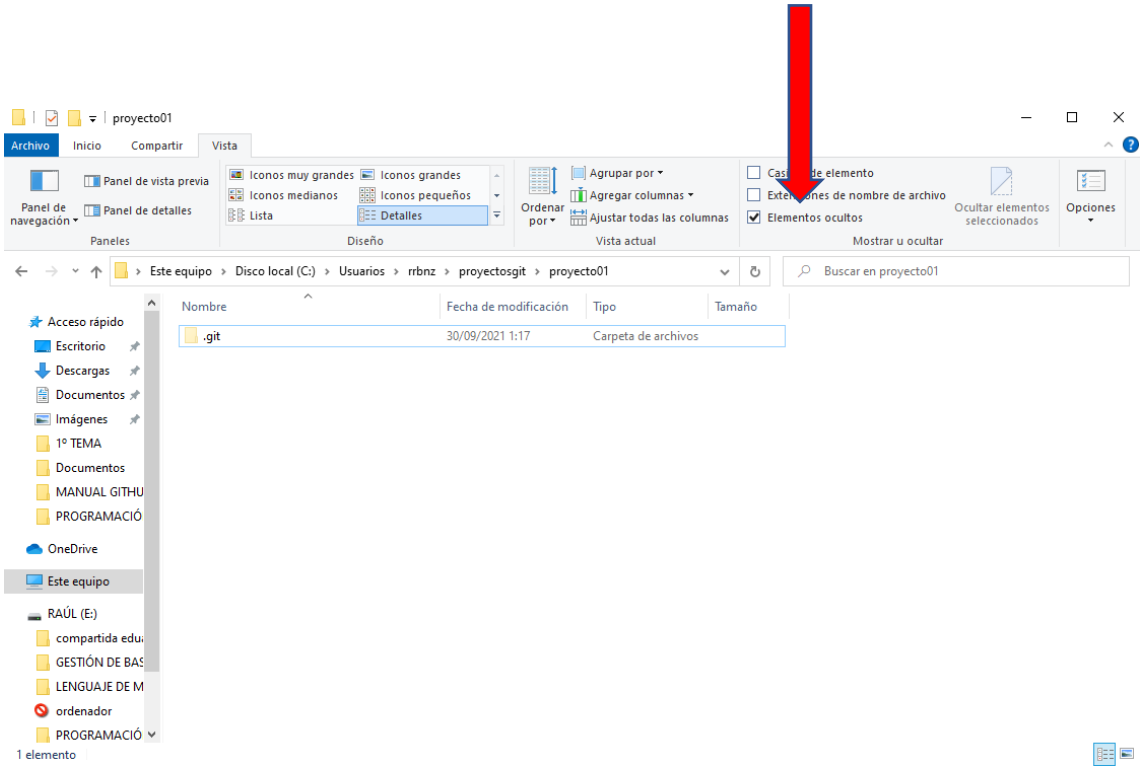
```
PS C:\Users\rrbnz> cd C:\Users\rrbnz\proyectosgit\proyecto01
PS C:\Users\rrbnz\proyectosgit\proyecto01>
```

Una vez hecho, tenemos que lanzar el primer comando Git de todos, "**git init**". Este comando nos sirve para inicializar un proyecto de Git simplemente en la carpeta principal de nuestro proyecto.

```
PS C:\Users\rrbnz> cd C:\Users\rrbnz\proyectosgit\proyecto01
PS C:\Users\rrbnz\proyectosgit\proyecto01> git init
Initialized empty Git repository in C:/Users/rrbnz/proyectosgit/proyecto01/.git/
PS C:\Users\rrbnz\proyectosgit\proyecto01>
```

Ya lanzado el "git init" este lo que nos va a hacer es crearnos una carpeta .git en la ruta que le hemos indicado antes.

Esta carpeta simplemente es el directorio raíz de tu proyecto principal de Git. Contiene la descripción del último commit que hayas hecho, las configuraciones del repositorio, las ramas de hayamos utilizado en Git, los errores que hayan surgido, etc.



Ya simplemente para terminar la configuración, nos falta por ejecutar dos comandos que son:

git config --global user.name "Tu nombre aquí"

git config --global user.email "tu_email_aquí@example.com"

Con estos comandos, indicas tu nombre de usuario (usas tu nombre y apellidos generalmente) y el email.

