

Introducción a Git

Guía básica para el despliegue de proyectos con Git



git

Git es un software diseñado como **sistema de control de versiones** muy utilizado por programadores para guardar los archivos creados y en especial el historial de cambios en ellos (fecha de modificación, autor de la modificación, líneas de código que se agregaron/modificaron, etc.).

Dicha colección de archivos y su historial de cambios se almacenan en **repositorios**. Hay dos tipos de repositorios: los **locales** y los **remotos**.

Un repositorio **local** se ubica en un directorio oculto llamado **.git** dentro de la carpeta raíz de un proyecto. Un repositorio **remoto** se ubica en un servidor y sirve como respaldo, pero también para distribuir a colaboradores. Las plataformas más conocidas para repositorios remotos son **GitHub**, **Bitbucket** y **GitLab**.

GitHub es muy popular en la comunidad Open Source dado que ofrece repositorios públicos de forma gratuita, lo que es aprovechado por programadores para distribuir sus desarrollos de forma libre, abierta y gratuita. En Enero de 2019 habilitó crear repositorios privados ilimitados gratuitamente (máximo hasta 3 colaboradores).



GitHub



Bitbucket GitLab

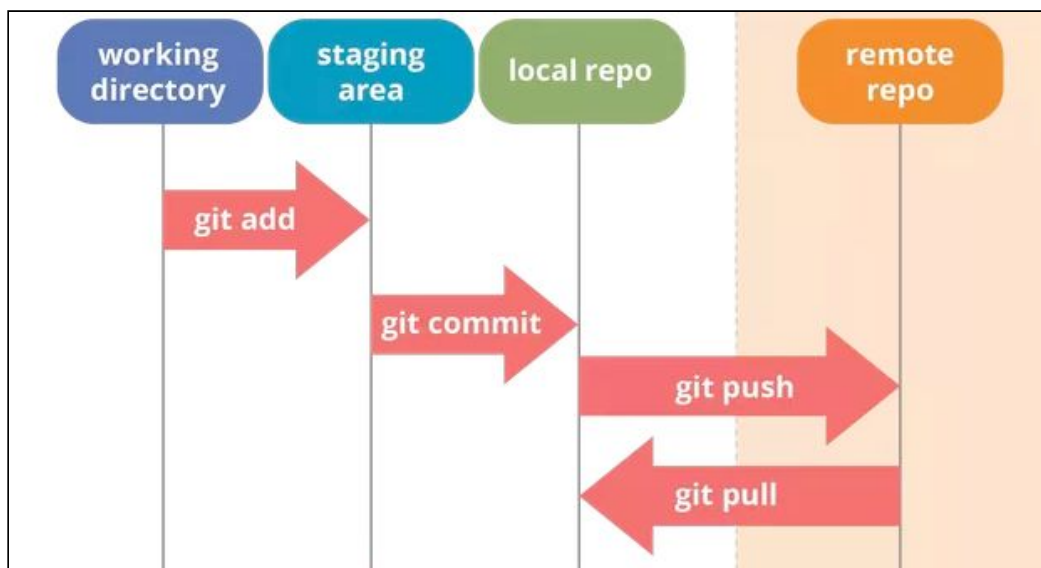
Bitbucket y **GitLab** son por otro lado son más utilizados para proyectos cerrados dado que ofrecen crear repositorios privados al igual que públicos, resultando ideal para tanto para desarrollos individuales como equipos de trabajo.

Trabajar con **Git** elimina la necesidad de usar clientes y conexiones FTP o SFTP para subir proyectos a un servidor web dado que una de sus características esenciales es ser un sistema distribuido mediante la **clonación** de repositorios. También elimina la necesidad de adjuntar y enviar archivos por email.

Son muchas las ventajas que se le atribuyen a la gestión de proyectos con **Git** como sistema de control de versiones, destacándose entre ellas:

1. Facilita el **trabajo colaborativo** ya que distintos programadores pueden estar editando el mismo archivo, o versiones distintas del mismo, y todos los cambios serán reflejados en el archivo final.
2. Permite **regresar a versiones anteriores** de forma sencilla y rápida. En caso de haber realizado cambios negativos en un proyecto, volver a la última versión estable es un simple comando, que retrocede a su estado previo todos los cambios realizados en la última modificación.
3. Empezar a **trabajar desde otro entorno** es tan fácil como "clonar" el proyecto a un nuevo espacio.

El siguiente gráfico detalla el flujo de trabajo que implica utilizar **Git** como sistema de versionado:



Para entenderlo mejor...

- **Working directory (directorio de trabajo):** es la carpeta raíz/principal donde se albergan todos los archivos y sub-directorios que componen el proyecto.
- **Staging area (área de preparación):** es un índice en donde se detallan los archivos creados o modificados en el proyecto.
- **Local repository (repositorio local):** es el espacio local en donde queda almacenada una copia de cada uno de los archivos manipulados, junto con el historial de los cambios realizados.
- **Remote repository (repositorio remoto):** es el espacio remoto en un servidor, en donde se guarda una copia del repositorio local (donde están los archivos manipulados y los cambios realizados).

Primeros pasos

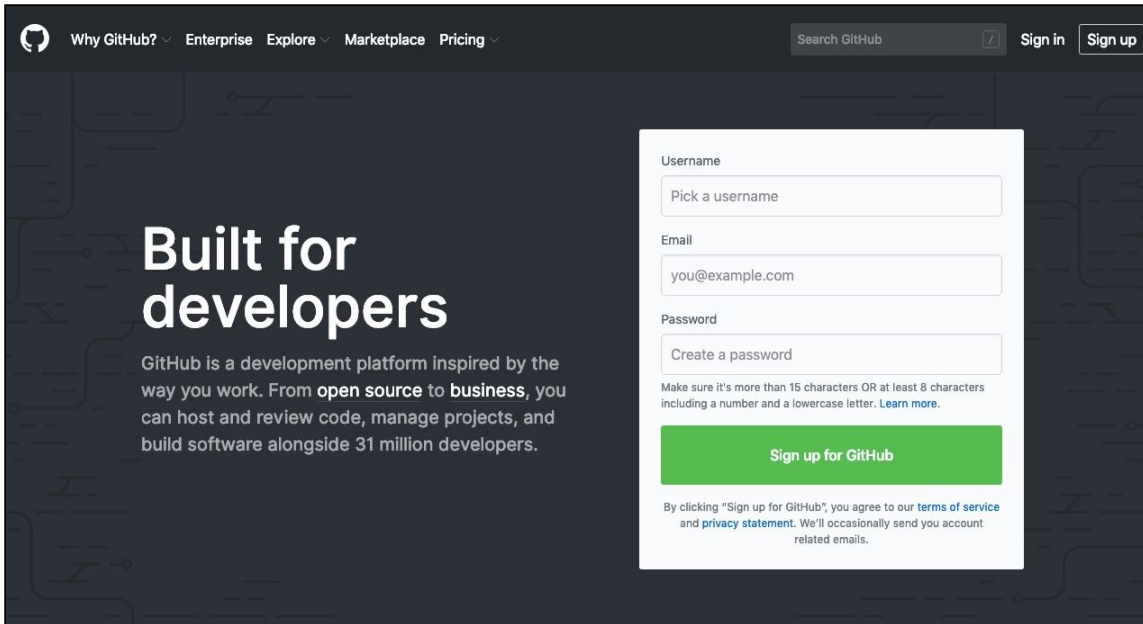
En la presente guía se pretende mostrar los pasos básicos para poder llevar a cabo un adecuado control de código de un proyecto. La práctica será necesaria para un manejo fluido de las instrucciones comunes. Los requisitos para implementar **Git** son:

1. Crear una cuenta en alguna de las plataformas (ej: GitHub)
2. Crear en dicha plataforma un repositorio por cada proyecto
3. Instalar en el equipo propio el software **Git**
4. Conocer y ejecutar los comandos básicos de **Git**

La clave de todo se centra en adquirir una simple pero estricta serie de pasos en la programación de un proyecto, los cuales conformarán un flujo de trabajo que con la práctica se volverá una rutina disciplinaria y productiva.

