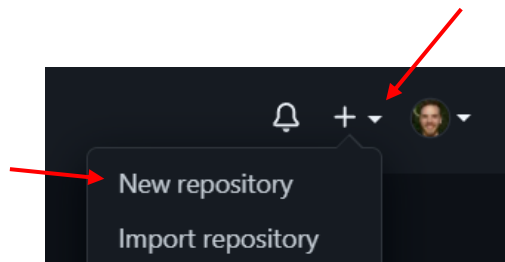


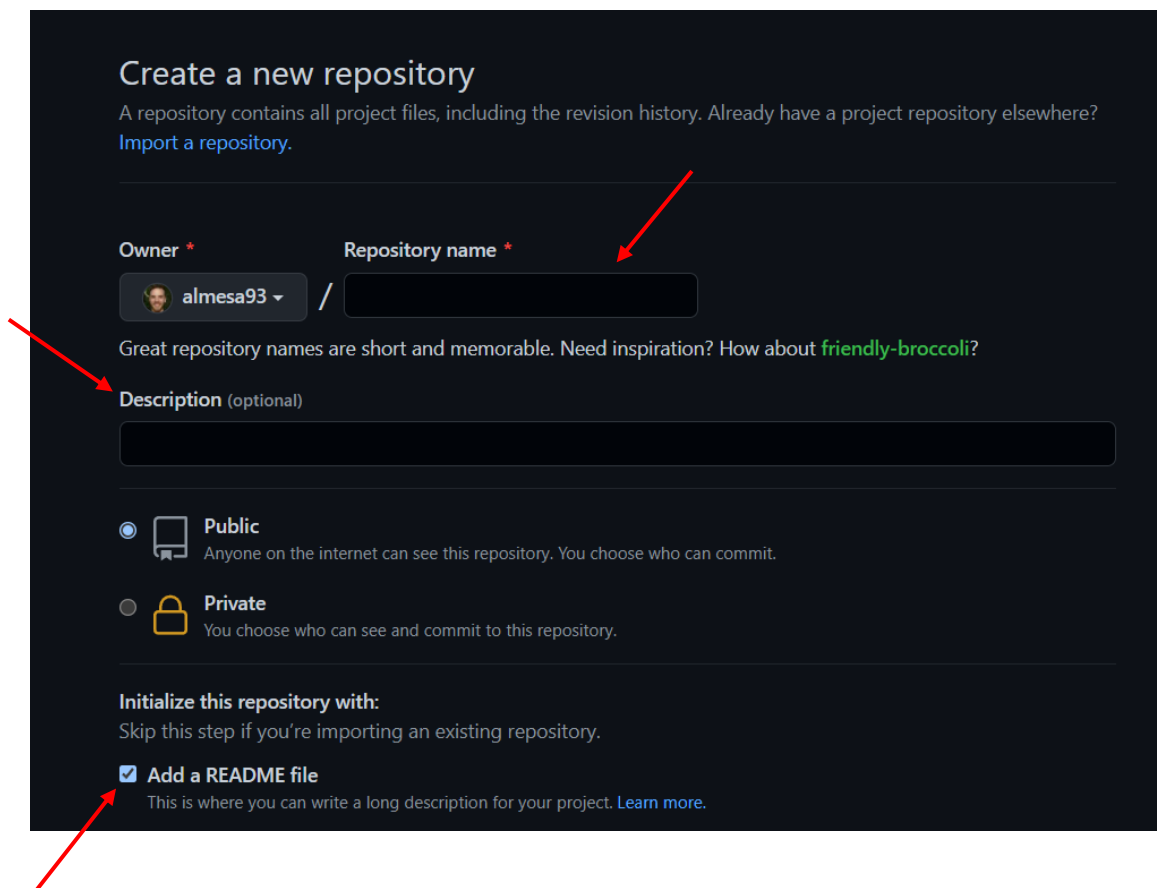
GUÍA GITHUB PARA CREAR Y TRABAJAR CON REPOSITORIOS

1. CREACIÓN DEL REPOSITORIO REMOTO EN GITHUB

Vamos a la página web de GitHub. Hacemos clic en la flecha desplegable al lado del símbolo (+) y damos en “New repository”.



Ponemos el nombre de nuestro repositorio, añadimos una descripción (no obligatorio pero recomendable) y marcamos “Add a README file”

A screenshot of the 'Create a new repository' form on GitHub. The form has a dark background. At the top, it says 'Create a new repository' and 'A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)'. Below this, there are two main sections: 'Owner' and 'Repository name'. The 'Owner' section shows a dropdown menu with 'almesa93' selected. The 'Repository name' section has a text input field. Below these, there is a text prompt: 'Great repository names are short and memorable. Need inspiration? How about [friendly-broccoli?](#)'. Then, there is a 'Description (optional)' section with a text input field. Below that, there are two radio button options: 'Public' (selected) and 'Private'. The 'Public' option has a description: 'Anyone on the internet can see this repository. You choose who can commit.' The 'Private' option has a description: 'You choose who can see and commit to this repository.' At the bottom, there is a section 'Initialize this repository with:' with the text 'Skip this step if you're importing an existing repository.' Below this, there is a checkbox labeled 'Add a README file' which is checked. A red arrow points to this checkbox. Below the checkbox, there is a text prompt: 'This is where you can write a long description for your project. [Learn more.](#)'



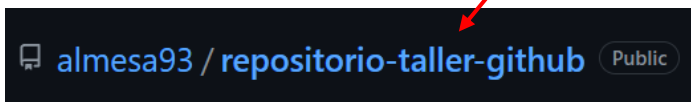
GitHub

Creamos el repositorio.