Esta configuración sirve para que cuando hagas commits en el repositorio local, éstos se almacenen con la referencia a ti mismo, meramente informativa.

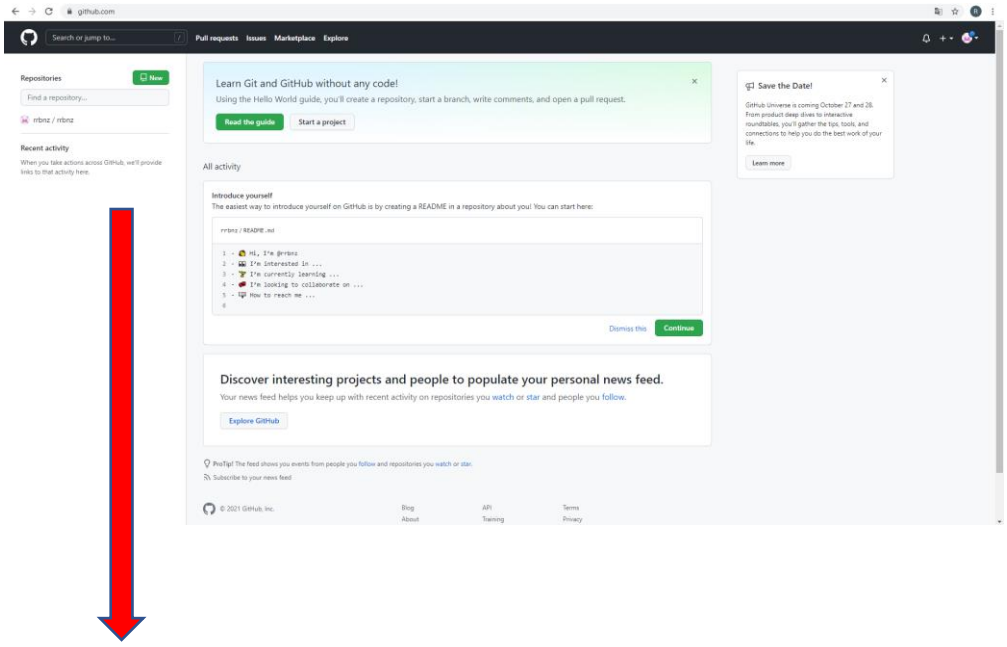
Gracias a ello, más adelante cuando obtengas información de los cambios realizados en el los archivos del "repo" local, te va a aparecer como responsable de esos cambios a este usuario y correo que has indicado.

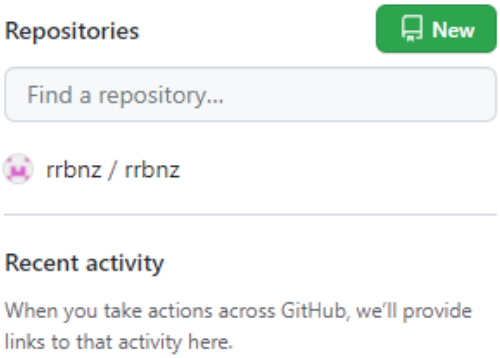
4. CREANDO NUESTRO PRIMER REPOSITORIO.

Nosotros en Git vamos a trabajar en LOCAL y vamos a ir actualizando el progreso del trabajo que estemos haciendo EN LA NUBE.