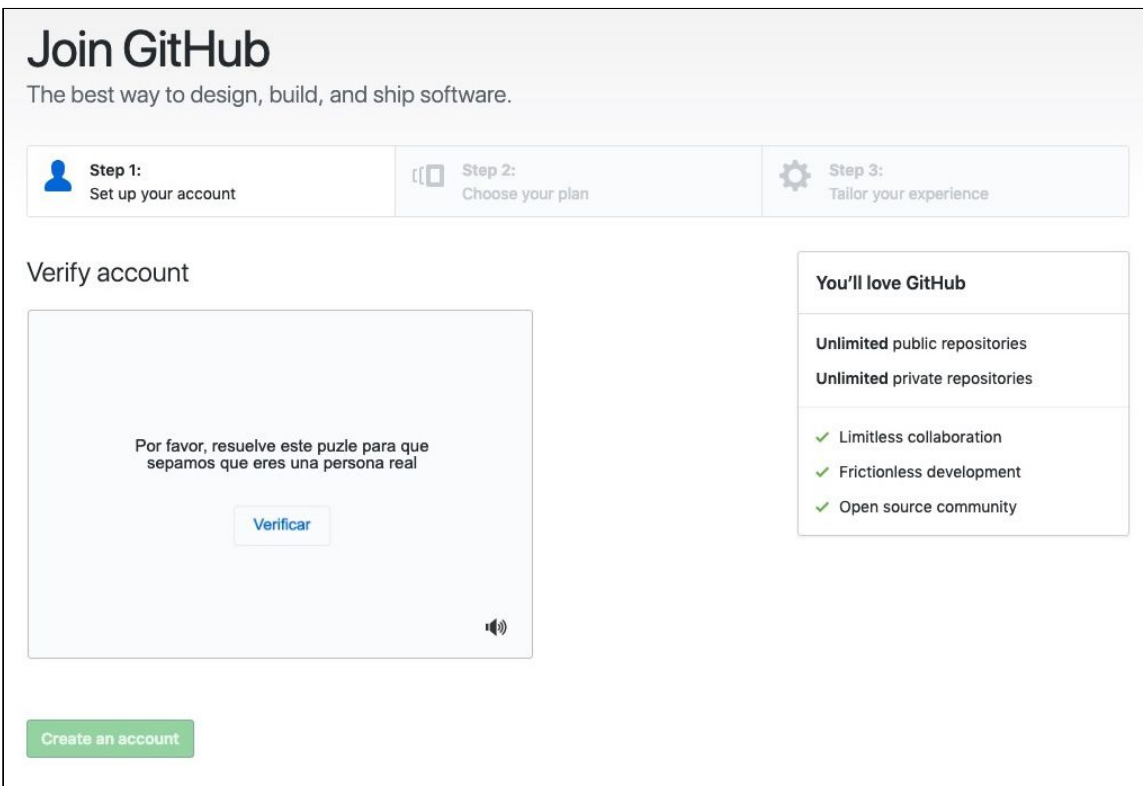
1. Creación de una cuenta de usuario en GitHub

Ingresa a github.com y proceder a registrarse mediante la opción **Sign up for GitHub**.



The screenshot shows the GitHub homepage with a dark background. On the left, the text "Built for developers" is prominent, followed by a description of GitHub as a development platform. On the right, a white sign-up form is displayed. The form includes fields for "Username" (with a placeholder "Pick a username"), "Email" (with a placeholder "you@example.com"), and "Password" (with a placeholder "Create a password"). Below the password field, there is a note: "Make sure it's more than 15 characters OR at least 8 characters including a number and a lowercase letter. [Learn more.](#)". A green button labeled "Sign up for GitHub" is at the bottom of the form. Below the button, a small disclaimer states: "By clicking 'Sign up for GitHub', you agree to our [terms of service](#) and [privacy statement](#). We'll occasionally send you account related emails."


Especificar un nombre de usuario único pero "recordable", ya que será necesario para configurar la conexión entre el repositorio **local** y el **remoto** ubicado en **GitHub**.



The screenshot shows the "Join GitHub" page. At the top, it says "Join GitHub" and "The best way to design, build, and ship software." Below this, there are three steps: "Step 1: Set up your account" (active), "Step 2: Choose your plan", and "Step 3: Tailor your experience". The "Verify account" section is highlighted. It contains a large box with the text "Por favor, resuelve este puzle para que sepamos que eres una persona real" and a "Verificar" button. At the bottom left of the page, there is a green button labeled "Create an account". On the right side, there is a section titled "You'll love GitHub" with a list of benefits: "Unlimited public repositories", "Unlimited private repositories", "Limitless collaboration", "Frictionless development", and "Open source community".

Choose your plan

With tools developers love and the world's largest open source community, there's no wrong choice.



Free

The basics of GitHub for every developer


\$0

per month

Includes:

- ∞ Unlimited public and private repositories
- ✓ 3 collaborators for private repositories
- ✓ Issues and bug tracking
- ✓ Project management

Are you a [student](#)? Get access to the best developer tools for free with the [GitHub Student Developer Pack](#).



Pro

Pro tools for developers with advanced requirements

\$7

per month

[\(view in ARS\)](#)

Includes:

- ∞ Unlimited public and private repositories
- ∞ Unlimited collaborators
- ✓ Issues and bug tracking
- ✓ Project management
- ✓ [Advanced tools and insights](#)

☐ **Help me set up an organization next**

Organizations are separate from personal accounts and are best suited for businesses who need to manage permissions for many employees. [Learn more about organizations](#)

☐ **Send me updates on GitHub news, offers, and events**

Unsubscribe anytime in your email preferences. [Learn more](#)


[Continue](#)


Basta con elegir el plan **Free** para acceder a todos los servicios y funcionalidades que ofrece **GitHub**.


NOTA: Si se crea la cuenta con una dirección de e-mail educativa (ej. @uba.ar, @itba.edu.ar, @utn.edu.ar, etc) es posible acceder al **GitHub Student Developer Pack** (<https://education.github.com/pack>) el cual ofrece acceso gratuito a varios servicios premium de diferentes plataformas de desarrollo profesional.

Welcome to GitHub

You'll find endless opportunities to learn, code, and create, @eant-online.

 **Completed**
Set up a personal account

 **Step 2:**
Choose your plan

 **Step 3:**
Tailor your experience

How would you describe your level of programming experience?

☐ Very experienced
 ☐ Somewhat experienced
 ☐ Totally new to programming

What do you plan to use GitHub for? (check all that apply)

☐ Project Management
 ☐ School projects
 ☐ Design
☐ Research
 ☐ Development
 ☐ Other (please specify)

Which is closest to how you would describe yourself?

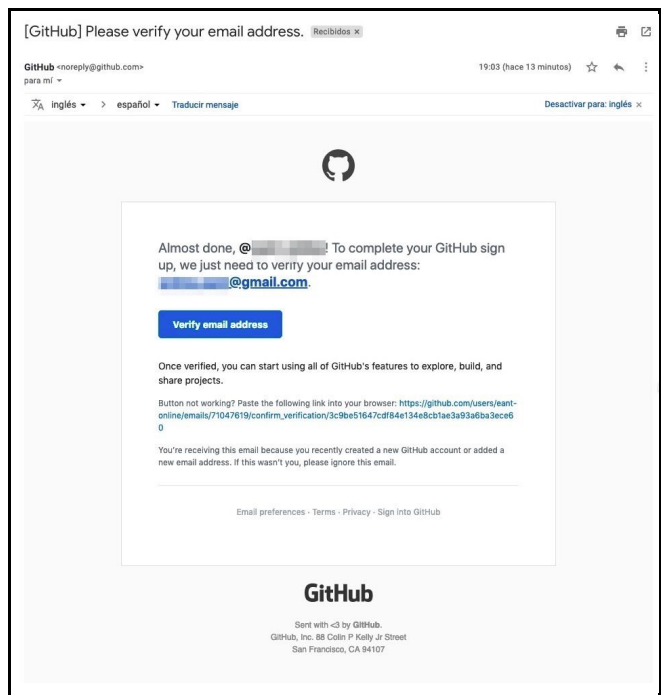
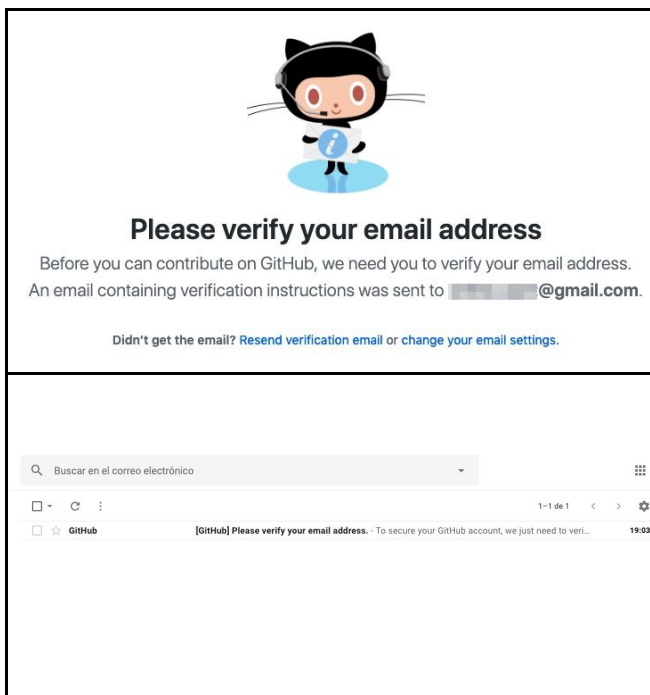
☐ I'm a professional
 ☐ I'm a student
 ☐ I'm a hobbyist
☐ Other (please specify)

What are you interested in?

