

MANUAL DE GITHUB



2º ASIR

Javier Luis Fernández

ÍNDICE

1. QUE ES GitHub ¿?	Pag.3
2. DESCARGA DE Git	Pag.3
3. CREAR CUENTA EN GitHub	Pag.4
4. ENVIO DE ARCHIVOS AL REPOSITORIO GitHub	Pag.7
5. ACTUALIZACIÓN DE ARCHIVOS EN LÍNEA	Pag.11
6. CLONAR REPOSITORIOS EN GitHub	Pag.12

1. QUE ES GitHub ¿?

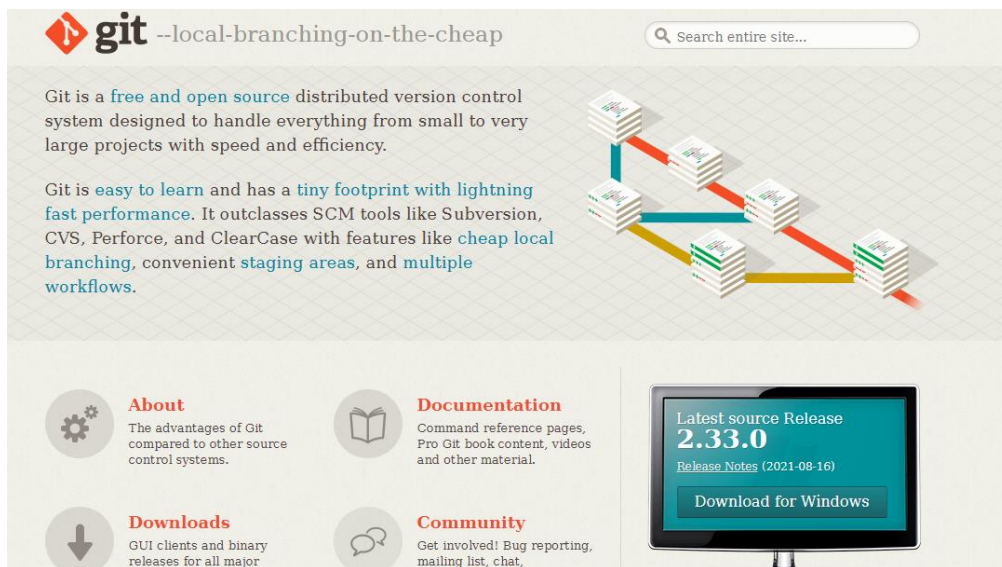
Se trata de una de las principales plataformas para crear proyectos abiertos de herramientas y aplicaciones, y se caracteriza sobre todo por sus funciones colaborativas que ayudan a que todos puedan aportar su granito de arena para mejorar el código.

Como buen repositorio, el código de los proyectos que sean abiertos puede ser descargado y revisado por cualquier usuario, lo que ayuda a mejorar el producto y crear ramificaciones a partir de él. Y si prefieres que tu código no se vea, también pueden crearse proyectos privados.

Como su nombre indica, la web utiliza el sistema de control de versiones Git . Un sistema de gestión de versiones es ese con el que los desarrolladores pueden administrar su proyecto, ordenando el código de cada una de las nuevas versiones que sacan de sus aplicaciones para evitar confusiones. Así, al tener copias de cada una de las versiones de su aplicación, no se perderán los estados anteriores cuando se va a actualizar.

2. DESCARGA DE Git

Para descargar este programa nos debemos de dirigir a la dirección <https://git-scm.com/>



Pulsaremos en el botón “Download for Windows” .

Una vez descargado el ejecutable , lo ejecutamos y la instalación se realizará siguiendo la configuración por defecto, por lo que no tenemos que cambiar ningún parámetro del proceso.

Si queremos comprobar que el programa se ha instalado correctamente en el equipo, podemos comprobarlo usando la herramienta Powershell.