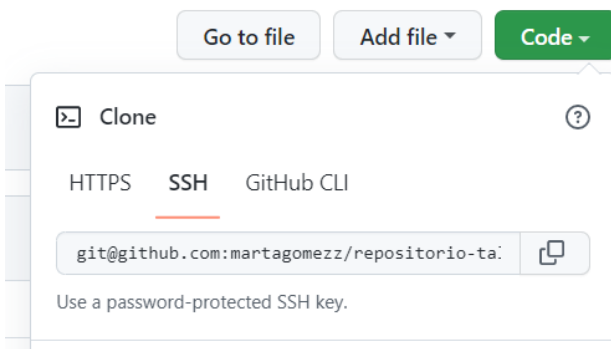
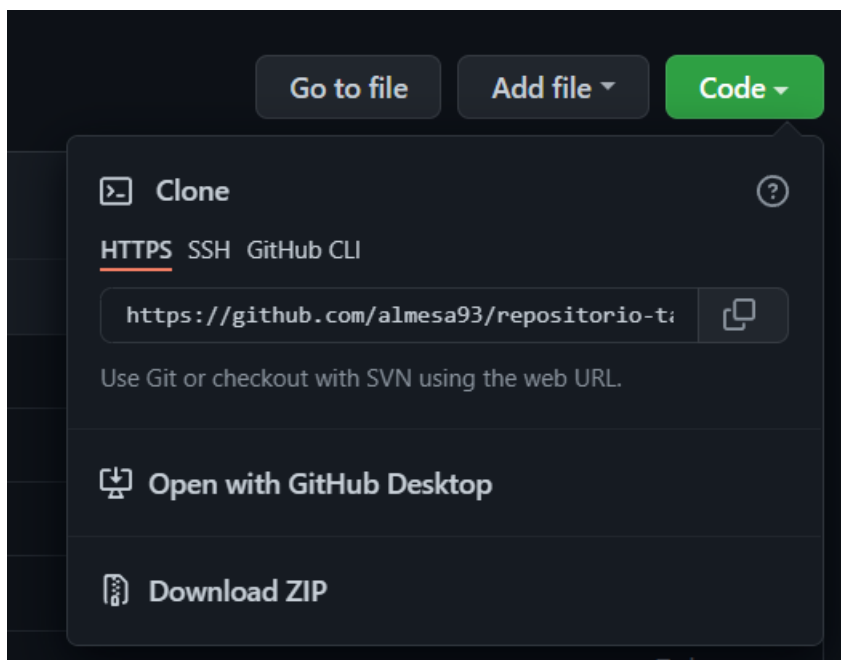
Create repository

2. CLONAMOS EL REPOSITORIO REMOTO PARA CREAR EL REPOSITORIO LOCAL

Hacemos clic en el nombre del repositorio. Esto nos llevará al repositorio.



Copiamos el enlace para clonar el repositorio. HTTPS para Windows, SSH para MAC.

