

INSTALACIÓN DE GITHUB

Contenido

INSTALACIÓN DE GITHUB	1
1.) ¿Qué es Git, y que es GitHub?	2
2.) Instalación de Git	2
3.) Creación de una cuenta en GitHub	5
4.) Creación del repositorio	9
5.) Comandos básicos con Git.	12
5.1) Vincular directorio local con el repositorio y subir fichero README	12
5.2) ¿Cómo subir una carpeta entera?	14
5.3) ¿Qué hacemos si modificamos un archivo y queremos actualizarlo en el repositorio?	15
5.4) ¿Qué pasa si modificamos un fichero en el repositorio?	16
5.5) ¿Cómo puedo seguir trabajando en mi proyecto en otro equipo o en un directorio diferente?	16
5.6) ¿Cómo puedo hacer para quitarle los permisos a un equipo de que me actualice mi propio repositorio?	17

1.) ¿Qué es Git, y que es GitHub?

Git es un sistema de control de versiones, es decir un software que facilita la administración de versiones de un mismo programa, subiendo a la nube todas las versiones de tu propio software. Git te da muchas ventajas:

- Poder gestionar varias versiones de tu propio programa, de manera que en todo momento tengas el código de todas las versiones por si hiciese falta en versiones futuras.
- Otros desarrolladores tienen acceso también a tu programa, de manera que facilita mucho el trabajo en equipo.
- Tener tu programa en la nube, de forma que puedas trabajar en cualquier equipo siempre que tengas Git instalado.

2.) Instalación de Git

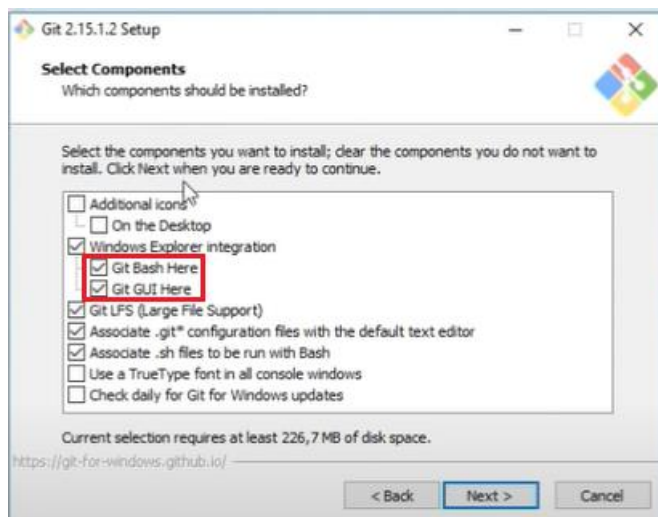
Para su instalación, debemos irnos a su página [oficial](#), y descargarnos su setup (En nuestro caso, para Windows):

The screenshot shows the Git website homepage. At the top left is the Git logo and the tagline "--fast-version-control". A search bar is on the top right. The main content area describes Git as a "free and open source" distributed version control system. Below this, it mentions Git is "easy to learn" and has a "tiny footprint with lightning fast performance". To the right of the text is a diagram showing a branching model with stacks of code blocks connected by lines. Below the text are four icons with labels: "About" (gears), "Documentation" (book), "Downloads" (downward arrow), and "Community" (speech bubbles). To the right of these is a monitor displaying the "Latest source Release 2.33.0" and a "Download for Windows" button, which is circled in red. A red arrow points from the "Downloads" section towards the monitor. At the bottom right, there are links for "Windows GUIs", "Tarballs", "Mac Build", and "Source Code". The footer section is titled "Companies & Projects Using Git" and features logos for Google, Facebook, Microsoft, Twitter, LinkedIn, Netflix, a camel, and PostgreSQL.

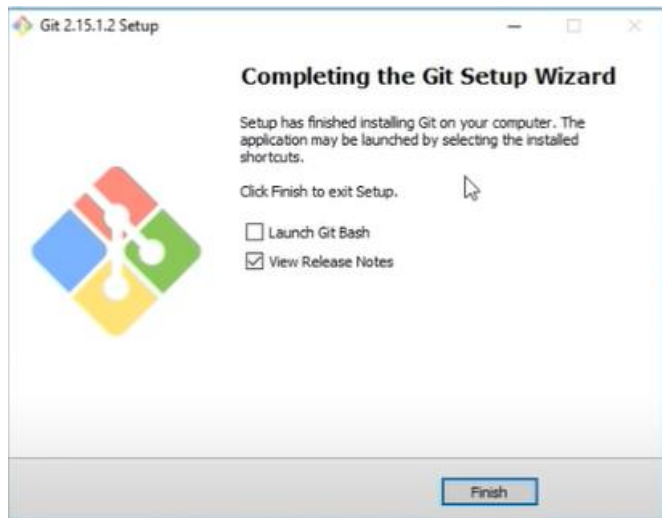
Una vez descargado, iniciaremos este mismo setup nos saldrá una ventana como esta, informándonos de que git es un programa con licencia GNU, es decir un programa de código libre. Le daremos a “**Next**”:



Después, nos saldrá una ventana en la que nos preguntara que componentes queremos instalar, y es muy importante **dejar marcados** los dos que resalto en la captura:



Las siguientes ventanas de personalización de la instalación del programa, en nuestro caso lo dejaremos por defecto todo. Una vez, finalice la instalación de Git, nos saldrá esta ventana, en la que habrá 2 opciones por marcar, iniciar el Bash de Git y ver la página web de las ultimas notas de lanzamiento. (**Es indiferente que las marquemos o no**, en mi caso deje sin marcar la de iniciar Bash y deje marcada la de ver las notas de lanzamiento, tal como viene por defecto)



Lo que habremos instalado aquí es el propio interprete de comandos de Git, de manera que si por ejemplo nosotros abrimos ahora el PowerShell de Windows y ponemos “**git --version**”, vemos que nos ejecuta sin problema estos comandos que hemos instalado con el propio Git:

```
PS C:\Users\Toni> git --version
git version 2.33.0.windows.2
PS C:\Users\Toni>
```

3.) Creación de una cuenta en GitHub.

Primero, debemos irnos a la página oficial de [GitHub](https://github.com). Cuando entremos en la página tendremos 2 opciones: "Sign in"; para iniciar sesión en una cuenta ya existente, o "**Sign up**"; para crear una cuenta nueva.

En mi caso, haré una cuenta nueva. Nos pedirá una serie de datos personales que tenemos que rellenar:



Welcome to GitHub!
Let's begin the adventure

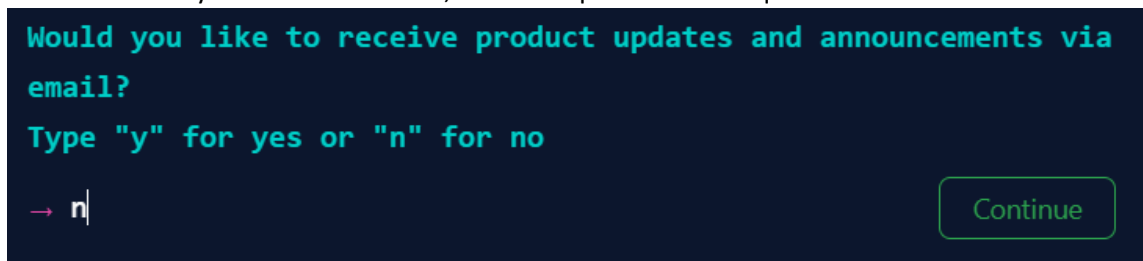
Enter your email
✓ amenram1711@g.educaand.es

Create a password
✓

Enter a username
→ MendezRamirezAntonioJose

Continue

Una vez rellenemos los datos y continuemos, nos preguntará que si queremos recibir actualizaciones y anuncios vía e-mail, de modo que le diremos que no:

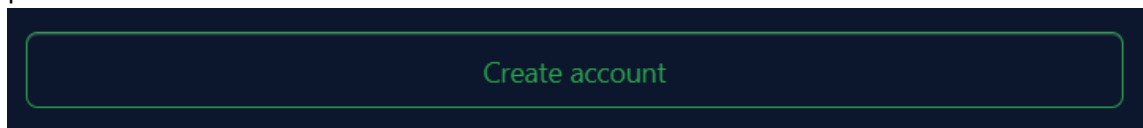


Would you like to receive product updates and announcements via email?
Type "y" for yes or "n" for no

→ n

Continue

Después de esto, tendremos que hacer un pequeño captcha, y una vez que lo realicemos podremos crear la cuenta:



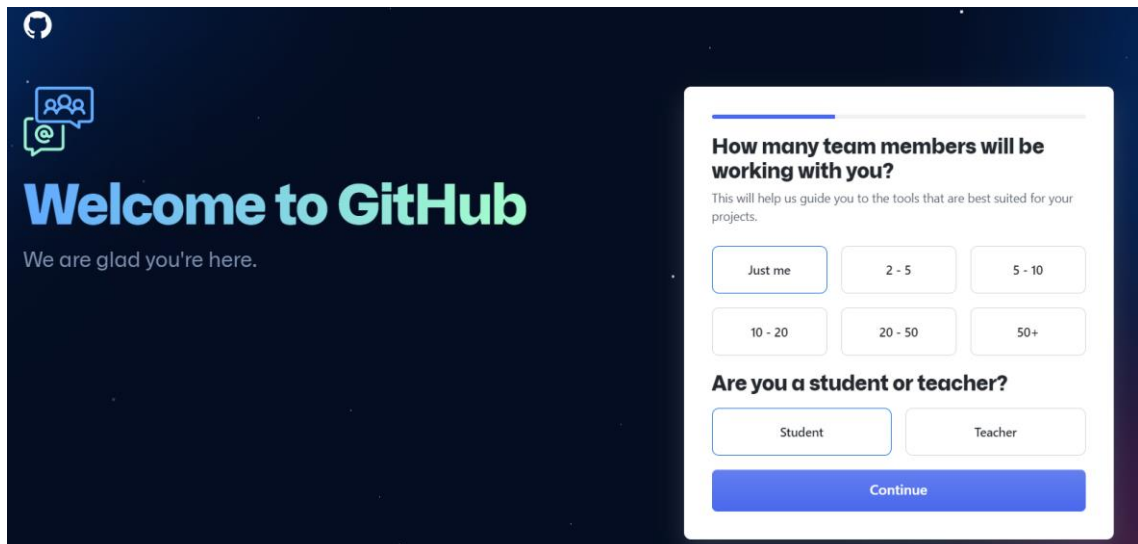
Create account

Nos pedirán que verifiquemos la cuenta poniendo un código que nos enviarán al email que hemos puesto:

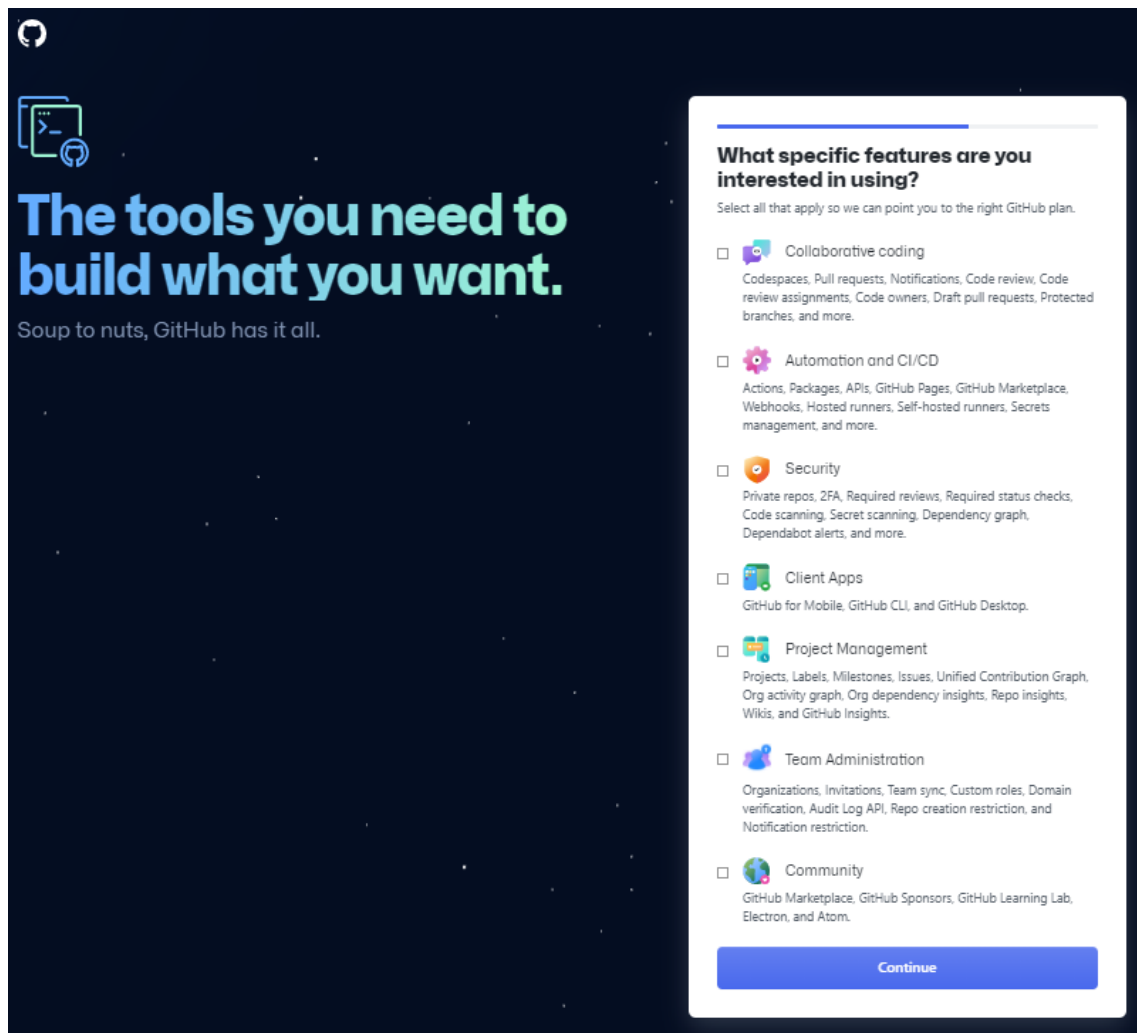


Una vez verificados, nos preguntaran algunas cosas:








En la primera pantalla, nos preguntaran cuantos miembros trabajan con nosotros (**Marcamos la opción de "Just Me"**) y si somos estudiantes o maestros; en mi caso: estudiante.



Después, nos preguntaran el tipo de características nos interesan, en lo que lo dejaremos en blanco:

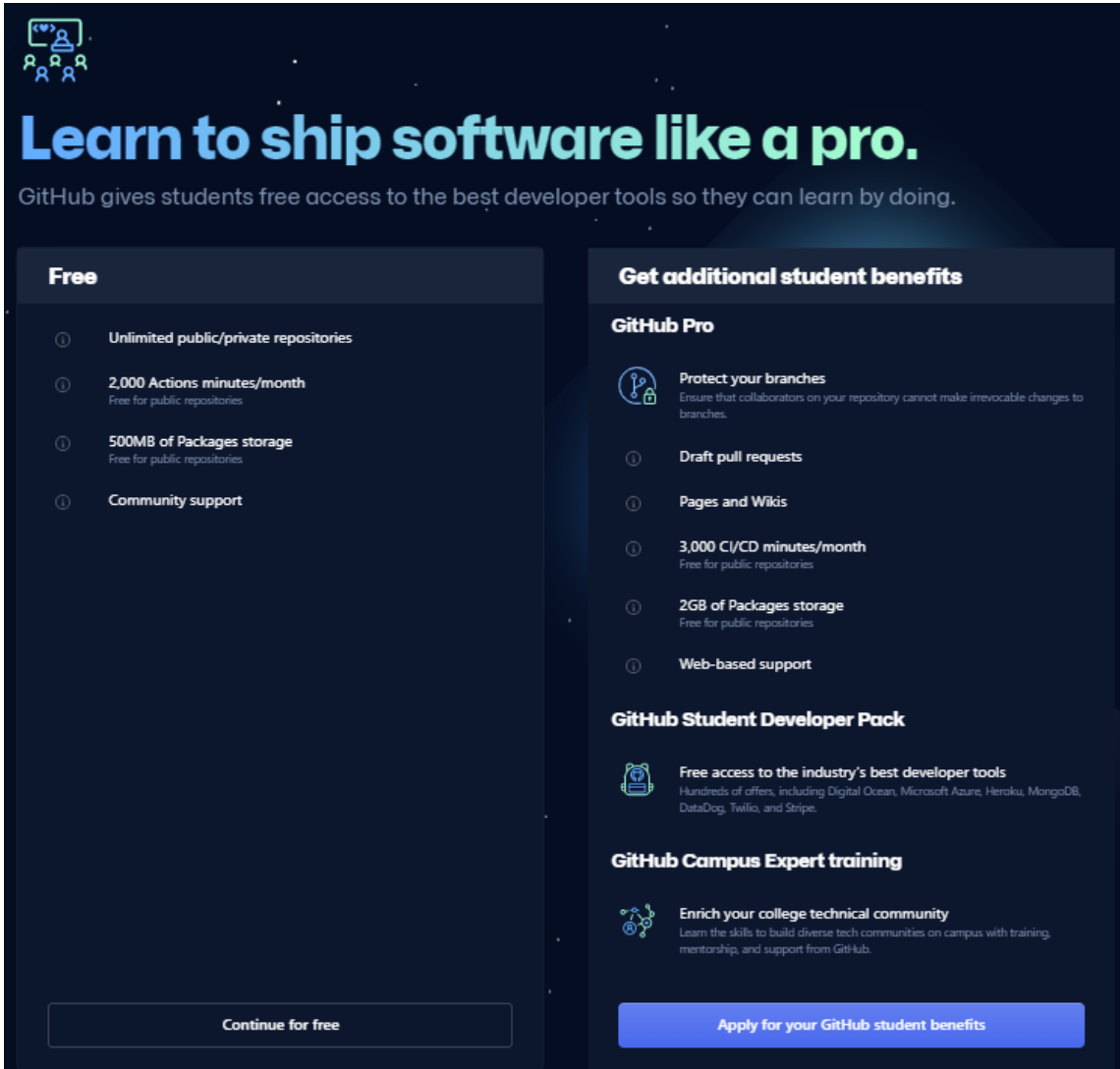


The image shows a GitHub onboarding form with a dark blue background. On the left, there is a GitHub logo and a code editor icon. The main text reads: "The tools you need to build what you want. Soup to nuts, GitHub has it all." On the right, there is a white box titled "What specific features are you interested in using?" with a subtext "Select all that apply so we can point you to the right GitHub plan." Below this, there are seven checkboxes, each with an icon and a list of features:

- ☐  Collaborative coding
Codespaces, Pull requests, Notifications, Code review, Code review assignments, Code owners, Draft pull requests, Protected branches, and more.
- ☐  Automation and CI/CD
Actions, Packages, APIs, GitHub Pages, GitHub Marketplace, Webhooks, Hosted runners, Self-hosted runners, Secrets management, and more.
- ☐  Security
Private repos, 2FA, Required reviews, Required status checks, Code scanning, Secret scanning, Dependency graph, Dependabot alerts, and more.
- ☐  Client Apps
GitHub for Mobile, GitHub CLI, and GitHub Desktop.
- ☐  Project Management
Projects, Labels, Milestones, Issues, Unified Contribution Graph, Org activity graph, Org dependency insights, Repo insights, Wikis, and GitHub Insights.
- ☐  Team Administration
Organizations, Invitations, Team sync, Custom roles, Domain verification, Audit Log API, Repo creation restriction, and Notification restriction.
- ☐  Community
GitHub Marketplace, GitHub Sponsors, GitHub Learning Lab, Electron, and Atom.

At the bottom of the white box is a blue button labeled "Continue".

Y por último, nos darán a elegir entre el plan free y el plan Premium, el cual obviamente elegiremos el free:



The image shows the GitHub Student Developer Pack landing page. At the top, there's a GitHub logo with a graduation cap. Below it, the headline reads "Learn to ship software like a pro." followed by the subtext "GitHub gives students free access to the best developer tools so they can learn by doing." The page is divided into two main columns. The left column, titled "Free", lists four benefits: "Unlimited public/private repositories", "2,000 Actions minutes/month" (with a note "Free for public repositories"), "500MB of Packages storage" (with a note "Free for public repositories"), and "Community support". At the bottom of this column is a button labeled "Continue for free". The right column, titled "Get additional student benefits", is further divided into three sections. The first section, "GitHub Pro", lists five benefits: "Protect your branches" (with a note "Ensure that collaborators on your repository cannot make irrevocable changes to branches"), "Draft pull requests", "Pages and Wikis", "3,000 CI/CD minutes/month" (with a note "Free for public repositories"), and "2GB of Packages storage" (with a note "Free for public repositories"). The second section, "GitHub Student Developer Pack", lists one benefit: "Free access to the industry's best developer tools" (with a note "Hundreds of offers, including Digital Ocean, Microsoft Azure, Heroku, MongoDB, DataDog, Twilio, and Stripe."). The third section, "GitHub Campus Expert training", lists one benefit: "Enrich your college technical community" (with a note "Learn the skills to build diverse tech communities on campus with training, mentorship, and support from GitHub."). At the bottom of this column is a button labeled "Apply for your GitHub student benefits".

Free

- ① Unlimited public/private repositories
- ① 2,000 Actions minutes/month
Free for public repositories
- ① 500MB of Packages storage
Free for public repositories
- ① Community support

[Continue for free](#)

Get additional student benefits

GitHub Pro

- ① Protect your branches
Ensure that collaborators on your repository cannot make irrevocable changes to branches.
- ① Draft pull requests
- ① Pages and Wikis
- ① 3,000 CI/CD minutes/month
Free for public repositories
- ① 2GB of Packages storage
Free for public repositories
- ① Web-based support

GitHub Student Developer Pack

- ① Free access to the industry's best developer tools
Hundreds of offers, including Digital Ocean, Microsoft Azure, Heroku, MongoDB, DataDog, Twilio, and Stripe.

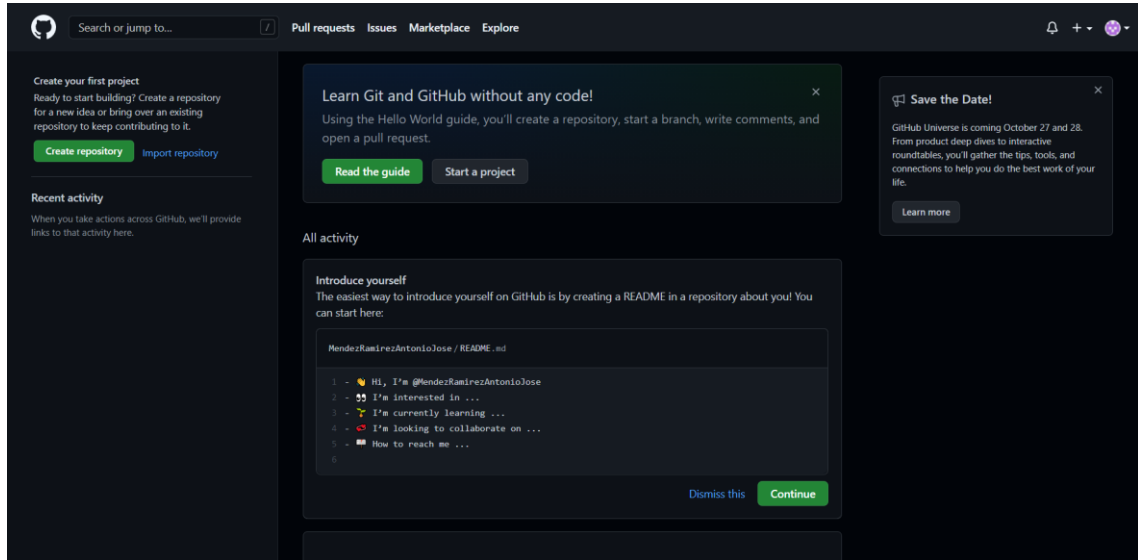
GitHub Campus Expert training

- ① Enrich your college technical community
Learn the skills to build diverse tech communities on campus with training, mentorship, and support from GitHub.

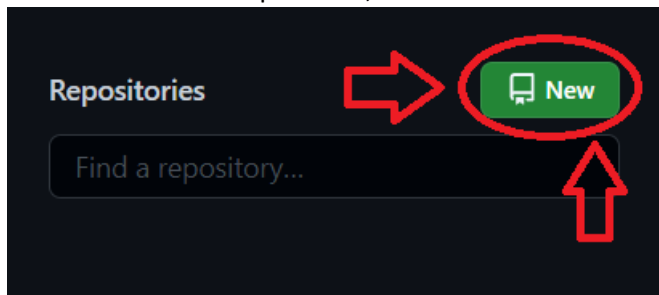
[Apply for your GitHub student benefits](#)

4.) Creación del repositorio.

Para esto, es obligatorio crearse una cuenta en GitHub como antes he enseñado. Una vez que creamos la cuenta nos saldrá algo así:



Para crear nuestro repositorio, le daremos al botón verde que dice **“New”**:



Después de esto, nos saldrá la siguiente ventana:

Create a new repository

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

Owner * MendezRamirezAntonioJose ▾ / **Repository name *** proyecto01 ✓

Nombre del proyecto. (No podrás tener en tu cuenta 2 proyectos con el mismo nombre.)

Great repository names are short and memorable. Need inspiration? How about **cautious-octo-rotary-phone**?

Description (optional) **Descripción del proyecto. (Es opcional)**

1º Actividad Base de datos.

☒ **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.

Lo pondremos en público para que el profesor pueda corregirnos.


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.](#)

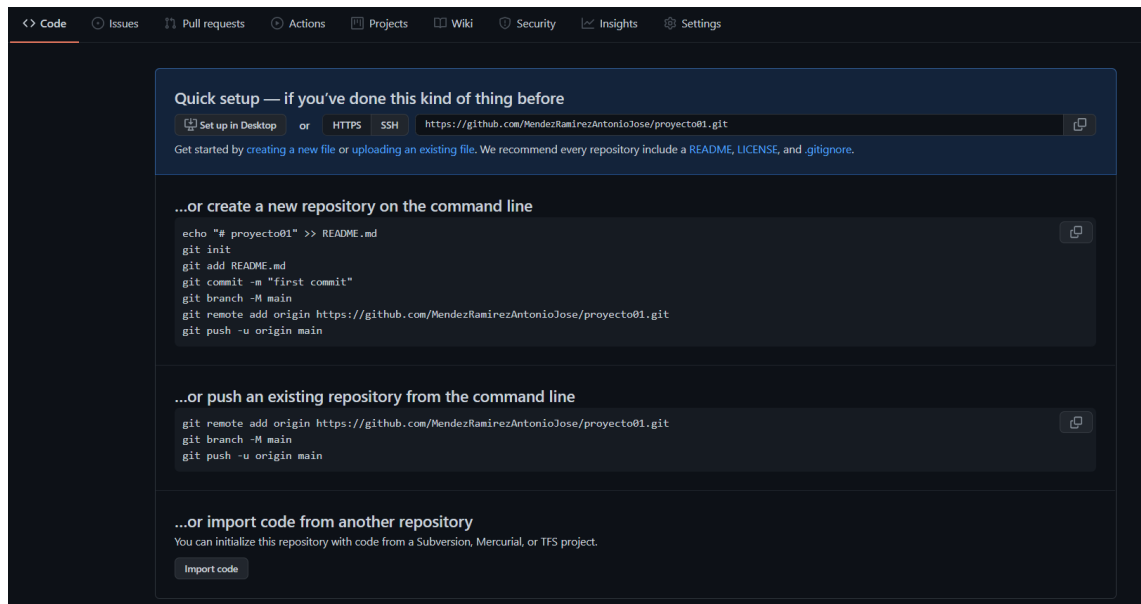
☐ **Choose a license**
A license tells others what they can and can't do with your code. [Learn more.](#)

Es muy importante que si marcamos esta opción, el fichero README se creara primero en el repositorio, por lo que lo 1º que tenemos que hacer entonces es un "pull"

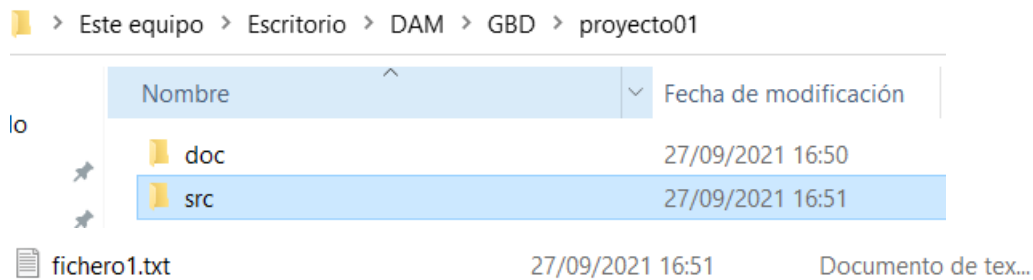
This will set  **main** as the default branch. Change the default name in your [settings](#).

Create repository

Después de configurar nuestro repositorio, le daremos al botón **“Create repository”**, y nos saldrán unas instrucciones básicas:

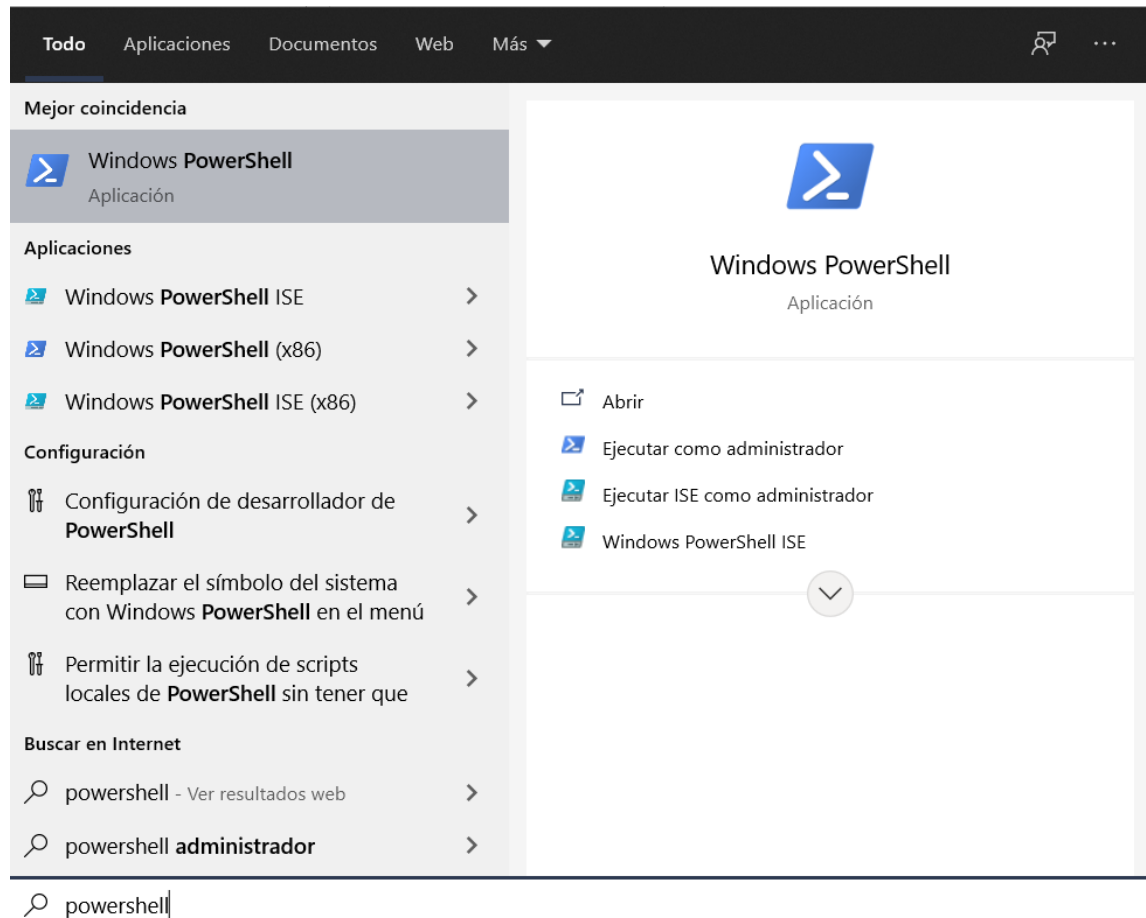


Ahora crearemos la estructura de directorios requerida por el ejercicio dentro de la carpeta del proyecto:



5.) Comandos básicos con Git.

Para empezar a ejecutar los comandos tendremos que abrir **powershell**, la línea de comandos de Windows:



5.1) Vincular directorio local con el repositorio y subir fichero README

Una vez estemos dentro de la powershell iremos al directorio que queramos vincular con el repositorio de Github:

```
PS C:\Users\Toni> cd C:\Users\Toni\Desktop\DAM\GBD\proyecto01
```

Una vez que estemos dentro del directorio, ejecutaremos el siguiente comando para vincular nuestro repositorio de Github con nuestro directorio local:

```
PS C:\Users\Toni\Desktop\DAM\GBD\proyecto01> git init
Initialized empty Git repository in C:/Users/Toni/Desktop/DAM/GBD/proyecto01/.git/
```

Si vemos nuestra carpeta local, vemos que se nos ha creado una carpeta oculta:

```
.git 27/09/2021 17:02 Carpeta de archivos
```

Ahora, añadiremos el fichero README.md a la carpeta del proyecto:

```
README.md 27/09/2021 17:11 Archivo MD 0 KB
```

Y editaremos el fichero README, con el formato de su archivo para hacer una introducción ([que podemos ver en su página oficial](#)) sobre el proyecto:

```
## Mi primer repositorio
**En este proyecto, en el que explicare los primeros pasos en Github, viendo todas las herramientas que hemos visto en clase, su instalación y su funcionamiento.**
***Hemos creado 2 carpetas aparte del readme:***
1. src: Carpeta source del proyecto, donde tenemos un fichero de prueba.
2. doc: Carpeta para la documentación del proyecto.
```

Realizado por Antonio José Méndez Ramírez, alumno de 1ºDAM en el IES Punta Del verde

Ahora, prepararemos el primer envío, con el comando **“add”**. Primero tendremos que añadir todo lo que queramos subir, **en mi caso solo subiremos el readme**:

```
PS C:\Users\Toni\Desktop\DAM\GBD\proyecto01> git add README.md
```

Después haremos un **“git status”** para comprobar que todo se ha añadido correctamente:

```
PS C:\Users\Toni\Desktop\DAM\GBD\proyecto01> git status
On branch master

No commits yet

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

Ahora, tendremos que “empaquetarlo”. Para esto, usaremos el comando **“commit”**, dándole un nombre al paquete:

```
PS C:\Users\Toni\Desktop\DAM\GBD\proyecto01> git commit -m "Primer envio"
[master (root-commit) 9e56e12] Primer envio
 1 file changed, 7 insertions(+)
 create mode 100644 README.md
```

IMPORTANTE: Si es la primera vez que usamos github en el equipo nos pedirá que pongamos nuestro nombre de usuario y el correo asociado a tu cuenta. Tendremos que poner los siguientes comandos cambiando los valores personales:

```
PS C:\Users\Toni\Desktop\DAM\GBD\proyecto01> git config --global user.email "amenram1711@g.educaand.es"
PS C:\Users\Toni\Desktop\DAM\GBD\proyecto01> git config --global user.name "MendezRamirezAntonioJose"
```

Ahora tendremos que decirle al repositorio cual rama del proyecto queremos dirigirnos. Nosotros nos quedaremos en el camino principal, es decir en el **“main”**. Para esto usaremos el comando **“branch”**:

```
PS C:\Users\Toni\Desktop\DAM\GBD\proyecto01> git branch -M main
PS C:\Users\Toni\Desktop\DAM\GBD\proyecto01> git status
On branch main
nothing to commit, working tree clean
```

Si hacemos un **“status”**, vemos que estamos en el camino main.

Ahora, tendremos que vincular el repositorio con el local, con el comando **“remote add origin”** y nuestra url del proyecto:

```
PS C:\Users\Toni\Desktop\DAM\GBD\proyecto01> git remote add origin https://github.com/MendezRamirezAntonioJose/proyecto01.git
```

Ahora lo que queda, es sincronizar nuestro directorio local con el repositorio de Github. Lo haremos con el comando “push”:

```
PS C:\Users\Toni\Desktop\DAM\GBD\proyecto01> git push -u origin main
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Delta compression using up to 16 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 527 bytes | 527.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/MendezRamirezAntonioJose/proyecto01.git
 * [new branch]      main -> main
Branch 'main' set up to track remote branch 'main' from 'origin'.
```

5.2) ¿Cómo subir una carpeta entera?

Ahora subiremos todo lo demás, con un “.”, que expresa todo lo que hay en el directorio:

```
PS C:\Users\Toni\Desktop\DAM\GBD\proyecto01> git add .
```

Ahora, preparemos un nuevo paquete:

```
PS C:\Users\Toni\Desktop\DAM\GBD\proyecto01> git commit -m "Segundo envio"
[main b266c93] Segundo envio
 2 files changed, 1 insertion(+)
 create mode 100644 doc/documentacion.pdf
 create mode 100644 src/fichero1.txt
```

Ahora, solo faltará hacer “push”, ya que hemos el “branch” y el “origin” anteriormente:

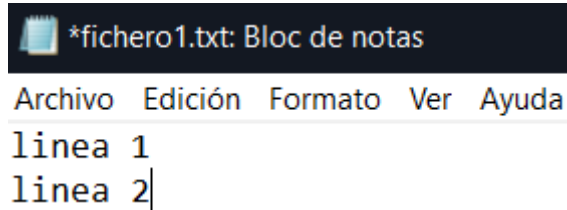
```
PS C:\Users\Toni\Desktop\DAM\GBD\proyecto01> git push -u origin main
Enumerating objects: 7, done.
Counting objects: 100% (7/7), done.
Delta compression using up to 16 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (6/6), 445 bytes | 445.00 KiB/s, done.
Total 6 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/MendezRamirezAntonioJose/proyecto01.git
 9e56e12..b266c93  main -> main
Branch 'main' set up to track remote branch 'main' from 'origin'.
```

Y ya vemos que se ha subido a nuestro repositorio:

	MendezRamirezAntonioJose Segundo envio	b266c93 10 minutes ago	🕒 2 commits
	doc	Segundo envio	10 minutes ago
	src	Segundo envio	10 minutes ago
	README.md	Primer envio	18 minutes ago

5.3) ¿Qué hacemos si modificamos un archivo y queremos actualizarlo en el repositorio?

Vamos a modificar el fichero que está dentro de src:



Ahora, tendremos que hacerle un “**add**” al directorio, e identificará que el único archivo modificado es ese:

```
PS C:\Users\Toni\Desktop\DAM\GBD\proyecto01> git add .
PS C:\Users\Toni\Desktop\DAM\GBD\proyecto01> 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)
        modified:   src/fichero1.txt
```






Después de eso, hacemos un “**commit**” para empaquetarlo:

```
PS C:\Users\Toni\Desktop\DAM\GBD\proyecto01> git commit -m "Tercer envio"
[main 033d5e3] Tercer envio
 1 file changed, 2 insertions(+), 1 deletion(-)
```

Ya solo quedaría hacer el “**push**”:

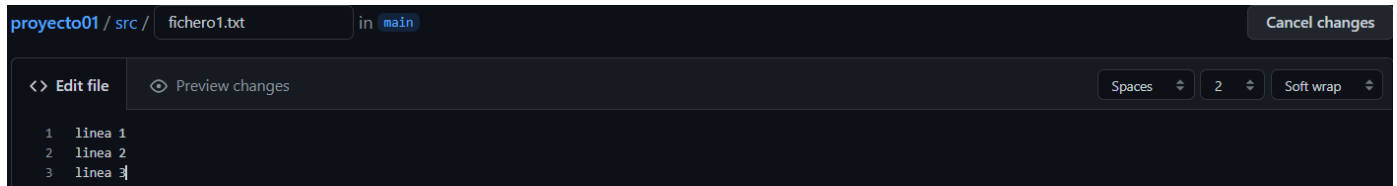
```
PS C:\Users\Toni\Desktop\DAM\GBD\proyecto01> git push -u origin main
Enumerating objects: 7, done.
Counting objects: 100% (7/7), done.
Delta compression using up to 16 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (4/4), 333 bytes | 333.00 KiB/s, done.
Total 4 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/MendezRamirezAntonioJose/proyecto01.git
   b266c93..033d5e3  main -> main
Branch 'main' set up to track remote branch 'main' from 'origin'.
```

Vemos que el repositorio queda tal que así:

	MendezRamirezAntonioJose Tercer envio	033d5e3 2 minutes ago	 3 commits
	doc	Segundo envio	17 minutes ago
	src	Tercer envio	2 minutes ago
	README.md	Primer envio	25 minutes ago

5.4) ¿Qué pasa si modificamos un fichero en el repositorio?

Vamos a modificar el fichero que tenemos dentro del directorio src:



Tendremos que ejecutar el comando “pull”

```
PS C:\Users\Toni\Desktop\DAM\GBD\proyecto01> git pull
remote: Enumerating objects: 7, done.
remote: Counting objects: 100% (7/7), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 4 (delta 1), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (4/4), 707 bytes | 78.00 KiB/s, done.
From https://github.com/MendezRamirezAntonioJose/proyecto01
   033d5e3..461eb0e  main      -> origin/main
Updating 033d5e3..461eb0e
Fast-forward
 src/fichero1.txt | 3 ++-
 1 file changed, 2 insertions(+), 1 deletion(-)
```

Y ya estaría totalmente sincronizado con el directorio local.

5.5) ¿Cómo puedo seguir trabajando en mi proyecto en otro equipo o en un directorio diferente?

Primero, tendremos que situarnos en la powershell en el directorio donde queramos tener el proyecto.

Para esto usaremos el comando “clone”. Iremos a otro equipo o a otro directorio donde queramos tener absolutamente todo lo que llevamos trabajado en el proyecto y lo traeremos de forma local, es decir lo **clonaremos** aportándole al comando la dirección url de nuestro repositorio:

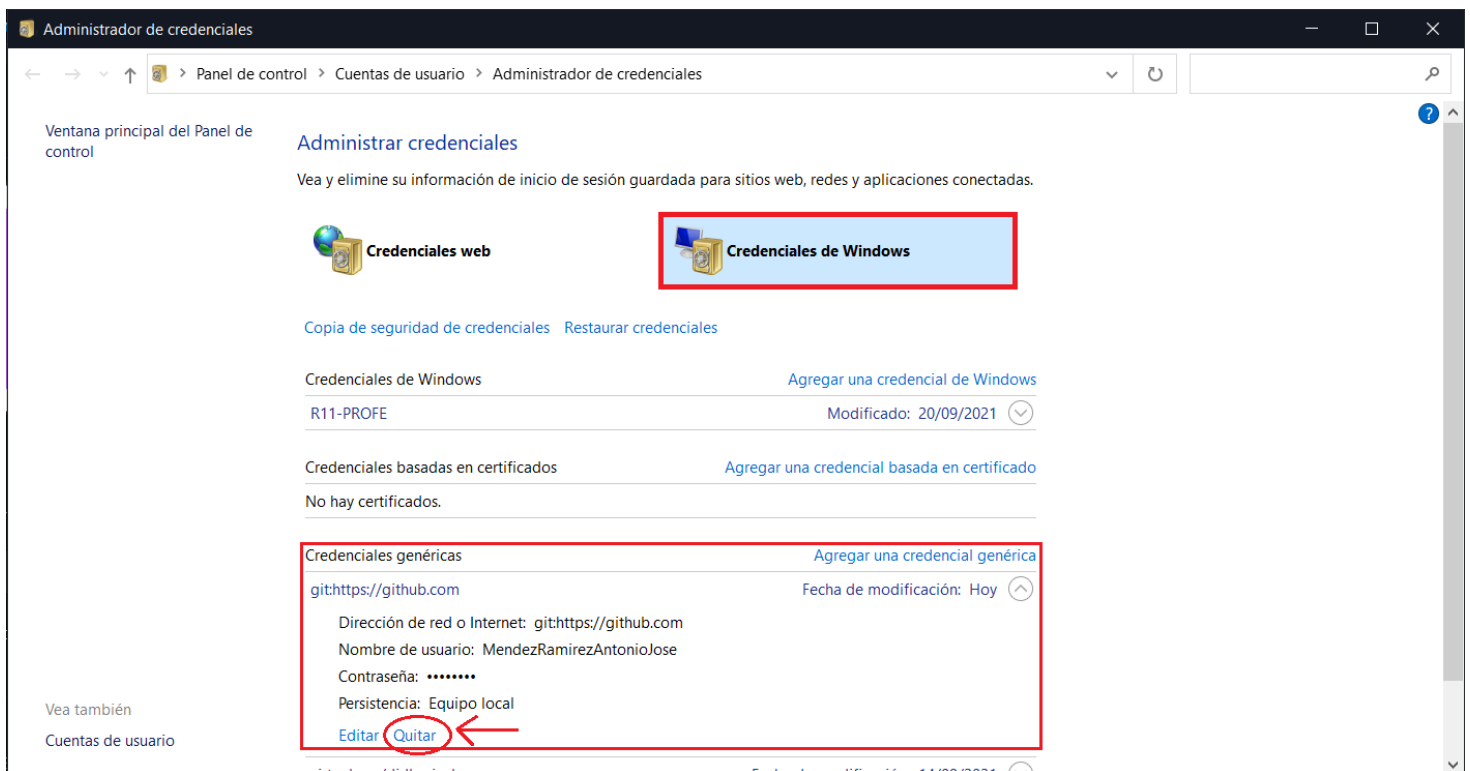
```
PS C:\Users\Toni\Desktop\DAM\GBD\Respaldo01> git clone https://github.com/MendezRamirezAntonioJose/proyecto01.git
Cloning into 'proyecto01'...
remote: Enumerating objects: 17, done.
remote: Counting objects: 100% (17/17), done.
remote: Compressing objects: 100% (7/7), done.
Receiving objects: 100% (17/17), done.
Resolving deltas: 100% (3/3), done.
remote: Total 17 (delta 3), reused 11 (delta 1), pack-reused 0
```


Si ahora vamos al directorio donde lo hemos clonado (en mi caso Respaldo01), veremos que se ha creado el directorio “proyecto01”, y que tiene el mismo contenido que el directorio original:

```
PS C:\Users\Toni\Desktop\DAM\GBD\Respaldo01> tree /F
Listado de rutas de carpetas
El número de serie del volumen es 20DD-002F
C:..
├── proyecto01
│   ├── README.md
│   ├── doc
│   │   └── documentacion.pdf
│   └── src
│       └── fichero1.txt
```

5.6) ¿Qué puedo hacer para quitarle los permisos a un equipo de que me actualice mi propio repositorio?

Para esto, tendremos que irnos al panel de control de Windows > Cuentas de usuario > Administrador de credenciales.



Así, nos aseguramos de que todos los cambios que haga esa persona con ese otro equipo solo se queden de manera local, y que no afecte de ninguna manera al repositorio.