e.g. tutorials, android, ruby, web-development, machine-learning, open-source

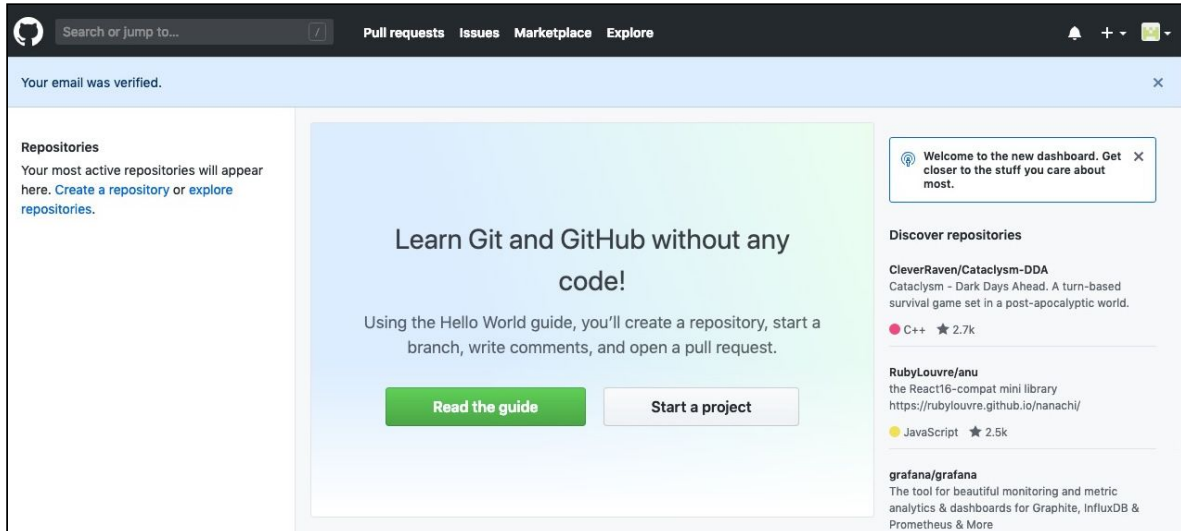
[Submit](#)
[skip this step](#)

Puede omitirse el paso 3 sin problema. Luego de ello, chequear la casilla de e-mail para activar la cuenta de **GitHub**.



2. Creación del primer repositorio en GitHub

Una vez ya confirmada la cuenta de usuario se puede proceder a crear el primer repositorio ingresando a **Start a project**.



Especificar el **nombre** del repositorio y configurarlo como **público** o **privado**. También indicar que **NO** se quiere inicializar con un **README** (será creado manualmente después).

Create a new repository

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

Owner

Repository name *

eant-online

/ MiProyecto

Great repository names are short and memorable. Need inspiration? How about **redesigned-parakeet**?

Description (optional)

Proyecto de prueba para aprender 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

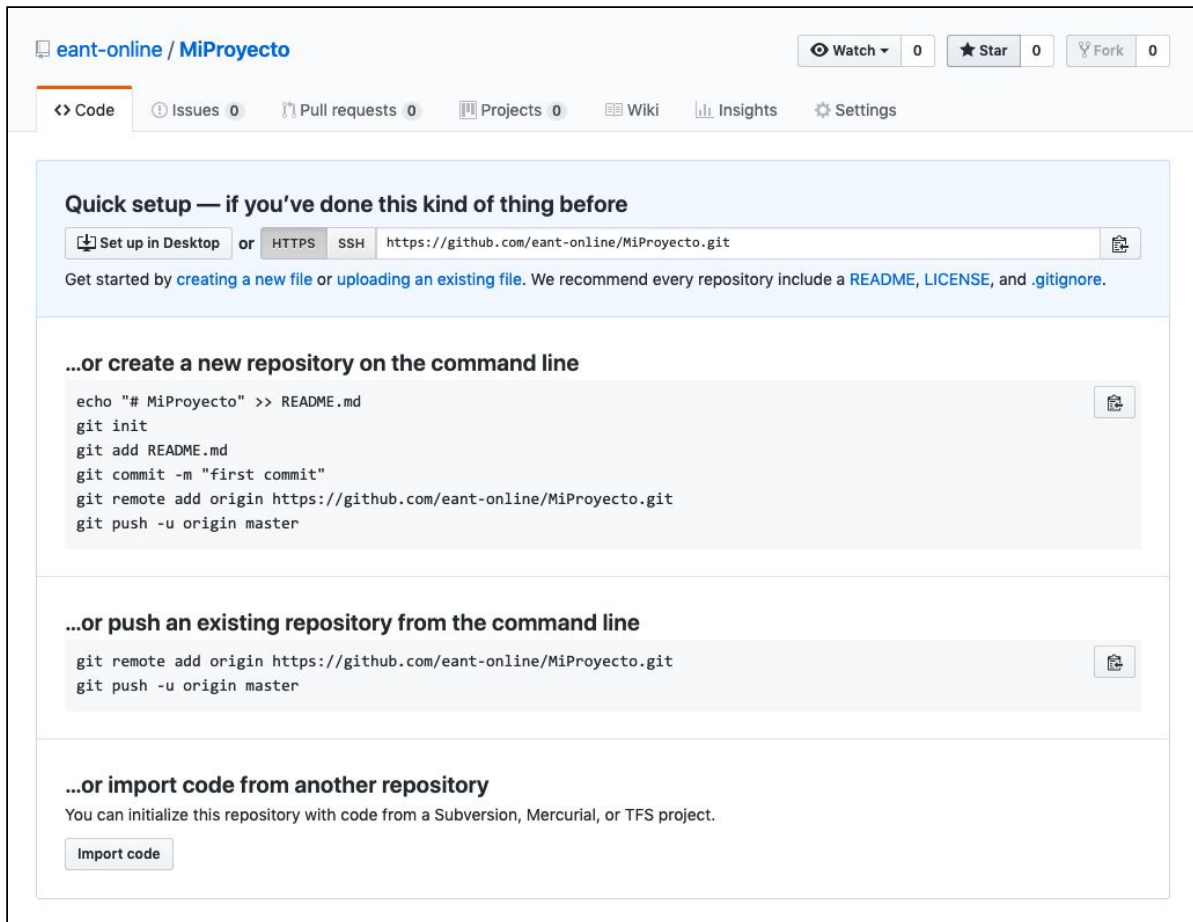
This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.

Add .gitignore: None

Add a license: None

Create repository

Una vez creado el primer **repositorio remoto** ya se está en condiciones de crear una conexión con un **repositorio local**.



Y listo, se ha creado el primer repositorio remoto. Ahora es turno de preparar el **entorno local** para vincularlo al **repositorio remoto**. Para lograr esto último, será necesario tomar nota del **primer comando** que se detalla en la opción "Estoy comenzando desde cero":

```
git clone https://github.com/MIUSUARIO/MIREPOSITORIO.git
```

Este comando (con los datos específicos del usuario y repositorio de cada uno) permitirá **clonar** el **repositorio remoto** en un **repositorio local** para mantener **sincronizados** y **distribuidos** los archivos que componen al proyecto.

Es momento de incorporar **Git** y configurar el **entorno local** en la PC donde se desarrollará el proyecto.

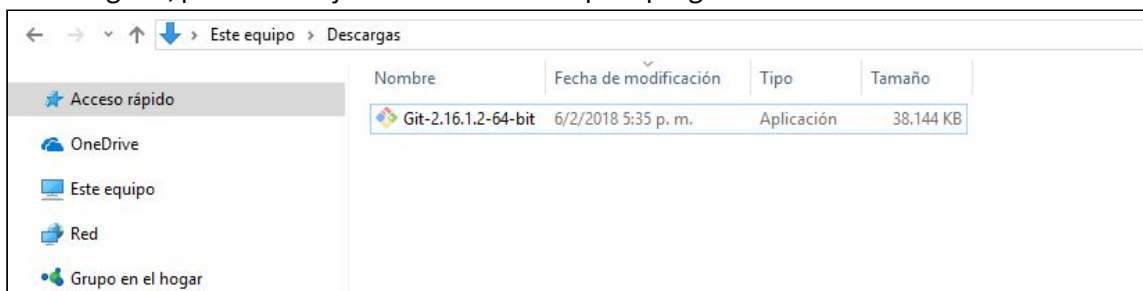
3. Instalación de Git en el equipo local

Ingresa a git-scm.com, y descargar el software **Git** correspondiente al sistema operativo sobre el que se trabajará (GNU/Linux, Mac o Windows).

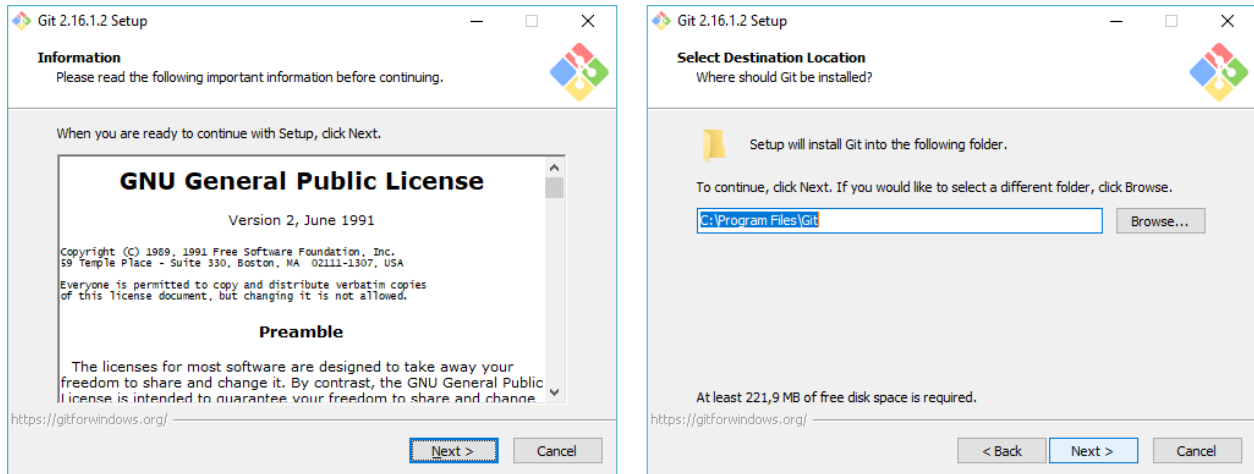
Esta guía se basa en la instalación sobre Windows por presentar algunas diferencias frente a GNU/Linux y Mac, pero al momento de su utilización la misma es idéntica en todas las plataformas.