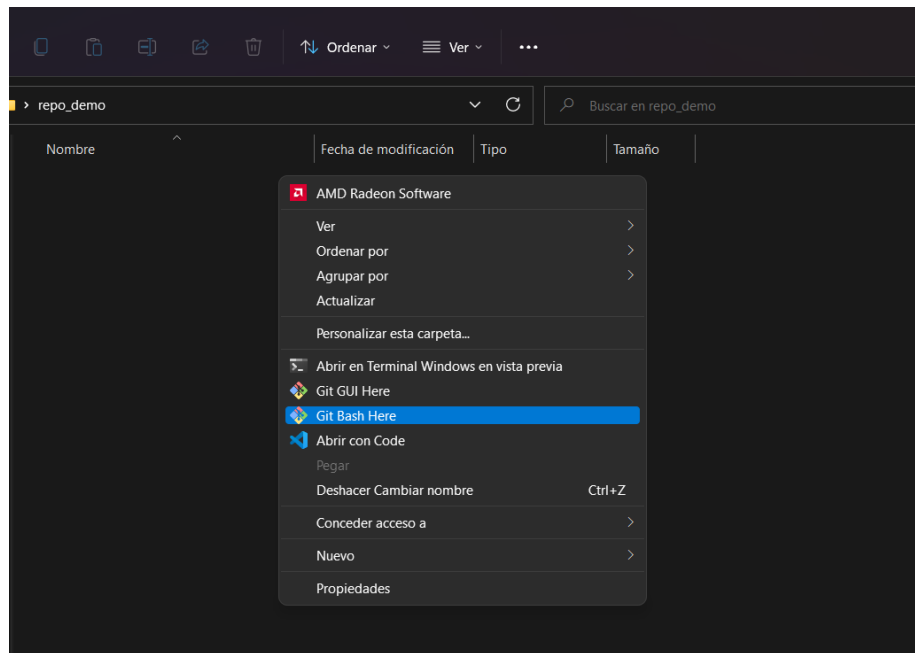


MAC

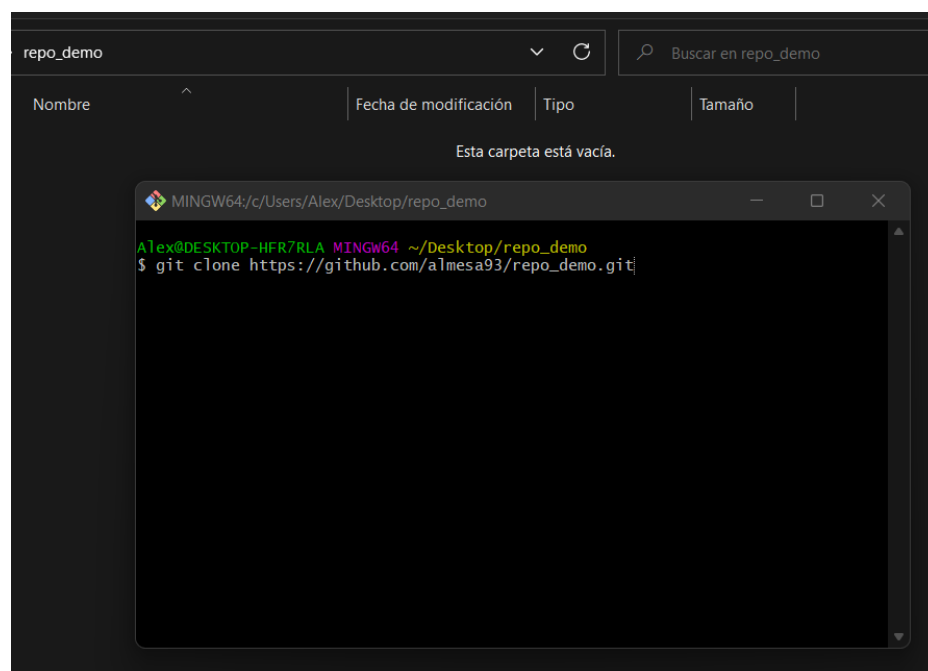
En el caso de los usuarios de MAC, para hacer el git clone es necesario hacerlo con el SSH en lugar del HTTPS. **IMPORTANTE**, el SSH solo se muestra estando logueado en Github y recuerda hacer conectado tu cuenta de Github con el equipo a través de la configuración de la clave (ver guía).



Vamos a la ubicación que queramos, abrimos el Bash de Git “Git Bash Here” para abrir la terminal de Bash dentro de la ruta del ordenador.



Ponemos en el terminal el comando **git clone [enlace_repositorio]**





GitHub

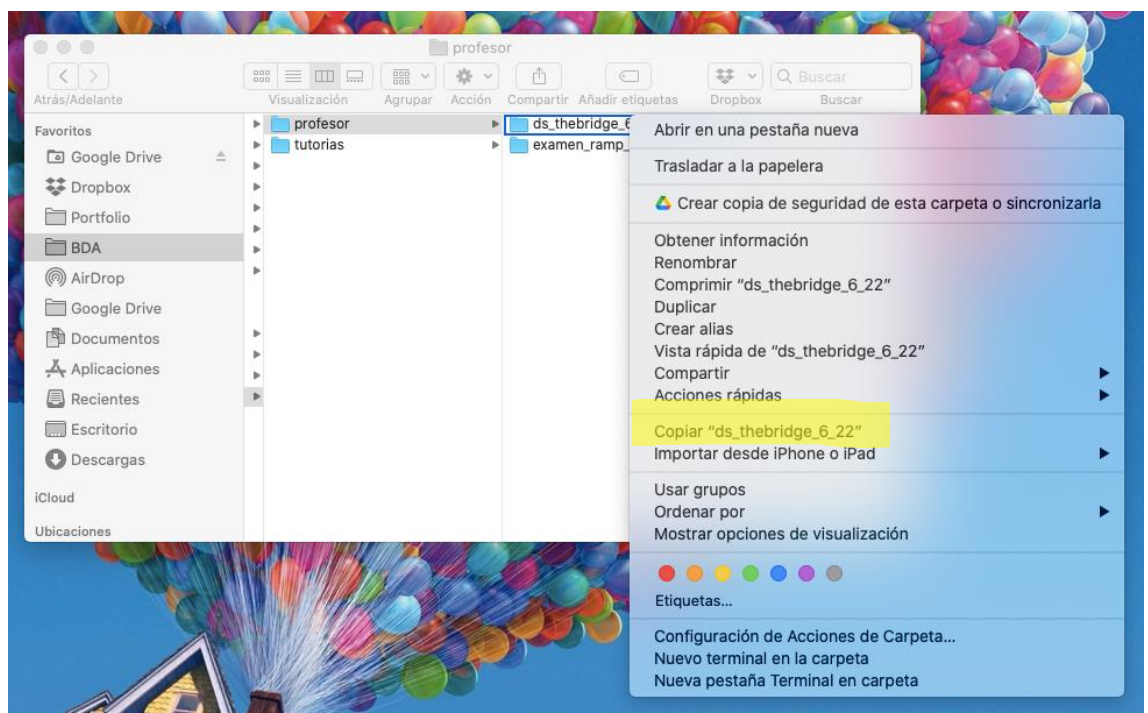
MAC

En el caso de los usuarios de Mac, no es necesario instalarse un software como Bash, trabajemos a través de la propia terminal de MAC o desde Visual Studio.

Buscaremos en Finder la carpeta donde queremos copiar el repositorio y haremos click con el botón derecho. Con el propio desplegable de opciones abierto, pulsaremos la tecla “all options”



Automáticamente nos mostrará nuevas opciones y pulsaremos [Copiar “xxxx” como ruta] que antes se mostraba como [Copiar “xxxx”]



Abriremos la terminal o visual y pondremos los siguientes comandos

1. **cd [ruta de la carpeta]**
2. **git clone [SSH]**



Una vez clonado, aparecerá una carpeta con el nombre del repositorio remoto creado, lo que significa que ya hemos creado el repositorio local.

