



# CONTROL DE VERSIONES

José Corrochano Pardo | Alfonso Rincón Cuerva  
1ºDAM

## ÍNDICE

¿Qué es un control de versiones, y para qué sirve? .....	2
¿Qué es Git y GitHub?.....	2
Otros sistemas de control de versiones .....	2
Sistemas centralizados y distribuidos .....	3
Definiciones de Git .....	3
Práctica con Git y GitHub .....	4
GitHub .....	5
Local .....	9
Bibliografía .....	15
Repositorios .....	15

## TEORÍA

### ¿Qué es un control de versiones y para que sirve?

El control de versiones consiste en la gestión de los diversos cambios que se realizan en un producto a lo largo de la creación de este.

Esto nos podrá servir para múltiples funciones, como volver a versiones anteriores del producto para poder modificarlas, encontrar un error más fácilmente al poder ir a versiones anteriores, colaborar con otras personas en el mismo proyecto o tener un historial de todos los cambios que se han ido realizando en un proyecto.

### ¿Qué es Git y Github?

Git es un sistema de control de versiones distribuido y local. Permite rastrear los cambios en los archivos, crear ramas y trabajar en versiones anteriores del código.

GitHub en cambio es la plataforma que nos permite almacenar nuestros repositorios en la nube. Funciona como un repositorio centralizado donde, mediante Git, puedes almacenar y compartir tus proyectos. Además, ofrece otras funcionalidades, como un seguimiento de problemas y proyectos en equipos.

### ¿Qué otros sistemas de control de versiones se suelen utilizar en la actualidad? Descríbelos.

- **CVS:** es un modelo cliente-servidor, que permite a varias personas trabajar en el mismo proyecto.



- **Apache Subversion:** es un sistema de control de versiones centralizado, gratuito y de código abierto. A diferencia de Git, almacena todas las versiones de los archivos en un repositorio centralizado, que suele estar ubicado en un servidor.



mercurial

- **Mercurial:** es un sistema de control de versiones distribuido, gratuito y de código abierto, que es característico por su flexibilidad y eficiencia. También tiene una comunidad activa, y, aunque puede ser un poco más difícil de aprender al principio que otros controles de versiones, tiene una interfaz muy intuitiva.
- **Monotone:** sistema de control de versiones distribuido, gratuito y de código abierto, fácil de usar y muy seguro. Al ser menos conocido, su comunidad es más pequeña y ofrece menor cantidad de herramientas y plugins.

## ¿Qué son los sistemas centralizados y distribuidos? Diferencias

- **SISTEMAS CENTRALIZADOS:** son aquellos que almacenan el repositorio en un servidor central, de manera que los usuarios trabajan con una copia local del repositorio la cual se sincroniza con el servidor central.
  - Requieren de una conexión constante al servidor.
  - Suelen ser más fáciles de usar
  - Existe un único historial de versiones para todo el proyecto
  - El flujo de trabajo es menos flexible y escalable
- **SISTEMAS DISTRIBUIDOS:** cada usuario tienen una copia completa del repositorio en su propio ordenador, sin que haya un servidor central único. En estos, los usuarios trabajan de forma independiente en sus proyectos, y luego estos se fusionan.
  - Se puede trabajar sin conexión a internet
  - Es más seguro, ya que cada copia del repositorio es una copia de seguridad
  - El flujo de trabajo es más flexible y escalable

## Define los siguientes conceptos en el SCV (Sistema de control de versiones) git:

### REPOSITORIO

Se trata de un espacio para tu proyecto donde se almacena código, archivos y el historial de cada cambio. En este repositorio se puede trabajar solo o en equipo, y elegir si compartir tu proyecto o mantenerlo privado.

### .GIT

Es una carpeta oculta que guarda toda la información necesaria para su funcionamiento, incluyendo la configuración del repositorio, el historial de las versiones del proyecto, las conexiones o enlaces a otros repositorios y scripts que ayudan a la automatización de tareas. En la configuración podremos ver por ejemplo el nombre de usuario y email que adjudicaremos al inicializar el repositorio con los comandos necesarios para ello. De forma predeterminada los archivos ocultos no se muestran en nuestro ordenador, por lo que tendremos que activar esta función para acceder a esta carpeta.

### ADD

Es un comando que se utiliza para preparar los cambios realizados en los archivos de tu proyecto para ser incluidos en la siguiente confirmación. Puedes usar tanto "git add ." que añadiría todos los nuevos cambios realizados, o "git add NombreArchivo", que añadiría el archivo especificado.

### COMMIT

Es un comando que sirve para crear una instantánea o registro de los cambios que has preparado, anteriormente con git add, en tu proyecto. Es como tomar una fotografía del estado de tu código en un momento determinado. Cuando realizas un commit se debe añadir un comentario indicando lo que ha sido modificado en el proyecto.

## PULL

Se utiliza para actualizar tu repositorio local con los cambios del repositorio remoto. Te permite sincronizar tu trabajo con el trabajo de otros colaboradores o con el repositorio principal del proyecto.

## PUSH

Se utiliza para lo contrario de git pull: en lugar de traer cambios del control remoto, sirve para subir los cambios locales que has realizado a un repositorio remoto.

## STATUS

Se utiliza para obtener una instantánea del estado actual de tu repositorio Git. Te informa sobre los cambios que has realizado en tu proyecto, tanto los que ya has preparado para confirmar como los que aún no.