The screenshot shows the Git website homepage. The main heading is "git --local-branching-on-the-cheap". Below this, there's a search bar and a description of Git as a free and open source distributed version control system. The page is divided into sections: About, Documentation, Downloads, and Community. The "Downloading Git" section is highlighted, showing a large download arrow and text indicating the latest version (2.16.1) is available for Windows. It lists various download options: Git for Windows Setup (32-bit and 64-bit), Git for Windows Portable (32-bit and 64-bit), and the source code.

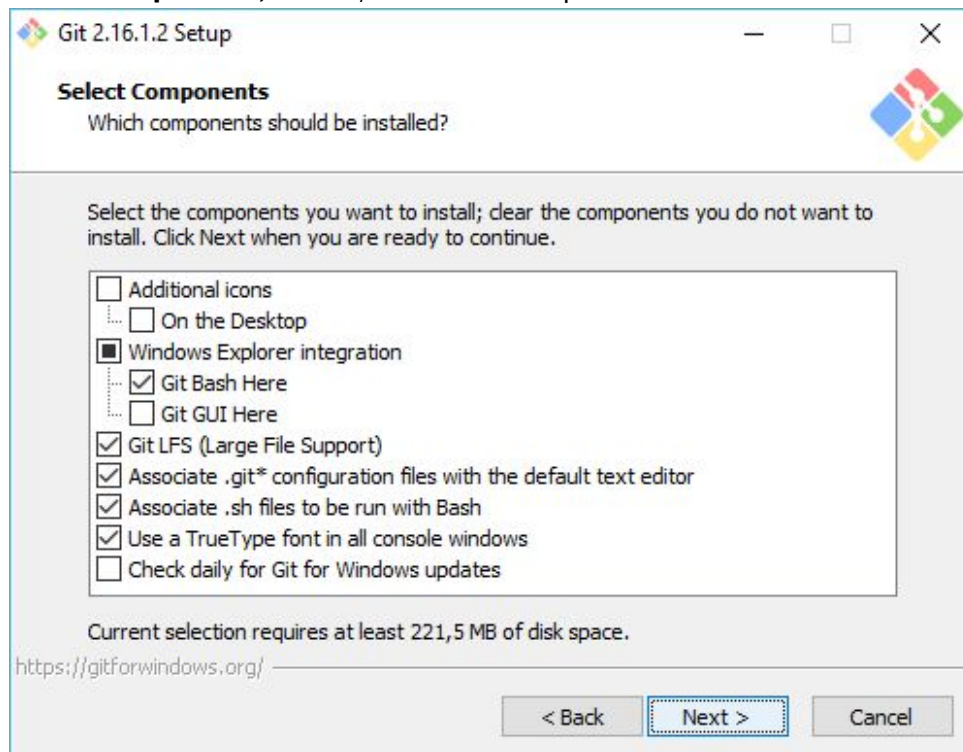
Una vez descargado, proceder a ejecutarlo como cualquier programa de instalación.



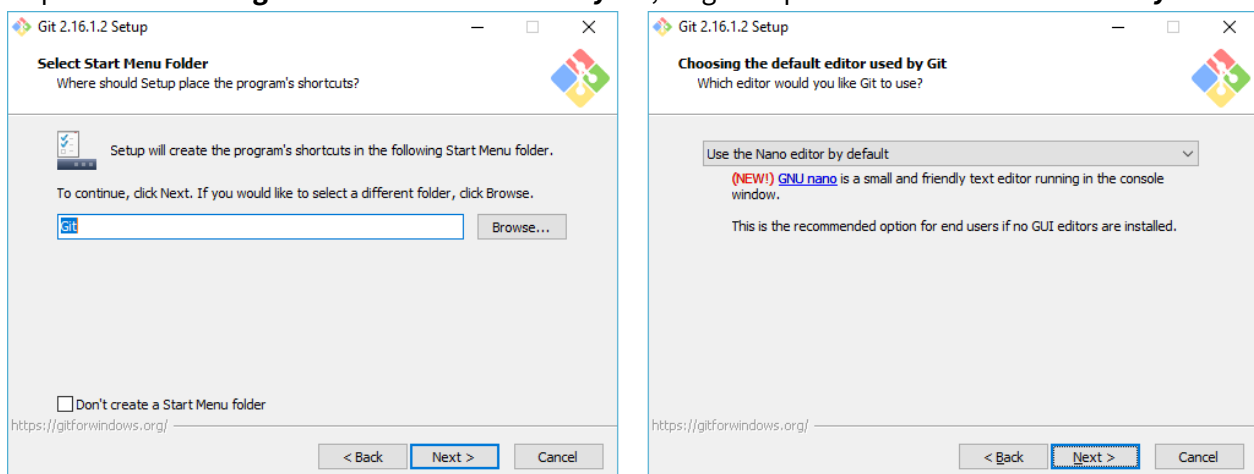
Acá se detalla paso a paso las pantallas del proceso de instalación, en donde se indicarán las opciones mínimas e indispensables para una una instalación óptima.



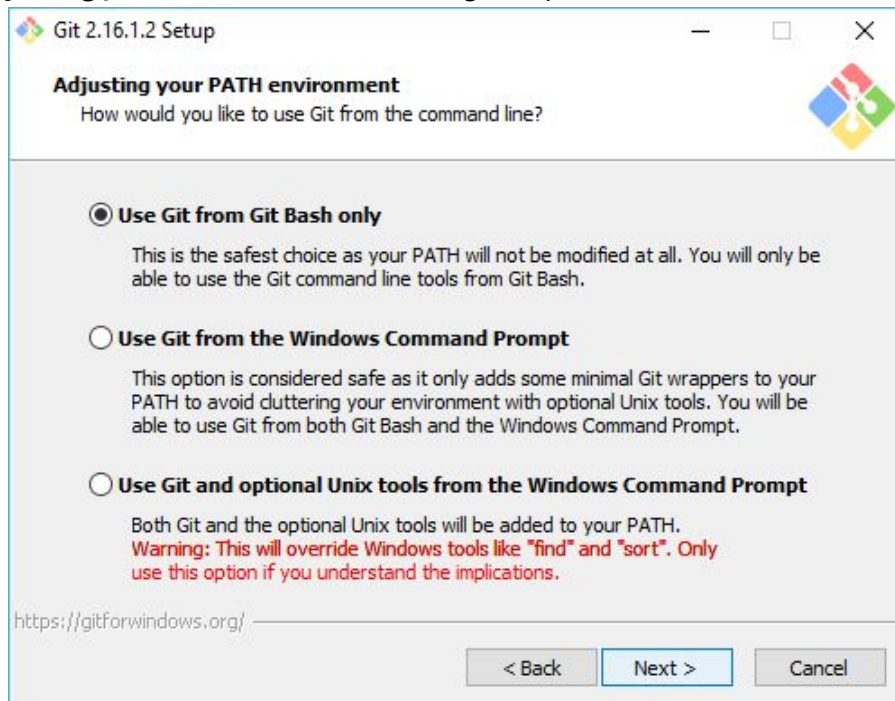
En la pantalla **Select Components**, marcar/desmarcar las opciones indicadas.



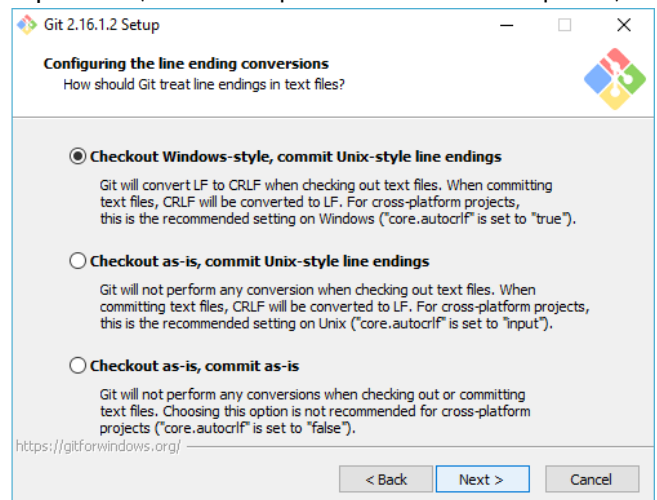
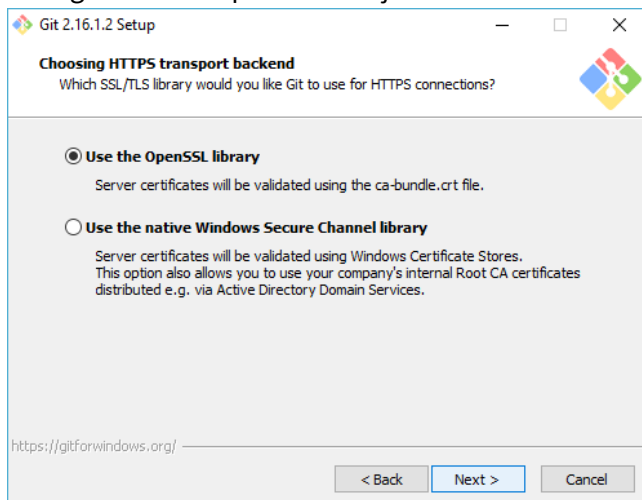
En la pantalla **Choosing the default editor used by Git**, elegir la opción "Use the Nano editor by default".