The screenshot shows a file explorer window on the left with a folder named 'repo_demo' circled in red. To the right is a terminal window with the following commands and output:

```
MINGW64/c/Users/Alex/Desktop/repo_demo
Alex@DESKTOP-HFR7RLA MINGW64 ~/Desktop/repo_demo
$ git clone https://github.com/almesa93/repo_demo.git
Cloning into 'repo_demo'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (3/3), done.

Alex@DESKTOP-HFR7RLA MINGW64 ~/Desktop/repo_demo
$
```

3. TRABAJANDO CON REPOSITORIOS

Una vez creado el repositorio local, entramos en él con el comando “**cd**”.

The screenshot shows the same file explorer with 'repo_demo' circled. The terminal window now shows additional commands:

```
Alex@DESKTOP-HFR7RLA MINGW64 ~/Desktop/repo_demo
$ git clone https://github.com/almesa93/repo_demo.git
Cloning into 'repo_demo'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (3/3), done.

Alex@DESKTOP-HFR7RLA MINGW64 ~/Desktop/repo_demo
$ cd repo_demo

Alex@DESKTOP-HFR7RLA MINGW64 ~/Desktop/repo_demo/repo_demo (main)
$
```

Si nos aparece “(main)” en azul a la derecha de la ruta de la ubicación, significará que estamos dentro del repositorio.

Actualizar repositorios

1. Actualizar la versión en local con la versión en remoto
git pull
2. Actualizar la versión en remoto con los cambios que hemos realizado en local
git add ..
git commit -m 'update'
git push -> s

Se recomienda hacer un **git pull** antes de realizar cualquier cambio en local para asegurarse que estamos trabajando sobre la última versión y no sobre una vieja para que no se generen conflictos cuando subamos dichos cambios posteriormente.



Ramas, trabajar en equipo y niveles

Cuando hacemos un proyecto, trabajaremos a varios niveles. Actualmente estamos en el nivel o rama (que es como llamaremos a esos niveles) con nombre “main”. La rama main, será como el “escaparate de una tienda”. Será lo que queremos que vea nuestro cliente final. Por lo tanto, trabajaremos en niveles inferiores para subir todo a la rama main cuando tengamos finalizado el proyecto. Para crear ramas en niveles inferiores usaremos el comando **git branch nombre_rama**. Con el comando **git branch** a secas, comprobamos las ramas que tenemos en nuestro repositorio local. Cuando creamos una rama, al crearse a un nivel inferior, la nueva rama tendrá los archivos desde la que se crea (lo veremos más adelante). Por ejemplo, si estamos en la rama main y creamos la develop, la rama develop tendrá los archivos que estaban en main.

```
MINGW64/c/Users/Alex/Desktop/repo_demo/repo_demo
Alex@DESKTOP-HFR7RLA MINGW64 ~/Desktop/repo_demo
$ git clone https://github.com/almesa93/repo_demo.git
Cloning into 'repo_demo'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (3/3), done.

Alex@DESKTOP-HFR7RLA MINGW64 ~/Desktop/repo_demo
$ cd repo_demo

Alex@DESKTOP-HFR7RLA MINGW64 ~/Desktop/repo_demo/repo_demo (main)
$ git branch develop

Alex@DESKTOP-HFR7RLA MINGW64 ~/Desktop/repo_demo/repo_demo (main)
$ git branch
develop
* main

Alex@DESKTOP-HFR7RLA MINGW64 ~/Desktop/repo_demo/repo_demo (main)
$
```

Salen que tenemos en nuestro repositorio local la rama main (en color verde porque es en la que estamos actualmente) y la rama develop. Para ver las ramas que tenemos tanto en nuestro repositorio local como el remoto, hacemos **git branch -a**.



```
MINGW64/c/Users/Alex/Desktop/repo_demo/repo_demo
remote: Counting objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (3/3), done.

Alex@DESKTOP-HFR7RLA MINGW64 ~/Desktop/repo_demo
$ cd repo_demo

Alex@DESKTOP-HFR7RLA MINGW64 ~/Desktop/repo_demo/repo_demo (main)
$ git branch develop

Alex@DESKTOP-HFR7RLA MINGW64 ~/Desktop/repo_demo/repo_demo (main)
$ git branch
  develop
* main

Alex@DESKTOP-HFR7RLA MINGW64 ~/Desktop/repo_demo/repo_demo (main)
$ git branch -a
  develop
* main
remotes/origin/HEAD -> origin/main
remotes/origin/main

Alex@DESKTOP-HFR7RLA MINGW64 ~/Desktop/repo_demo/repo_demo (main)
$
```

En rojo muestra nuestras ramas en remoto. Como se puede observar, la rama develop que hemos creado solo está en local, no en el remoto.

Para subir las ramas a nuestro repositorio remoto, cambiaremos a la rama que queremos subir a nuestro repositorio remoto (**git checkout nombre_rama**) y hacemos **git push origin nombre_rama**.

```
MINGW64/c/Users/Alex/Desktop/repo_demo/repo_demo

Alex@DESKTOP-HFR7RLA MINGW64 ~/Desktop/repo_demo/repo_demo (main)
$ git branch -a
  develop
* main
remotes/origin/HEAD -> origin/main
remotes/origin/main

Alex@DESKTOP-HFR7RLA MINGW64 ~/Desktop/repo_demo/repo_demo (main)
$ git checkout develop
Switched to branch 'develop'

Alex@DESKTOP-HFR7RLA MINGW64 ~/Desktop/repo_demo/repo_demo (develop)
$ git push origin develop
Total 0 (delta 0), reused 0 (delta 0), pack-reused 0
remote:
remote: Create a pull request for 'develop' on GitHub by visiting:
remote:   https://github.com/almesa93/repo_demo/pull/new/develop
remote:
To https://github.com/almesa93/repo_demo.git
 * [new branch]      develop -> develop

Alex@DESKTOP-HFR7RLA MINGW64 ~/Desktop/repo_demo/repo_demo (develop)
$
```

Si ahora hacemos **git branch -a**, veremos que aparece en rojo la rama develop.