## PRACTICAMOS CON GIT Y GITHUB

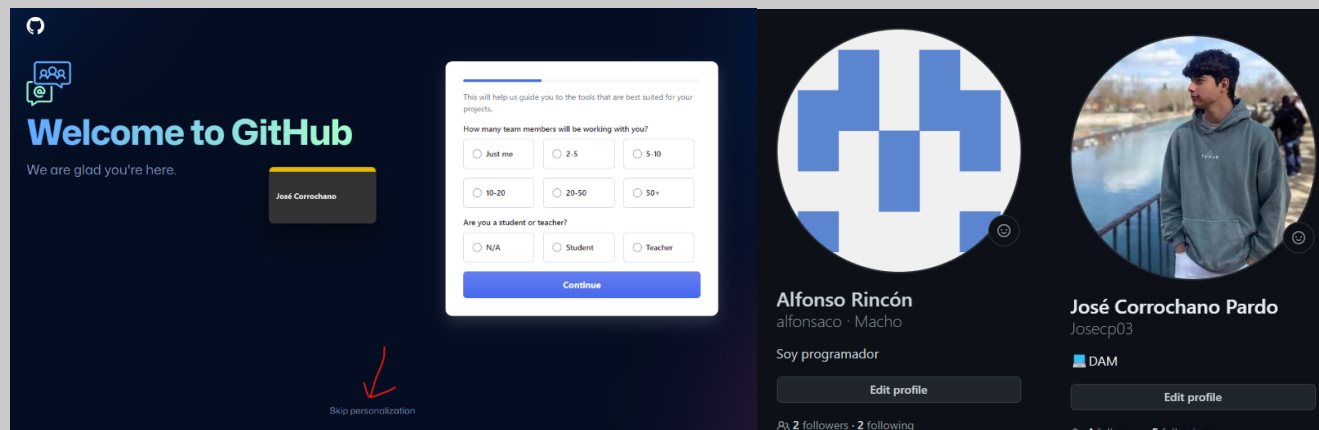
### CREAR CUENTA

Para crearnos una cuenta en Github necesitaremos acceder a este enlace e introducir nuestros datos:

[https://github.com/signup?ref\\_cta=Sign+up&ref\\_loc=header+logged+out&ref\\_page=%2F&source=header-home](https://github.com/signup?ref_cta=Sign+up&ref_loc=header+logged+out&ref_page=%2F&source=header-home)

Una vez introduzcamos nuestro correo, contraseña y nombre, nos mandarán un código al correo proporcionado para seguir con la creación de nuestra cuenta. Después, podremos ir personalizando esta cuenta, o bien saltarnos esta parte:

Si decidimos saltarnos la personalización, ya tendríamos la cuenta creada



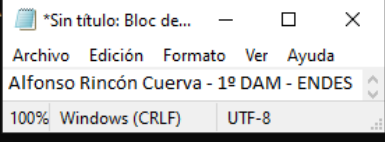
## COMPROBAR INSTALACIÓN GIT

Para comprobar si git está instalado, tendremos que acceder al CMD de nuestro ordenador. Para acceder al CMD, pulsaremos la tecla Windows y escribiremos “CMD”

Una vez dentro, escribiremos el siguiente comando:

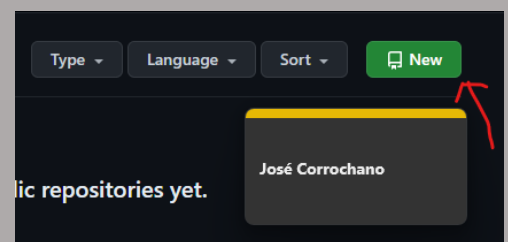
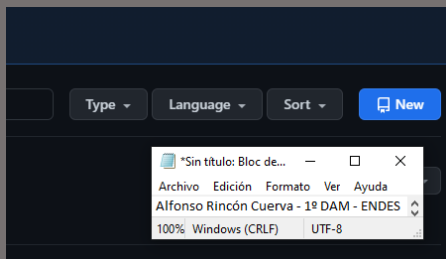
```
Microsoft Windows [version 10.0.19041.1415]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\Users\PROGRAMACION>git --version
git version 2.44.0.windows.1
```

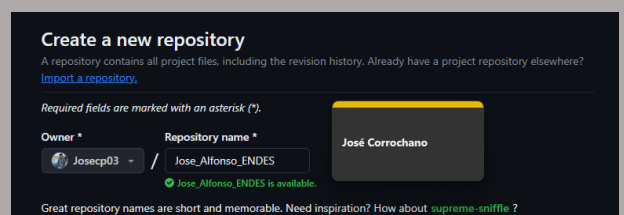
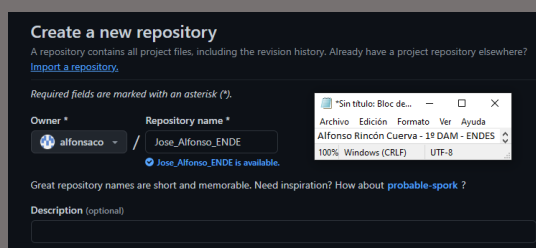


## EN GITHUB

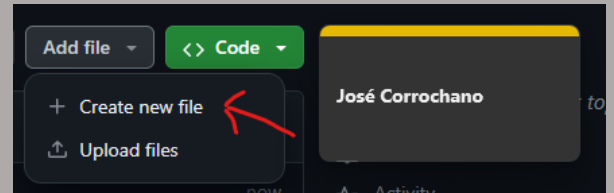
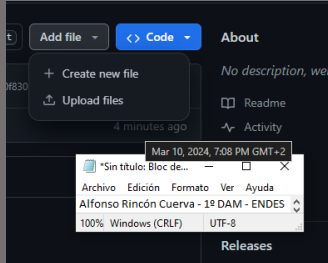
Para crear un repositorio en GitHub, vamos a Perfil, donde encontraremos un botón, que nos permitirá crearlo.