En la pantalla **Adjusting your PATH environment**, elegir la opción "Use Git from Git Bash only".

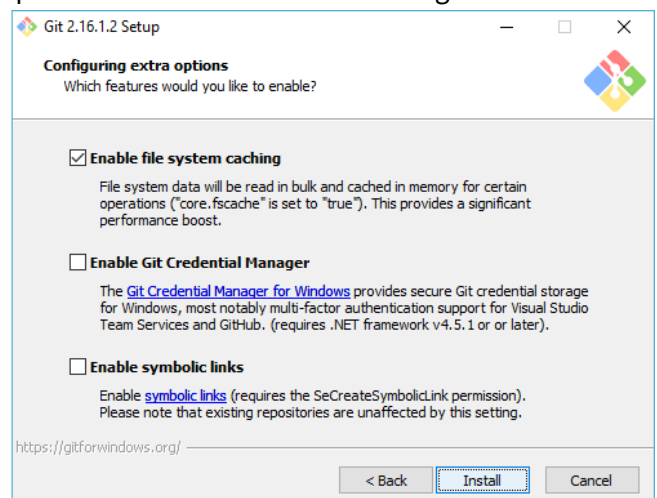
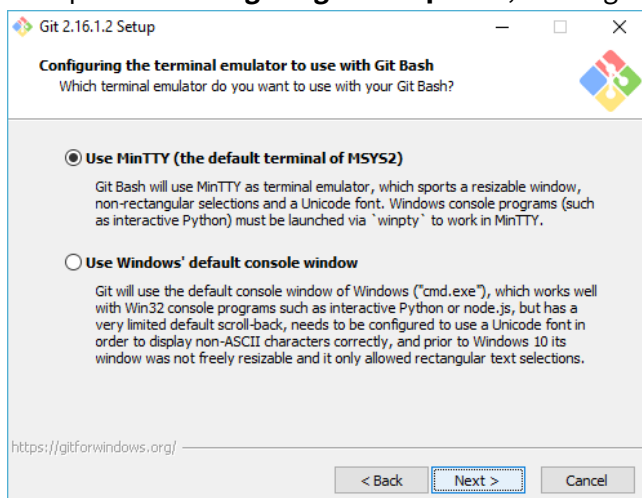


Las siguientes dos pantallas dejarlas sin cambios en sus opciones (corroborar que coincidan con las capturas).

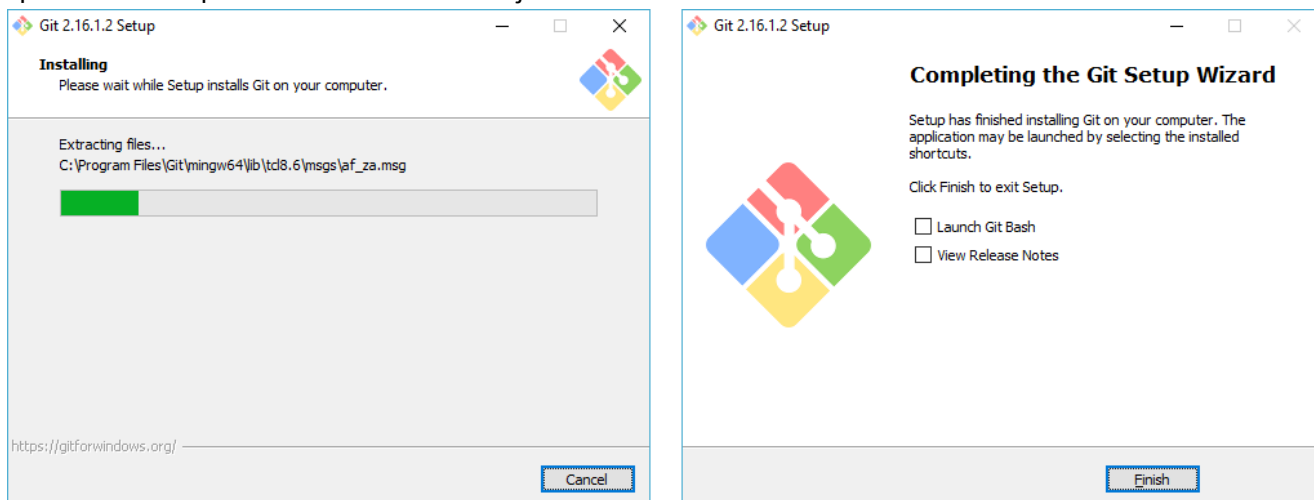


En la pantalla **Configuring the terminal emulator to use with Git Bash**, elegir la opción "Use MinTTY...".

En la pantalla **Configuring extra options**, **NO** elegir la opción "Enable Git Credential Manager".



Proceder a iniciar la instalación con las opciones configuradas. Al culminar el proceso, destildar cualquier opción ofrecida para finalizar sin abrir ni ejecutar nada más adicional.



...y listo, ya se ha incorporado **Git** al sistema local. El siguiente paso es aprender a crear **repositorios locales** y vincularlos con **repositorios remotos**.

4. Clonación de un repositorio de GitHub

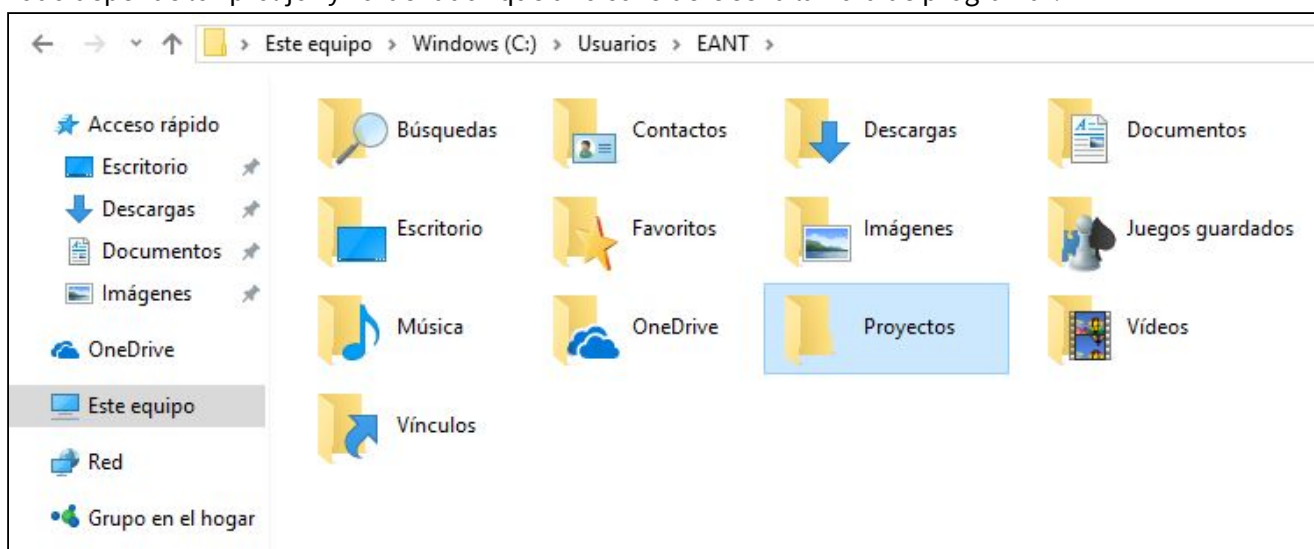
Los siguientes pasos muestran un modelo más o menos "común" para la creación de **repositorios locales** y su vinculación con **repositorios remotos**.

Con la práctica y asimilación de trabajar con **Git**, uno podrá crear/adaptar su propio flujo de trabajo a la forma más cómoda y productiva que mejor resulte.

Se puede empezar creando un directorio llamado Proyectos dentro de la carpeta de usuario del sistema. Allí se crearán un subdirectorio por cada proyecto vinculado a un **repositorio local** y **remoto**.

Otra opción, por ejemplo, si se trabaja con **PHP** sobre **Apache**, podría usarse la carpeta **htdocs** para crear los proyectos y vincularlos a su respectivo **repositorio local** y **remoto**.