GitHub

```
MINGW64/c:/Users/Alex/Desktop/repo_demo/repo_demo
Alex@DESKTOP-HFR7RLA MINGW64 ~/Desktop/repo_demo/repo_demo (main)
$ git checkout develop
Switched to branch 'develop'

Alex@DESKTOP-HFR7RLA MINGW64 ~/Desktop/repo_demo/repo_demo (develop)
$ git push origin develop
Total 0 (delta 0), reused 0 (delta 0), pack-reused 0
remote:
remote: Create a pull request for 'develop' on GitHub by visiting:
remote:   https://github.com/almesa93/repo_demo/pull/new/develop
remote:
To https://github.com/almesa93/repo_demo.git
 * [new branch]      develop -> develop

Alex@DESKTOP-HFR7RLA MINGW64 ~/Desktop/repo_demo/repo_demo (develop)
$ git branch -a
* develop
  main
remotes/origin/HEAD -> origin/main
remotes/origin/develop
remotes/origin/main

Alex@DESKTOP-HFR7RLA MINGW64 ~/Desktop/repo_demo/repo_demo (develop)
$
```

Ya tenemos hecha la rama en la que vamos a trabajar. La rama develop será donde crearemos nuestra beta del proyecto, por lo que el paso final será fusionar la rama develop con main.

Un proyecto, normalmente, se divide en varias tareas. Por ejemplo, un Hundir la Flota tendrá que tener una parte en la que construya el tablero con los barcos y otra con el juego en sí. Sabiendo esto, tendremos que crear una rama para cada una de esas tareas. Cambiamos a la rama desde la que deseamos crear esas ramas de tareas (“develop” en nuestro caso), y volvemos a hacer **git branch nombre_rama**.

```
Alex@DESKTOP-HFR7RLA MINGW64 ~/Desktop/repo_demo/repo_demo (develop)
$ git branch epic-branch-1

Alex@DESKTOP-HFR7RLA MINGW64 ~/Desktop/repo_demo/repo_demo (develop)
$ git branch epic-branch-2

Alex@DESKTOP-HFR7RLA MINGW64 ~/Desktop/repo_demo/repo_demo (develop)
$
```

A su vez, cada tarea puede tener una subtarea. En nuestro ejemplo anterior, cuando queremos construir el tablero, tendremos dos fases: la de crear ese tablero y la de posicionar los barcos. Por lo tanto, tendremos que crear esas dos sub tareas desde la rama creada en el paso anterior.



GitHub

```
Alex@DESKTOP-HFR7RLA MINGW64 ~/Desktop/repo_demo/repo_demo (develop)
$ git checkout epic-branch-1
Switched to branch 'epic-branch-1'

Alex@DESKTOP-HFR7RLA MINGW64 ~/Desktop/repo_demo/repo_demo (epic-branch-1)
$ git branch ticket-1

Alex@DESKTOP-HFR7RLA MINGW64 ~/Desktop/repo_demo/repo_demo (epic-branch-1)
$ git branch ticket-2

Alex@DESKTOP-HFR7RLA MINGW64 ~/Desktop/repo_demo/repo_demo (epic-branch-1)
$
```

```
Alex@DESKTOP-HFR7RLA MINGW64 ~/Desktop/repo_demo/repo_demo (epic-branch-1)
$ git branch -a
develop
* epic-branch-1
epic-branch-2
main
ticket-1
ticket-2
remotes/origin/HEAD -> origin/main
remotes/origin/develop
remotes/origin/main
```

Ahora mismo, dentro de nuestro repositorio tendremos lo siguiente (estamos en la carpeta de Windows o de Finder):

Nombre	Fecha de modificación	Tipo	Tamaño
.git	03/03/2022 11:50	Carpeta de archivos	
README	03/03/2022 11:10	Archivo de origen ...	1 KB

Vamos a realizar una de las subtareas. Primero nos cambiamos a ticket-1 (primera subtarea de epic-branch-1) y vamos a realizar alguna modificación. Por ejemplo, creamos un archivo txt. Con un **git status**, nos mostrará en qué estado están esos cambios.



GitHub

```
Alex@DESKTOP-HFR7RLA MINGW64 ~/Desktop/repo_demo/repo_demo (ticket-1)
$ git status
On branch ticket-1
Untracked files:
  (use "git add <file>..." to include in what will be committed)
       doc_prueba.txt
```

Dice literalmente “usa git add nombre_archivo para poder ser commiteado”.

Hacemos **git add** para subir los cambios al “staging area” de una de las siguientes formas:

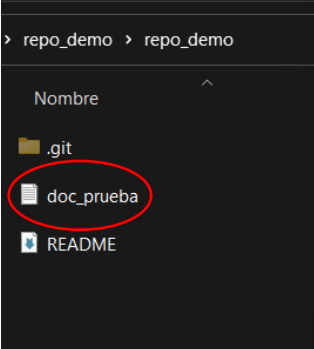
- “**git add .**” para guardar todos los cambios
- “**git add nombre_archivo**” para guardar los cambios del archivo especificado

```
Alex@DESKTOP-HFR7RLA MINGW64 ~/Desktop/repo_demo/repo_demo (ticket-1)
$ git add .

Alex@DESKTOP-HFR7RLA MINGW64 ~/Desktop/repo_demo/repo_demo (ticket-1)
$ git status
On branch ticket-1
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
       new file:   doc_prueba.txt
```

“Cambios para ser commiteados”.