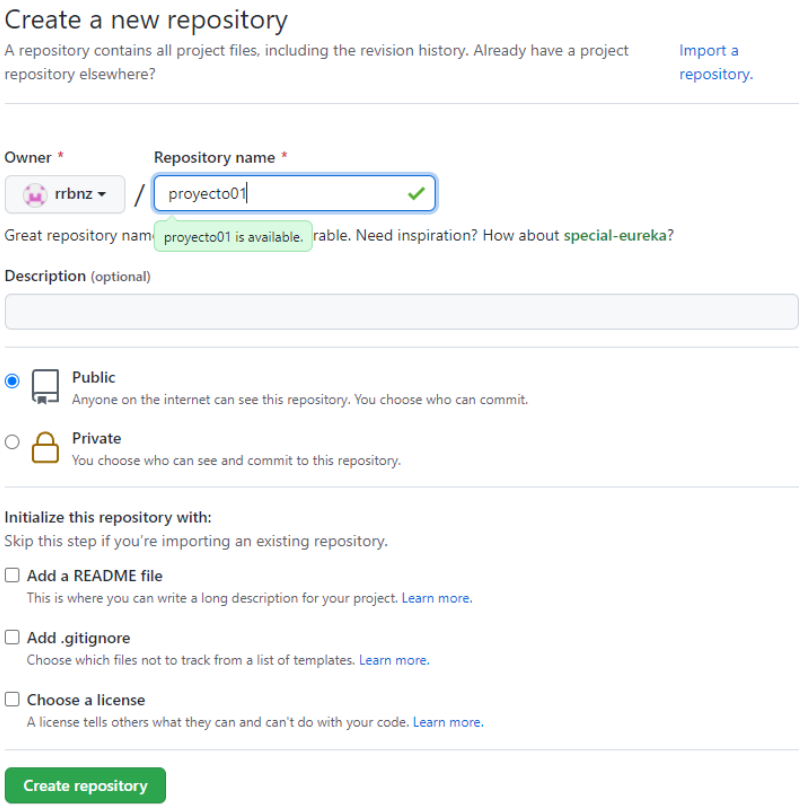
Es decir, nosotros vamos a estar trabajando en nuestra carpeta que hemos elegido antes (actualizando los datos, las carpetas, etc) y enviando esos cambios a nuestro REPOSITORIO EN LA NUBE para hacer como un doble guardado de todos nuestros proyectos.

Empezamos creando nuestro Repositorio en GitHub.



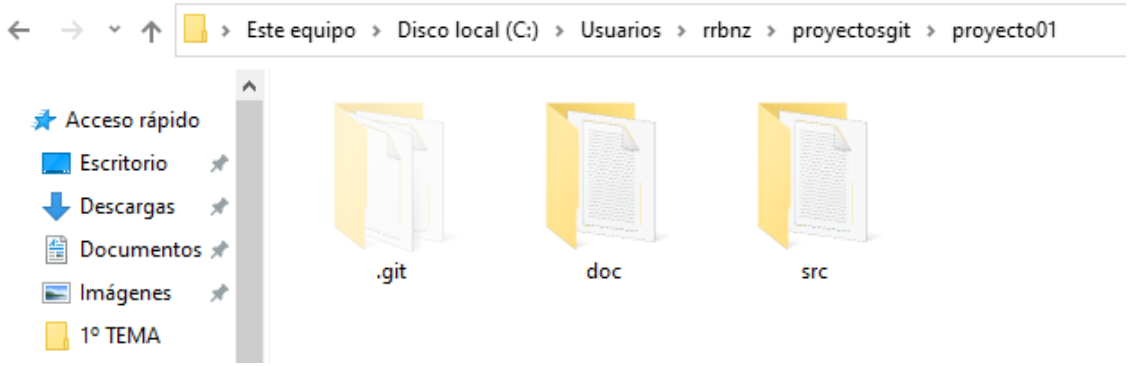


En la página principal de GitHub, en la esquina superior izquierda le damos a "New" y automáticamente se nos va a abrir una ventana para crear nuestro repositorio en la nube.

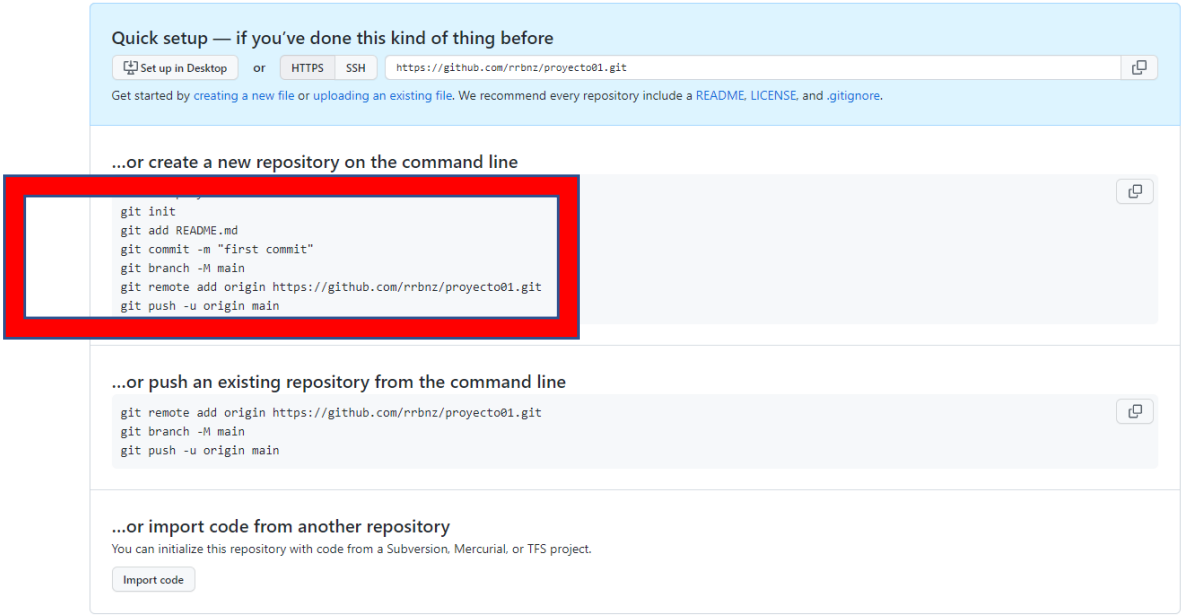


Personalizamos el repositorio como queramos y le damos a “Crear repositorio”. Le damos un nombre, elegir si el repositorio va a ser público o privado y, una descripción si queremos. La parte que dice "Add a README file" siempre la dejo desmarcada.

Ahora, vamos a suponer que estamos trabajando en nuestra carpeta “proyecto01” con una carpeta src que contiene los archivos fuente codificados para este trabajo en lenguaje Java y una carpeta doc que contiene información del programa



Entonces, ¿cómo subimos esta información a nuestro repositorio en GitHub?
Es tan simple como seguir las instrucciones que hay en
“...or create a new repository on the command line”



Pero, ¿qué estamos haciendo en realidad con todos estos comandos? Lo que estamos haciendo es simplemente ENLAZAR nuestra carpeta donde hemos estado trabajando con el repositorio del GitHub que hemos creado.

Primero nos dice que iniciemos el "git init". Hacemos eso lo primero si no lo hemos hecho todavía.

Después, escribimos en la línea de comandos,

```
git add .
```

Este comando lo que hace es desde la carpeta de haces el comando "add" para agregar todos los archivos al "staging area" (área de preparación)

.
.

Opcional: Si queremos añadir el archivo "README.md" lo podemos hacer ahora creándolo directamente como un bloc de notas con ese nombre y esa extensión. Simplemente, este archivo contiene información acerca de otros archivos en un directorio.

.
.

```
git commit -m 'MiPrimerCommit'
```

Después del "git add README" o de crearlo en la carpeta directamente y hacer un "git add ." para subirlo al repositorio, lanzamos directamente esta instrucción.
Esto lo que hace es llevarse los comandos al repositorio que hemos creado en GitHub.

Siguiendo la estructura que hemos dicho antes, simplemente vamos lanzando todos los comandos hasta el último.

```
git branch -M main
```

.
.

```
git remote add origin https://github.com/aqui-tu-repo.git
```

En mi caso, sería:

git remote add origin https://github.com/rrbnz/proyecto01.git

Y, por último:

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

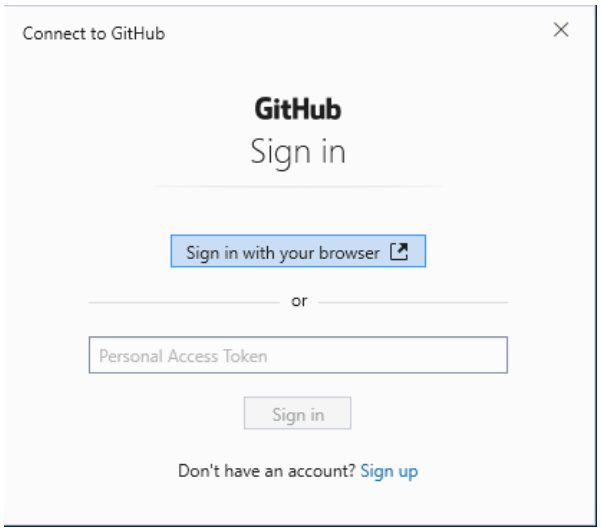
Básicamente lo que estamos haciendo es vincular el repositorio EN LA NUBE con la carpeta principal del proyecto con el link de repositorio del GitHub. Luego estamos haciendo un "push" desde el repositorio local al remoto con los comandos que aparecen en la página de Github que hay justo después de haber creado el repositorio.

Si queremos, podemos hacer todos estos comandos de un tirón copiándolos en el cortapapeles y pegándolos en la línea de comandos, aunque yo recomiendo hacerlo todo poco a poco e ir haciendo un.....

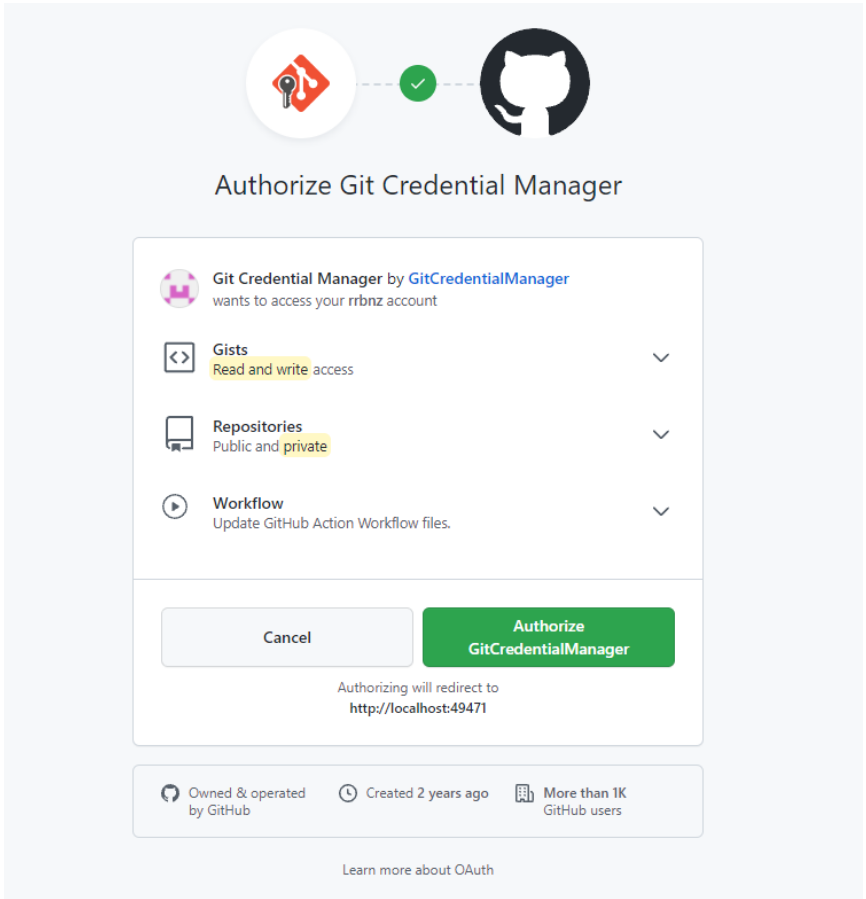
git status

Para ir comprobando que todo se este haciendo correctamente.

Probablemente, al hacer todos estos comandos, al final, nos salga esto.