Una vez dentro introducimos el comando git :

 A screenshot of a Windows PowerShell terminal window. The title bar says "Windows PowerShell". The text inside the terminal shows the command 'git' being entered and the resulting help information. The output includes the copyright notice, a link to a PowerShell multiplatform technology, and a detailed list of Git commands categorized into: "These are common Git commands used in various situations:", "start a working area", "work on the current change", "examine the history and state", "grow, mark and tweak your common history", and "collaborate".


```
Windows PowerShell
Copyright (C) Microsoft Corporation. Todos los derechos reservados.

Prueba la nueva tecnología PowerShell multiplataforma https://aka.ms/pscore6

PS C:\Users\javier-cole> git
usage: git [--version] [--help] [-C <path>] [-c <name>=<value>]
        [--exec-path[=<path>]] [--html-path] [--man-path] [--info-path]
        [-p | --paginate | -P | --no-pager] [--no-replace-objects] [--bare]
        [--git-dir=<path>] [--work-tree=<path>] [--namespace=<name>]
        <command> [<args>]

These are common Git commands used in various situations:

start a working area (see also: git help tutorial)
  clone                Clone a repository into a new directory
  init                 Create an empty Git repository or reinitialize an existing one

work on the current change (see also: git help everyday)
  add                  Add file contents to the index
  mv                   Move or rename a file, a directory, or a symlink
  restore              Restore working tree files
  rm                   Remove files from the working tree and from the index
  sparse-checkout      Initialize and modify the sparse-checkout

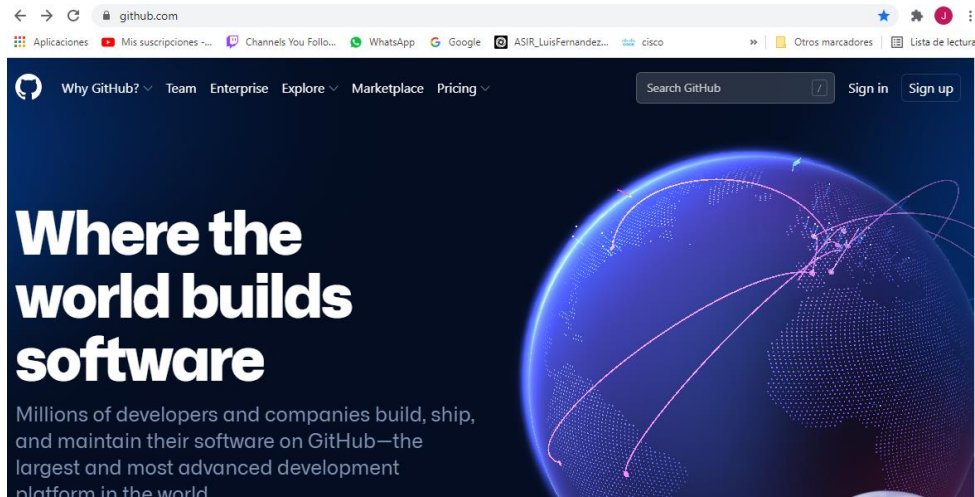
examine the history and state (see also: git help revisions)
  bisect               Use binary search to find the commit that introduced a bug
  diff                 Show changes between commits, commit and working tree, etc
  grep                 Print lines matching a pattern
  log                  Show commit logs
  show                 Show various types of objects
  status               Show the working tree status

grow, mark and tweak your common history
  branch               List, create, or delete branches
  commit               Record changes to the repository
  merge                Join two or more development histories together
  rebase               Reapply commits on top of another base tip
  reset                Reset current HEAD to the specified state
  switch               Switch branches
  tag                  Create, list, delete or verify a tag object signed with GPG

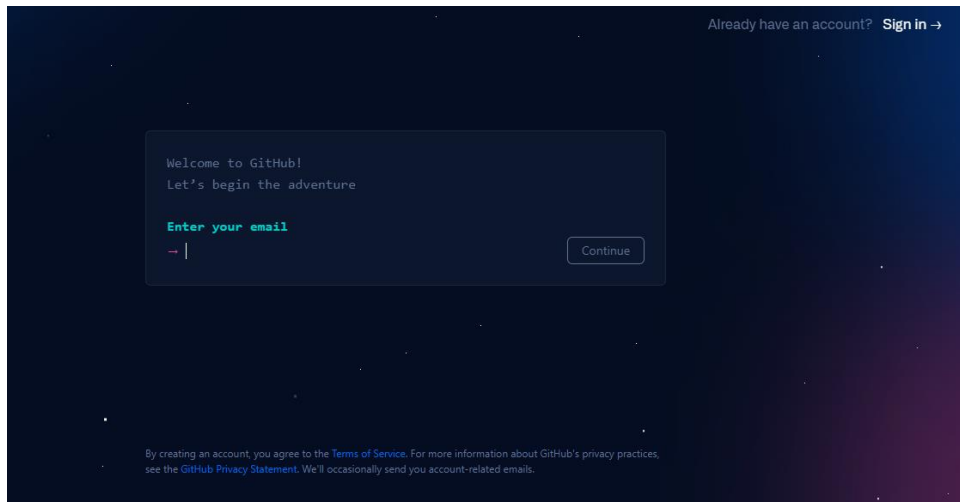
collaborate (see also: git help workflows)
  fetch                Download objects and refs from another repository
  pull                 Fetch from and integrate with another repository or a local branch
  push                 Update remote refs along with associated objects
```

3. CREAR CUENTA EN GitHub

Nos dirigimos a la siguiente dirección : <https://github.com/>

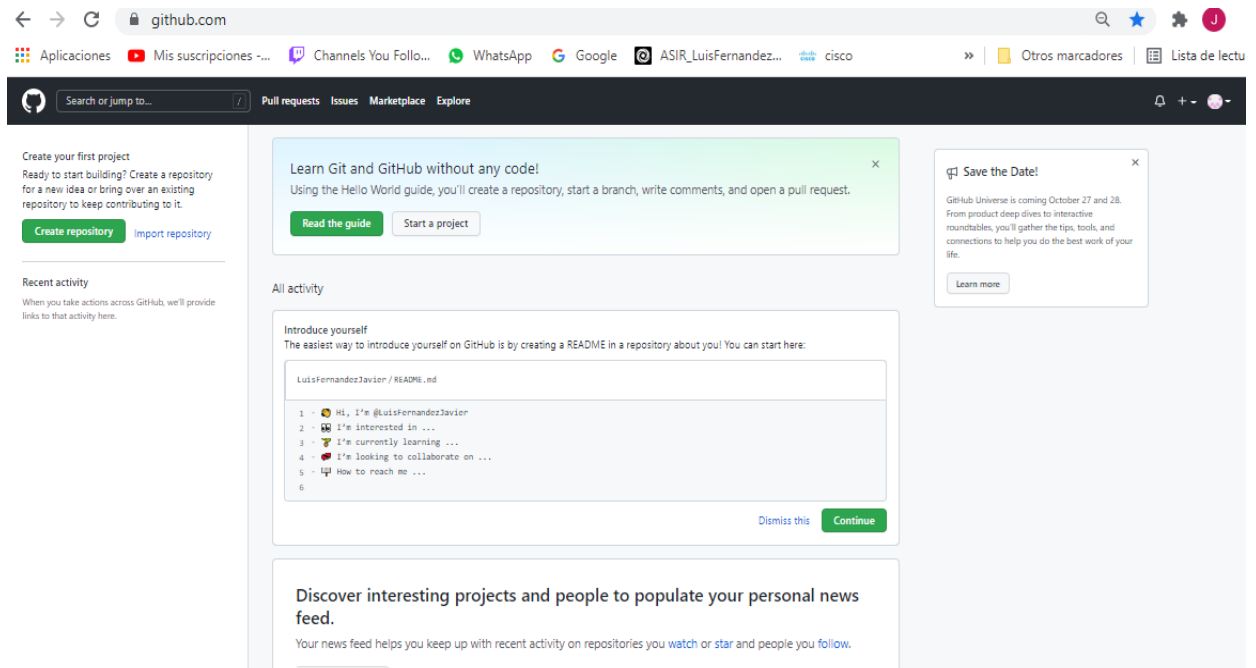


Pulsamos en “Sign up”



Vamos introduciendo nuestros datos hasta terminar el proceso para crear una cuenta.

Ahora vamos a crear nuestro primer repositorio .




Pulsamos en el botón “create repository”. E introducimos el nombre del repositorio.


Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Owner *

 LuisFernandezJavier

Repository name *

/ prueba01 

Great repository names are short and **prueba01** is available. Inspiration? How about **friendly-garbanzo**?

Description (optional)

☒  **Public**

Anyone on the internet can see this repository. You choose who can commit.

☐  **Private**

You choose who can see and commit to this repository.

Initialize this repository with:

Skip this step if you're importing an existing repository.

☐ **Add a README file**

This is where you can write a long description for your project. [Learn more.](#)

☐ **Add .gitignore**

Choose which files not to track from a list of templates. [Learn more.](#)

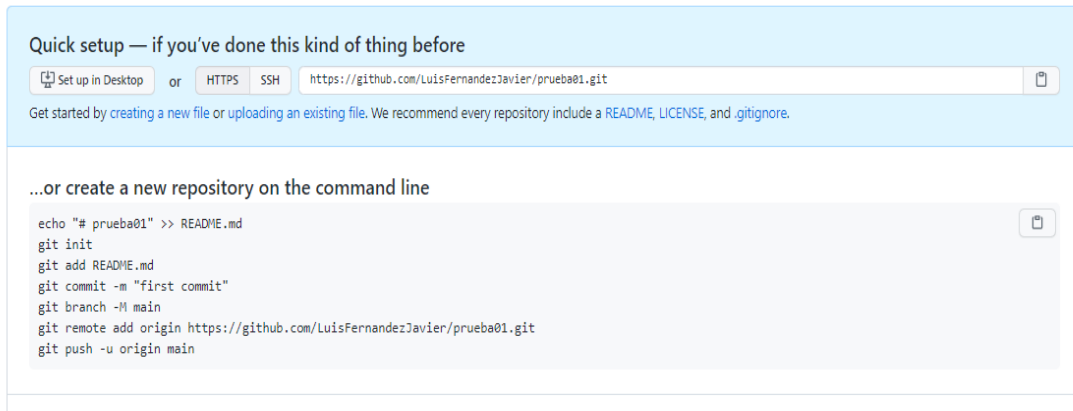
☐ **Choose a license**

A license tells others what they can and can't do with your code. [Learn more.](#)

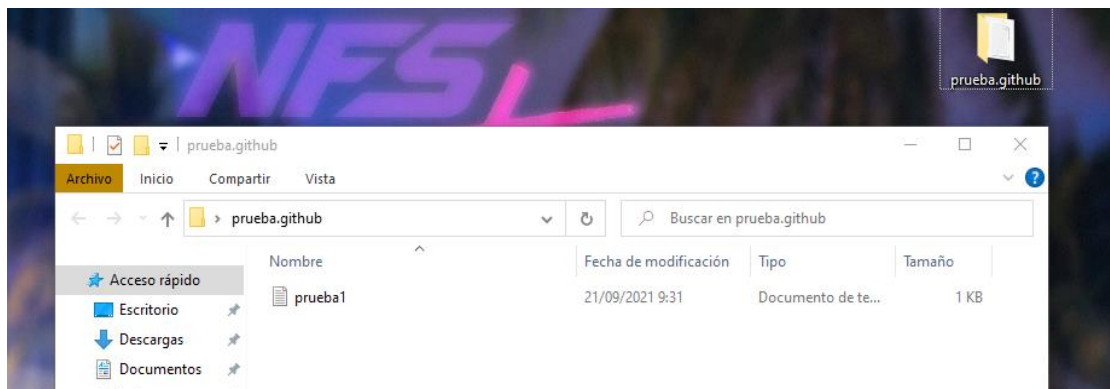
Create repository

4. ENVIO DE ARCHIVOS AL REPOSITORIO GitHub

Para ello seguiremos los pasos que nos recomienda la página.



Abrimos la powershell con git , este será nuestro fichero de prueba .



Nos situamos en nuestro fichero .

```
PS C:\Users\javie> cd desktop
PS C:\Users\javie\desktop> cd prueba.github
PS C:\Users\javie\desktop\prueba.github>
```

Introducimos “ git init “ para arrancar el programa “Git”.

```
PS C:\Users\javie\desktop\prueba.github> git init
Initialized empty Git repository in C:/Users/javie/Desktop/prueba.github/.git/
PS C:\Users\javie\desktop\prueba.github>
```

A continuación configuramos el nombre de usuario y nuestro email. Esto solo ocurrirá la primera vez , quedará guardado para proximos repositorios.

```

*** Please tell me who you are.

Run

  git config --global user.email "you@example.com"
  git config --global user.name "Your Name"

to set your account's default identity.
Omit --global to set the identity only in this repository.

fatal: unable to auto-detect email address (got 'javier@LAPTOP-DNG6T3KQ.(none)')
PS C:\Users\javier\desktop\prueba.github> git config --global user.email "jluifer361@g.educaand.es"
PS C:\Users\javier\desktop\prueba.github> git config --global user.name "LuisFernandezJavier"
PS C:\Users\javier\desktop\prueba.github>

```

Para enviar archivos de nuestro directorio conectado con el repositorio en GitHub usaremos “git add <file>”.

```

PS C:\Users\javier\desktop\prueba.github> git add .
PS C:\Users\javier\desktop\prueba.github>

```

Al usar “.” subimos todos los archivos del directorio en el que nos encontramos.

Con “git status” podemos ver el estado de los archivos subidos.

Ahora introducimos “git commit -m "first commit"”, con esto guardaremos los cambios en el repositorio.

```

PS C:\Users\javier\desktop\prueba.github> git commit -m "first commit"
On branch master

Initial commit

Untracked files:
  (use "git add <file>..." to include in what will be committed)
  prueba1.txt

nothing added to commit but untracked files present (use "git add" to track)
PS C:\Users\javier\desktop\prueba.github>

```

Usaremos los siguientes comandos para indicar la rama del repositorio, la url del repositorio y con el ultimo terminaremos la subida. Una vez hecho esto tendremos que autentificarnos con nuestra cuenta de GitHub en una ventana emergente.

“git branch -M main”

“git remote add origin https://github.com/LuisFernandezJavier/prueba01.git”

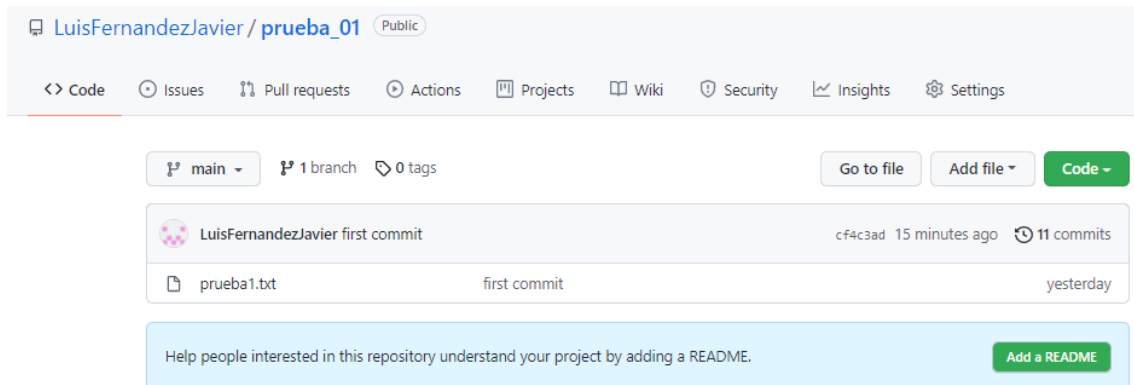
“git push -u origin main”

```

PS C:\Users\javier\desktop\prueba.github> git branch -M main
PS C:\Users\javier\desktop\prueba.github> git remote add origin https://github.com/LuisFernandezJavier/prueba01.git
error: remote origin already exists.
PS C:\Users\javier\desktop\prueba.github> git push -u origin main
info: please complete authentication in your browser...
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Writing objects: 100% (3/3), 235 bytes | 235.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/LuisFernandezJavier/prueba01.git
 * [new branch]      main -> main
Branch 'main' set up to track remote branch 'main' from 'origin'.
PS C:\Users\javier\desktop\prueba.github>

```

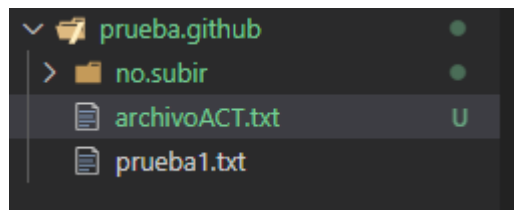
Si todo ha ido bien nuestro repositorio estará subido.



[IMPORTANTE]

A continuación vamos a ver el uso de “git ignore” sirve para ignorar archivo o ficheros que no queremos subir a nuestro repositorio.

Para ver esto he creado la siguiente carpeta dentro de nuestra carpeta principal del ejemplo anterior , va a ser la que no queremos subir , también crearé un archivo que si subiré a mi repositorio. Para ver que se actualiza pero esta carpeta no se subirá.



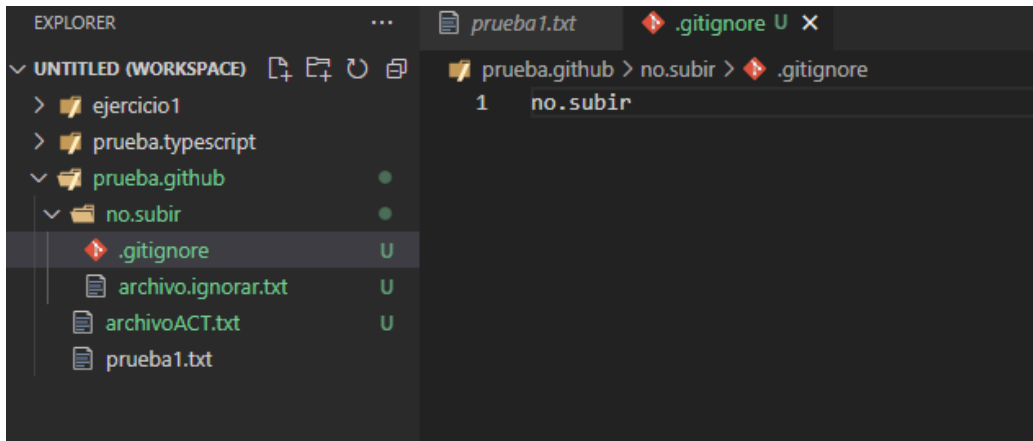
Si usamos “git status” detecta los nuevos ficheros y carpetas.

```
PS C:\Users\javie\Desktop\prueba.github> git status
On branch main
Your branch is up to date with 'origin/main'.

Untracked files:
  (use "git add <file>..." to include in what will be committed)
  archivoACT.txt
  no.subir/

nothing added to commit but untracked files present (use "git add" to track)
PS C:\Users\javie\Desktop\prueba.github> |
```

A continuación crearemos el archivo “,gitignore” en el que indico la carpeta que no quiero subir.



Una vez hecho esto , chequeamos el status y nos parecerá tal que así

```

PROBLEMS  OUTPUT  TERMINAL  CONSOLA DE DEPURACIÓN

PS C:\Users\javie\Desktop\prueba.github> git status
On branch main
Your branch is up to date with 'origin/main'.

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

nothing added to commit but untracked files present (use "git add" to track)
PS C:\Users\javie\Desktop\prueba.github>

```

Añadimos los archivos .

```

PS C:\Users\javie\Desktop\prueba.github> git add .
PS C:\Users\javie\Desktop\prueba.github> git status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   .gitignore
        new file:   archivoACT.txt

```

Hacemos el commit

```

PS C:\Users\javie\Desktop\prueba.github> git commit -m "gitignore"
[main 52c5b4b] gitignore
2 files changed, 1 insertion(+)
create mode 100644 .gitignore
create mode 100644 archivoACT.txt

```

Al pushear veremos como se actualiza nuestro repositorio . Sin subir nuestra carpeta ignorada.

LuisFernandezJavier / prueba_01 Public

<> Code Issues Pull requests Actions Projects Wiki Security Insights Settings

main 1 branch 0 tags Go to file Add file Code

LuisFernandezJavier gitignore		52c5b4b 3 minutes ago 12 commits
.gitignore	gitignore	3 minutes ago
archivoACT.txt	gitignore	3 minutes ago
prueba1.txt	first commit	2 days ago

Help people interested in this repository understand your project by adding a README. [Add a README](#)

5. ACTUALIZACIÓN DE ARCHIVOS EN LÍNEA

Nuestros archivos pueden ser editados en github , para actualizarlos en nuestra versión local procederemos de la siguiente manera.

Editaré uno de los archivos que estoy usando como ejemplo en puntos anteriores.

LuisFernandezJavier Update prueba1.txt

1 contributor

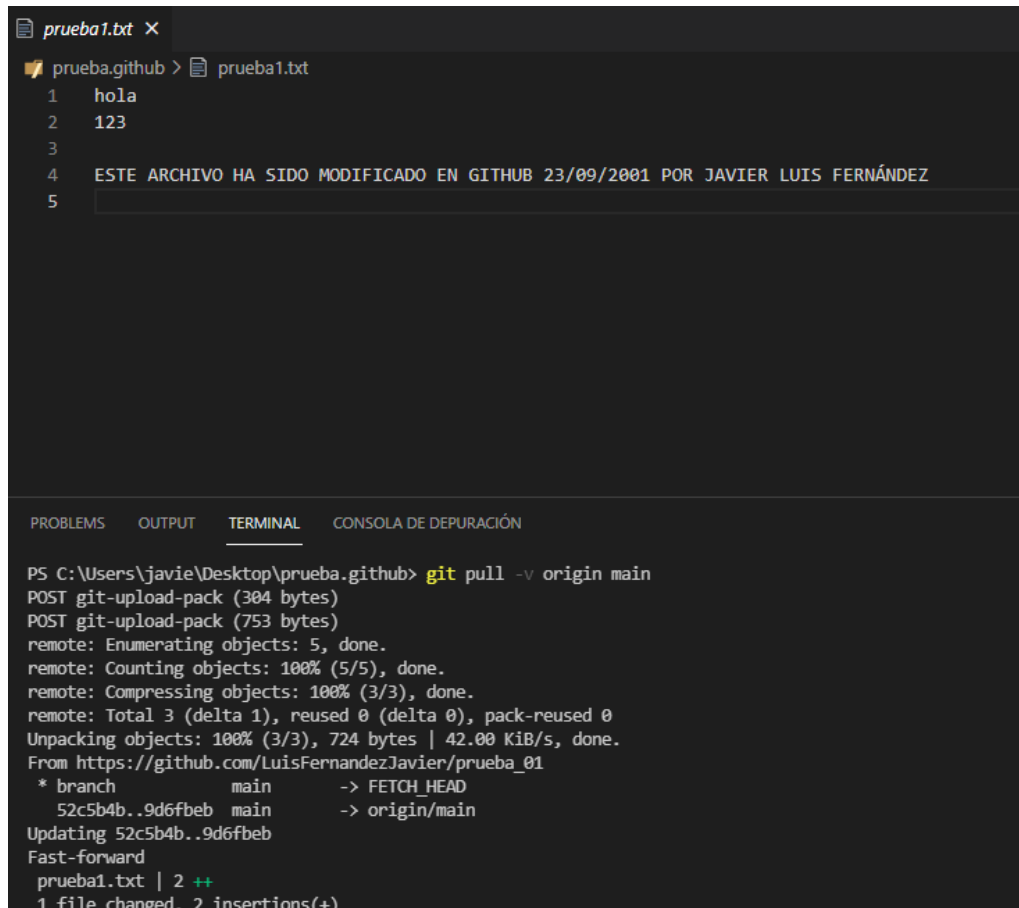
4 lines (3 sloc) | 91 Bytes

```

1  hola
2  123
3
4  ESTE ARCHIVO HA SIDO MODIFICADO EN GITHUB 23/09/2001 POR JAVIER LUIS FERNÁNDEZ

```

Con el comando “git pull -v origin main” lo actualizaremos en local.



The screenshot shows a code editor with a file named `prueba1.txt` open. The file contains the following text:

```
1 hola
2 123
3
4 ESTE ARCHIVO HA SIDO MODIFICADO EN GITHUB 23/09/2001 POR JAVIER LUIS FERNÁNDEZ
5
```

Below the editor, the terminal window shows the output of a `git pull` command:

```
PS C:\Users\javie\Desktop\prueba.github> git pull -v origin main
POST git-upload-pack (304 bytes)
POST git-upload-pack (753 bytes)
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), 724 bytes | 42.00 KiB/s, done.
From https://github.com/LuisFernandezJavier/prueba_01
* branch          main          -> FETCH_HEAD
  52c5b4b..9d6fbeb  main          -> origin/main
Updating 52c5b4b..9d6fbeb
Fast-forward
 prueba1.txt | 2 ++
1 file changed, 2 insertions(+)
```

6. CLONAR REPOSITORIOS

Para clonar repositorios usaremos el comando : “git clone https: //...”

Haré la prueba con un repositorio de un compañero de clase.

```
PS C:\Users\javie\Desktop\prueba.github> git clone https://github.com/NaranjoJimenezAndres/1000_BASIC0
Cloning into '1000_BASIC0'...
remote: Enumerating objects: 40, done.
remote: Counting objects: 100% (40/40), done.
Receiving mpressing objects: 100% (32/32), done.
remote: Total 40 (delta 7), reused 21 (delta 2), pack-reused 0
Receiving objects: 100% (40/40), 2.48 MiB | 3.77 MiB/s, done.
Resolving deltas: 100% (7/7), done.
PS C:\Users\javie\Desktop\prueba.github>
```