Y luego “**git commit -m ‘comentario’**” para guardar los cambios.

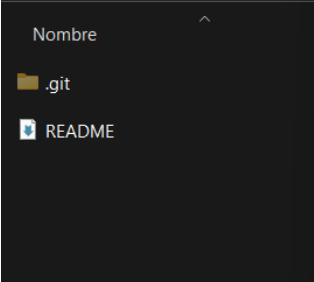


```
Alex@DESKTOP-HFR7RLA MINGW64 ~/Desktop/repo_demo/repo_demo (epic-branch-1)
$ git branch -a
* epic-branch-1
  epic-branch-2
  main
  ticket-1
  ticket-2
  remotes/origin/HEAD -> origin/main
  remotes/origin/develop
  remotes/origin/main

Alex@DESKTOP-HFR7RLA MINGW64 ~/Desktop/repo_demo/repo_demo (epic-branch-1)
$ git checkout ticket-1
Switched to branch 'ticket-1'

Alex@DESKTOP-HFR7RLA MINGW64 ~/Desktop/repo_demo/repo_demo (ticket-1)
$
```

Si volvemos a cambiar a “epic-branch-1” (que sería el nivel justo superior), nos encontraremos con que desaparece el archivo.



```
Alex@DESKTOP-HFR7RLA MINGW64 ~/Desktop/repo_demo/repo_demo (ticket-1)
$ git commit -m 'comentario'
[ticket-1 3fcdbe1] comentario
1 file changed, 1 insertion(+)
create mode 100644 doc_prueba.txt

Alex@DESKTOP-HFR7RLA MINGW64 ~/Desktop/repo_demo/repo_demo (ticket-1)
$ git checkout epic-branch-1
Switched to branch 'epic-branch-1'

Alex@DESKTOP-HFR7RLA MINGW64 ~/Desktop/repo_demo/repo_demo (epic-branch-1)
$
```



Esto es debido a que los cambios se han realizado en “ticket-1”, no en la que estamos ahora. Para que aparezca también aquí, tendremos que hacer un **git merge ticket-1** (se hace en la rama a la que queremos llevar los cambios y se pone el nombre de la rama en la que se han hecho esos cambios).

```
Switched to branch 'epic-branch-1'
Alex@DESKTOP-HFR7RLA MINGW64 ~/Desktop/repo_demo/repo_demo (epic-branch-1)
$ git merge ticket-1
Updating 78a9cb7..3fcdbe1
Fast-forward
 doc_prueba.txt | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 doc_prueba.txt
Alex@DESKTOP-HFR7RLA MINGW64 ~/Desktop/repo_demo/repo_demo (epic-branch-1)
$
```

Como veis, ahora aparece el archivo en la “epic-branch-1”. Si ahora mismo, volviera a crear otra rama (por ejemplo, ticket-3), esta nueva rama contendría los 3 archivos que de la imagen.

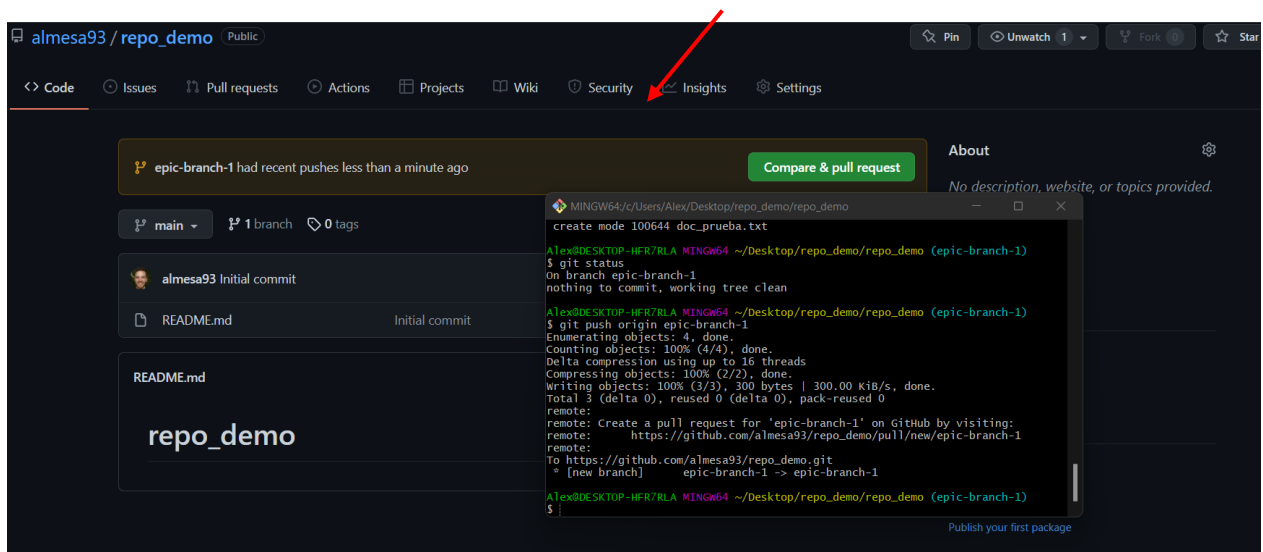
Si tuviera que hacer más subtareas, procedemos del mismo modo. Entramos en la rama de la subtask, hacemos los cambios pertinentes, volvemos a la epic y hacemos el **git merge**.