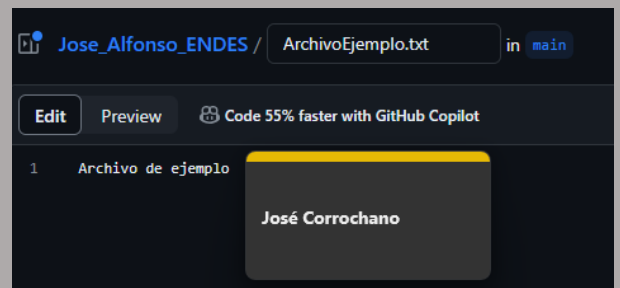
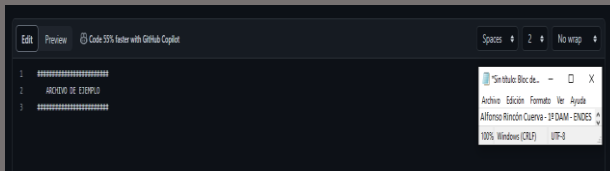
Una vez dentro, tendremos que indicar el nombre del repositorio. También podremos indicar si queremos que nuestro repositorio sea público o privado, además de añadir un REEDME y otras funciones.



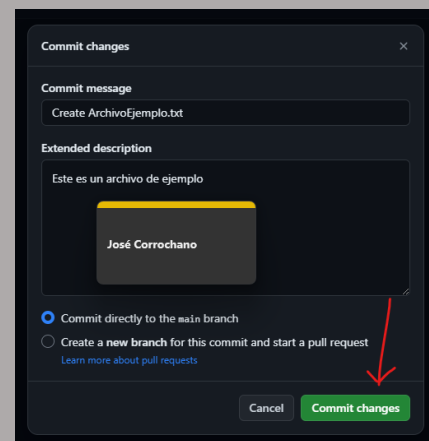
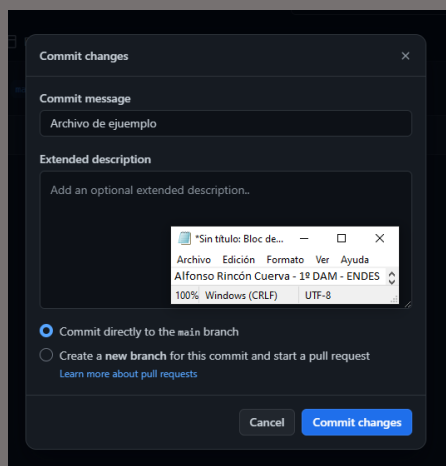
Una vez tenemos el repositorio creado, para crear un archivo debemos irnos a la sección “Add File” y crear manualmente un archivo



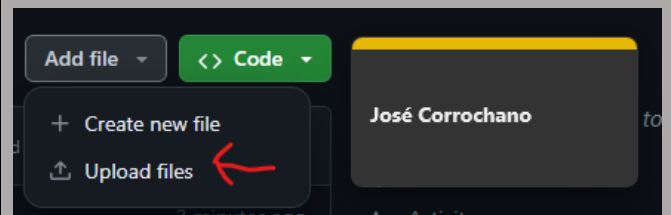
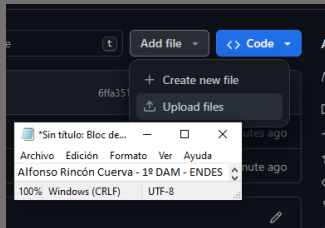
Dentro del editor de texto crearíamos el archivo



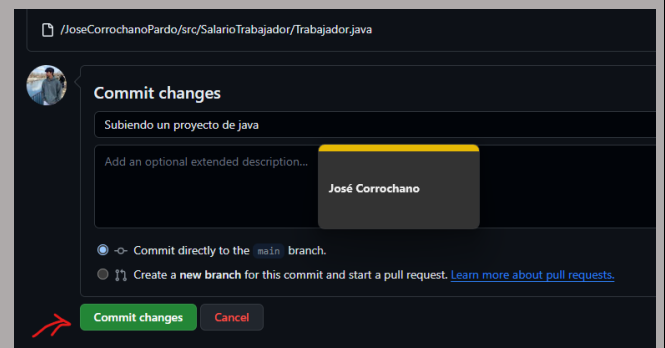
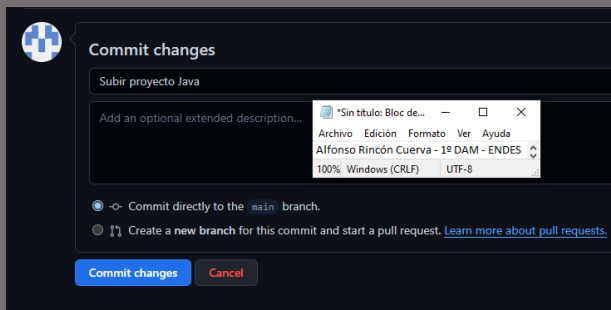
Para finalizar la creación de este archivo tendremos que confirmar los cambios haciendo un commit de este archivo como a continuación:



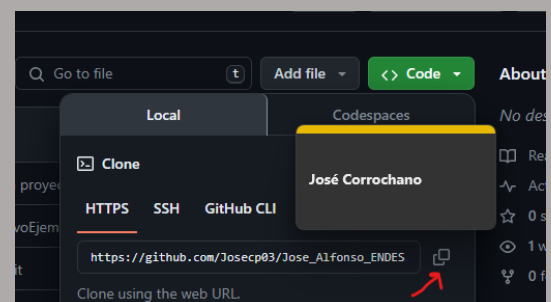
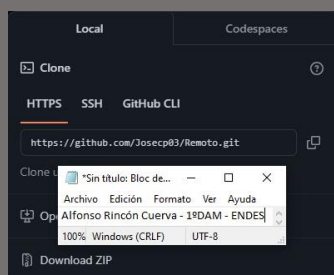
Ahora, para subir un proyecto en vez de crear un archivo y similar, tenemos que ir a la misma sección donde creamos el archivo, pero esta vez en vez de crearlo, lo subiremos



Simplemente con arrastrar los archivos ya se subirán, y después de eso tendremos que hacer un commit de los cambios como hemos hecho anteriormente introduciendo un mensaje representativo del cambio que hemos hecho

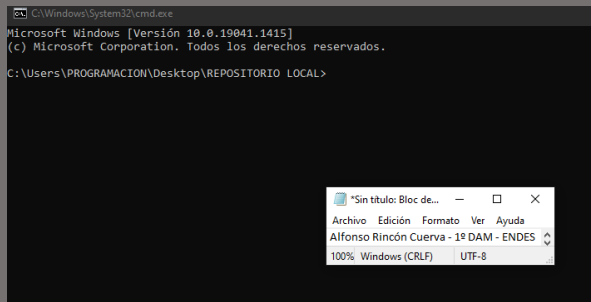
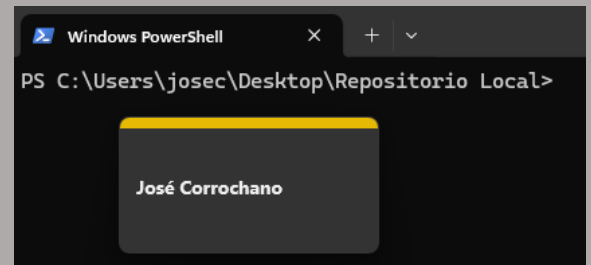


Ahora, para clonar un repositorio de GitHub tendremos que utilizar una serie de comandos que ejecutaremos en la terminal de nuestro ordenador o bien en el GitBash. Pero antes, tenemos que adquirir un enlace para clonar dicho repositorio. Este enlace se encuentra en la sección “Code” de nuestro repositorio:

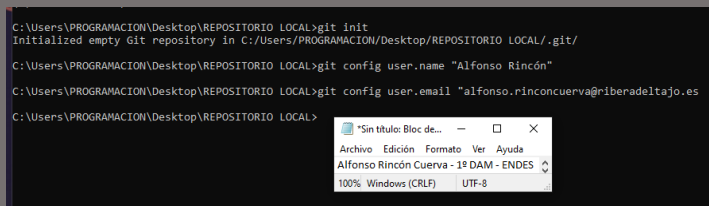
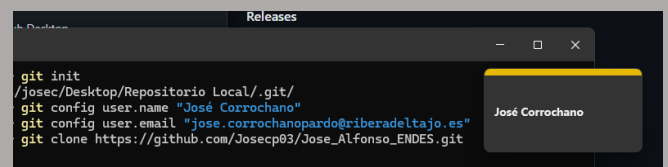


Una vez tenemos el enlace, tendremos que dirigirnos a la carpeta donde vamos a clonar el repositorio y abrir el terminal desde esa carpeta

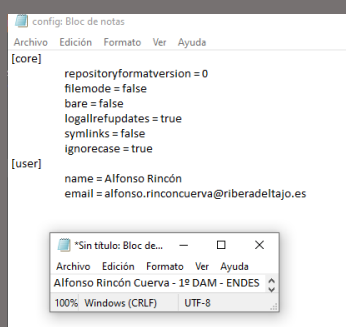
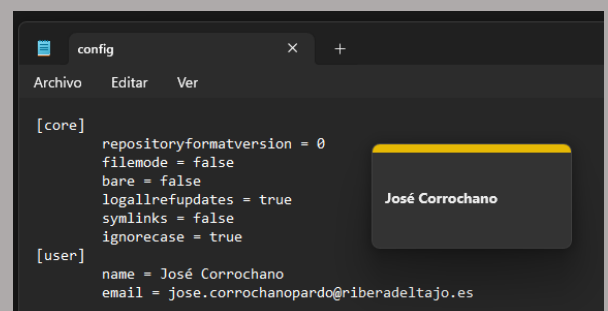


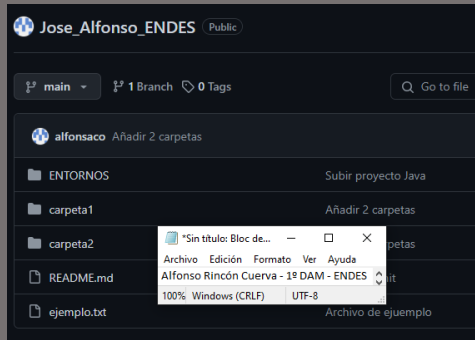
Una vez estamos en la carpeta que queremos, tendremos que ejecutar una serie de comandos. Tendremos que inicializar el repositorio con “git init”, después asignaremos un nombre y email con “git config user.”, y por último clonaremos el repositorio con “git clone” + el enlace que hemos copiado anteriormente de nuestro repositorio.