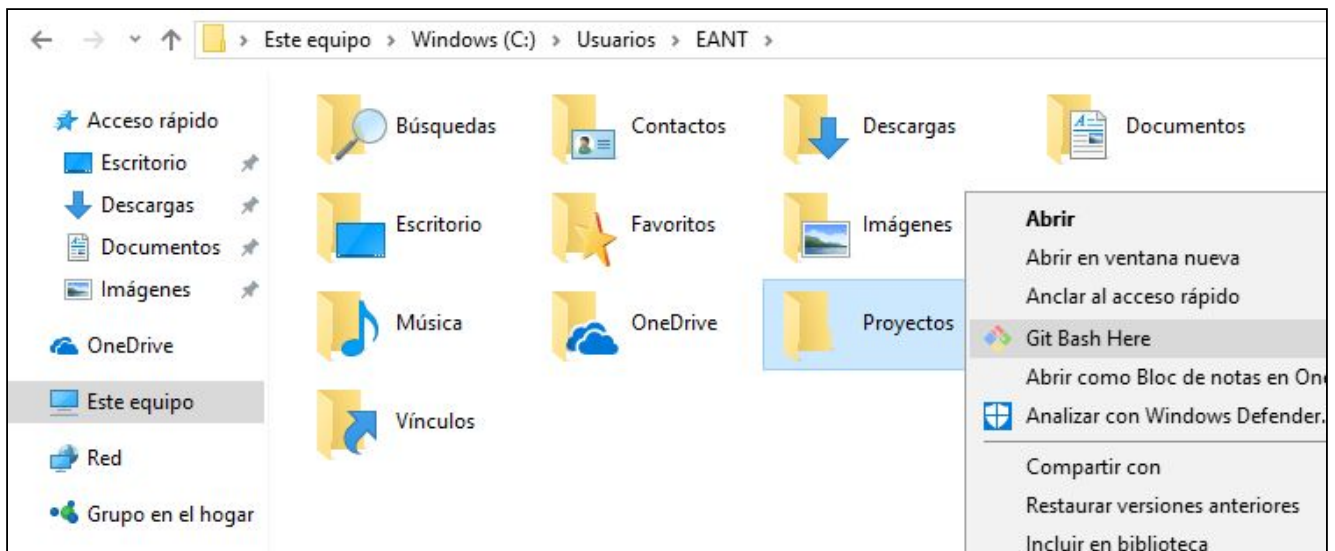
Todo depende lo "prolijo" y "ordenado" que uno considere ser a la hora de programar.



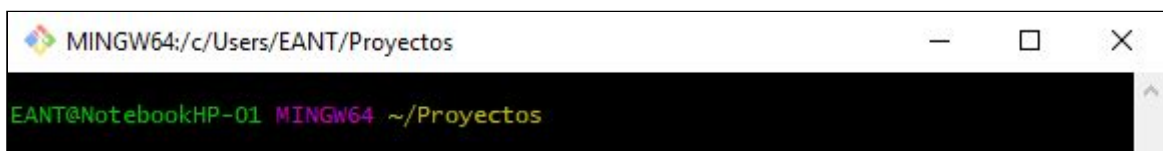
Una vez definido el directorio para alojar proyectos, es momento de crear un **repositorio local** y vincularlo con el **repositorio remoto** creado en **GitHub**, todo ello mediante comandos de **Git**.

Hay dos formas de poder abordarlo:

1) Haciendo click derecho sobre la carpeta de Proyectos y elegir la opción **Git Bash Here**.

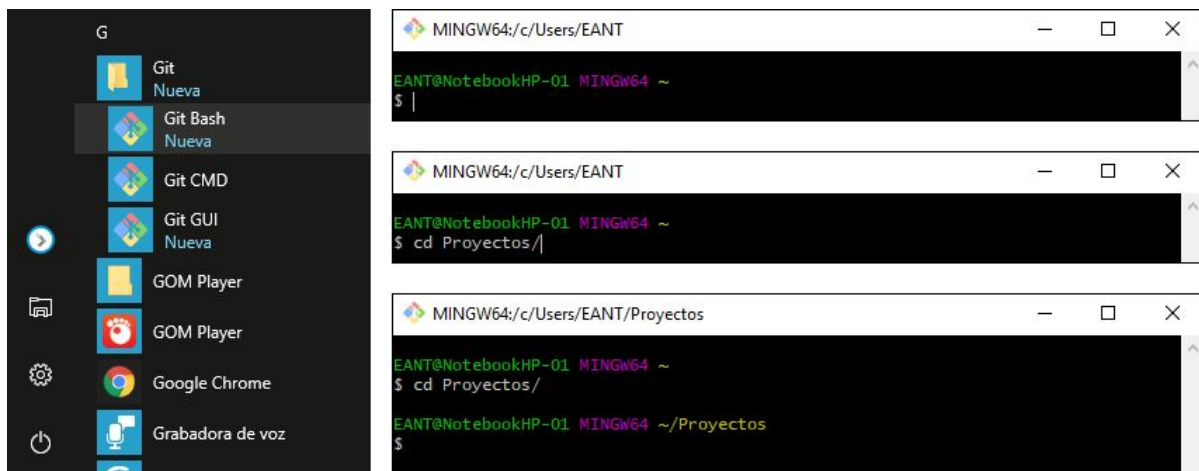


Se abrirá una consola/terminal que indicara que se esta ubicado dentro del directorio de Proyectos



2) Desde el Menú Inicio, abriendo la aplicación **Git Bash**, se abrirá una **consola/terminal** que indicara que se está ubicado dentro de la carpeta de usuario.

Luego habrá que ingresar al directorio de Proyectos mediante el comando `cd Proyectos/`.



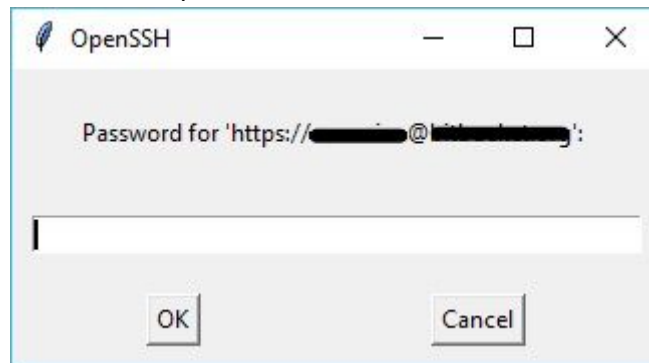
En cualquiera de los dos casos se puede anticipar que trabajar con **Git** se hace a través de la **consola/terminal** del sistema, sin depender de programas con interfaz gráfica y solo mediante la ejecución de comandos que ordenen hacer algo.

Como se indicó anteriormente, esta guía pretende mostrar los comandos básicos para armar una rutina de trabajo que permita entender la dinámica de **Git** y eventualmente sirva para profundizar en su uso profesional.

Proceder a la creación de un **repositorio local** y su vinculación a uno **remoto** se hace mediante el comando `git clone ...` que se obtuvo desde **GitHub**.

```
EANT@NotebookHP-01 MINGW64 ~/Proyectos
$ git clone https://github.com/amosciana/MiProyecto.git
Cloning into 'MiProyecto'...
```

Al ejecutar el comando `git clone ...` saldrá un cuadro solicitando la contraseña de la cuenta de **GitHub** para autorizar la clonación del repositorio indicado.



Una vez ingresada correctamente la contraseña, se creará una carpeta con el nombre del **repositorio remoto** y en ella se descargarán todos los archivos del mismo (o bien indicará que el repositorio está vacío).

```
EANT@NotebookHP-01 MINGW64 ~/Proyectos
$ git clone https://github.com/amosciana/MiProyecto.git
Cloning into 'MiProyecto'...
warning: You appear to have cloned an empty repository.

EANT@NotebookHP-01 MINGW64 ~/Proyectos
$
```



Antes de empezar a trabajar sobre el repositorio local, desde la consola hay que ingresar al repositorio mediante el comando `cd NombreRepo/`.

```
EANT@NotebookHP-01 MINGW64 ~/Proyectos
$ git clone https://github.com/amosciana/MiProyecto.git
Cloning into 'MiProyecto'...
warning: You appear to have cloned an empty repository.

EANT@NotebookHP-01 MINGW64 ~/Proyectos
$ cd MiProyecto/
```

Aparecerá una aclaración indicando `(master)` indicando que el **repositorio local** se encuentra en la **rama maestra**. Las **ramas** son una funcionalidad más avanzada del trabajo con **Git**, por lo cual ahora solo servirá para corroborar que los comandos se ejecutan sobre el **repositorio local**.

```
EANT@NotebookHP-01 MINGW64 ~/Proyectos
$ git clone https://github.com/amosciana/MiProyecto.git
Cloning into 'MiProyecto'...
warning: You appear to have cloned an empty repository.

EANT@NotebookHP-01 MINGW64 ~/Proyectos
$ cd MiProyecto/

EANT@NotebookHP-01 MINGW64 ~/Proyectos/MiProyecto (master)
$
```


Lo siguiente es configurar el usuario autorizado para realizar y subir los cambios en el proyecto hacia el repositorio remoto. Dicha configuración se especifica mediante los siguientes comandos:

```
git config user.email "miusuario@miemail.com"
git config user.name "Nombre de Usuario"
```

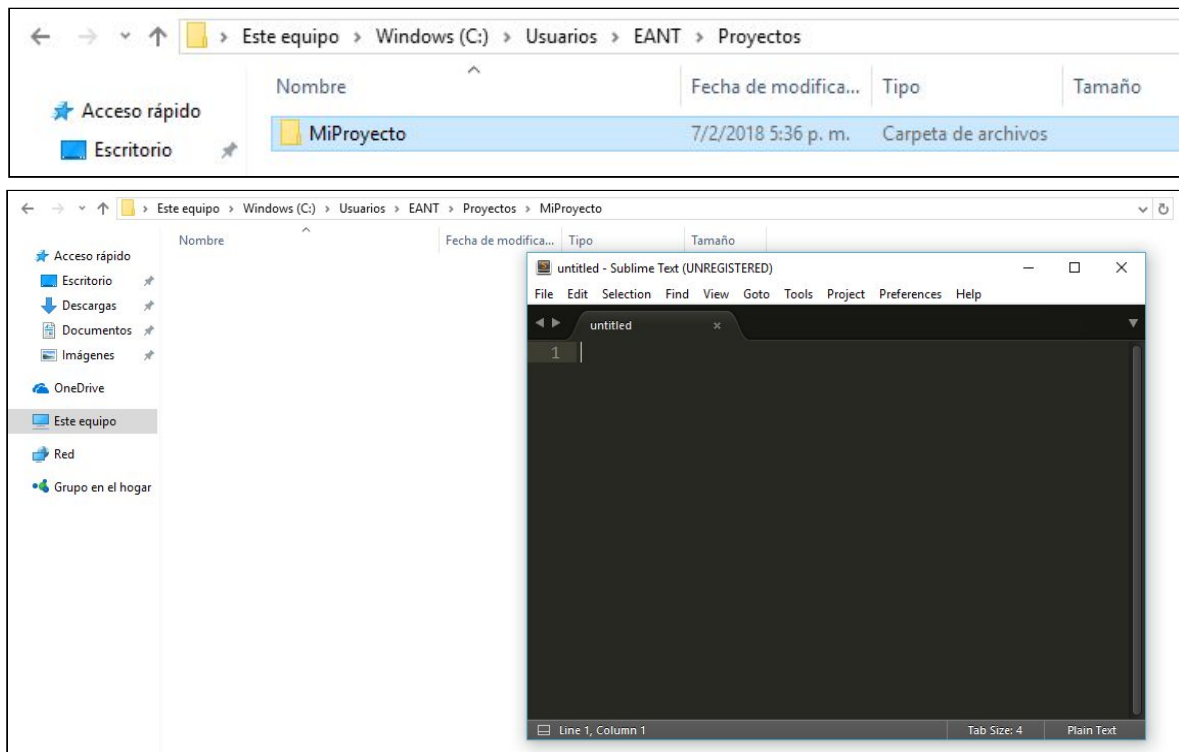
```
EANT@NotebookHP-01 MINGW64 ~/Proyectos/MiProyecto (master)
$ git config user.email "miusuario@miemail.com"
```

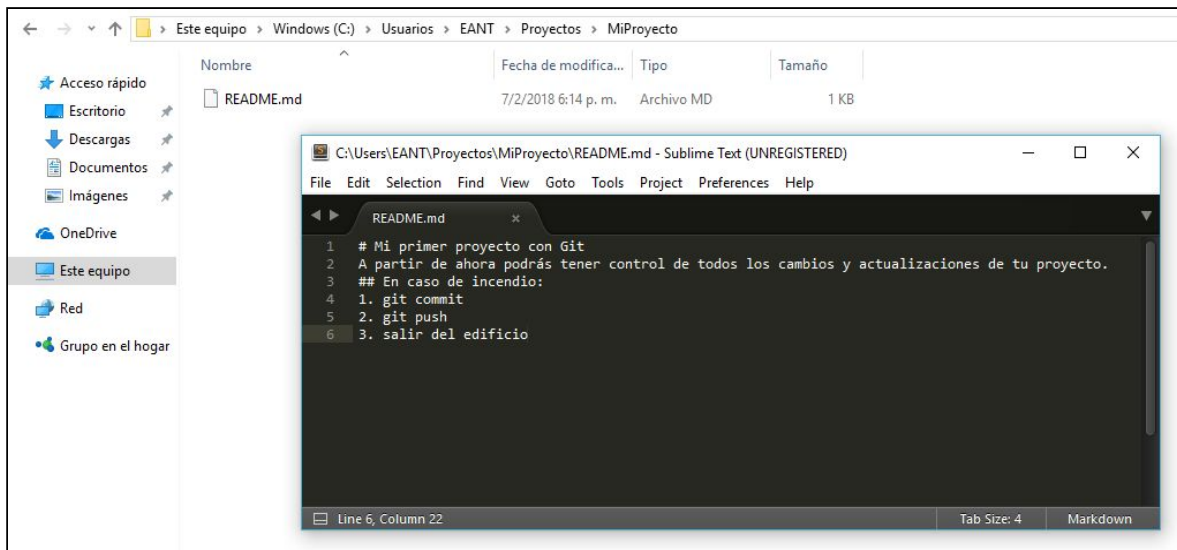
```
EANT@NotebookHP-01 MINGW64 ~/Proyectos/MiProyecto (master)
$ git config user.name "Cosme Fulanito"
```

Esta configuración solo aplica para el repositorio actual. Al trabajar con otro repositorio deberá establecerse nuevamente. Es posible definir de forma predeterminada email/usuario mediante los siguientes comandos:

```
git config --global user.email "miusuario@miemail.com"
git config --global user.name "Nombre de Usuario"
```

Ahora está todo listo para empezar a trabajar sobre el proyecto. Un primer paso muy recomendado es crear una archivo llamado **README.md**, donde se coloca una breve descripción del proyecto, autor, recursos, etc.





Volviendo a la consola se puede proceder a subir el primer cambio en el repositorio.

```
EANT@NotebookHP-01 MINGW64 ~/Proyectos/MiProyecto (master)
$ git status
```

Con el comando `git status` podemos ver estado del **repositorio local**.

```
EANT@NotebookHP-01 MINGW64 ~/Proyectos/MiProyecto (master)
$ git status
On branch master

No commits yet

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)
EANT@NotebookHP-01 MINGW64 ~/Proyectos/MiProyecto (master)
$ |
```

En este caso arroja dos cosas: **"No commits yet"** y **"Untracked files"**.

Los **commits** son los cambios realizados en el **repositorio local** que están confirmados para ser aplicados al **repositorio remoto** de forma permanente.

Los archivos **untracked** son aquellos que difieren de los que están en el repositorio. Puede ser por tratarse de nuevos archivos o modificaciones hechas a los archivos.

```
EANT@NotebookHP-01 MINGW64 ~/Proyectos/MiProyecto (master)
$ git add README.md
```

Mediante el comando `git add NombreDelArchivo` se agrega el archivo nuevo o modificado.

Para no hacer un `git add NombreDelArchivo` por cada archivo nuevo/modificado, se puede ejecutar `git add -A` y así agregar varios cambios en un solo comando. Hay que usarlo con cuidado, corroborando que realmente se quieren agregar todos los archivos manipulados.

```
EANT@NotebookHP-01 MINGW64 ~/Proyectos/MiProyecto (master)
$ git status
On branch master

No commits yet

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

        new file:   README.md

EANT@NotebookHP-01 MINGW64 ~/Proyectos/MiProyecto (master)
$
```

Si se ejecuta nuevamente `git status` luego de un `git add ...` se puede ver que los archivos antes marcados como **untracked** ahora se muestran como **"new file: ..."**. Los archivos se encuentran en la **staging area**, es decir que están marcados para integrarse al **repositorio local**.

```
EANT@NotebookHP-01 MINGW64 ~/Proyectos/MiProyecto (master)
$ git commit -m "Haciendo mi primer commit"
```

Una vez que se han marcado, el siguiente paso es integrarlos al **repositorio local** mediante el comando:

```
git commit -m "Un comentario sobre lo trabajado"
```

```
EANT@NotebookHP-01 MINGW64 ~/Proyectos/MiProyecto (master)
$ git commit -m "Haciendo mi primer commit"
[master (root-commit) 5e53afb] Haciendo mi primer commit
1 file changed, 6 insertions(+)
create mode 100644 README.md
```

Es de mucha utilidad poner comentarios ya que permitirá encontrar viejos **commits** a medida que se avance en el proyecto. Una vez hecho el **commit** sobre el **repositorio local**, la consola resume la cantidad de cambios que hay entre el **commit actual** y el **anterior** (o bien desde el estado inicial del repositorio).

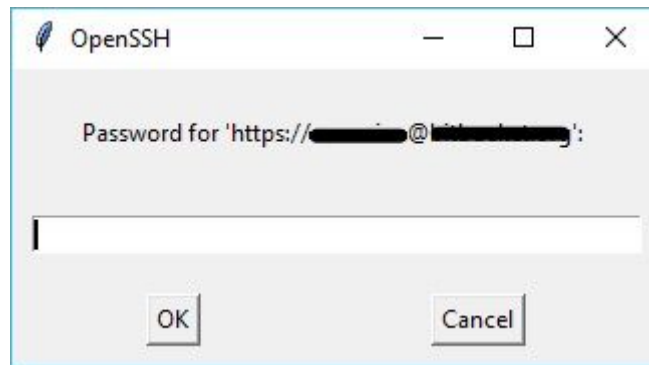
Puede continuar trabajándose en el proyecto, agregando/cambiando archivos y repitiendo el proceso de `git add ...` y luego `git commit -m "Un comentario sobre lo trabajado"`.

El paso final es subir/integrar los cambios del **repositorio local** sobre el **remoto** ubicado en **GitHub** mediante el comando:

```
git push origin master
```

```
EANT@NotebookHP-01 MINGW64 ~/Proyectos/MiProyecto (master)
$ git push origin master
```

Nuevamente solicitará la contraseña del usuario de **GitHub** para así autorizar la subida de los cambios realizados hacia el **repositorio remoto**.



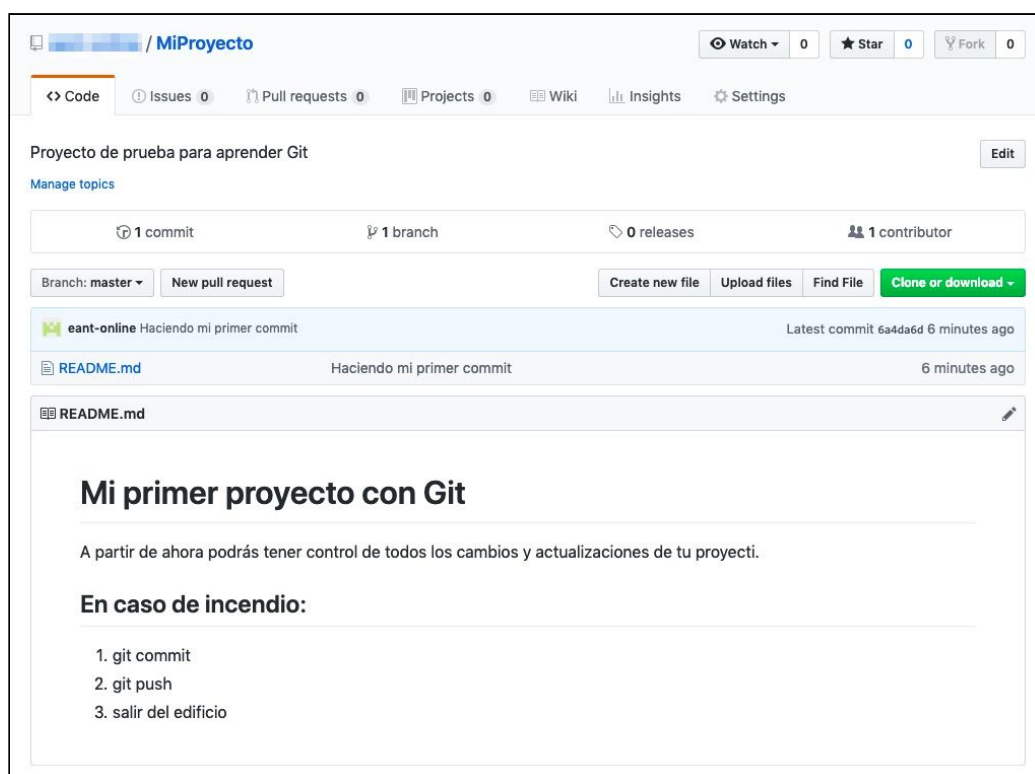
Confirmada la contraseña los archivos y cambios son enviados a la plataforma, mostrando el progreso y un resumen de la operación.

```
EANT@NotebookHP-01 MINGW64 ~/Proyectos/MiProyecto (master)
$ git push origin master
Counting objects: 3, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 371 bytes | 16.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To https://github.com/[redacted]/MiProyecto.git
 * [new branch]      master -> master

EANT@NotebookHP-01 MINGW64 ~/Proyectos/MiProyecto (master)
$
```

Volviendo a **GitHub** se podrá visualizar la actualización en el **repositorio remoto**. La actividad reciente mostrará todos los commits realizados, el usuario que lo realizó y el respectivo comentario.

También podrá apreciarse que el contenido del archivo **README.md** sirve como detalle del proyecto.



Si nuevamente desde la consola se ejecuta el comando `git status` luego de ejecutar un `git push origin master` confirmará que no quedan cambios pendientes.

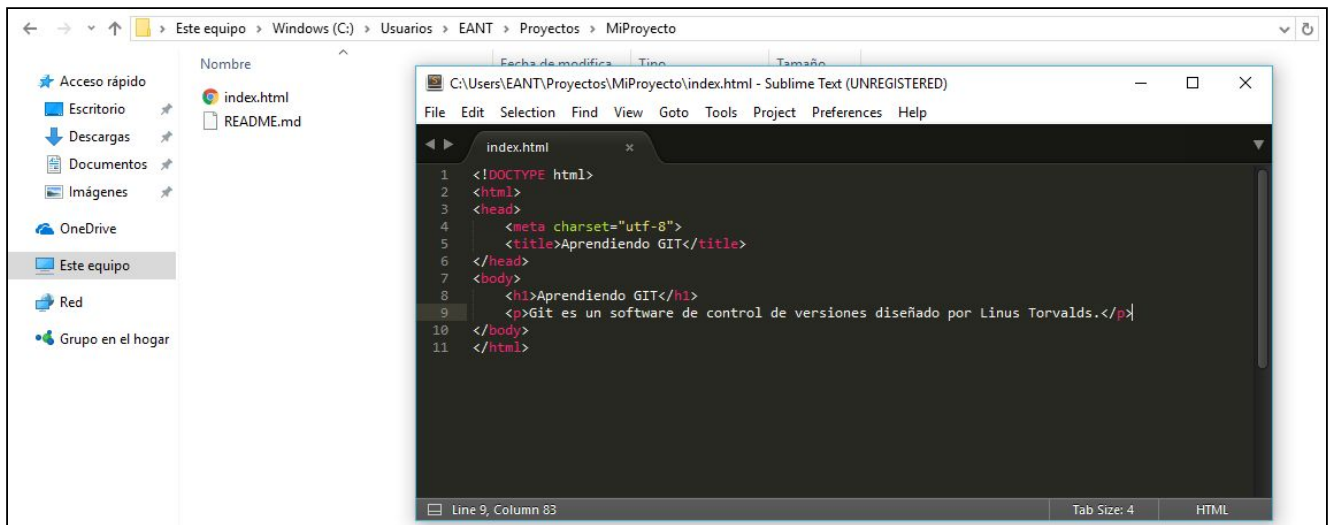
```
EANT@NotebookHP-01 MINGW64 ~/Proyectos/MiProyecto (master)
$ git status
On branch master
Your branch is up to date with 'origin/master'.

nothing to commit, working tree clean

EANT@NotebookHP-01 MINGW64 ~/Proyectos/MiProyecto (master)
$ |
```

Acá es donde empieza la **disciplinaria rutina** de cambios sobre el proyecto y comandos **git** para ir registrando dichos cambios tanto en el **entorno local** como en el **remoto**.

5. Repaso del procedimiento



Crear un nuevo archivo (por ej: index.html) y codear en el mismo algún contenido. No necesariamente debe ser uno solo, tranquilamente puede ser todo un conjunto de archivos (html, css, js, php, py, etc.).

Luego, realizar la respectiva secuencia de comandos **Git** ...

- `git status` → chequea el estado del repositorio local (**opcional**)
- `git add NombreArchivo` o `git add -A` → agrega el/los archivo/s manipulados (**necesario**)
- `git commit -m "Comentario X"` → confirma el/los archivo/s agregados (**necesario**)
- `git push origin master` → realiza la subida de archivos/cambios al repositorio remoto (**necesario**)

```
EANT@NotebookHP-01 MINGW64 ~/Proyectos/MiProyecto (master)
$ git status
On branch master
Your branch is up to date with 'origin/master'.

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

        index.html

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

EANT@NotebookHP-01 MINGW64 ~/Proyectos/MiProyecto (master)
$ git add index.html

EANT@NotebookHP-01 MINGW64 ~/Proyectos/MiProyecto (master)
$ git status
On branch master
Your branch is up to date with 'origin/master'.

Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        new file:   index.html

EANT@NotebookHP-01 MINGW64 ~/Proyectos/MiProyecto (master)
$ git commit -m "Subiendo un index en HTML"
[master 5695449] Subiendo un index en HTML
 1 file changed, 11 insertions(+)
 create mode 100644 index.html

EANT@NotebookHP-01 MINGW64 ~/Proyectos/MiProyecto (master)
$ git push origin master
Counting objects: 3, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 463 bytes | 231.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To https://github.com/mauricio/MiProyecto.git
 5e53afb..5695449 master -> master
```

Tal como se indicó al comienzo de esta guía, los pasos detallados representan un uso básico pero eficaz para entender cómo funciona trabajar con un **sistema de control de versiones**.

Incorporarlo puede no resultar sencillo al principio, pero con constancia se verá reflejado una mejora en la calidad de los desarrollos y su fluido manejo será una habilidad muy apreciada en un entorno de trabajo profesional.

Para cerrar, quedan a disposición algunos links para ampliar, completar y complementar el uso de **Git** en proyectos de desarrollo de software.

- <https://git-scm.com/book/es/v2>
- <https://diegobersano.com.ar/2017/06/13/introduccion-a-git-repaso-a-los-conceptos-generales>
- <https://www.uco.es/aulasoftwarelibre/curso-de-git>
- <https://codeburst.io/git-good-a-practical-introduction-to-git-and-github-in-git-we-trust-f18fa263ec48>