Una vez hemos hecho el merge de todos los archivos, hacemos un **git status** para comprobar que no hay nada de commitar.

```
Alex@DESKTOP-HFR7RLA MINGW64 ~/Desktop/repo_demo/repo_demo (epic-branch-1)
$ git status
On branch epic-branch-1
nothing to commit, working tree clean
```

Cuando tengamos esto, haremos un **git push origin epic-branch-1** para subir la tarea al repositorio remoto.

Si vamos a GitHub, nos damos cuenta de que ha salido una notificación:





GitHub

De este modo, hemos creado una **PULL REQUEST**. Esta Pull Request nos indica que alguien está intentando subir un archivo al repositorio remoto. Si hacemos clic en el botón, nos llevará a la siguiente pantalla:

Open a pull request

Create a new pull request by comparing changes across two branches. If you need to, you can also [compare across forks](#).

base: main ← compare: epic-branch-1 ✓ **Able to merge.** These branches can be automatically merged.

comentario

Write Preview

Leave a comment

Attach files by dragging & dropping, selecting or pasting them.

Create pull request

Remember, contributions to this repository should follow our [GitHub Community Guidelines](#).

Aquí le pondremos una descripción de los cambios que se hayan hecho, para que todo el que quiera ver los cambios realizados sepa qué se ha hecho (es buena praxis).

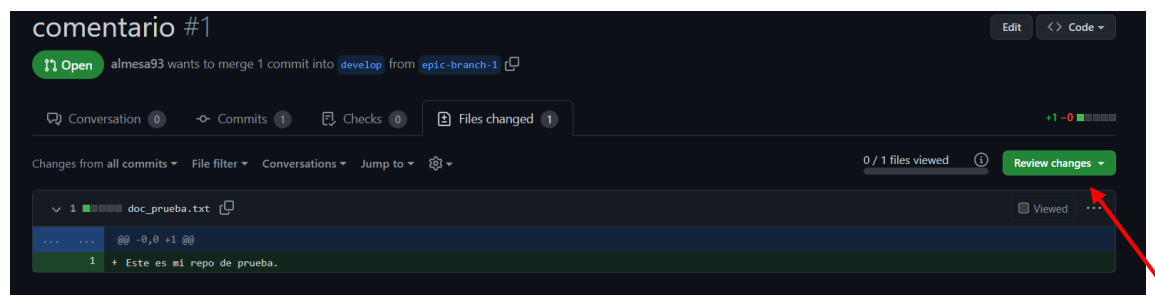
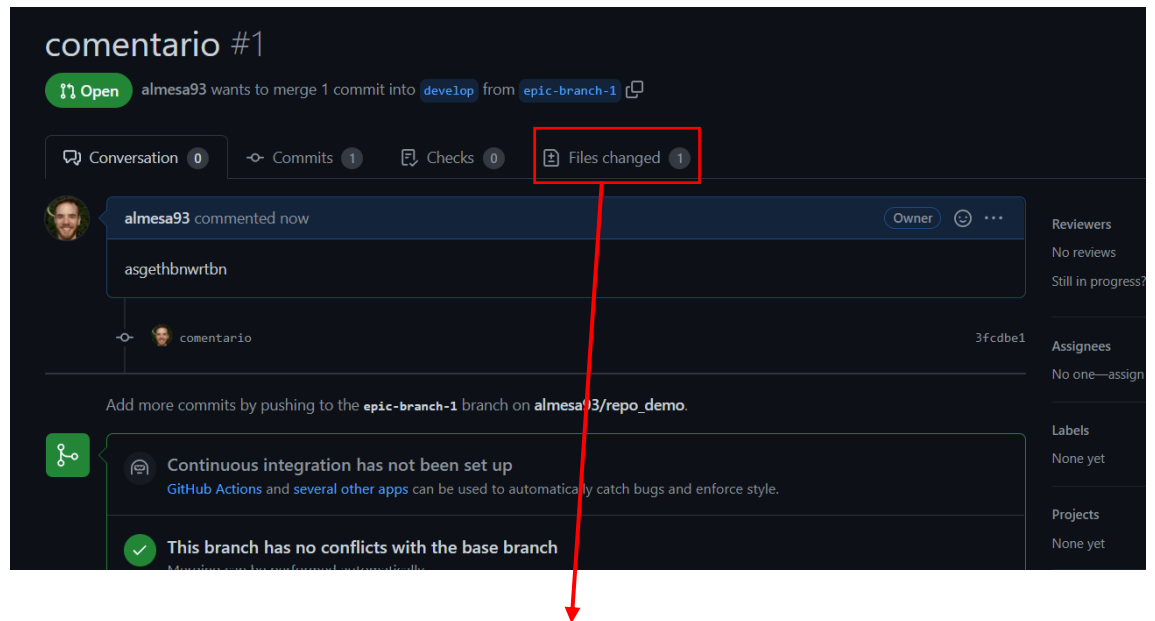
Como hemos dicho que trabajamos en develop y no en main, habrá que hacer el siguiente cambio:

base: develop ← compare: epic-branch-1 ✓ **Able to merge.** These branches can be automatically merged.



Ahora damos en **“Create pull request”**.

De esas 4 pestañas, en la que pone **“Files changed”** podemos ver los cambios que se han realizado.



Una vez comprobado todo, hacemos clic en **“Review changes”** y ahí nos dará opción de poner un comentario, aprobar o solicitar cambios (como nos lo estamos haciendo a nosotros mismos la pull request, no nos deja cambiar de opción).