En la carpeta oculta .git podremos ver como se ha guardado la configuración de nuestro nombre y email. Esta configuración es indispensable, ya que cuando trabajemos en proyectos en grupo, a la hora de hacer un commit, este nombre y correo se verá reflejado en él, por lo que en todo momento podremos saber quién ha hecho cada cambio y acceder a esta información

Una vez clonado el repositorio, para subir más archivos como otras dos carpetas con archivos dentro, basta con hacer exactamente lo mismo que hemos hecho en el apartado de subir un proyecto. Hay que arrastrarlos y añadir un commit con un mensaje descriptivo.



## EN LOCAL

Para iniciar un repositorio local de un proyecto que tenemos en nuestro ordenador deberemos de inicializar el repositorio primero con `git init`, y después configurar nuestro nombre y email con `git config user`. Una vez creado el repositorio local, para subirlo a GitHub, debemos de crear un nuevo repositorio como ya hemos hecho anteriormente, y copiar el enlace para utilizarlo con el `git remote add origin`.

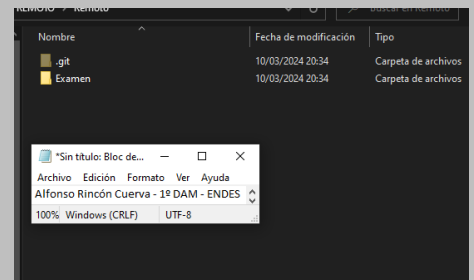
Después de eso, añadimos los cambios con `git add` y realizamos un commit con un comentario descriptivo. Una vez hecho el commit, lo subimos al repositorio con `git push origin master`. Lo subimos a la rama master, porque si al crear el repositorio y no creas ningún archivo, no se crea ninguna rama, pero en el momento que ya creas un archivo directamente con GitHub se creará la rama main. Entonces, al no haber ninguna rama creada anteriormente se podrá subir los archivos a la rama master, sino tendríamos que cambiarnos a la rama main si quisiéramos que todos los archivos esten en la misma rama.

```
PS C:\Users\josec\Desktop\Remoto> git init
Initialized empty Git repository in C:/Users/josec/Desktop/Remoto/.git/
PS C:\Users\josec\Desktop\Remoto> git config user.name "José Corrochano Pardo"
PS C:\Users\josec\Desktop\Remoto> git config user.email "jose.corrochanopardo@riberadeltajo.es"
PS C:\Users\josec\Desktop\Remoto> git remote add origin https://github.com/Josecp03/Remoto.git
PS C:\Users\josec\Desktop\Remoto> git add .
PS C:\Users\josec\Desktop\Remoto> git commit -m "Proyecto subido"
[master (root-commit) 5b4c788] Proyecto subido
12 files changed, 295 insertions(+)
create mode 100644 Examen/Ejercicios XML/Ejercicio1.xml
create mode 100644 Examen/Ejercicios XML/Ejercicio2.xml
create mode 100644 Examen/Examen/estilos/estilos.css
create mode 100644 Examen/Examen/imagenes/imagen1.jpg
create mode 100644 Examen/Examen/imagenes/imagen2.jpg
create mode 100644 Examen/Examen/imagenes/imagen3.jpg
create mode 100644 Examen/Examen/imagenes/imagen4.jpg
create mode 100644 Examen/Examen/index.html
create mode 100644 Examen/Examen/paginas/apartado2.html
create mode 100644 Examen/Examen/paginas/apartado3.html
create mode 100644 Examen/Examen/paginas/apartado4.html
create mode 100644 Examen/Examen/paginas/formularioEnviado.html
PS C:\Users\josec\Desktop\Remoto> git push origin master
```

José Corrochano

Ahora, este repositorio, si está público, el compañero puede clonarlo en su ordenador y tener lo que está subido en la nube.

```
C:\Windows\System32\cmd.exe
C:\Users\PROGRAMACION\Desktop\REMOTO>git config user.name "Alfonso Rincón Cuerva"
C:\Users\PROGRAMACION\Desktop\REMOTO>git config user.email "alfonso.rinconcuerva@riberadeltajo.es"
C:\Users\PROGRAMACION\Desktop\REMOTO>git config user.email alfonso.rinconcuerva@riberadeltajo.es
C:\Users\PROGRAMACION\Desktop\REMOTO>git clone https://github.com/Josecp03/Remoto.git
Cloning into 'Remoto'...
remote: Enumerating objects: 20, done.
remote: Counting objects: 100% (20/20), done.
remote: Compressing objects: 100% (18/18), done.
remote: Total 20 (delta 0), reused 20 (delta 0), pack-reused 0
Receiving objects: 100% (20/20), 2.19 MiB | 10.47 MiB/s, done.
C:\Users\PROGRAMACION\Desktop\REMOTO>git status
On branch master
No commits yet
Untracked files:
  (use "git add <file>..." to include in what will be committed)
  Remoto/
nothing added to commit but untracked files present (use "git add" to track)
C:\Users\PROGRAMACION\Desktop\REMOTO>git add .
```

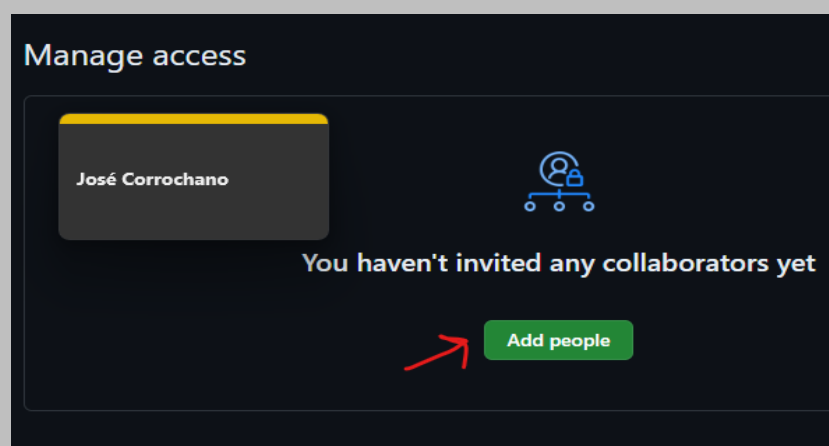


Pero si quiere realizar algún cambio en el proyecto y empezar a realizar commits, el usuario que ha creado el repositorio debe de darle permisos primero, para que pueda realizar estas operaciones.

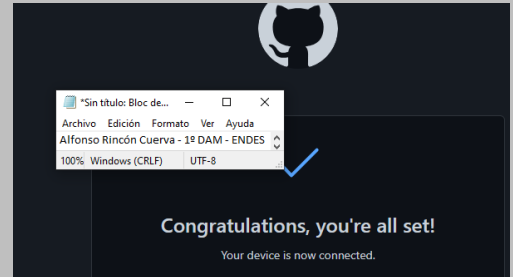
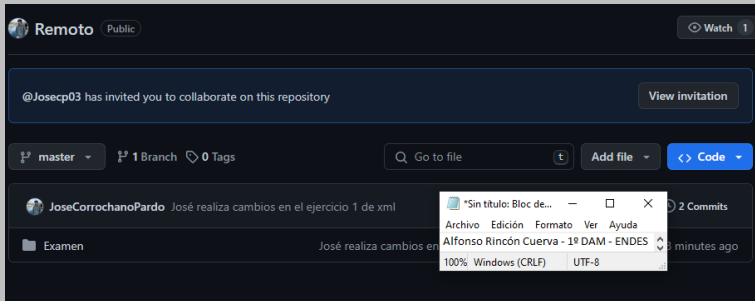
```
on: git remote prune [-n | --dry-run] <name>
on: git remote [-v | --verbose] update [-p | --prune] [(<group> | <remote>)...]
on: git remote set-branches [---add]
on: git remote get-url [---push] [---a
on: git remote set-url [---push] <nam
on: git remote set-url --add <name>
on: git remote set-url --delete <nam
-v, --[no-]verbose be verbose; must be placed before a subcommand

C:\Users\PROGRAMACION\Desktop\REMOTO\Remoto\Examen>git push origin master
remote: Permission to Josecp03/Remoto.git denied to alfonso.
fatal: unable to access 'https://github.com/Josecp03/Remoto.git/': The requested URL returned error: 403
```

Para añadir estos permisos, si te vas al repositorio en concreto en el que quieres aplicar estos cambios, en ajustes, en el apartado de colaboradores, ahí tienes que poner el nombre del usuario al que quieres darle permisos para realizar cambios en tu proyecto.



El compañero visualizaría que el usuario le concede los permisos de la siguiente manera:



En este momento, tendríamos los dos el mismo repositorio en cada uno de nuestros ordenadores. Si ahora queremos hacer cambios sobre este proyecto pero en archivos distintos se harían sin problemas siempre y cuando no afecten los cambios. Al realizarse los cambios, cada uno añade ese cambio desde el repositorio local con git add . Después realiza el commit correspondiente con el comentario descriptivo del cambio, y lo subiremos ambos a la rama master:



Llegado a este momento, partiendo del mismo commit los dos, el último de todos que está subido al repositorio remoto, vamos a modificar el mismo archivo. Primero, el usuario principal del repositorio realiza un cambio con un comentario y lo sube, y no da ningún problema. Pero, a la hora de que el otro usuario al que hemos invitado previamente, y que sube los cambios después del primer cambio subido, al haber una nueva modificación en el código, le da un error:

```
C:\Users\PROGRAMACION\Desktop\REMOTO\Remoto\Examen>git add .
C:\Users\PROGRAMACION\Desktop\REMOTO\Remoto\Examen>git status
On branch master
Your branch is up to date with 'origin/master'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   Examen/index.html

C:\Users\PROGRAMACION\Desktop\REMOTO\Remoto\Examen>git
[master 4dbf256] Comentario Alfonso
1 file changed, 1 insertion(+)
C:\Users\PROGRAMACION\Desktop\REMOTO\Remoto\Examen>git
To https://github.com/Josecp03/Remoto.git
! [rejected]        master -> master (fetch first)
error: failed to push some refs to 'https://github.com/Josecp03/Remoto.git'
hint: Updates were rejected because the remote contains work that you do not
hint: have locally. This is usually caused by another repository pushing to
hint: the same ref. If you want to integrate the remote changes, use
hint: 'git pull' before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
```

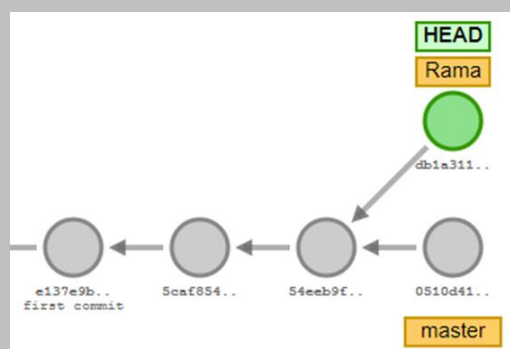
Para solucionar este error, lo que debe de hacer este otro usuario es ir haciendo los cambios en una rama independiente para que no entre en conflicto con los otros cambios que está realizando el usuario principal en la rama master. Para crear una rama ejecutaremos el comando `git checkout -b Nombre_de_la_rama`, o directamente, al subir los cambios, en vez de subirlo a la master, indicamos el nombre de la nueva rama y se crearía automáticamente:

```
C:\Users\PROGRAMACION\Desktop\REMOTO\Remoto\Examen>git push origin rama
Enumerating objects: 35, done.
Counting objects: 100% (35/35), done.
Delta compression using up to 4 threads
Compressing objects: 100% (28/28), done.
Writing objects: 100% (35/35), 2.19 MiB | 1.67 MiB/s,
Total 35 (delta 6), reused 23 (delta 2), pack-reused 0
remote: Resolving deltas: 100% (6/6), done.
remote:
remote: Create a pull request for 'rama' on GitHub by
remote:   https://github.com/Josecp03/Remoto/pull/new/rama
remote:
To https://github.com/Josecp03/Remoto.git
 * [new branch]      rama -> rama

C:\Users\PROGRAMACION\Desktop\REMOTO\Remoto\Examen>git status
On branch rama
nothing to commit, working tree clean
```

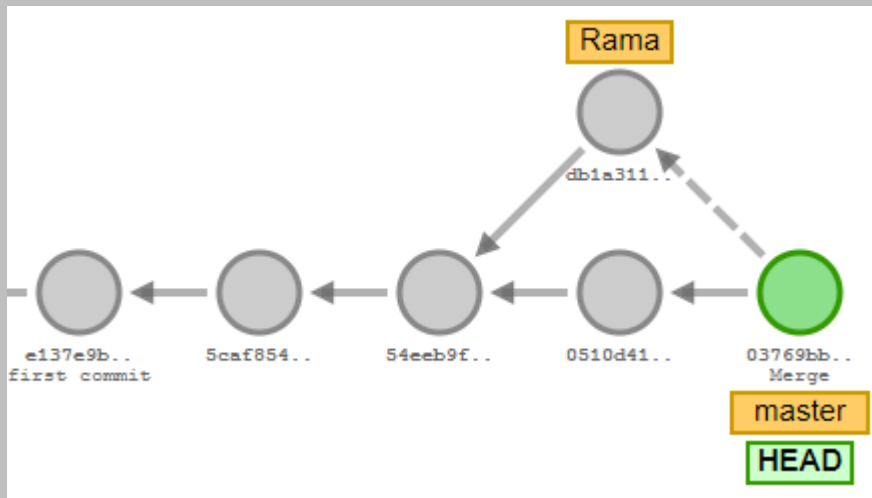
Para visualizar esto mejor nos ayudaremos de esta herramienta que nos ayuda a ver mejor las distintas ramas que se crean y los commits que se van realizando.

<https://git-school.github.io/visualizing-git/>



Aquí podemos ver como se ha creado una rama independiente a la master y hemos realizado un commit en esta. El HEAD representa donde estamos apuntando en ese momento.

Para que tengamos todas las nuevas modificaciones en una única rama, en vez de tener dos ramas, deberemos de juntarlas para que se vea de la siguiente manera:

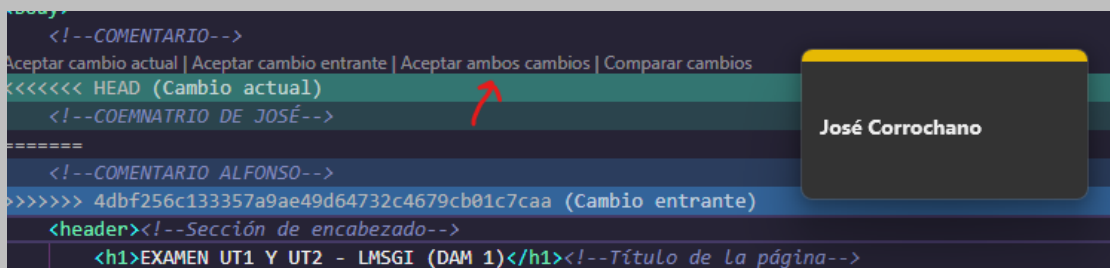


Una vez el segundo usuario ha juntado todo en la rama master, el usuario principal trae esos cambios a su repositorio local con el comando *git pull origin master*

```
PS C:\Users\josec\Desktop\Remoto> git pull origin master
remote: Enumerating objects: 9, done.
remote: Counting objects: 100% (9/9), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 5 (delta 2), reused 5 (delta 2), pack-reused 0
Unpacking objects: 100% (5/5), 432 bytes | 25.00 KiB/s, done.
From https://github.com/Josecp03/Remoto
* branch      master      -> FETCH_HEAD
2855470..f016c0c master    -> origin/master
Updating 2855470..f016c0c
Fast-forward
 Examen/Examen/index.html | 1 +
 1 file changed, 1 insertion(+)
PS C:\Users\josec\Desktop\Remoto>
```

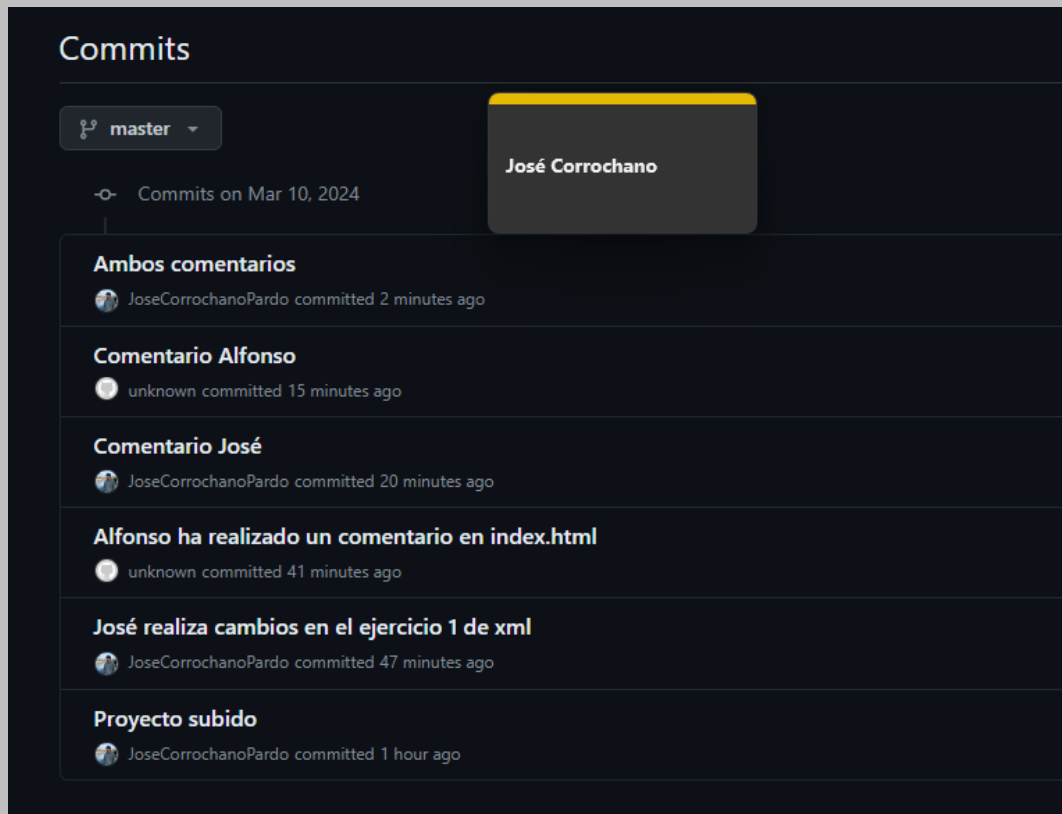
José Corrochano

Al realizar esto, los cambios, al partir de la misma versión, crearán conflicto, por lo que, desde el editor de código, por ejemplo el Visual Studio Code, al abrir nuestro código, deberemos de decidir si aplicar un cambio u otro, o bien, aceptar ambos cambios. En este caso vamos a aceptar ambos cambios pues son compatibles.



José Corrochano

Después de decidir la versión de nuestro proyecto, subimos los cambios a la rama master realizando su respectivo commit, viéndose finalmente los commits de la rama master de la siguiente manera:



### Commits

master

Commits on Mar 10, 2024

**José Corrochano**

- Ambos comentarios**  
JoseCorrochanoPardo committed 2 minutes ago
- Comentario Alfonso**  
unknown committed 15 minutes ago
- Comentario José**  
JoseCorrochanoPardo committed 20 minutes ago
- Alfonso ha realizado un comentario en index.html**  
unknown committed 41 minutes ago
- José realiza cambios en el ejercicio 1 de xml**  
JoseCorrochanoPardo committed 47 minutes ago
- Proyecto subido**  
JoseCorrochanoPardo committed 1 hour ago

## BIBLIOGRAFÍA

### REPOSITORIO

<https://docs.github.com/es/repositories/creating-and-managing-repositories/about-repositories>

### COMANDOS

<https://git-scm.com/book/es/v2/Fundamentos-de-Git-Guardando-cambios-en-el-Repositorio>

### ¿QUÉ ES GIT Y GITHUB?

<https://kinsta.com/es/base-de-conocimiento/git-vs-github/>

### SISTEMAS DE CONTROL DE VERSIONES

<https://www.drauta.com/5-softwares-de-control-de-versiones>

### SISTEMAS CENTRALIZADOS Y DISTRIBUIDOS

<https://chaui201521701115540.wordpress.com/2015/11/09/sistema-centralizado-vs-sistema-distribuido/>

## REPOSITORIOS

### ALFONSO

<https://github.com/alfonsaco>

### JOSÉ

<https://github.com/Josecp03>