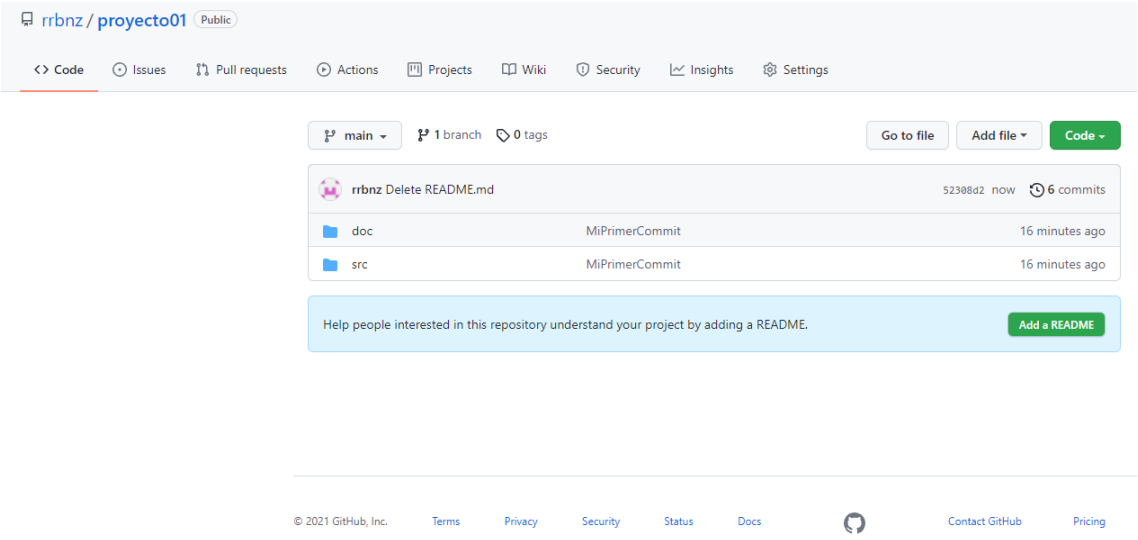


Le damos a "Sign in with your browser" y a "Authorize GitCredentialManager".



Ahora, después de haber hecho todos estos pasos, por fin hemos sincronizado el repositorio y la carpeta del proyecto.

Si volvemos a nuestro GitHub y nos metemos en el repositorio que hemos creado, veremos que ya nos sale los archivos que tenemos en nuestra carpeta principal del proyecto.



Entonces, simplemente la función que hacemos ahora con Git es trabajar con nuestra carpeta principal del proyecto y subir los cambios o los archivos que vayamos añadiendo al repositorio.

Por ejemplo:

Vamos a crear el readme.ME con detalles de proyecto y hemos actualizado información en la carpeta src y doc.

Lo que tenemos que hacer es con nuestro Powershell abierto, es un

.
.
git add .
.
.

Para enviar todos los archivos nuevos y los ya existentes editados a la "stating area".

Después hacer un

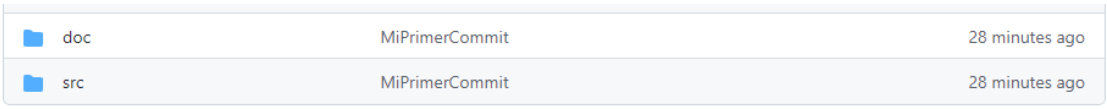
.
.
git commit -m "mensaje del commit"
.
.

Para subirlos al repositorio remoto y, por último, hacer un

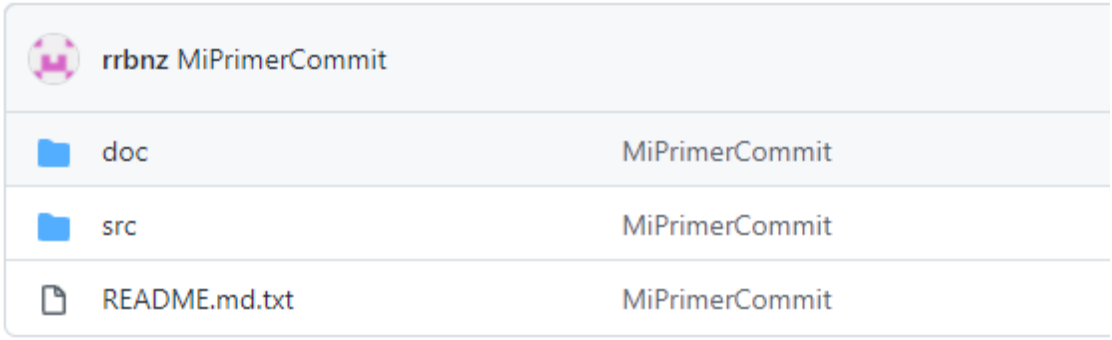
.
.
git push -u origin master
.
.

Para enviar todos los commits que hayamos hecho.

Entonces, como estamos viendo en la siguiente imagen, los archivos que hay están modificados hace bastante tiempo y enviados en el primer commit que hicimos y, no está creado el README.

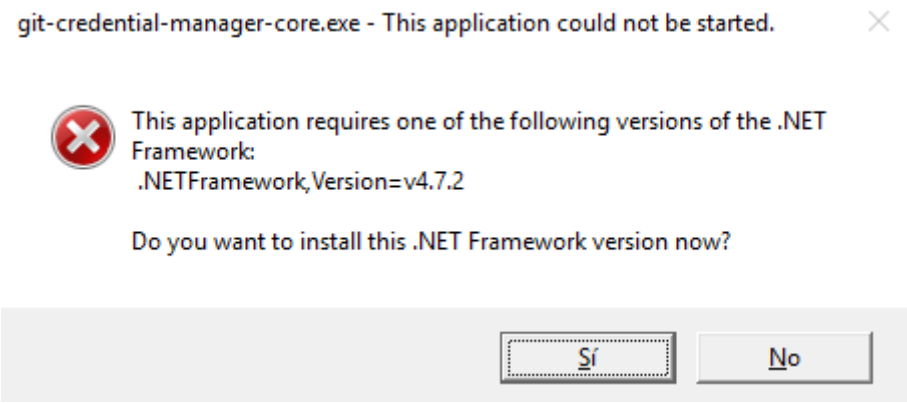


Pues al ejecutar todos los comandos anteriormente nombrados en el Windows Powershell.....



Automáticamente se actualizan todos los datos y se van añadiendo al repositorio remoto. Dentro de este repositorio también se pueden editar los archivos de texto y demás.

IMPORTANTE:



Si durante la configuración os sale esta ventana, darle a “Sí”.

Eso es que os falta una estructura previa para poder usar Git.

Aquí te voy a dejar un pequeño resumen de todos los comandos que hemos visto por si alguna vez la necesitas.

HOJA DE REFERENCIA DE GITHUB

5. COMANDO ".GITIGNORE"

No todos los archivos y carpetas son necesarios de gestionar a partir del sistema de control de versiones. Hay código que no necesitas enviar a Git.

El ejemplo más claro que se puede dar surge cuando se trabaja con sistemas de gestión de dependencias, como npm, Bower, Composer.

Al instalar las dependencias se descargan muchos archivos con documentos, tests, demos, etc. Todo eso no es necesario que se mantenga en el sistema de gestión de versiones, porque no forma parte del código de nuestro proyecto en concreto, sino que es código de terceros.

Si Git ignora todos esos archivos, el peso total de proyecto será mucho menor y eso redundará en un mejor mantenimiento y distribución del código.

¿Cómo implementamos el .gitignore?

Dentro de ese archivo le añadimos simplemente que archivos no queremos que tenga en cuenta:

.

nombrearchivo.ext

.

O podemos decirle que no tenga en cuenta todos los archivos de una extensión específica:

.

***.extensión**

.

O podemos decirle que no tome en cuenta una carpeta entera con todo su contenido:

.

nombrecarpeta/

6. CLONAR UN PROYECTO.

Para clonar algún proyecto simplemente vamos a GitHub, buscamos el link del proyecto que queremos clonar y desde la línea de comando, ejecutamos un:

git clone (enlace del repositorio)

Primero copiarás la URL del respositorio remoto que deseas clonar.

Luego abrimos nuestro Powershell, para situarte sobre la carpeta de tu proyecto que quieras clonar.

Creamos directamente una carpeta con el nombre del proyecto que estás clonando, o cualquier otro nombre que te parezca mejor para este repositorio.

Nos situamos dentro de esa carpeta (cd) y desde ella lanzamos el comando para hacer el clon, que sería algo como esto:

git clone https://github.com/rrbnz/proyecto01.git

Ya simplemente se nos clonaría el repositorio remoto en la nueva carpeta y ya haríamos lo mismo de subirlo a otro repositorio en nuestro perfil, o también nos vendría bien si cambiamos de ordenador porque estemos trabajando desde casa y desde la oficina con distintos portátiles.

Sin embargo, tenemos también la opción de “descargar” el repositorio que, a primera vista parece más fácil, pero te voy a explicar porque no es más fácil que clonar y que desventajas tiene.

Descargar un repositorio no te permite algunas de las utilidades interesantes de clonarlo, como:

- No crea un repositorio Git en local con los cambios que el repositorio remoto ha tenido a lo largo del tiempo. Es decir, te descargas el código, pero nada más.

- No podrás luego enviar cambios al repositorio remoto, una vez los hayas realizado en local.

En resumen, no podrás usar en general las ventajas de Git en el código descargado. Así que es mejor clonar, ya que de todas formas no requiere mucha dificultad hacerlo.

REFERENCIAS

- HTTPS://GITHUB.COM/ACADEMIACODER/GITGITHUB-GUIA-RAPIDA

- HTTPS://MEDIUM.COM/@STHEFANY/PRIMEROS-PASOS-CON-GITHUB-7D5E0769158C

- HTTPS://DESARROLLOWEB.COM/MANUALES/MANUAL-DE-GIT.HTML

- HTTPS://GIST.GITHUB.COM/WIMARBUENO/5516DE693639FBDCCAC2