GitHub

The screenshot shows the 'Finish your review' dialog box in GitHub. At the top, it says '0 / 1 files viewed' and has a 'Review changes' button. The dialog has two tabs: 'Write' (selected) and 'Preview'. The 'Write' tab contains a text area for 'Leave a comment' and a section for 'Attach files by dragging & dropping, selecting or pasting them.' Below this, there are three radio button options: 'Comment' (selected), 'Approve', and 'Request changes'. Each option has a description: 'Submit general feedback without explicit approval.', 'Submit feedback and approve merging these changes.', and 'Submit feedback that must be addressed before merging.' respectively. At the bottom, there is a green 'Submit review' button.

Ponemos un comentario (esta vez sí es obligatorio) y damos a “**Submit review**”.

Aparece lo siguiente. Hacemos el “**Merge pull request**”.

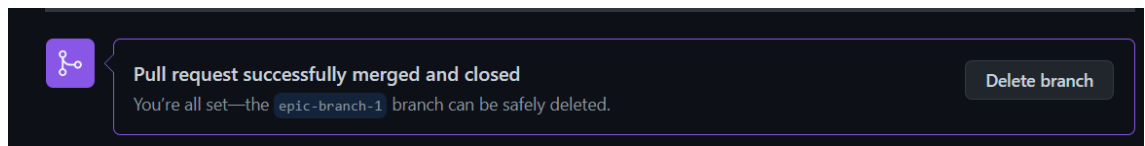
The screenshot shows the merge status bar in GitHub. It has a title 'Add more commits by pushing to the epic-branch-1 branch on almesa93/repo_demo.' Below the title, there are two status boxes. The first box has a red icon and says 'Continuous integration has not been set up' with a link to 'GitHub Actions and several other apps'. The second box has a green checkmark icon and says 'This branch has no conflicts with the base branch' with the text 'Merging can be performed automatically.' Below these boxes, there is a green 'Merge pull request' button and a link to 'You can also open this in GitHub Desktop or view command line instructions.'

Confirmamos:

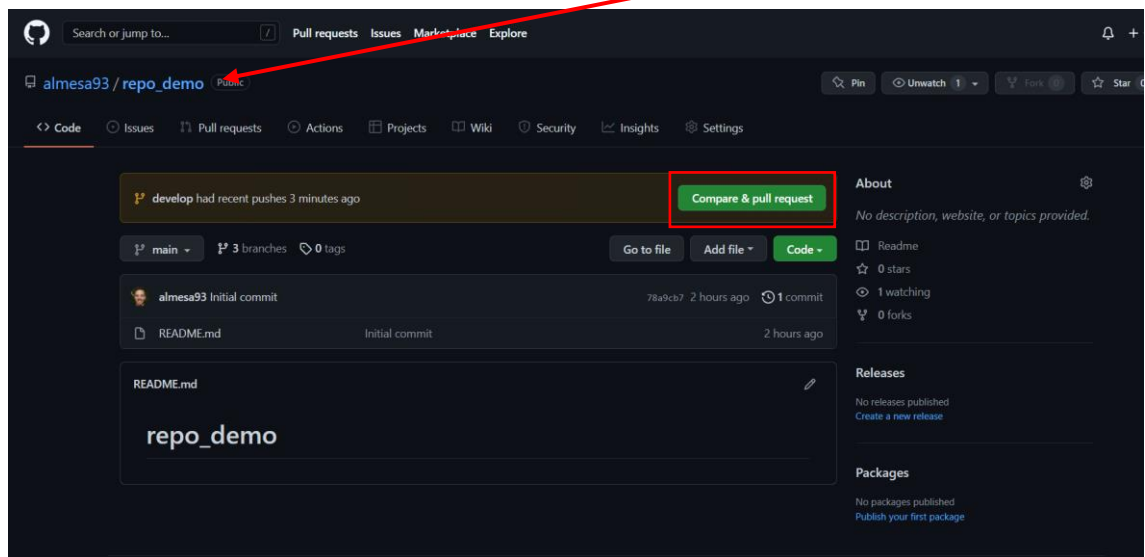
The screenshot shows the 'Confirm merge' dialog box in GitHub. It has a title 'Add more commits by pushing to the epic-branch-1 branch on almesa93/repo_demo.' Below the title, there is a profile picture of a person. To the right of the profile picture, there is a text input field with the text 'Merge pull request #1 from almesa93/epic-branch-1'. Below this, there is another text input field with the text 'comentario'. At the bottom, there are two buttons: 'Confirm merge' (green) and 'Cancel' (grey).



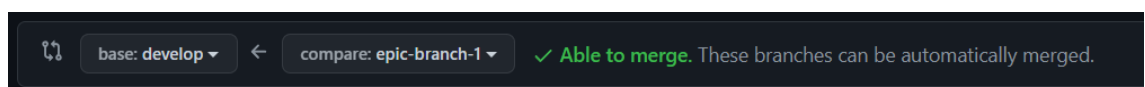
Y ya tendremos nuestra rama develop actualizada en el repositorio remoto.



Si volvemos a la página principal del repositorio (haciendo clic en el nombre del repo), volverá a aparecer una Pull Request:



Esto es debido a que los cambios no han llegado al main, ya que anteriormente lo hicimos desde epic-branch-1 a la rama develop, y te da la opción de hacerlo desde develop a main. Esta imagen es de lo que hicimos en la **Pull Request** anterior.



En principio no le haremos caso hasta no terminar el proyecto.

Una vez hayamos terminado, sí que habrá que hacerlo. Será del mismo modo que el anterior pero de develop a main:

