

# GIT y GitHub

*Martín Gil Blanco*



<b>Git y GitHub</b>	<b>3</b>
<b>Comandos básicos</b>	<b>3</b>
<b>Ramas</b>	<b>8</b>
<b>Manejo de ramas y solución de conflictos</b>	<b>11</b>
<b>GitHub</b>	<b>22</b>
<b>Manejo de la seguridad de nuestro repositorio</b>	<b>32</b>
<b>Uso de Tags y Alias</b>	<b>40</b>
<b>Ramas remotas en GitHub</b>	<b>44</b>
<b>README y .gitignore</b>	<b>53</b>
<b>MarkDown</b>	<b>54</b>
<b>Git Rebase</b>	<b>58</b>
<b>Git Stash</b>	<b>61</b>
<b>Git Amend</b>	<b>63</b>
<b>Git Reset</b>	<b>65</b>
<b>Git Grep</b>	<b>66</b>
<b>Git Squash</b>	<b>66</b>
<b>Issues y Milestones</b>	<b>68</b>
<b>Wiki</b>	<b>71</b>
<b>Configura tu perfil de Git</b>	<b>72</b>

# Git y GitHub

**GIT** es un sistema de control de versiones distribuido de código abierto diseñado para gestionar proyectos de software de manera eficiente. Fue creado por Linus Torvalds en 2005 y se ha convertido en una herramienta fundamental para desarrolladores y equipos de programación. La función principal de GIT es rastrear cambios en el código fuente durante el desarrollo de software, lo que permite a los equipos trabajar de manera colaborativa y mantener un historial detallado de todas las modificaciones realizadas en el código.

Las características clave de GIT incluyen la capacidad de ramificar y fusionar fácilmente, lo que facilita el desarrollo paralelo de funciones o correcciones de errores sin afectar al código principal. También ofrece un rendimiento rápido y eficiente, incluso en proyectos grandes.

**GitHub** es una plataforma web que utiliza el sistema de control de versiones GIT para facilitar la colaboración en proyectos de desarrollo de software. Fue lanzado en 2008 y se ha convertido en uno de los repositorios de código más grandes y populares del mundo. GitHub proporciona servicios de alojamiento para proyectos de GIT, así como herramientas adicionales para la gestión de proyectos y la colaboración en equipo.

## Comandos básicos

**git init**: Este comando se utiliza para iniciar un nuevo repositorio de Git en un directorio existente. Cuando ejecutas `git init`, se crea un subdirectorio llamado `.git` que contiene todos los archivos necesarios para el repositorio. Este directorio es esencial porque almacena la información sobre el historial de cambios, las configuraciones y otros datos relacionados con el repositorio.

**git status**: Al usar `git status`, obtienes una descripción del estado actual de tu repositorio. Te mostrará los cambios realizados en tu directorio de trabajo en comparación con la última versión confirmada en el repositorio. Puedes ver qué archivos están sin seguimiento, qué archivos han sido modificados y qué archivos están listos para ser confirmados.

**git add**: Este comando se utiliza para agregar cambios al área de preparación (también conocida como "*staging area*"). Antes de realizar un commit, necesitas agregar los archivos modificados al área de preparación. Puedes hacer esto para archivos específicos (`git add nombre_archivo`) o para todos los archivos modificados (`git add .`).

**git commit**: Una vez que has agregado los archivos al área de preparación, ejecutas `git commit` para confirmar esos cambios. Un commit en Git representa un conjunto de cambios que forman una unidad lógica. Cada commit tiene un mensaje asociado que proporciona una descripción breve de los cambios realizados en ese commit. Por ejemplo:

```
git commit -m "Agrega nuevas funcionalidades al sistema"
```

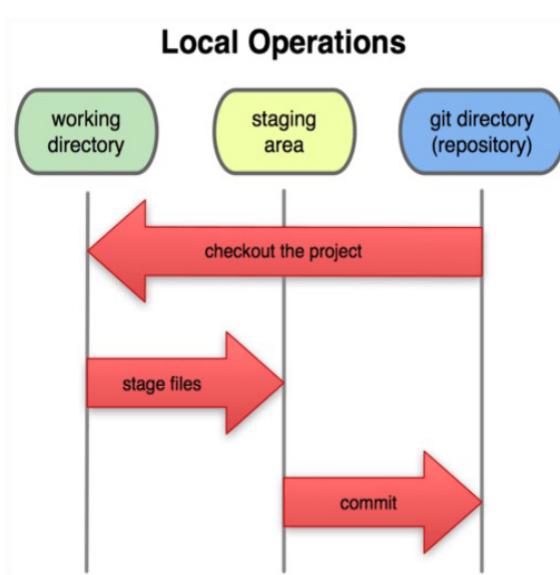
Los cambios en el área de preparación y los guarda en el historial de versiones de Git.

A modo de resumen y en especial es importante conocer el estado de los archivos en un repositorio git. Un archivo puede estar en uno de los siguientes estados.

**Modificado:** El archivo ha sido modificado en el directorio de trabajo del repositorio.

**Preparado:** Tras ser modificado se ha marcado como preparado y se ha enviado al área de preparación (staging area). Todo archivo modificado cuyos cambios van a ser enviados de forma permanente al repositorio deben de ser marcados como preparados (staged).

**Confirmado:** el archivo tras haber sido modificado y marcado como preparado se envía, mediante una operación commit, al directorio git (git directory) y pasa a formar parte del histórico de cambios del repositorio git.



Si nunca hemos usado git, es posible que tengamos que indicar nuestro usuario y correo electrónico para dejar constancia de quien realiza las confirmaciones de código. Para ello empleamos los siguientes comandos:

```
git config --global user.name "Tu Nombre"
git config --global user.email TuEmail@Email.com
```

A continuación te muestro un ejemplo básico de lo que hemos visto por comandos:

```
prueba.txt
1 Hola a todos!

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

● PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git init
Initialized empty Git repository in C:/Users/gilbl/OneDrive/Escritorio/IES SAN CLEMENTE/Ordinario/SI/Curso Git y GitHub/.git/
● PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git status
On branch master

No commits yet

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

nothing added to commit but untracked files present (use "git add" to track)
● PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git add .\prueba.txt
● PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git status
On branch master

No commits yet

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

● PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git commit -m "Primer commit"
[master (root-commit) 4ad77c5] Primer commit
 1 file changed, 1 insertion(+)
 create mode 100644 prueba.txt
● PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git log
commit 4ad77c57b60c0a18e08d02edca6bc3c3bf628f0d (HEAD -> master)
Author: MARTÍN GIL BLANCO <gilblancomartin@gmail.com>
Date:   Tue Nov 14 14:02:13 2023 +0100

Primer commit
○ PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub>
```

Imaginemos que queremos añadir otra línea de código a nuestro archivo. En ese caso el proceso sería el siguiente:



```
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
    modified:   prueba.txt

no changes added to commit (use "git add" and/or "git commit -a")
● PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git add .\prueba.txt
● PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    modified:   prueba.txt
```

Pudiera darse el caso de que quisiéramos volver al punto anterior y eliminar la línea dos de nuestro seguimiento si nos hubiéramos equivocado. Para ello empleamos el comando:

`git restore --stage <file>`



```
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
    modified:   prueba.txt

no changes added to commit (use "git add" and/or "git commit -a")
● PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git add .\prueba.txt
● PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    modified:   prueba.txt
```

Es decir, deshace los cambios que hayas realizado en el archivo especificado y lo quita del área de preparación, lo que significa que esos cambios no se incluirán en tu próxima confirmación a menos que los vuelvas a agregar al área de preparación.

A continuación vamos a añadir de nuevo el cambio empleando el comando `git add` y hacemos un segundo commit. Vamos a ver con `git log` el historial de commits.

```
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git commit -m "Segundo commit"
[master ba110cf] Segundo commit
 1 file changed, 2 insertions(+), 1 deletion(-)
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git log
commit ba110cf8bc75eb0b0586994833bedcebf9de34522 (HEAD -> master)
Author: MARTÍN GIL BLANCO <gilblancmartin@gmail.com>
Date:   Tue Nov 14 14:19:41 2023 +0100

        Segundo commit

commit 4ad77c57b60c0a18e08d02edca6bc3c3bf628f0d
Author: MARTÍN GIL BLANCO <gilblancmartin@gmail.com>
Date:   Tue Nov 14 14:02:13 2023 +0100

        Primer commit
```

El comando `git log` muestra un historial de confirmaciones (commits) en un repositorio de Git. Muestra información detallada acerca de cada confirmación, lo que puede ser útil para rastrear la evolución del proyecto. Aquí hay algunos de los detalles más importantes que puedes encontrar al usar `git log`:

1. **Hash del commit:** Un identificador único para cada confirmación. Se muestra como una cadena hexadecimal. Este hash es esencial para referenciar commits en Git.
2. **Autor y fecha:** Muestra quién hizo la confirmación y cuándo lo hizo. Esto incluye el nombre del autor y la dirección de correo electrónico, así como la fecha y la hora.
3. **Mensaje del commit:** Un mensaje descriptivo que proporciona información sobre los cambios realizados en esa confirmación. Un buen mensaje de confirmación debería ser conciso pero informativo, describiendo los cambios realizados.
4. **Ramas y etiquetas:** Si estás trabajando con ramas o etiquetas, `git log` mostrará a qué rama o etiqueta pertenece cada confirmación. Esto puede ser útil para entender la estructura del desarrollo en el repositorio.
5. **Formato del árbol:** Muestra el árbol de confirmaciones que representa la estructura de la historia del proyecto.

Actualmente el archivo que estamos empleando tiene el siguiente contenido.

Hola a todos! → Primer commit

Adios a todos! → Segundo commit

Pudiera darse el caso de que quisiéramos volver al punto del primer commit, esto podemos hacerlo gracias al comando `git checkout HASH_Commit`, tal y como podemos ver a continuación. En este punto el contenido debería ser únicamente:

Hola a todos! → Primer commit

```
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git checkout 4ad77c57b60c0a18e08d02edca6bc3c3bf628f0d
Note: switching to '4ad77c57b60c0a18e08d02edca6bc3c3bf628f0d'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by switching back to a branch.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -c with the switch command. Example:

  git switch -c <new-branch-name>

Or undo this operation with:

  git switch ->

Turn off this advice by setting config variable advice.detachedHead to false

HEAD is now at 4ad77c5 Primer commit
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git log
commit 4ad77c57b60c0a18e08d02edca6bc3c3bf628f0d (HEAD)
Author: MARTÍN GIL BLANCO <gilblancmartin@gmail.com>
Date:   Tue Nov 14 14:02:13 2023 +0100

  Primer commit
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git checkout master
Previous HEAD position was 4ad77c5 Primer commit
Switched to branch 'master'
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git log
commit ba110cf8c75eb0b0586994833bedceb9de34522 (HEAD -> master)
Author: MARTÍN GIL BLANCO <gilblancmartin@gmail.com>
Date:   Tue Nov 14 14:19:41 2023 +0100

  Segundo commit
commit 4ad77c57b60c0a18e08d02edca6bc3c3bf628f0d
Author: MARTÍN GIL BLANCO <gilblancmartin@gmail.com>
Date:   Tue Nov 14 14:02:13 2023 +0100

  Primer commit
```

Si nos fijamos, en este momento el nombre de la rama se ha modificado a raíz de cambiar a otro commit, estamos en este momento en una rama temporal en vez de en la rama master. Esto es un proceso temporal hasta que volvamos a cambiar a la rama master.

Si quisieramos ver la configuración inicial de nuestro git podríamos emplear el comando `git config --list`.

```
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git config --list
diff.astextplain.textconv=astextplain
filter.lfs.clean=git-lfs clean -- %f
filter.lfs.smudge=git-lfs smudge -- %f
filter.lfs.process=git-lfs filter-process
filter.lfs.required=true
http.sslbackend=openssl
http.sslcainfo=C:/Program Files/Git/mingw64/ssl/certs/ca-bundle.crt
core.autocrlf=true
core.fscache=true
core.symlinks=false
pull.rebase=false
credential.helper=manager-core
credential.https://dev.azure.com.usehttppath=true
init.defaultbranch=master
user.name=MARTÍN GIL BLANCO
user.email=gilblancomartin@gmail.com
safe.directory=C:/Users/gilbl
filter.lfs.clean=git-lfs clean -- %f
filter.lfs.smudge=git-lfs smudge -- %f
filter.lfs.process=git-lfs filter-process
filter.lfs.required=true
core.repositoryformatversion=0
core.filemode=false
core.bare=false
core.logallrefupdates=true
core.symlinks=false
core.ignorecase=true
```

Además podemos ver en qué parte de nuestro PC se guarda esta información. Para ello hacemos uso del comando `git config --list --show-origin`. En nuestro caso, como se puede ver en la siguiente imagen podemos apreciar que se almacenan en la ruta C:/Program Files/Git/etc/gitconfig.

```
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git config --list --show-origin
file:C:/Program Files/Git/etc/gitconfig diff.astextplain.textconv=astextplain
file:C:/Program Files/Git/etc/gitconfig filter.lfs.clean=git-lfs clean -- %f
file:C:/Program Files/Git/etc/gitconfig filter.lfs.smudge=git-lfs smudge -- %f
file:C:/Program Files/Git/etc/gitconfig filter.lfs.process=git-lfs filter-process
file:C:/Program Files/Git/etc/gitconfig filter.lfs.required=true
file:C:/Program Files/Git/etc/gitconfig http.sslbackend=openssl
file:C:/Program Files/Git/etc/gitconfig http.sslcainfo=C:/Program Files/Git/mingw64/ssl/certs/ca-bundle.crt
file:C:/Program Files/Git/etc/gitconfig core.autocrlf=true
file:C:/Program Files/Git/etc/gitconfig core.fscache=true
file:C:/Program Files/Git/etc/gitconfig core.symlinks=false
file:C:/Program Files/Git/etc/gitconfig pull.rebase=false
file:C:/Program Files/Git/etc/gitconfig credential.helper=manager-core
file:C:/Program Files/Git/etc/gitconfig credential.https://dev.azure.com.usehttppath=true
file:C:/Program Files/Git/etc/gitconfig init.defaultbranch=master
file:C:/Users/gilbl/.gitconfig user.name=MARTÍN GIL BLANCO
file:C:/Users/gilbl/.gitconfig user.email=gilblancomartin@gmail.com
file:C:/Users/gilbl/.gitconfig safe.directory=C:/Users/gilbl
file:C:/Users/gilbl/.gitconfig filter.lfs.clean=git-lfs clean -- %f
file:C:/Users/gilbl/.gitconfig filter.lfs.smudge=git-lfs smudge -- %f
file:C:/Users/gilbl/.gitconfig filter.lfs.process=git-lfs filter-process
file:C:/Users/gilbl/.gitconfig filter.lfs.required=true
file:.git/config core.repositoryformatversion=0
file:.git/config core.filemode=false
file:.git/config core.bare=false
file:.git/config core.logallrefupdates=true
file:.git/config core.symlinks=false
file:.git/config core.ignorecase=true
```

Por otra parte, podemos ver también los últimos cambios con respecto al commit anterior y el actual. Por este motivo existe el comando `git show` que nos está indicando que la última línea fue Adiós a todos! y después de ella se añadió la línea: Línea para el comando git show.

```
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git show
commit f5f8a24e1831e92c8a2b0ef131cbae330bd6250e (HEAD -> master)
Author: MARTÍN GIL BLANCO <gilblancomartin@gmail.com>
Date:   Tue Nov 14 17:14:27 2023 +0100

    Tercer commit para ver comando git show

diff --git a/prueba.txt b/prueba.txt
index 1df5914..82953b5 100644
--- a/prueba.txt
+++ b/prueba.txt
@@ -1,2 +1,3 @@
 Hola a todos!
-Adios a todos!
\ No newline at end of file
+Adios a todos!
+Línea para el comando git show
\ No newline at end of file
```

También podemos emplear el comando `git diff HASH_COMMIT_ANTERIOR HASH_COMMIT_ACTUAL` que nos permite ver diferencias de un commit con respecto a otro.

## Ramas

En Git, el término "rama" se refiere a una línea de desarrollo independiente. Cada rama en Git es como una línea separada de cambios en tu proyecto. Puedes tener varias ramas en un repositorio, y cada una puede representar una característica, una corrección de error, una experimentación, etc. Las ramas permiten trabajar en diferentes aspectos del proyecto sin afectar directamente el código en otras ramas.

Aquí hay algunos conceptos clave relacionados con las ramas en Git:

**Rama Maestra (Master):** La rama principal en Git se llama comúnmente "master". Esta rama es la línea principal de desarrollo y generalmente refleja la versión estable del proyecto.

### Crear una Rama:

- Puedes crear una nueva rama para trabajar en una característica específica o solucionar un problema. El comando para crear una nueva rama es:  
`git branch nombre-de-la-rama`
- Para cambiar a la nueva rama, puedes usar:  
`git checkout nombre-de-la-rama`
- O, en versiones más recientes de Git  
`git switch nombre-de-la-rama`
- También puedes crear y cambiar de rama en un único comando con:  
`git checkout -b nombre-de-la-rama`

**Hacer Cambios en una Rama:**

Después de cambiar a una rama, puedes realizar cambios en los archivos de tu proyecto.

**Fusionar Ramas:**

Una vez que has completado el trabajo en una rama y estás satisfecho con los cambios, puedes fusionar esa rama de vuelta a la rama principal (por ejemplo, `master`). El comando para hacer esto es:

```
git merge nombre-de-la-rama
```

Esto combinará los cambios de la rama especificada en la rama actual.

**Eliminar una Rama:**

Después de fusionar los cambios de una rama y asegurarte de que no necesitas la rama para futuros desarrollos, puedes eliminarla. El comando para eliminar una rama es:

```
git branch -d nombre-de-la-rama
```

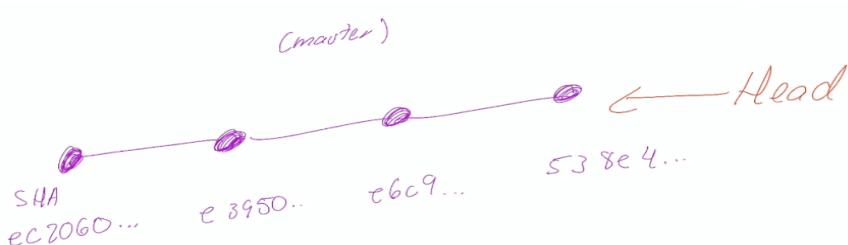
**Visualización de Ramas:**

Puedes ver todas las ramas en tu repositorio y ver en cuál estás actualmente usando:

```
git branch
```

**Ramificación Remota:**

Además de las ramas locales, Git también permite la ramificación remota, lo que significa que puedes trabajar con ramas en repositorios remotos. Hasta este punto nosotros tenemos una única rama `master` con diferentes commits realizados. El escenario es por tanto similar al siguiente:



Existe además en este sentido un comando de interés que es `git reset`. Este comando tiene dos opciones.

- `git reset HASH_COMMIT --soft`: Vuelve al punto de otro commit desde el actual pero conservando el espacio de staging.
- `git reset HASH_COMMIT --hard`: Vuelve desde el commit actual al estado de otro commit diferente.

Podemos hacer uso de la opción hard en momentos donde una modificación de código hace que nuestro desarrollo deje de ser estable. Un ejemplo de esto lo podemos ver a continuación con el siguiente código escrito en prueba.txt:

```
Hola a todos! → Primer commit
Adios a todos! → Segundo commit
Comando git show → Tercer commit
```

```
1 Hola a todos!
2 Adios a todos!
```

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
TERMINAL

PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git log
commit f5f8a24e1831e92c8a2b00ef131cbae330bd6250e (HEAD -> master)
Author: MARTÍN GIL BLANCO <gilblancomartin@gmail.com>
Date:   Tue Nov 14 17:14:27 2023 +0100

Tercer commit para ver comando git show

commit ba110cf8bc75eb0b0586994833bedceb9de34522
Author: MARTÍN GIL BLANCO <gilblancomartin@gmail.com>
Date:   Tue Nov 14 14:19:41 2023 +0100

Segundo commit

commit 4ad77c57b60c0a18e08d02edca6bc3c3bf628f0d
Author: MARTÍN GIL BLANCO <gilblancomartin@gmail.com>
Date:   Tue Nov 14 14:02:13 2023 +0100

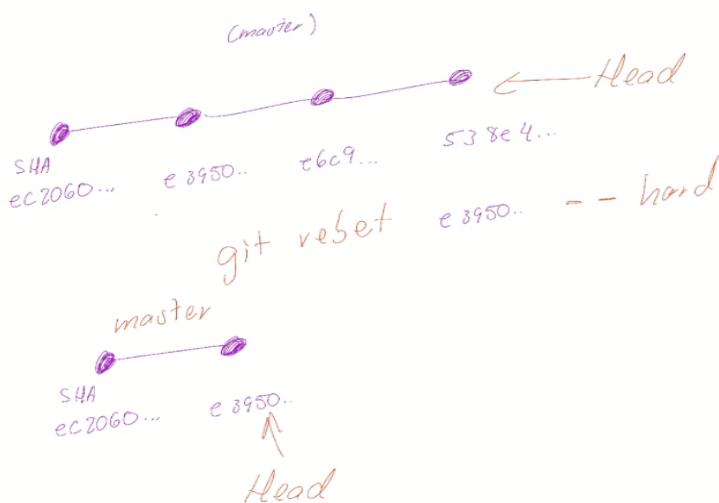
Primer commit
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git reset ba110cf8bc75eb0b0586994833bedceb9de34522 --hard
HEAD is now at ba110cf Segundo commit
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git log
commit ba110cf8bc75eb0b0586994833bedceb9de34522 (HEAD -> master)
Author: MARTÍN GIL BLANCO <gilblancomartin@gmail.com>
Date:   Tue Nov 14 14:19:41 2023 +0100

Segundo commit

commit 4ad77c57b60c0a18e08d02edca6bc3c3bf628f0d
Author: MARTÍN GIL BLANCO <gilblancomartin@gmail.com>
Date:   Tue Nov 14 14:02:13 2023 +0100

Primer commit
```

La realización de este código hace que se pierda el contenido del tercer commit y HEAD se coloca en el segundo commit por lo que el desarrollo que teníamos creado en el tercer commit desaparece así como se pierde el contenido que teníamos en staging. Un símil con el dibujo de antes es como si quedase de la siguiente manera la rama master:



# Manejo de ramas y solución de conflictos

Vamos a mostrar un flujo de funcionamiento real de desarrollo. Para este punto vamos a emplear un código JS junto a un HTML que se adjuntan a continuación:

HTML (index.html)	JavaScript (codigo.js)
<pre>&lt;!DOCTYPE html&gt; &lt;html lang="es-ES"&gt;  &lt;head&gt;     &lt;meta charset="UTF-8"&gt;     &lt;title&gt;Edad del usuario&lt;/title&gt;     &lt;script src="edad.js"&gt;&lt;/script&gt; &lt;/head&gt;  &lt;/html&gt;</pre>	<pre>let edad = prompt("Por favor, introduce a túa idade:"); edad = parseInt(edad);  if (isNaN(edad)) {     alert("Por favor, introduce un número válido para a idade."); } else {     let categoria;      switch (true) {         case edad &gt;= 0 &amp;&amp; edad &lt;= 12:             categoria = "neno";             break;         case edad &gt;= 13 &amp;&amp; edad &lt;= 18:             categoria = "adolescente";             break;         case edad &gt;= 19 &amp;&amp; edad &lt;= 30:             categoria = "xoven";             break;         case edad &gt;= 31 &amp;&amp; edad &lt;= 64:             categoria = "adulto";             break;         case edad &gt;= 65 &amp;&amp; edad &lt;= 100:             categoria = "xubilado";             break;         default:             alert("Idade fora do rango permitido.");     }      if (categoria) {         alert("Eres un " + categoria + ".");     } }</pre>

- Vamos a crear un nuevo proyecto con los archivos que se adjuntan anteriormente y ejecutamos: `git init`
- Hacemos el estado del área de staging haciendo uso del comando: `git status`
- Añadimos todos los nuevos archivos al área de staging con: `git add .`
- Creamos nuestro commit con: `git commit -m "Mi primer commit"`

```

index.html × JS codigo.js
> index.html > html
1  <!DOCTYPE html>
2  <html lang="es-ES">
3
4    <head>
5      <meta charset="UTF-8">
6      <title>Edad del usuario</title>
7      <script src="edad.js"></script>
8    </head>
9
10   </html>

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
TERMINAL

● PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git init
Initialized empty Git repository in C:/Users/gilbl/OneDrive/Escritorio/IES SAN CLEMENTE/Ordinario/SI/Curso Git y GitHub/.git/
● PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git status
On branch master

No commits yet

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

nothing added to commit but untracked files present (use "git add" to track)
● PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git add .
● PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
    new file:   codigo.js
    new file:   index.html

● PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git commit -m "Mi primer commit"
[master (root-commit) 39cb2f6] Mi primer commit
  2 files changed, 42 insertions(+)
  create mode 100644 codigo.js
  create mode 100644 index.html
● PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git log
commit 39cb2f6849f5707298a403113d00e5de1fe60194 (HEAD -> master)
Author: MARTÍN GIL BLANCO <gilblancomartin@gmail.com>
Date:   Tue Nov 14 18:40:18 2023 +0100

  Mi primer commit
○ PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub>

```

Una vez inicializado git y creado el esqueleto de nuestro primer proyecto lo suyo sería emplear una repositorio remoto de control de versiones como github o gitlab. En mi caso voy a hacer uso de github aunque el funcionamiento de ambas plataformas es bastante similar. Lo que nos va a permitir github es tener un repositorio remoto donde almacenar información en la nube y en el que puedan trabajar más de una persona. Un funcionamiento en diagrama podría ser el siguiente (en el siguiente diagrama todavía no se contempla el uso de ramas):

**git clone:** El comando `git clone` se utiliza para crear una copia local de un repositorio remoto. Al ejecutar este comando, se descarga la versión completa del repositorio, incluyendo todo su historial de versiones y ramas, en la máquina local del usuario. Este comando es comúnmente utilizado al iniciar un nuevo proyecto o al colaborar en un proyecto existente.

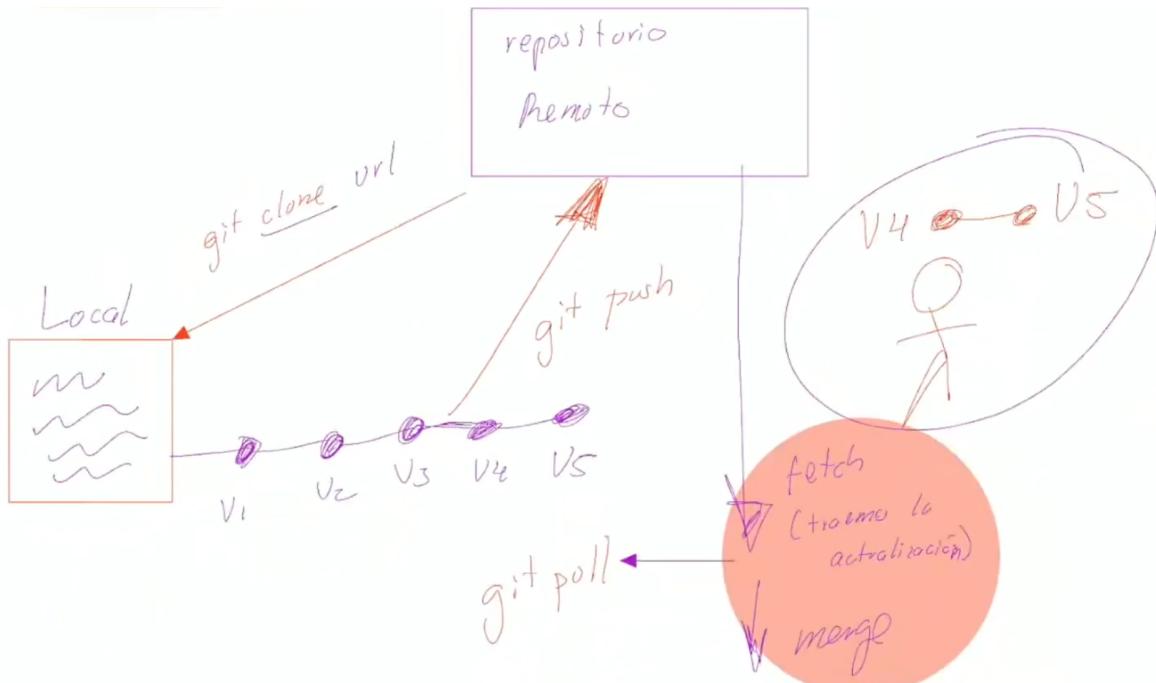
**git push:** El comando `git push` se emplea para enviar los cambios locales de una rama específica al repositorio remoto. Despues de realizar modificaciones en la rama local y

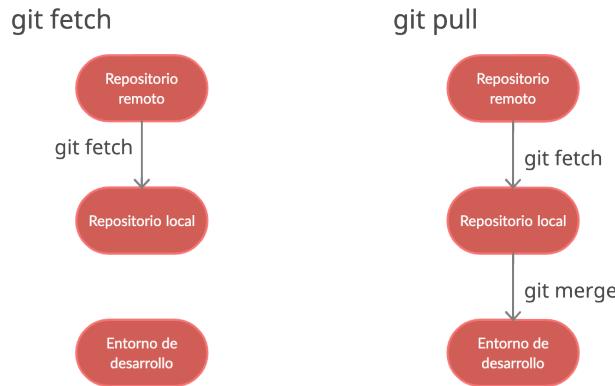
realizar un commit, `git push` se utiliza para actualizar el repositorio remoto con los cambios locales. Este comando requiere especificar el nombre del repositorio remoto y la rama que se está enviando

**git pull:** El comando `git pull` se utiliza para obtener los cambios más recientes desde el repositorio remoto y fusionarlos automáticamente con la rama local. Este comando combina dos acciones: primero, obtiene los cambios del repositorio remoto mediante `git fetch` y luego los fusiona con la rama local utilizando `git merge`. Es útil para mantener actualizada la rama local con los cambios realizados por otros colaboradores en el repositorio remoto.

**git fetch:** El comando `git fetch` se emplea para obtener información sobre los cambios más recientes en el repositorio remoto, sin realizar automáticamente la fusión con la rama local. A diferencia de `git pull`, este comando no modifica la rama local, pero actualiza los puntos de referencia (como las ramas remotas) para que se pueda decidir cuándo fusionar los cambios utilizando `git merge`. Es útil para revisar los cambios antes de combinarlos en la rama local.

**git merge:** El comando `git merge` se utiliza para combinar los cambios de una rama con otra. Por lo general, se utiliza después de ejecutar `git fetch` para obtener cambios desde el repositorio remoto. `git merge` toma los cambios recuperados y los fusiona en la rama local actual. Puede haber conflictos de fusión que deben resolverse manualmente en caso de cambios simultáneos en las mismas líneas de código. Este comando es fundamental para incorporar cambios en la rama local después de obtenerlos con `git fetch`.





Vamos a crear una nueva rama a partir de la master y vamos hacer modificaciones en la nueva rama sobre el código anterior eliminando el último case, esto nos va a permitir ver cuestiones interesantes del manejo de git. Para ello desde la rama master voy a realizar el siguiente comando:

```
git branch nuevaRama
git checkout nuevaRama
```

También es cierto que estos dos comandos podemos agruparlos en un único comando y hacer automáticamente la creación y cambio de rama. El comando en este caso sería:

```
git checkout -b nuevaRama
```

```
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git branch
* master
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git branch nuevaRama
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git branch
* master
  nuevaRama
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git checkout nuevaRama
Switched to branch 'nuevaRama'
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git branch
  master
* nuevaRama
```

Dentro de la nueva rama vamos a realizar la modificación de código por lo que estos cambios solo van a existir en la rama nuevaRama y no en master. El proceso siguiente nos muestra el proceso de commit de los cambios. Podemos apreciar que el primer commit se ha llevado a cabo en la rama master y a continuación el siguiente commit ya es propio de nuevaRama, en la cual estamos situados (esto se puede ver gracias al indicador HEAD).

```
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git status
On branch nuevaRama
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
    (use "git restore <file>..." to discard changes in working directory)
      modified:   codigo.js

no changes added to commit (use "git add" and/or "git commit -a")
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git commit -am "Eliminación del último case"
[nuevaRama f921996] Eliminación del último case
 1 file changed, 1 insertion(+), 4 deletions(-)
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git log
commit f9219961ff482414db38077afc574c134db471a (HEAD -> nuevaRama)
Author: MARTÍN GIL BLANCO <gilblancomartin@gmail.com>
Date:   Tue Nov 14 19:20:49 2023 +0100

  Eliminación del último case

commit 39cb2f6849f5707298a403113d00e5de1fe60194 (master)
Author: MARTÍN GIL BLANCO <gilblancomartin@gmail.com>
Date:   Tue Nov 14 18:40:18 2023 +0100

  Mi primer commit
```

A continuación, podemos ver estas diferencias haciendo uso de los comandos:

`git show` → Nos permite ver la diferencia entre el commit actual con respecto al anterior.

`git diff HASH_COMMIT_ANTERIOR HAS_COMMIT_VER_DIFERENCIA` → Nos permite ver la diferencia entre el commit que indicamos con el HASH y el commit sobre el que queremos ver la diferencia.

```
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git show
commit f9219961ff482414db38077afc574c134d1b471a (HEAD -> nuevaRama)
Author: MARTÍN GIL BLANCO <gilblancomartin@gmail.com>
Date: Tue Nov 14 19:20:49 2023 +0100

    Eliminación del último case

diff --git a/codigo.js b/codigo.js
index c15b847..df0f160 100644
--- a/codigo.js
+++ b/codigo.js
@@ -16,12 +16,9 @@ if (isNaN(edad)) {
    categoria = "xoven";
    break;
-   case edad >= 31 && edad <= 64:
+   case edad >= 31 && edad <= 100:
    categoria = "adulto";
    break;
-   case edad >= 65 && edad <= 100:
-      categoria = "xubilado";
-      break;
-   default:
-      alert("Idade fora do rango permitido.");
}
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git diff 39cb2f6849f5707298a403113d00e5de1fe60194 f9219961ff482414db38077afc574c134d1b471a
diff --git a/codigo.js b/codigo.js
index c15b847..df0f160 100644
--- a/codigo.js
+++ b/codigo.js
@@ -16,12 +16,9 @@ if (isNaN(edad)) {
    categoria = "xoven";
    break;
-   case edad >= 31 && edad <= 64:
+   case edad >= 31 && edad <= 100:
    categoria = "adulto";
    break;
-   case edad >= 65 && edad <= 100:
-      categoria = "xubilado";
-      break;
-   default:
-      alert("Idade fora do rango permitido.");
}
```

Fijémonos ahora en lo siguiente:

```
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git branch
  master
* nuevaRama
```

```
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git log
```

```
commit f9219961ff482414db38077afc574c134d1b471a (HEAD -> nuevaRama)
```

```
Author: MARTÍN GIL BLANCO <gilblancomartin@gmail.com>
```

```
Date: Tue Nov 14 19:20:49 2023 +0100
```

Eliminación del último case

```
commit 39cb2f6849f5707298a403113d00e5de1fe60194 (master)
```

```
Author: MARTÍN GIL BLANCO <gilblancomartin@gmail.com>
```

```
Date: Tue Nov 14 18:40:18 2023 +0100
```

Mi primer commit

```
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git checkout master
Switched to branch 'master'
```

```
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git log
```

```
commit 39cb2f6849f5707298a403113d00e5de1fe60194 (HEAD -> master)
```

```
Author: MARTÍN GIL BLANCO <gilblancomartin@gmail.com>
```

```
Date: Tue Nov 14 18:40:18 2023 +0100
```

Mi primer commit

```
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git log nuevaRama
```

```
commit f9219961ff482414db38077afc574c134d1b471a (nuevaRama)
```

```
Author: MARTÍN GIL BLANCO <gilblancomartin@gmail.com>
```

```
Date: Tue Nov 14 19:20:49 2023 +0100
```

Eliminación del último case

```
commit 39cb2f6849f5707298a403113d00e5de1fe60194 (HEAD -> master)
```

```
Author: MARTÍN GIL BLANCO <gilblancomartin@gmail.com>
```

```
Date: Tue Nov 14 18:40:18 2023 +0100
```

Mi primer commit

En la imagen anterior muestro el cambio entre las dos ramas desde las que puede ver el git log, en función de la rama en que me encuentre tendré un historial de commits u otro, de todos modos, puedes consultar desde una rama el historial de commits de otra rama, esto lo hago con el comando: `git log nombreRama`

También es interesante fijarnos en la palabra HEAD que nos indica sobre qué commit estamos, es decir, podemos ver que cuando cambiamos de rama siempre nos situamos con HEAD en el último commit, salvo en el caso del último git log que nos permite situarnos sobre el primer commit, el resultado es que veremos nuestro código desde la versión propia del primer commit, es decir, sin cambios.

También puedo desplazarme a un commit en concreto como vemos en la siguiente imagen.

```
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git checkout nuevaRama
Switched to branch 'nuevaRama'
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git log
commit f9219961ff482414db38077afc574c134d1b471a (HEAD -> nuevaRama)
Author: MARTÍN GIL BLANCO <gilblancmartin@gmail.com>
Date:   Tue Nov 14 19:20:49 2023 +0100

    Eliminación del último case

commit 39cb2f6849f5707298a403113d00e5de1fe60194 (master)
Author: MARTÍN GIL BLANCO <gilblancmartin@gmail.com>
Date:   Tue Nov 14 18:40:18 2023 +0100

    Mi primer commit
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git checkout 39cb2f6849
Note: switching to '39cb2f6849'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by switching back to a branch.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -c with the switch command. Example:

git switch -c <new-branch-name>

Or undo this operation with:

git switch -

Turn off this advice by setting config variable advice.detachedHead to false

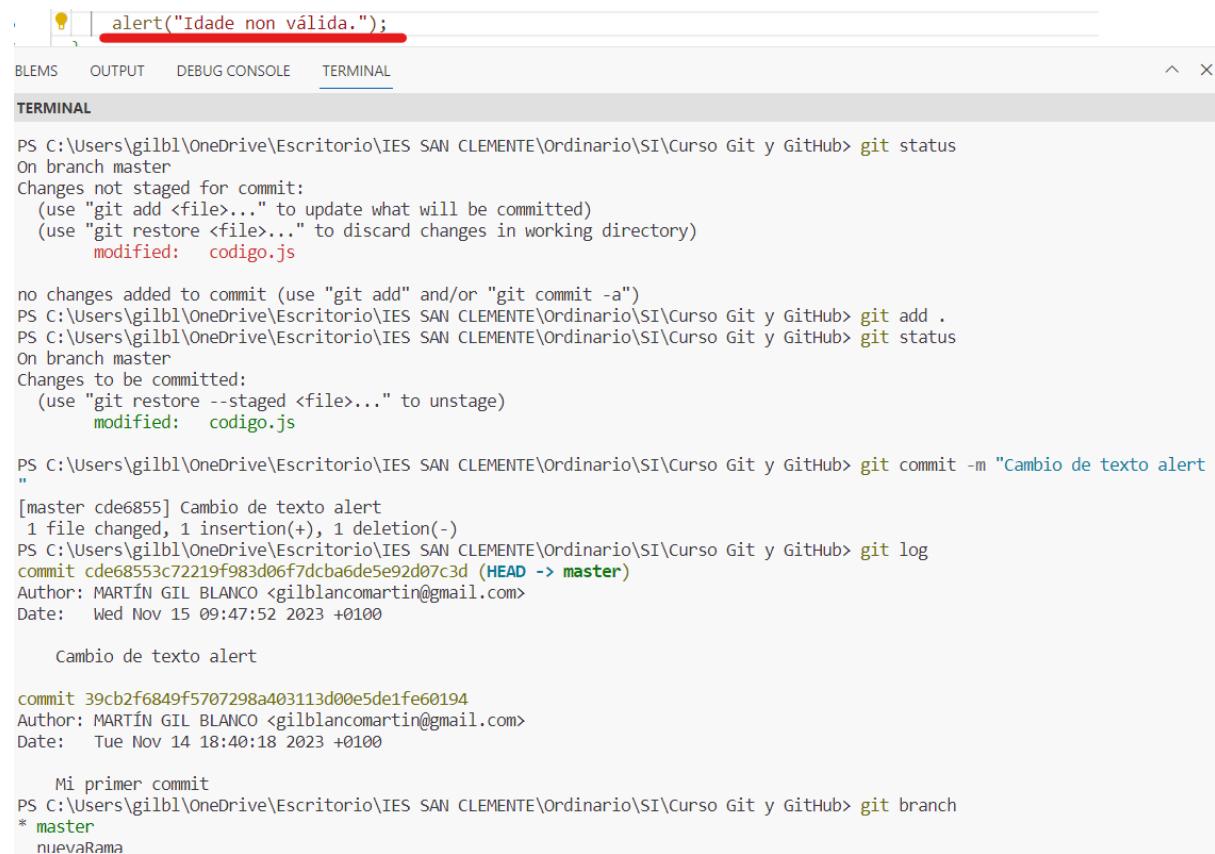
HEAD is now at 39cb2f6 Mi primer commit
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git log
commit 39cb2f6849f5707298a403113d00e5de1fe60194 (HEAD, master)
Author: MARTÍN GIL BLANCO <gilblancmartin@gmail.com>
Date:   Tue Nov 14 18:40:18 2023 +0100

    Mi primer commit
```

Una vez llegados hasta este punto si nos volvemos a situar en nuevaRama tenemos dos códigos, el de master donde solo hay un commit y el de nuevaRama con el último case eliminado. Lo suyo sería llevar a master estas modificaciones ya que esta es nuestra rama de producción donde tenemos que situar nuestras versiones estables de código. Dicho de forma clara, el código de master siempre ha de funcionar.

Aunque no sea lo correcto si estuviéramos desarrollando código, en este caso y por simplicidad del escenario vamos a hacer una modificación sobre la rama master y hacer a posteriori un merge de los cambios de nuevaRama (que no va a incluir esa modificación) con master.

Vamos a modificar el `alert("Idade fora do rango permitido.");` de nuestro código en master por el nuevo `alert("Idade non válida.");`



```

alert("Idade non válida.");

```

TERMINAL

```

PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
    modified:   codigo.js

no changes added to commit (use "git add" and/or "git commit -a")
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git add .
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    modified:   codigo.js

PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git commit -m "Cambio de texto alert"
[master cde6855] Cambio de texto alert
 1 file changed, 1 insertion(+), 1 deletion(-)
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git log
commit cde6855c72219f983d06f7dcba6de5e92d07c3d (HEAD -> master)
Author: MARTÍN GIL BLANCO <gilblancomartin@gmail.com>
Date:   Wed Nov 15 09:47:52 2023 +0100

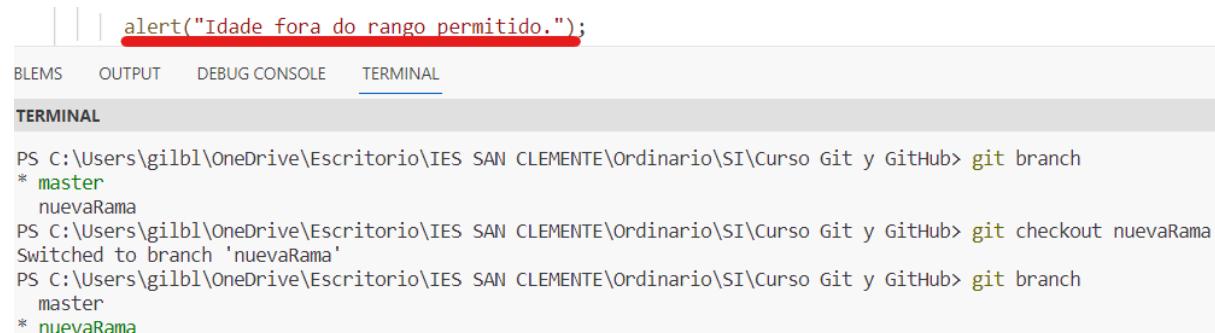
  Cambio de texto alert

commit 39cb2f6849f5707298a403113d00e5de1fe60194
Author: MARTÍN GIL BLANCO <gilblancomartin@gmail.com>
Date:   Tue Nov 14 18:40:18 2023 +0100

  Mi primer commit
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git branch
* master
  nuevaRama

```

Si yo ahora hago un cambio de rama a nuevaRama no tendré estas modificaciones:



```

alert("Idade non válida.");

```

TERMINAL

```

PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git branch
* master
  nuevaRama
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git checkout nuevaRama
Switched to branch 'nuevaRama'
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git branch
  master
* nuevaRama

```

Actualmente nuestro escenario es el siguiente:

**master** → Tenemos el alert editado.

**nuevaRama** → Tenemos el último case del switch eliminado pero no tenemos la última modificación de master sobre el alert.

Lo suyo sería hacer ahora un merge de los cambios, el `git merge nombreRama` siempre se va a producir desde la rama en la que estoy situado, es decir, el comando lo escribo estando en master ya que quiero guardar aquí los cambios y traigo a master los cambios de la rama nuevaRama.

El resultado de hacer el merge generará un nuevo commit resultado de este proceso. También es cierto que podrían producirse conflictos si se estuviera intentando hacer un merge entre dos ramas con modificaciones de código sobre el mismo elemento, este caso lo veremos más adelante.

```

let edad = prompt("Por favor, introduce tu edad:");
edad = parseInt(edad);

if (isNaN(edad)) {
    alert("Por favor, introduce un número válido para la edad.");
} else {
    let categoria;

    switch (true) {
        case edad >= 0 && edad <= 12:
            categoria = "neno";
            break;
        case edad >= 13 && edad <= 18:
            categoria = "adolescente";
            break;
        case edad >= 19 && edad <= 30:
            categoria = "xoven";
            break;
        case edad >= 31 && edad <= 100:
            categoria = "adulto";
            break;
        default:
            alert("Edad no válida.");
    }
}

```

The screenshot shows the VS Code interface with the terminal tab selected. The terminal output is as follows:

```

BLEMS OUTPUT DEBUG CONSOLE TERMINAL
TERMINAL
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git checkout master
Switched to branch 'master'
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git branch
* master
  nuevaRama
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git merge nuevaRama
Auto-merging codigo.js
Merge made by the 'ort' strategy.
codigo.js | 5 +---
 1 file changed, 1 insertion(+), 4 deletions(-)
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git log
commit 45b4d84b2998143553b9bb01da5a3f471fd6d0e1 (HEAD -> master)
Merge: cde6855 f921996
Author: MARTÍN GIL BLANCO <gilblancomartin@gmail.com>
Date:   Wed Nov 15 10:02:11 2023 +0100

  Merge branch 'nuevaRama'

commit cde68553c72219f983d06f7dcba6de5e92d07c3d
Author: MARTÍN GIL BLANCO <gilblancomartin@gmail.com>
Date:   Wed Nov 15 09:47:52 2023 +0100

  Cambio de texto alert

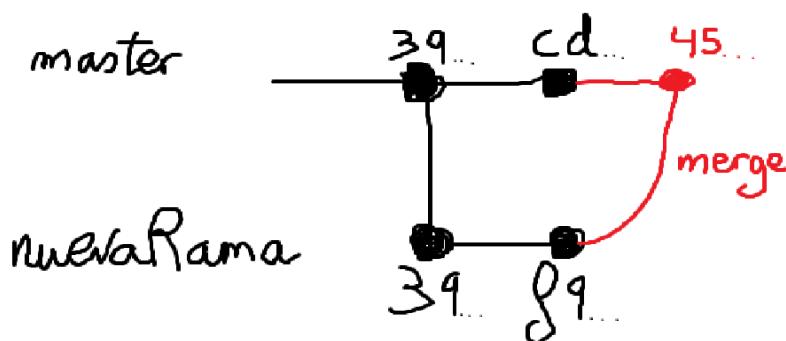
commit f9219961ff482414db38077afc574c134d1b471a (nuevaRama)
Author: MARTÍN GIL BLANCO <gilblancomartin@gmail.com>
Date:   Tue Nov 14 19:20:49 2023 +0100

  Eliminación del último case

commit 39cb2f6849f5707298a403113d00e5de1fe60194
Author: MARTÍN GIL BLANCO <gilblancomartin@gmail.com>
Date:   Tue Nov 14 18:40:18 2023 +0100

  Mi primer commit

```



El escenario que hemos creado sería el siguiente donde se nos ha creado un nuevo commit (Su HASH comienza por 45...) con el resultado de la unión de los cambios de nuevaRama en master.

Por otra parte, muy interesante es fijarse en la imagen del histórico de confirmaciones donde podemos apreciar que nuevaRama se ha quedado en el commit f9... tal y como podemos ver entre paréntesis.

Hasta este momento no se ha producido ningún conflicto entre los códigos que tenemos en las dos ramas, si este caso se diera sería necesario solucionarlos. Vamos a generar un conflicto para entender bien este punto y solucionarlo en nuestro directorio de trabajo local. Vamos a crear una nueva rama y hacer modificaciones en master para que nuestro entorno de trabajo esté en el siguiente punto:

```
master → alert("Modificado en master.");
ramaConflicto → alert("Modificado en ramaConflicto.");
```

A continuación voy a crear una nueva rama y hacer la modificación de código para el conflicto. El proceso en master será similar.

```
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git branch
* master
  nuevaRama
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git checkout -b ramaConflicto
Switched to a new branch 'ramaConflicto'
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git add .
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git status
On branch ramaConflicto
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    modified:   codigo.js

PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git commit -m "Conflicto ramaConflicto"
[ramaConflicto f6f3056] conflicto ramaConflicto
 1 file changed, 1 insertion(+), 1 deletion(-)
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git branch
  master
  nuevaRama
* ramaConflicto
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git checkout master
Switched to branch 'master'
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git branch
* master
  nuevaRama
  ramaConflicto
```

Si yo veo el historial de log de las ramas master y ramaConflicto obtengo lo siguiente:

**master**

```
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git log
commit f6f30564a7f80881c543edd4e6aa55a0791f8525 (HEAD -> ramaConflict)
Author: MARTÍN GIL BLANCO <gilblancomartin@gmail.com>
Date:   Wed Nov 15 10:32:36 2023 +0100

    Conflicto ramaConflict

commit 45b4d84b2998143553b9bb01da5a3f471fd6d0e1
Merge: cde6855 f921996
Author: MARTÍN GIL BLANCO <gilblancomartin@gmail.com>
Date:   Wed Nov 15 10:02:11 2023 +0100

    Merge branch 'nuevaRama'

commit cde68553c72219f983d06f7dcba6de5e92d07c3d
Author: MARTÍN GIL BLANCO <gilblancomartin@gmail.com>
Date:   Wed Nov 15 09:47:52 2023 +0100

    Cambio de texto alert

commit f9219961ff482414db38077afc574c134d1b471a (nuevaRama)
Author: MARTÍN GIL BLANCO <gilblancomartin@gmail.com>
Date:   Tue Nov 14 19:20:49 2023 +0100

    Eliminación del último case

commit 39cb2f6849f5707298a403113d00e5de1fe60194
Author: MARTÍN GIL BLANCO <gilblancomartin@gmail.com>
Date:   Tue Nov 14 18:40:18 2023 +0100

    Mi primer commit
```

**ramaConflict**

```
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git log
commit bed46240138ba36e7e25dcd7ec8fdeda7c436477 (HEAD -> master)
Author: MARTÍN GIL BLANCO <gilblancomartin@gmail.com>
Date:   Wed Nov 15 10:37:24 2023 +0100

    Conflicto master

commit 45b4d84b2998143553b9bb01da5a3f471fd6d0e1
Merge: cde6855 f921996
Author: MARTÍN GIL BLANCO <gilblancomartin@gmail.com>
Date:   Wed Nov 15 10:02:11 2023 +0100

    Merge branch 'nuevaRama'

commit cde68553c72219f983d06f7dcba6de5e92d07c3d
Author: MARTÍN GIL BLANCO <gilblancomartin@gmail.com>
Date:   Wed Nov 15 09:47:52 2023 +0100

    Cambio de texto alert

commit f9219961ff482414db38077afc574c134d1b471a (nuevaRama)
Author: MARTÍN GIL BLANCO <gilblancomartin@gmail.com>
Date:   Tue Nov 14 19:20:49 2023 +0100

    Eliminación del último case

commit 39cb2f6849f5707298a403113d00e5de1fe60194
Author: MARTÍN GIL BLANCO <gilblancomartin@gmail.com>
Date:   Tue Nov 14 18:40:18 2023 +0100

    Mi primer commit
```

Si me situo en la rama master y hago un merge de los cambios de ramaConflicto con el comando `git merge ramaConflicto` se producirá un conflicto el cual tendremos que resolver indicando cual es el código (el de master o ramaConflicto que es correcto). En mi caso voy a asumir que el código correcto es el de ramaConflicto, voy a resolvélo con el propio editor de VS code que estoy empleando en este documento.

```

23 <<<<< HEAD (Current Change)
24     alert("Modificado en master.");
25 =====
26     alert("Modificado en ramaConflicto.");
27 >>>>> ramaConflicto (Incoming Change)
28 }

ROBLEMS 3 OUTPUT DEBUG CONSOLE TERMINAL
▼ TERMINAL
● PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git branch
* master
  nuevaRama
  ramaConflicto
● PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git merge ramaConflicto
Auto-merging codigo.js
CONFLICT (content): Merge conflict in codigo.js
Automatic merge failed: fix conflicts and then commit the result.

f6f3056 · ramaConflicto
...
|  categoria = "xoven";
|  break;
| case edad >= 31 && edad <= 100:
|   categoria = "adulito";
|   break;
| default:
|   ...
|   Accept Incoming | Ignore
|   alert("Modificado en ramaConflicto.");
| }
|
if (categoria) {
|   alert("Eres un " + categoria + ".");
}
codigo.js
...
|  categoria = "xoven";
|  break;
| case edad >= 31 && edad <= 100:
|   categoria = "adulito";
|   break;
| default:
|   ...
|   No Changes Accepted
|   alert("Idade non válida.");
|
if (categoria) {
|   alert("Eres un " + categoria + ".");
}
}

bed4624 · master
...
|  categoria = "xoven";
|  break;
| case edad >= 31 && edad <= 100:
|   categoria = "adulito";
|   break;
| default:
|   ...
|   Accept Current | Ignore
|   alert("Modificado en master.");
|
if (categoria) {
|   alert("Eres un " + categoria + ".");
}
...
Current bed4624 · master
...
|  categoria = "xoven";
|  break;
| case edad >= 31 && edad <= 100:
|   categoria = "adulito";
|   break;
| default:
|   ...
|   Accept Current | Ignore
|   alert("Modificado en master.");
|
if (categoria) {
|   alert("Eres un " + categoria + ".");
}
...
|  categoria = "xoven";
|  break;
| case edad >= 31 && edad <= 100:
|   categoria = "adulito";
|   break;
| default:
|   ...
|   Incoming | Remove Incoming
|   alert("Modificado en ramaConflicto.");
|
if (categoria) {
|   alert("Eres un " + categoria + ".");
}
}

Complete Merge

```

Como este proceso no ha sido automático tengo que completar el merge haciendo commit yo mismo para confirmar el proceso, recordemos que antes el merge generó un commit automáticamente al realizar el proceso.

```
}; | | alert("Modificado en ramaConflicto.");
```

TERMINAL

```
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git status  
On branch master  
All conflicts fixed but you are still merging.  
(use "git commit" to conclude merge)  
  
Changes to be committed:  
    modified:   codigo.js  
  
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git commit -m "Conflict solucionado"  
[master ed14395] Conflicto solucionado  
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git log  
commit ed1439597146ee1ecb9f5ae275d122b0aa85734e (HEAD -> master)  
Merge: bed4624 f6f3056  
Author: MARTÍN GIL BLANCO <gilblancomartin@gmail.com>  
Date:   Wed Nov 15 10:50:50 2023 +0100  
  
    Conflicto solucionado  
  
commit bed46240138ba36e7e25dcd7ec8fdeda7c436477  
Author: MARTÍN GIL BLANCO <gilblancomartin@gmail.com>  
Date:   Wed Nov 15 10:37:24 2023 +0100  
  
    Conflicto master  
  
commit f6f30564a7f80881c543edd4e6aa55a0791f8525 (ramaConflicto)  
Author: MARTÍN GIL BLANCO <gilblancomartin@gmail.com>  
Date:   Wed Nov 15 10:32:36 2023 +0100  
  
    Conflicto ramaConflicto
```

## GitHub

Como ya se dijo GitHub es una plataforma de desarrollo colaborativo basada en la nube que facilita la gestión de proyectos de software mediante sistemas de control de versiones, como Git. Permite a los desarrolladores trabajar de manera conjunta en proyectos, realizar un seguimiento de cambios en el código, colaborar en el desarrollo de software y coordinar esfuerzos a través de funciones como solicitudes de extracción y problemas. GitHub proporciona herramientas para alojar repositorios de código, controlar revisiones, y gestionar flujos de trabajo de desarrollo, facilitando la colaboración efectiva entre equipos distribuidos geográficamente. Su interfaz web intuitiva y sus características avanzadas, como integración continua y despliegue automático, han convertido a GitHub en una plataforma central para la comunidad de desarrollo de software. La página web es la siguiente: <https://github.com/>

Como curiosidad, aunque git haga un manejo de la rama principal llamandola master, GitHub comenzó a realizar cambios en el nombre predeterminado de la rama principal de los repositorios a "main" en lugar de "master". Este cambio fue parte de un esfuerzo más amplio para hacer que la plataforma sea más inclusiva y evitar la connotación negativa asociada con la palabra "master" en el contexto de la esclavitud. La idea era eliminar términos que pudieran tener connotaciones racistas o discriminatorias y utilizar un lenguaje más neutral y amigable. "main" es simplemente un término más neutral y no tiene las connotaciones históricas que algunos encuentran problemáticas en "master".

Es importante destacar que este cambio no se limita a GitHub; otras plataformas de desarrollo también han adoptado prácticas similares para fomentar la inclusividad y eliminar lenguaje potencialmente ofensivo.

Lo primero de todo va a ser crear un repositorio en el cual vamos a poder añadir el código que estamos empleando.

## Create a new repository

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

Required fields are marked with an asterisk (\*).

Owner \*



martin2745 ▾

Repository name \*



Great repository names are short and memorable. Need inspiration? How about [vigilant-octo-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:

Add a README file

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

Add .gitignore

.gitignore template: None ▾

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

Choose a license

License: None ▾

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

You are creating a public repository in your personal account.

Create repository

Como podemos ver existen dos tipos de repositorios.

**Repositorios Públicos en GitHub:** Los repositorios públicos en GitHub son aquellos que están disponibles para que cualquier persona los vea y clone. Cualquier usuario puede acceder al código, realizar contribuciones (si han sido autorizadas por el titular del repositorio) y seguir el desarrollo del proyecto. Esta opción es ideal para proyectos de código abierto y colaboración comunitaria, ya que fomenta la transparencia y la participación abierta.

**Repositorios Privados en GitHub:** En contraste, los repositorios privados en GitHub son accesibles solo para usuarios autorizados. Estos repositorios ofrecen mayor seguridad y privacidad, ya que solo aquellos con permisos específicos pueden acceder al código y colaborar en el proyecto. Los repositorios privados son adecuados para proyectos comerciales o aquellos que requieren confidencialidad, ya que permiten un control más estricto sobre quién puede ver o contribuir al código.

A mayores de esto existe un área de inicialización del repositorio donde tenemos que mencionar tres conceptos que ya veremos en detalle más adelante.

**README en GitHub:** El archivo README.md en un repositorio GitHub es crucial para proporcionar información clave sobre el proyecto. Se utiliza para describir el propósito del proyecto, proporcionar instrucciones para la instalación y uso, y dar orientación sobre cómo contribuir. GitHub automáticamente muestra el contenido del README en la página principal del repositorio, lo que lo convierte en la primera fuente de información para cualquier visitante. Es una herramienta valiosa para comunicar de manera efectiva la esencia y el uso de tu proyecto. Más adelante veremos la forma de darle formato al README. Este archivo está en formato .md, es decir, MarkDown.

**.gitignore en GitHub:** El archivo .gitignore es fundamental para gestionar qué archivos y directorios deben ser ignorados por Git al realizar seguimiento de cambios. En GitHub, este archivo es especialmente útil para excluir archivos temporales, datos generados y otros elementos que no deben ser parte del repositorio. Al especificar patrones en el archivo .gitignore, puedes evitar que estos archivos innecesarios se incluyan accidentalmente en el control de versiones, manteniendo así el repositorio limpio y enfocado en archivos esenciales del proyecto.

**Opción de Licencias en GitHub:** GitHub proporciona una opción para agregar una licencia a tu proyecto, lo que establece los términos bajo los cuales otros pueden usar, modificar y distribuir tu código. Esto es esencial para clarificar los derechos y restricciones asociados con tu trabajo. Al crear un repositorio, puedes elegir una licencia de software común, como MIT, GPL o Apache, o incluso personalizar los términos según tus necesidades. La elección de una licencia adecuada es crucial para definir el marco legal y ético para la colaboración y el uso del proyecto. Existen diferentes tipos de licencias y cada una con sus particularidades.

- **MIT License:** Permite a las personas hacer cualquier cosa que deseen con tu código, siempre que incluyan el aviso de copyright y la licencia original en cualquier copia o parte sustancial del trabajo.
- **GNU General Public License (GPL):** Impone restricciones más estrictas, exigiendo que cualquier trabajo derivado también se distribuya bajo la misma licencia GPL. Hay diferentes versiones de la GPL, como GPL-2.0 y GPL-3.0, cada una con sus propias disposiciones específicas.
- **Apache License:** Permite a las personas hacer cualquier cosa con tu trabajo, siempre que se otorgue el debido crédito. También proporciona una

exención de responsabilidad. Es una licencia permisiva que equilibra la protección de derechos de autor con la flexibilidad para los usuarios.

En mi caso voy a añadir la siguiente configuración del repositorio, más adelante nos centraremos en otros aspectos importantes sobre el repositorio como es el caso de .gitignore de una forma práctica.

## Create a new repository

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

Required fields are marked with an asterisk (\*).

Owner \* Repository name \*



martin2745

/ cursoGit

✓ cursoGit is available.

Great repository names are short and memorable. Need inspiration? How about [musical-enigma](#) ?

Description (optional)

Repositorio para el Curso de Git y GitHub empleando VScode

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:

Add a README file

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

Add .gitignore

.gitignore template: None ▾

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

Choose a license

License: None ▾

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

You are creating a public repository in your personal account.

Create repository

Aunque podríamos añadir un README.md no lo vamos a hacer, ya que existe una forma muy sencilla de crear un repositorio desde cero cómo vamos a ver a continuación. En mi caso, simplemente añado un título y descripción del repositorio y lo mantengo como público

(importante mencionar que estos ajustes se pueden editar más adelante desde settings del repositorio).

The screenshot shows a GitHub repository page for 'cursoGit'. At the top, there are buttons for Pin, Unwatch (1), Fork (0), and Star (0). Below the header, there are sections for 'Set up GitHub Copilot' and 'Add collaborators to this repository'. A 'Quick setup' section provides instructions for desktop setup or cloning via HTTPS or SSH, and includes links for creating a new file or uploading an existing one. It also recommends including README, LICENSE, and .gitignore files. Below this, there are two code snippets: one for creating a new repository from the command line and another for pushing an existing repository. Each snippet has a copy icon.

```

echo "# cursoGit" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin https://github.com/martin2745/cursoGit.git
git push -u origin main

```

```

git remote add origin https://github.com/martin2745/cursoGit.git
git branch -M main
git push -u origin main

```

Una vez creado el repositorio tenemos una serie de comandos a introducir por parte del usuario para inicializar el repositorio desde línea de comandos. Si bien estos comandos están pensados para un repositorio que no esté iniciado (en nuestro caso). Aún así, vamos a proceder a explicar qué hace cada comando y ver como hacemos nosotros el proceso.

`echo "# cursoGit" >> README.md`: Este comando crea un archivo llamado README.md en el directorio actual y agrega el texto "# cursoGit" al archivo. El `echo` se utiliza para imprimir texto en la consola, y el operador `>>` se utiliza para redirigir la salida al archivo README.md.

`git init`: Este comando inicializa un nuevo repositorio de Git. Crea un subdirectorio oculto llamado `.git` que contiene todos los archivos necesarios para el repositorio.

`git add README.md`: Agrega el archivo README.md al área de preparación de Git. Esto significa que Git ahora está al tanto de los cambios realizados en ese archivo y está listo para realizar un commit.

`git commit -m "first commit"`: Realiza un commit de los cambios en el área de preparación. El mensaje entre comillas después de `-m` es un comentario que describe brevemente los cambios realizados en este commit.

`git branch -M main`: Renombra la rama predeterminada de Git de "master" a "main". Este comando es opcional y se ha vuelto más común para reflejar un lenguaje más inclusivo y culturalmente consciente.

`git remote add origin https://github.com/martin2745/cursoGit.git`: Establece un enlace remoto llamado "origin" al repositorio en GitHub con la URL proporcionada. Esto le dice a Git dónde se encuentra el repositorio remoto al que se enviarán los cambios.

`git push -u origin main`: Sube los cambios al repositorio remoto en GitHub. El `-u` establece la conexión entre la rama local y la rama remota, por lo que en futuros `git push` no será necesario especificar la rama y simplemente se utilizará la rama configurada previamente. La rama especificada es "main", que es la rama predeterminada en este caso.

Llegados a este punto podemos tener dudas con que nos referimos con "origin", podemos decir que en el contexto de Git y GitHub, "origin" es el nombre convencional que damos al repositorio remoto por defecto cuando clonamos un repositorio. Utilizamos el término "origin" como un alias que apunta al URL del repositorio remoto desde el cual hemos clonado nuestro repositorio local.

Cuando ejecutamos el comando `git remote add origin <url>`, estamos configurando un enlace remoto llamado "origin" que apunta al repositorio remoto en la dirección proporcionada (en este caso, <https://github.com/martin2745/cursoGit.git>). Este enlace remoto permite a Git saber dónde enviar los cambios locales cuando ejecutamos comandos como `git push`. Lo primero que vamos a hacer, si bien no es necesario, va a ser eliminar las dos ramas que creamos anteriormente de prueba y quedarnos solo con la rama master dentro de nuestro repositorio local. Para ello empleamos el comando `git branch -d nombreRama`.

```
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git branch
* master
  nuevaRama
  ramaConflictio
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git branch -d ramaConflictio
Deleted branch ramaConflictio (was f6f3056).
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git branch -d nuevaRama
Deleted branch nuevaRama (was f921996).
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git branch
* master
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> echo "# cursoGit" >> README.md
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git add README.md
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git commit -m "Añadido README.md"
[master df615d7] Añadido README.md
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 README.md
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git branch -M main
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git branch
* main
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git remote add origin https://github.com/martin2745/cursoGit.git
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git push -u origin main
Enumerating objects: 23, done.
Counting objects: 100% (23/23), done.
Delta compression using up to 12 threads
Compressing objects: 100% (22/22), done.
Writing objects: 100% (23/23), 2.52 KiB | 2.52 MiB/s, done.
Total 23 (delta 8), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (8/8), done.
To https://github.com/martin2745/cursoGit.git
 * [new branch]  main -> main
branch 'main' set up to track 'origin/main'.
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> █
```

Ya que en nuestro caso teníamos ya el repositorio iniciado en local, simplemente creamos el README.md, lo añadimos al área de staging, confirmamos esta nueva modificación, renombramos la rama predeterminada de Git de "master" a "main", establece un enlace remoto (demonios que origin es la URL del repositorio), establecemos la conexión entre la rama remota y la local para posteriormente poder ejecutar simplemente el push. Como resultado si vemos GitHub tendremos lo siguiente.

Como podemos ver, una vez creado el repositorio, tenemos diferentes opciones en la cabecera.

#### 1. Code (Código):

- En esta sección podemos ver el código existente en el repositorio, así como las diferentes ramas remotas que existan, podremos cambiar entre estas ramas, acceder al contenido de cada archivo, modificarlo, ver el historial de commits...

#### 2. Issues (Problemas):

- Los issues son una herramienta para realizar un seguimiento de tareas, mejoras, errores y otras colaboraciones en tu proyecto. Puedes abrir un issue para discutir y asignar trabajo, y otros colaboradores pueden comentar y contribuir a la resolución del problema.

#### 3. Pull Requests (Solicitudes de extracción):

- Las pull requests son propuestas de cambios que los colaboradores envían al proyecto. Después de hacer cambios en una rama, puedes enviar una pull request para fusionar esos cambios en la rama principal. Facilitan la revisión del código y la colaboración antes de que se incorporen los cambios.

#### 4. Actions (Acciones):

- GitHub Actions es un sistema de automatización que te permite definir flujos de trabajo personalizados para tu proyecto. Puedes automatizar tareas como pruebas, compilaciones y despliegues. Actions se configura mediante archivos YAML dentro del repositorio y proporciona una forma flexible de integrar y automatizar procesos.

#### 5. Projects (Proyectos):

- Los proyectos de GitHub son tableros de colaboración que te permiten organizar y priorizar tareas. Puedes crear columnas para representar

diferentes etapas de tu flujo de trabajo y mover tarjetas entre ellas para indicar el progreso.

#### 6. Wiki:

- La wiki de GitHub te permite crear y mantener documentación para tu proyecto directamente en el repositorio. Puedes utilizarla para proporcionar información detallada sobre el proyecto, instrucciones de instalación y cualquier otra documentación relevante.

#### 7. Security (Seguridad):

- La pestaña "Security" en GitHub te proporciona información sobre las vulnerabilidades de seguridad en tu proyecto. Puedes recibir alertas sobre vulnerabilidades conocidas en las dependencias de tu proyecto y tomar medidas para solucionar esos problemas.

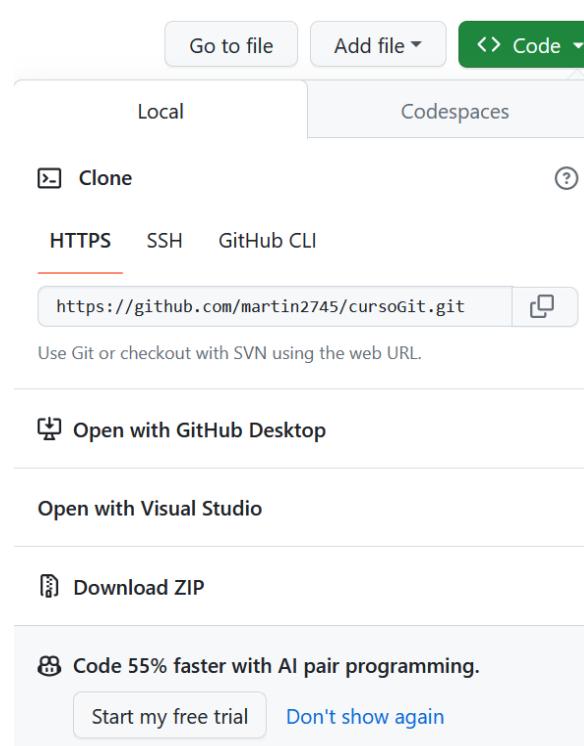
#### 8. Insights (Información):

- La sección "Insights" en GitHub proporciona información detallada sobre la actividad y la salud de tu repositorio. Puedes ver estadísticas sobre contribuyentes, tráfico y clonaciones, lo que te ayuda a comprender cómo se está utilizando tu proyecto.

#### 9. Settings (Ajustes):

- La sección de ajustes te permite configurar opciones específicas para tu repositorio, como la configuración de colaboradores, la administración de acceso, la integración con servicios externos y la configuración de branch protection, entre otras cosas. Esta parte es de especial importancia ya que nos permite hacer acciones como cambiar la visibilidad del repositorio o eliminarlo.

Cuando hacemos clic en el botón "Code" en GitHub y se despliegan las opciones, estamos accediendo a las diferentes formas de clonar el repositorio en tu máquina local.



**HTTPS:** Al seleccionar HTTPS, obtendremos la URL del repositorio que podemos usar para clonar el proyecto utilizando HTTPS. Podemos copiar esta URL y usarla con el comando git clone.

**SSH (Secure Shell):** Si elegimos SSH, obtendremos la URL SSH del repositorio. Esta opción utiliza claves SSH para autenticación, lo que puede ser más conveniente si ya hemos configurado nuestras claves SSH. Para clonar el repositorio con SSH, necesitaremos configurar y agregar nuestras claves SSH a nuestra cuenta de GitHub.

**Descargar ZIP:** Esta opción nos permite descargar el código fuente del repositorio en formato ZIP. Podemos extraer el ZIP en nuestro ordenador sin utilizar Git.

Si clicamos en commits podemos ver el histórico de commits del sistema y podemos ver las modificaciones que se realizan en cada commit, vamos a ver el commit “Conflicto solucionado”.

### Commits

The screenshot shows the GitHub commits page for a repository. At the top, there's a dropdown menu set to 'main'. Below it, a section titled 'Commits on Nov 15, 2023' lists several commits:

- Añadido README.md (commit df615d7)
- Conflicto solucionado (commit ed14395)
- Conflicto master (commit bed4624)
- Conflicto ramaConflicto (commit f6f3056)
- Merge branch 'nuevaRama' (commit 45b4d84)
- Cambio de texto alert (commit cde6855)

Below this, another section titled 'Commits on Nov 14, 2023' shows two more commits:

- Eliminación del último case (commit ff21996)
- Mi primer commit (commit 39cb2f6)

### Commit

The screenshot shows the GitHub commit page for the commit 'Conflicto solucionado' (commit ed14395). It includes the commit message, author ('martin2745'), date ('committed 4 hours ago'), and the fact that it has 2 parents (bed4624 + f6f3056) and a commit message ('commit ed14395').

Below the commit message, it says 'Showing 1 changed file with 1 addition and 1 deletion.' A code diff viewer shows the changes made to 'codigo.js':

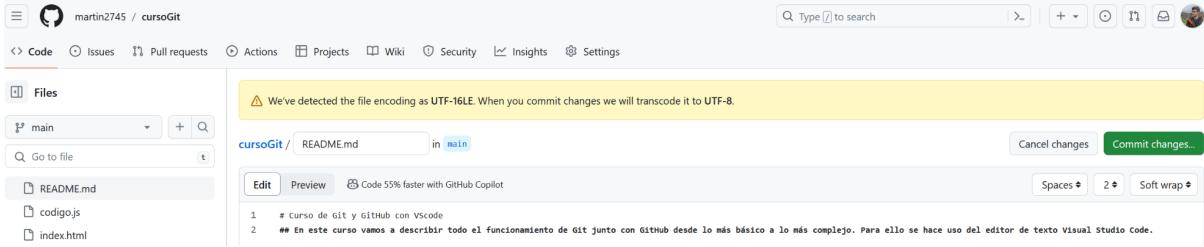
```

diff --git a/codigo.js b/codigo.js
@@ -20,7 +20,7 @@ if (isNaN(edad)) {
 20   20     categoria = "adulto";
 21   21     break;
 22   22     default:
 23 -     alert("Modificado en master.");
 23 +     alert("Modificado en ramaConflicto.");
 24   24   }
 25   25   if (categoria) {
 26   26

```

At the bottom, there's a comment section with a placeholder 'Leave a comment' and a green button 'Comment on this commit'.

Si nos fijamos, el README.md se ha creado de forma incorrecta y muestra mal su contenido, podemos modificar el código y hacer push al repositorio o editararlo desde el propio repositorio. Vamos a editararlo desde github clicando sobre el README.md y clicando en la opción de editar.



We've detected the file encoding as UTF-16LE. When you commit changes we will transcode it to UTF-8.

**curoGit / README.md**

**Commit changes**

**Commit message**

Corrección del README.md

**Extended description**

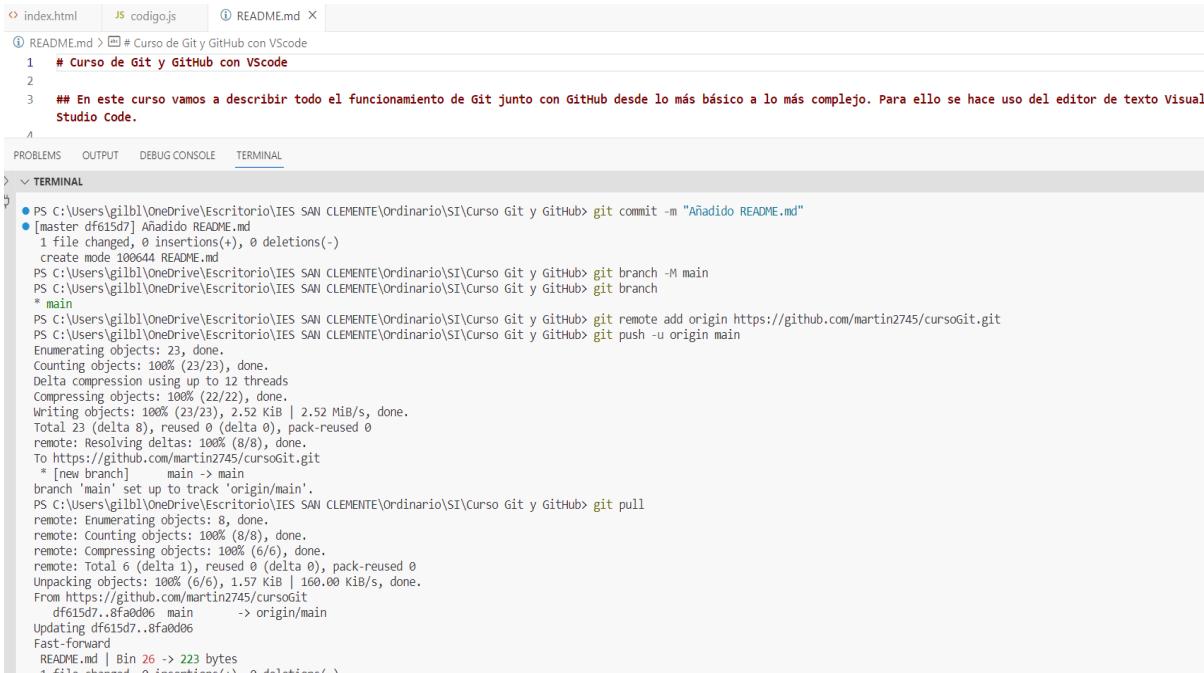
Se ha corregido el mensaje del README.md

Commit directly to the main branch  
 Create a new branch for this commit and start a pull request

[Learn more about pull requests](#)

**Commit changes**

**git pull**



```

PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\curso Git y GitHub> git commit -m "Añadido README.md"
[master df615d7] Añadido README.md
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 README.md
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\curso Git y GitHub> git branch -M main
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\curso Git y GitHub> git branch
* main
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\curso Git y GitHub> git remote add origin https://github.com/martin2745/cursoGit.git
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\curso Git y GitHub> git push -u origin main
Enumerating objects: 23, done.
Counting objects: 100% (23/23), done.
Delta compression using up to 12 threads
Compressing objects: 100% (22/22), done.
Writing objects: 100% (23/23), 2.52 KiB | 2.52 MiB/s, done.
Total 23 (delta 8), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (8/8), done.
To https://github.com/martin2745/cursoGit.git
 * [new branch]  main -> main
branch 'main' set up to track 'origin/main'.
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\curso Git y GitHub> git pull
remote: Enumerating objects: 8, done.
remote: Counting objects: 100% (8/8), done.
remote: Compressing objects: 100% (6/6), done.
remote: Total 6 (delta 1), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (6/6), 1.57 KiB | 160.00 KiB/s, done.
From https://github.com/martin2745/cursoGit
  df615d7..8fa0d06  main      -> origin/main
Updating df615d7..8fa0d06
Fast-forward
 README.md | Bin 26 -> 223 bytes
 1 file changed, 0 insertions(+), 0 deletions(-)

```

Hasta este punto hemos visto el manejo básico de Git y GitHub, con esto, sería ya posible trabajar en diferentes proyecto haciendo uso de la herramienta, de todos modos, existen muchos conceptos que veremos en los siguientes capítulos.

# Manejo de la seguridad de nuestro repositorio

El cifrado simétrico y asimétrico son dos técnicas fundamentales en el campo de la criptografía, utilizadas para proteger la confidencialidad y la integridad de la información. Ambos métodos implican el uso de claves, pero difieren en cómo se gestionan y comparten estas claves.

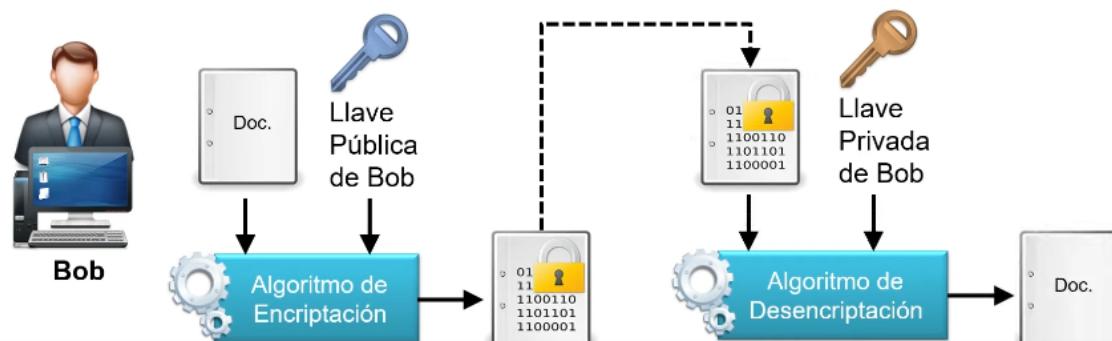
## 1. Cifrado Simétrico:

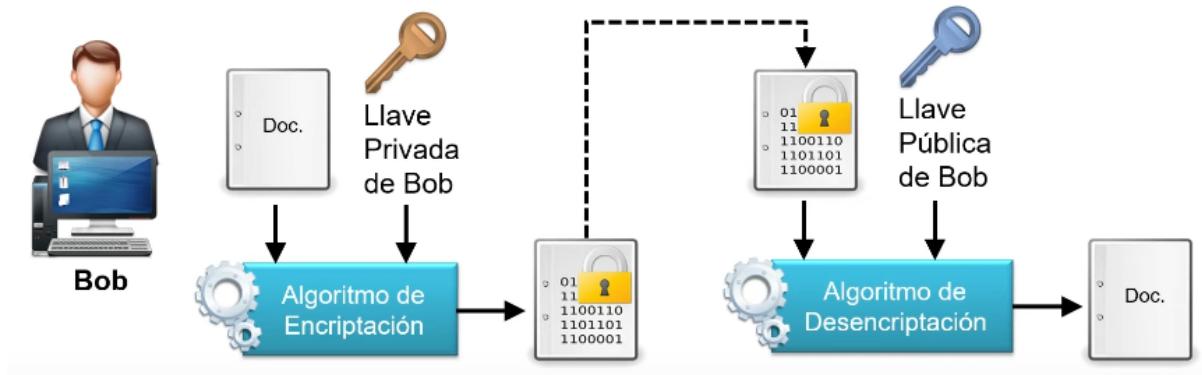
- En el cifrado simétrico, la misma clave se utiliza tanto para cifrar como para descifrar la información. Es un proceso más rápido que el cifrado asimétrico, lo que lo hace adecuado para grandes cantidades de datos.
- La principal desventaja es la necesidad de compartir la clave de manera segura entre las partes que desean comunicarse. Si un tercero no autorizado obtiene la clave, puede descifrar la información.
- Ejemplos de algoritmos de cifrado simétrico son DES (Data Encryption Standard), AES (Advanced Encryption Standard) y 3DES (Triple DES).

## 2. Cifrado Asimétrico (o de Clave Pública):

- En el cifrado asimétrico, se utilizan dos claves diferentes pero relacionadas: una clave pública y una clave privada. La clave pública se comparte abiertamente, mientras que la clave privada se mantiene en secreto.
- Un mensaje cifrado con la clave pública solo puede descifrarse de manera efectiva con la clave privada correspondiente y viceversa. Esto proporciona un nivel adicional de seguridad, ya que no es necesario compartir la clave privada.
- Se utiliza comúnmente en situaciones donde la distribución segura de claves simétricas sería difícil. Además, se emplea para la firma digital, que proporciona autenticación y verificación de la integridad del mensaje.
- Ejemplos de algoritmos de cifrado asimétrico son RSA (Rivest-Shamir-Adleman) y ECC (Elliptic Curve Cryptography).

En muchos casos, se utiliza una combinación de cifrado simétrico y asimétrico para aprovechar las ventajas de ambos. Por ejemplo, en un protocolo TLS/SSL, se utiliza el cifrado asimétrico para el intercambio seguro de claves simétricas, que luego se utiliza para cifrar la comunicación a granel de manera eficiente con cifrado simétrico. Esto se conoce como un sistema de "cifrado híbrido".



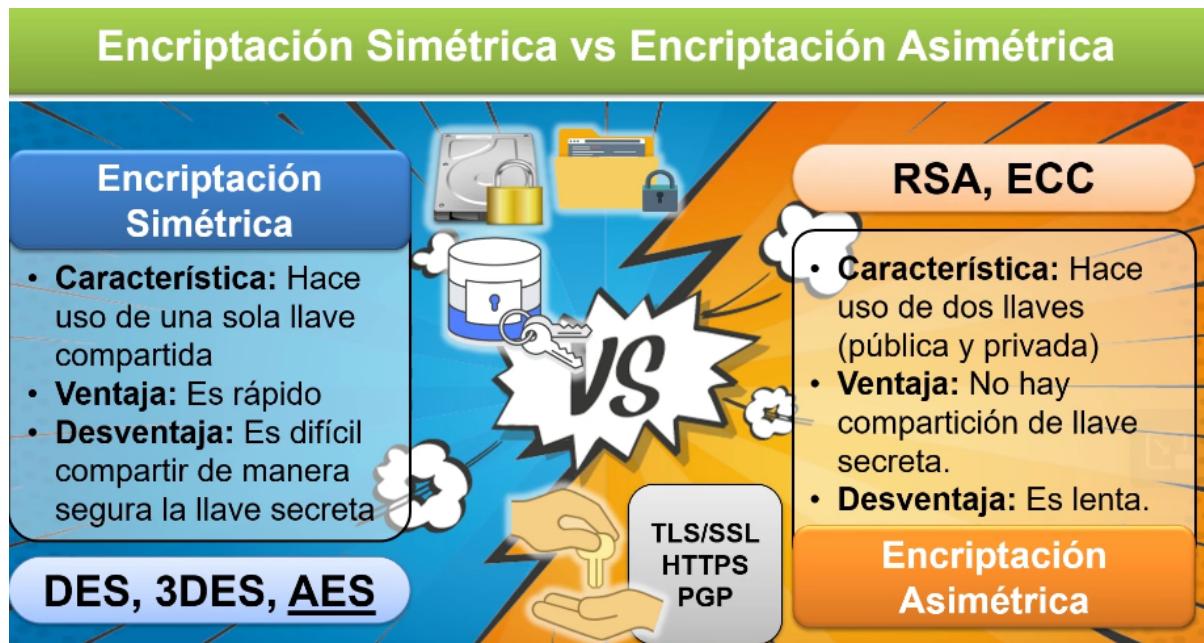
**Tiene 2 usos:**

- **Uso 1:** Proteger la confidencialidad de información secreta.

**Encriptación Asimétrica (Llave Pública)****Tiene 2 usos:**

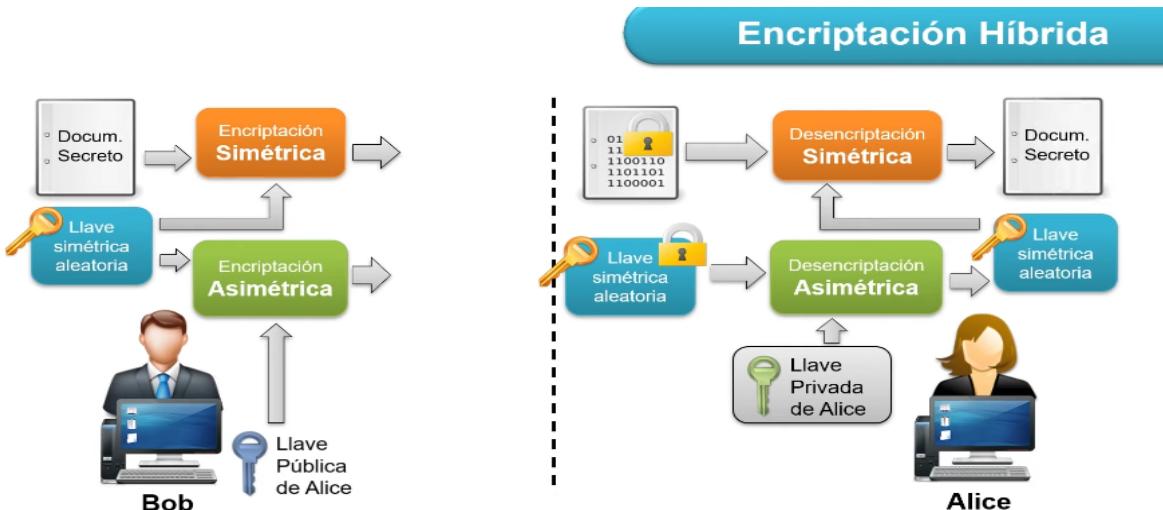
- **Uso 2:** Verificar la autenticidad/origen de la información (no repudio).

**Encriptación Asimétrica (Llave Pública)**



## Encriptación Híbrida (Sobre Digital)

Mezcla las ventajas de la encriptación simétrica y asimétrica



La ventaja de utilizar SSH en lugar de HTTPS en un proyecto de GitHub radica principalmente en la comodidad y seguridad que ofrece. SSH (Secure Shell) utiliza un método de autenticación basado en claves criptográficas, lo que significa que no es necesario ingresar sus credenciales cada vez que realiza una operación, como clonar un repositorio o realizar un push. Esto simplifica el proceso y proporciona una experiencia más fluida. Además, SSH utiliza un canal seguro para la comunicación, cifrando los datos transmitidos entre el cliente y el servidor, lo que mejora la seguridad en comparación con HTTPS. Al elegir SSH, los desarrolladores pueden beneficiarse de una autenticación eficiente y una capa adicional de protección para las interacciones con el repositorio en GitHub.

A continuación, vamos a realizar el proceso de configuración de llaves para el uso de SSH en GitHub. Estas llaves se crean a nivel de equipo, es decir, los proyectos que estén en un mismo ordenador van a tener una misma clave SSH.

La configuración a nivel de generación de claves SSH para su uso en GitHub depende del correo electrónico de la configuración, este ha de ser el mismo que el de nuestra cuenta en GitHub. Esto podemos comprobarlo a nivel de configuración con el comando.

```
git config --list
```

En caso de que el correo electrónico no sea correcto podemos configurarlo (al igual que el nombre del usuario) mediante.

```
git config --global user.name "Tu Nombre"  
git config --global user.email TuEmail@Email.com
```

Lo recomendable para la generación de la clave SSH es colocarnos en el directorio Users/TuUsuario. Una vez situados en esta ruta debemos ejecutar el comando:

```
ssh-keygen -t rsa -b 4096 -C "TuEmail@Email.com"
```

Podemos resumir el comando anterior de la siguiente forma:

**ssh-keygen** es un comando que genera pares de claves SSH, como RSA, para autenticación segura en sistemas Unix. Se utiliza para crear una clave privada y su correspondiente clave pública.

**-t rsa**: Especifica el tipo de algoritmo de clave a utilizar, en este caso, RSA. RSA es un algoritmo de criptografía asimétrica ampliamente utilizado para la seguridad de claves públicas.

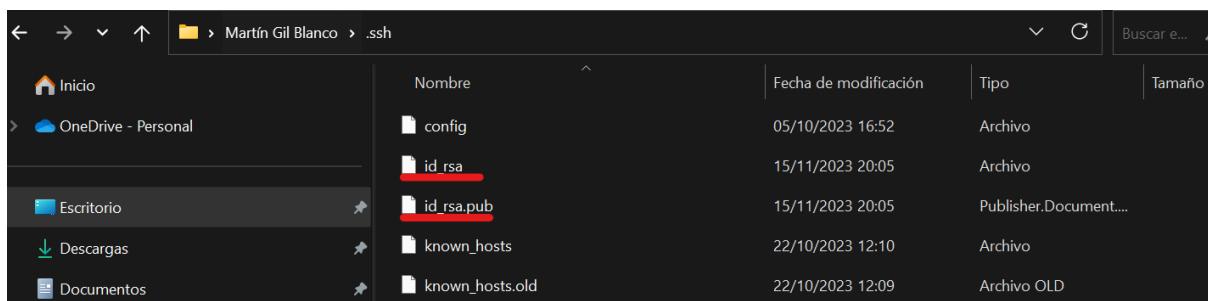
**-b 4096**: Indica el tamaño de bits de la clave generada. En este caso, se está generando una clave RSA de 4096 bits, que es más segura que tamaños de clave más pequeños.

**-C "TuEmail@Email.com"**: Agrega un comentario a la clave generada. Este comentario generalmente se utiliza para identificar la clave, y en este caso, se ha establecido como la dirección de correo electrónico asociada a la clave.

Este proceso como resultado nos va a generar nuestro par de claves RSA tal y como vemos a continuación.

```
PS C:\Users\gilbl> pwd
Path
-----
C:\Users\gilbl

PS C:\Users\gilbl> ssh-keygen -t rsa -b 4096 -C "gilblancomartin@gmail.com"
Generating public/private rsa key pair.
Enter file in which to save the key (C:\Users\gilbl/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in C:\Users\gilbl/.ssh/id_rsa
Your public key has been saved in C:\Users\gilbl/.ssh/id_rsa.pub
The key fingerprint is:
SHA256:uUcUMYrkLppa9rG3lvybVC7suGiDA6xJpzFOYlkTkJ0 gilblancomartin@gmail.com
The key's randomart image is:
+---[RSA 4096]---+
| .+ . . +.
| . E o . . o
| . . O . .
| . o . o
| . o o . S o
| oo + . . =
| BoX... . = o
| o*o.o+=+ +
| . o++++=.
+---[SHA256]-----+
```



A continuación vamos a revisar si el servidor de llaves se encuentra en funcionamiento. Pudiera darse el caso de que para iniciar el agente tengamos que emplear la consola de git para lanzar el comando siguiente ya que puede dar problemas desde la terminal de VScode.

Una vez verificado que el servidor SSH está funcionando voy a tener que agregar mi llave privada para poder cifrar mi tráfico, esta llave privada no debe de ser compartida. Posteriormente, añadiremos en GitHub la llave pública que se generó anteriormente y de esta forma el tráfico que se envíe desde mi máquina local irá encriptado y se podrá desencriptar con mi llave pública, garantizando así que somos nosotros, poseedores únicos de la llave privada quienes estamos actuando sobre el repositorio. Para ello empleamos los siguientes comandos como vemos también en la imagen:

```
eval $(ssh-agent -s)
```

```
ssh-add ~/.ssh/id_rsa
```

```
gilbl@Matebook MINGW64 ~ (master)
$ cd ~

gilbl@Matebook MINGW64 ~ (master)
$ eval "$(ssh-agent -s)"
Agent pid 1442

gilbl@Matebook MINGW64 ~ (master)
$ ssh-add ~/.ssh/id_rsa
Identity added: /c/Users/gilbl/.ssh/id_rsa (gilblancomartin@gmail.com)
```

A continuación, vamos a realizar la conexión con GitHub por lo que vamos a tener que añadir la clave pública que se generó junto a la llave privada. Para ello vamos a .ssh en nuestro equipo, copiamos la llave pública y en nuestro perfil de GitHub vamos a la sección de SSH and GPG Keys y añadimos la llave.

The screenshot shows the GitHub user settings interface. On the left, there's a sidebar with various account management options like Public profile, Account, Appearance, Accessibility, Notifications, Access, Billing and plans, Emails, Password and authentication, Sessions, SSH and GPG keys (which is currently selected), Organizations, Enterprises, and Moderation. The main content area has tabs for SSH keys and GPG keys. Under SSH keys, it says "There are no SSH keys associated with your account." and provides a link to generate one. Under GPG keys, it says "There are no GPG keys associated with your account." and provides a link to generate one. At the bottom, there's a "Vigilant mode" section with a checkbox for "Flag unsigned commits as unverified".

## SSH keys

New SSH key

This is a list of SSH keys associated with your account. Remove any keys that you do not recognize.

### Authentication Keys

The screenshot shows a list of authentication keys under the "SSH keys" tab. There is one key listed: "Curso Git", which was added on Nov 16, 2023, and is described as "Never used — Read/write". To the right of the key entry is a "Delete" button.

Check out our guide to [generating SSH keys](#) or troubleshoot [common SSH problems](#).

A continuación tenemos que cambiar la configuración de nuestro repositorio que está adaptado para el uso de HTTPS. Para ello lo primero es ir a nuestro repositorio en GitHub e ir a code/SSH para copiar la ruta SSH del repositorio.

The screenshot shows a GitHub repository named 'cursoGit'. On the right side, there is a 'Clone' menu with options for Local and Codespaces. Under the 'Clone' section, the SSH URL 'git@github.com:martin2745/cursoGit.git' is highlighted with a red box. Below the URL, it says 'Use a password-protected SSH key.' At the bottom of the menu, there are buttons for 'Start my free trial' and 'Don't show again'.

Una vez realizado este paso vamos a VSCode y cambiamos la configuración. Para ello usamos los siguientes comandos:

```
git remote -v
git remote set-url origin urlSSH
```

```
PS C:\Users\gilbl> git remote -v
origin https://github.com/martin2745/gilbl.git (fetch)
origin https://github.com/martin2745/gilbl.git (push)
PS C:\Users\gilbl> git remote set-url origin git@github.com:martin2745/cursoGit.git
PS C:\Users\gilbl> git remote -v
origin git@github.com:martin2745/cursoGit.git (fetch)
origin git@github.com:martin2745/cursoGit.git (push)
```

Una vez llegados hasta este punto podemos simplemente hacer una pequeña modificación en uno de nuestros archivos y hacer push al repositorio. Como vamos a poder apreciar, en el momento de subir los cambios se nos dice que no se da establecido la autenticación, esto es normal la primera vez que hagamos este proceso, simplemente escribimos "yes" y se realiza la subida de los cambios.

Hemos pues terminado el proceso de autenticación con SSH para nuestro repositorio en GitHub, lo que proporciona una capa adicional de seguridad al acceder y colaborar en el código fuente de nuestro proyecto. Ahora podemos realizar operaciones como clonar, hacer push y pull de manera segura utilizando claves SSH.

```
# Curso de Git y GitHub con vscode

### En este curso vamos a describir todo el funcionamiento de Git junto con GitHub desde lo más básico a lo más complejo.

TERMINAL
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git status
On branch main
Your branch is up to date with 'origin/main'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
    modified:   README.md

no changes added to commit (use "git add" and/or "git commit -a")
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git add .
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git commit -m "README.md editado para probar SSH"
[main 6b8acd4] README.md editado para probar SSH
 1 file changed, 1 insertion(+), 1 deletion(-)
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git status
On branch main
Your branch is ahead of 'origin/main' by 1 commit.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git push
The authenticity of host 'github.com (140.82.121.3)' can't be established.
ED25519 key fingerprint is SHA256:+DiY3wVV6TuJJhbpzisF/zLDA0zPMsvHdkr4UvC0qu.
This key is not known by any other names
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'github.com' (ED25519) to the list of known hosts.
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 12 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 380 bytes | 380.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To github.com:martin2745/cursoGit.git
  2bbdbc8..6b8acd4  main -> main
```

 cursoGit Public

 Pin  Unwatch 1

 main  1 branch  0 tags Go to file Add file Code

 martin2745	README.md editado para probar SSH	6b8acd4 1 minute ago	 12 commits
	README.md	README.md editado para probar SSH	1 minute ago
	codigo.js	Conflicto ramaConflicto	yesterday
	index.html	Mi primer commit	2 days ago

 README.md 

## Curso de Git y GitHub con VScode

En este curso vamos a describir todo el funcionamiento de Git junto con GitHub desde lo más básico a lo más complejo.

```
2 README.md
@@ -1,3 +1,3 @@
1 1 # Curso de Git y GitHub con VScode
2 2
3 - ### En este curso vamos a describir todo el funcionamiento de Git junto con GitHub desde lo más básico a lo más complejo. Para ello se hace uso del editor de texto Visual Studio Code.
3 + ### En este curso vamos a describir todo el funcionamiento de Git junto con GitHub desde lo más básico a lo más complejo.
```

## Uso de Tags y Alias

El uso de tags en Git es esencial para marcar versiones específicas de tu código, proporcionando puntos de referencia históricos y facilitando la identificación de versiones estables. Estas tags son fundamentales para el versionamiento, despliegue y documentación del software, permitiendo a los equipos trabajar de manera coordinada y referenciar cambios significativos en el desarrollo.

Por otro lado, los alias en Git son herramientas clave para simplificar el flujo de trabajo diario. Permiten la creación de atajos para comandos comunes, mejorando la eficiencia al reducir la escritura necesaria y personalizando el entorno de desarrollo. Los alias no solo facilitan la legibilidad del código Git, sino que también pueden integrarse con scripts, contribuyendo a una automatización más efectiva del proceso de desarrollo y colaboración.

Category	Commit Message	Author	Date
.circleci	Reduce install network flake (#27464)		last month
.codesandbox	Codesandbox: upgrade to Node.js 18 (#26330)		8 months ago
.github	Use content hash for react-native builds (#26734)		7 months ago
fixtures	Bump browserify-sign from 4.0.4 to 4.2.2 in /fixtures/expiration (#27600)		2 weeks ago
packages	[Fizz] handle errors in onHeaders (#27712)	ee68446	13 hours ago
scripts	refactor(ci/build): dont generate sourcemaps for BROWSER_SCRIPT bun...		last week

Hasta este punto, el historial de commits que hemos realizado en nuestro repositorio es el siguiente que podemos ver fácilmente con el comando:

```
git log --all --graph --decorate --oneline
```

```
C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git log --all --graph --decorate --oneline
* 4715ac5 (HEAD -> main, origin/main) Definición del temario del README
* 6b8acd4 README.md editado para probar SSH
* 2bbbcd8 Editada la descripción del README.md
* 8fa0d06 Update README.md
* 1e65405 Corrección del README.md
* df615d7 Añadido README.md
* ed14395 Conflicto solucionado
  \
  | * f6f3056 Conflicto ramaConflicto
  | | bed4624 Conflicto master
  | /
  | * 45b4d84 Merge branch 'nuevaRama'
  |
  | * f921996 Eliminación del último case
  | | cde6855 Cambio de texto alert
  |
  * 39cb2f6 Mi primer commit
```

Si bien algún commit que hemos realizado no se ha contemplado en las explicaciones anteriores podemos ver de forma muy clara nuestro funcionamiento hasta este punto con las diferentes ramas. Este comando es de gran interés para ver el flujo de trabajo de nuestro proyecto. Lógicamente algunos comandos son difíciles de recordar por la cantidad

de opciones que tienen, por este motivo es de gran interés el manejo de **alias** de comandos. Voy a crear un nuevo alias llamado ramas que contenga el comando anterior para que sea más fácil de invocar en el futuro.

```
alias nombreAlias="comando a ejecutar como alias"
unalias nombreAlias
```

```
gilbl@Matebook MINGW64 ~/OneDrive/Escritorio/IES SAN CLEMENTE/Ordinario/SI/Curso
  Git y GitHub (main)
$ alias ramas="git log --all --graph --decorate --oneline"

gilbl@Matebook MINGW64 ~/OneDrive/Escritorio/IES SAN CLEMENTE/Ordinario/SI/Curso
  Git y GitHub (main)
$ ramas
* 4715ac5 (HEAD -> main, origin/main) Definición del temario del README
* 6b8acd4 README.md editado para probar SSH
* 2bbbdc8 Editada la descripción del README.md
* 8fa0d06 Update README.md
* 1e65405 Corrección del README.md
* df615d7 Añadido README.md
* ed14395 Conflicto solucionado
|\ \
| * f6f3056 Conflicto ramaConflicto
| | bed4624 Conflicto master
|/
* 45b4d84 Merge branch 'nuevaRama'
|\
| * f921996 Eliminación del último case
| | cde6855 Cambio de texto alert
|/
* 39cb2f6 Mi primer commit

gilbl@Matebook MINGW64 ~/OneDrive/Escritorio/IES SAN CLEMENTE/Ordinario/SI/Curso
  Git y GitHub (main)
$ unalias ramas

gilbl@Matebook MINGW64 ~/OneDrive/Escritorio/IES SAN CLEMENTE/Ordinario/SI/Curso
  Git y GitHub (main)
$ ramas
bash: ramas: command not found
```

Por otra parte tenemos los **tags**, como ya dijimos son etiquetas que marcan puntos específicos en la historia de commits, generalmente asociados con versiones del software. Estos puntos proporcionan referencias estables para identificar y acceder fácilmente a versiones específicas del código, facilitando el seguimiento del desarrollo y la gestión de versiones en proyectos. Vamos a crear un tag en el commit ed14395 Conflicto solucionado como versión estable de nuestro código. Para ello empleamos el siguiente comando:

```
git tag -a v1.0 -m "Primera versión de mi app" ed14395
```

```
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git log --all --graph --decorate --oneline
* 4715ac5 (HEAD -> main, origin/main) Definición del temario del README
* 6b8acd4 README.md editado para probar SSH
* 2bbbdc8 Editada la descripción del README.md
* 8fa0d06 Update README.md
* 1e65405 Corrección del README.md
* df615d7 Añadido README.md
* ed14395 Conflicto solucionado
|\ \
| * f6f3056 Conflicto ramaConflicto
| | bed4624 Conflicto master
|/
* 45b4d84 Merge branch 'nuevaRama'
|\
| * f921996 Eliminación del último case
| | cde6855 Cambio de texto alert
|/
* 39cb2f6 Mi primer commit
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git tag -a v1.0 -m "Primera versión de mi app" ed14395
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git tag
v1.0
```

Una vez creado nuestro primer tag hemos creado la etiqueta v1.0 que corresponde con el commit donde se soluciona el conflicto entre ramas y que todavía no teníamos un README.md. Yo puedo subir estos tags a mi repositorio y ver el estado de este, para ello hago uso del comando:

```
git push origin --tags
```

```
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git push origin --tags
Enumerating objects: 1, done.
Counting objects: 100% (1/1), done.
Writing objects: 100% (1/1), 195 bytes | 195.00 KiB/s, done.
Total 1 (delta 0), reused 0 (delta 0), pack-reused 0
To github.com:martin2745/cursogit.git
 * [new tag]      v1.0 -> v1.0
```

Este es mi repositorio actual con el README.md actualizado y ya me aparece un tag creado.

The screenshot shows a GitHub repository page for 'cursogit'. At the top, there are navigation buttons: 'main' (selected), '1 branch', '1 tag', 'Go to file', 'Add file', and 'Code'. Below this is a list of commits. The first commit is by 'martin2745' titled 'Definición del temario del README' with hash '4715ac5' and timestamp '52 minutes ago'. It includes three files: 'README.md', 'codigo.js', and 'index.html'. The second commit is also by 'martin2745' titled 'Conflicto ramaConflict' with timestamp 'yesterday'. The third commit is by 'martin2745' titled 'Mi primer commit' with timestamp '2 days ago'. At the bottom of the commit list, there is a section titled 'Curso de Git y GitHub con VScode' with a description: 'En este curso vamos a describir todo el funcionamiento de Git junto con GitHub desde lo más básico a lo más complejo.' Below this is a 'Temario' section with a bulleted list of topics: Conceptos básicos, Ramas, Manejo de ramas y solución de conflictos, GitHub, and Manejo de la seguridad de nuestro repositorio.

Si accedemos al tag, veo la versión de mi código donde no existía el README.md.

The screenshot shows the same GitHub repository page as before, but now the 'v1.0' tag is selected in the top navigation bar. The commit list shows the same three commits as before, but the 'index.html' file has been modified. The first commit's description is now 'Conflicto solucionado'. The second commit's description is 'Conflicto ramaConflict'. The third commit's description is 'Mi primer commit'. The 'index.html' file is listed under the second commit.

También es posible eliminar los tag si no nos interesas, vamos a crear otro tag y eliminarlo.

Releases Tags

v2.0 ...  
2 minutes ago 4715ac5 zip tar.gz

v1.0 ...  
17 minutes ago ed14395 zip tar.gz

Si bien yo puedo eliminar las tags que no sean de mi interés con el comando `git tags -d nombreRama`. A pesar de ello y aunque hagamos un `git push origin --tags` las tags en remoto se mantienen por lo que tenemos que ejecutar otro comando para ello.

```
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git tag
v1.0
v2.0
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git push origin --tags
Enumerating objects: 1, done.
Counting objects: 100% (1/1), done.
Writing objects: 100% (1/1), 189 bytes | 189.00 KiB/s, done.
Total 1 (delta 0), reused 0 (delta 0), pack-reused 0
To github.com:martin2745/cursoGit.git
 * [new tag]      v2.0 -> v2.0
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git tag -d v2.0
Deleted tag 'v2.0' (was a76be1d)
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git tag
v1.0
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git push origin --tags
Everything up-to-date
```

cursoGit Public

main 1 branch 2 tags

martin2745 Definición del temario del README 4715ac5 1 hour ago 13 commits

README.md Definición del temario del README 1 hour ago

codigo.js Conflicto ramaConflictido yesterday

index.html Mi primer commit 2 days ago

Para solucionar este problema hacemos uso del siguiente comando:

```
git push origin :refs/tags/v2.0
```

```
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git push origin :refs/tags/v2.0
To github.com:martin2745/cursoGit.git
 - [deleted]          v2.0
```

cursoGit Public

main 1 branch 1 tag

martin2745 Definición del temario del README 4715ac5 1 hour ago 13 commits

README.md Definición del temario del README 1 hour ago

codigo.js Conflicto ramaConflictido yesterday

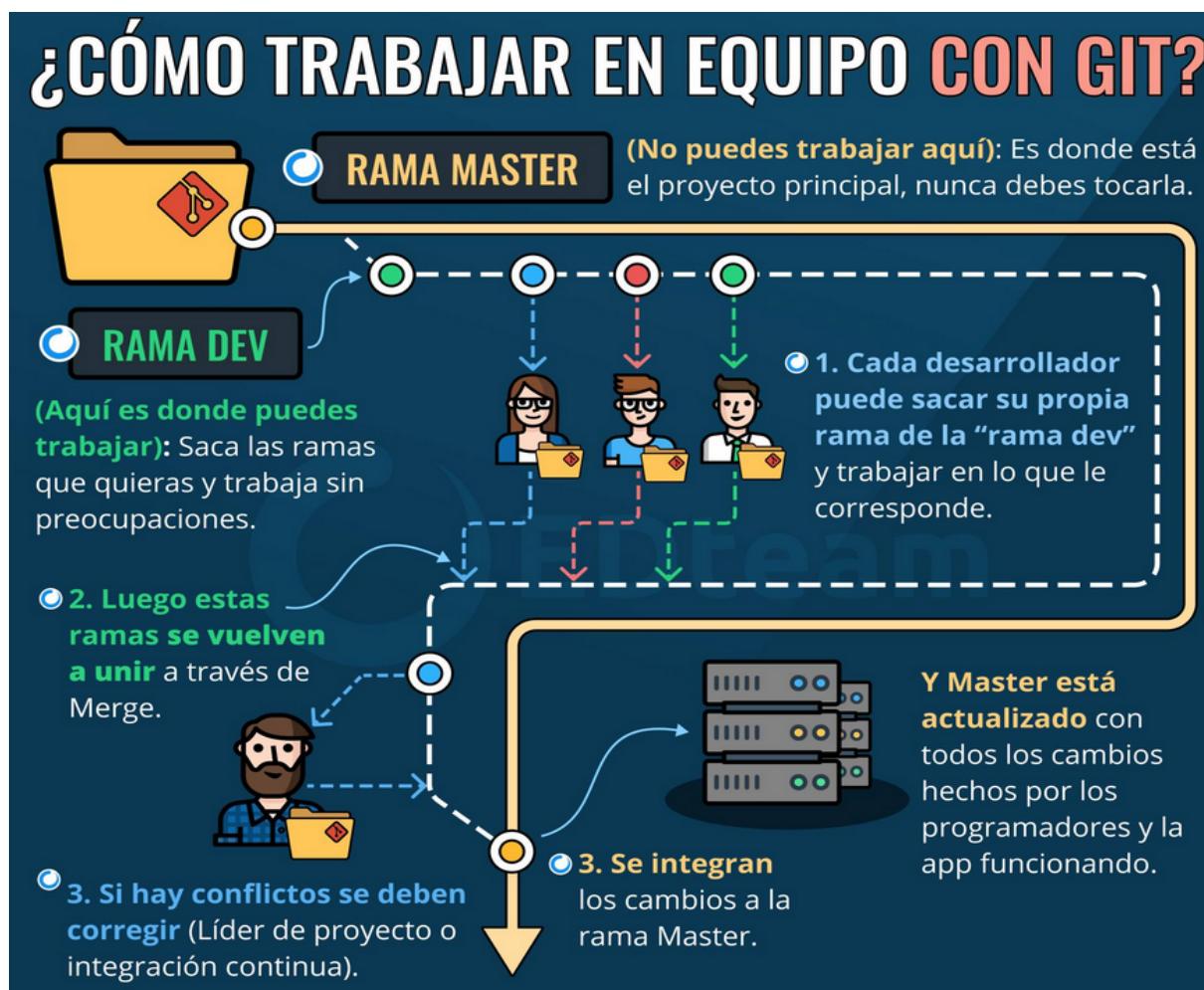
index.html Mi primer commit 2 days ago

## Ramas remotas en GitHub

En el contexto de Git y GitHub, las ramas locales y remotas son conceptos clave para el desarrollo colaborativo y la gestión de versiones. En Git, una rama local es una versión independiente del proyecto que existe solo en tu máquina. Puedes crear ramas locales para trabajar en nuevas características o solucionar problemas sin afectar la rama principal, conocida comúnmente como "master" o "main". Las ramas locales te permiten experimentar sin afectar directamente el código base.

Por otro lado, las ramas remotas existen en el repositorio en línea, como GitHub. Estas ramas reflejan el estado de las ramas locales de varios colaboradores y actúan como un punto de referencia común para la colaboración. Cuando trabajas en equipo, subes tus ramas locales a GitHub para que otros puedan ver, revisar y colaborar en tu código. Las ramas remotas son esenciales para facilitar la colaboración en proyectos distribuidos.

Es importante destacar que las ramas locales no se comparten automáticamente en GitHub. Debes realizar un "push" explícito para enviar tus cambios locales a la rama correspondiente en el repositorio remoto en GitHub. Si no subes tus ramas locales, los demás colaboradores no serán conscientes de su existencia ni podrán acceder a tus cambios. La comunicación efectiva a través de ramas locales y remotas es fundamental para asegurar una colaboración sin problemas en proyectos de desarrollo de software.



En la imagen anterior se describe el flujo normal de trabajo de varias personas en un repositorio común de trabajo. Lo que viene a decir la imagen es que existe una rama común que es la rama master (main actualmente) cuyo código debe de ser siempre funcional y no se puede trabajar directamente sobre esta rama, es decir, si varias personas trabajan en conjunto nadie debería hacer push directamente a main.

A partir de main se genera una rama de trabajo (en el caso del esquema se la llama RAMA DEV) y cada desarrollador emplea una rama creada a partir de esa rama de trabajo. A medida que se terminen las funcionalidades de código y estas hayan sido verificadas como correctas se hará un merge de estas ramas a la rama de desarrollo. El resto de integrantes del equipo tendrán que hacer un pull de las nuevas modificaciones que se han añadido en remoto. Estas verificaciones han de realizarse por el líder del proyecto y según se completen versiones estables de código que pudieran ser de interés, será el líder del proyecto quien haga el merge de la rama de trabajo con todas las nuevas funcionalidades sobre la rama main. Lo que conseguimos de esta forma es siempre tener una versión estable de código en la rama main que pueda ser desplegada en cualquier momento.

En nuestro proyecto hemos creado dos ramas (que posteriormente hemos eliminado) en local pero no las hemos subido al repositorio remoto por lo que no existe constancia de su existencia para otras personas que puedan ver nuestro repositorio o colaborar en él. Vamos a crear una nueva rama y hacer la subida del código al repositorio remoto.

```
● PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git branch
* main
● PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git show-branch
[main] Definición del temario del README
● PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git show-branch --all
* [main] Definición del temario del README
! [origin/main] Definición del temario del README
-
*+ [main] Definición del temario del README
● PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> gitk
○ PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub>
```

Curso Git y GitHub: All files - gitk

File Edit View Help

main → remotes/origin/main Definición del temario del README

- README.md editado para probar SSH
- Editada la descripción del README.md
- Update README.md
- Corrección del README.md
- Añadido README.md
- v1.0** Conflicto solucionado
- Conflicto ramaConflicto
- Conflicto master
- Merge branch 'nuevaRama'
- Eliminación del último case
- Cambio de texto alert
- Mi primer commit

MARTIN GIL BLANCO <gilblancmarti> 2023-11-16 10:54:40

MARTIN GIL BLANCO <gilblancmarti>	2023-11-16 10:32:02
MARTIN GIL BLANCO <gilblancmarti>	2023-11-16 10:27:41
MARTIN GIL BLANCO <80607504+ml>	2023-11-15 15:26:21
MARTIN GIL BLANCO <80607504+ml>	2023-11-15 15:24:54
MARTIN GIL BLANCO <gilblancmarti>	2023-11-15 15:05:54
MARTIN GIL BLANCO <gilblancmarti>	2023-11-15 10:50:50
MARTIN GIL BLANCO <gilblancmarti>	2023-11-15 10:32:36
MARTIN GIL BLANCO <gilblancmarti>	2023-11-15 10:37:24
MARTIN GIL BLANCO <gilblancmarti>	2023-11-15 10:02:11
MARTIN GIL BLANCO <gilblancmarti>	2023-11-14 19:20:49
MARTIN GIL BLANCO <gilblancmarti>	2023-11-15 09:47:52
MARTIN GIL BLANCO <gilblancmarti>	2023-11-14 18:40:18

SHA1 ID: 4715ac50c8e2e8632893d8a1f3d1d45992a36aaa4 ← → Row | 1 / 13 |

Find ↓ ↑ commit containing: ↴

Search

Diff  Old version  New version Lines of context: 3  Ignore space changes

Author: MARTÍN GIL BLANCO <gilblancmartin@gmail.com> 2023-11-16 10  
Committer: MARTÍN GIL BLANCO <gilblancmartin@gmail.com> 2023-11-16  
Parent: [6b9acd4d0bda8e26e8563cffb3648563dfc18577](#) (README.md editado)  
Branches: [main](#), [remotes/origin/main](#)  
Follows: [v1.0](#)  
Precedes:

Definición del temario del README

index 76f04c3..f754025 100644 README.md

```
@@ -1,3 +1,11 @@
 # Curso de Git y GitHub con VScode

 ### En este curso vamos a describir todo el funcionamiento de Git j
```

Patch Tree

Comments

README.md

Existen diferentes comandos para ver información interesante de las ramas como los que se adjuntan a continuación.

**git branch:** Este comando se utiliza para listar, crear o borrar ramas en tu repositorio local. Al ejecutar simplemente `git branch`, obtendrás una lista de las ramas locales presentes en tu repositorio. Si deseas crear una nueva rama, puedes usar `git branch nombre_de_la_rama`. Además, puedes cambiar a una rama específica utilizando `git checkout nombre_de_la_rama` o la versión más reciente de Git utiliza `git switch nombre_de_la_rama`.

**git show-branch:** Este comando muestra la relación entre las ramas locales. Proporciona una visión general de cómo las diferentes ramas han divergido en términos de sus commits. Es útil para entender la historia del proyecto y la interacción entre las distintas líneas de desarrollo.

**git show-branch --all:** Al agregar `--all`, se incluirán también las ramas remotas en la visualización. Esto es útil cuando trabajas en proyectos colaborativos y quieres ver cómo se comparan las ramas locales y remotas.

A continuación creó una nueva rama llamada ramaDev y la subo al repositorio.

```
### En este curso vamos a describir todo el funcionamiento de Git junto con GitHub desde lo más básico a lo más complejo.
```

#### ## Temario

- **Conceptos básicos:** Se explica el funcionamiento de comandos de manejo básicos como git init status, add, commit, push, pull, merge, show, diff, entre otros
- **Ramas:** Explicación del funcionamiento de las ramas de Git.
- **Manejo de ramas y solución de conflictos:** Manejo de ramas y solución de conflictos entre ramas, además se explica el funcionamiento de las ramas remotas y locales al repositorio.
- **GitHub:** Configuración y creación de un repositorio desde cero.
- **Manejo de la seguridad de nuestro repositorio:** Configuración de SSH en el repositorio.
- **Uso de Tags y Alias:** Configuración y creación de tags para establecer la versión y alias para comandos.
- **Ramas remotas en GitHub:** Creación de ramas en local y manejo en remoto.

The screenshot shows a terminal window with several tabs at the top: BEMS, OUTPUT, DEBUG CONSOLE, and TERMINAL. The TERMINAL tab is active, displaying the following command-line session:

```
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git branch
* main
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git checkout -b ramaDev
Switched to a new branch 'ramaDev'
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git status
On branch ramaDev
nothing to commit, working tree clean
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git status
On branch ramaDev
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
    (use "git restore <file>..." to discard changes in working directory)
      modified: README.md

no changes added to commit (use "git add" and/or "git commit -a")
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git add .
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git commit -m "Añadido nuevos temas al README.md"
[ramaDev 445c129] Añadido nuevos temas al README.md
  1 file changed, 2 insertions(+)
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git push origin ramaDev
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 12 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 522 bytes | 522.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
remote:
remote: Create a pull request for 'ramaDev' on GitHub by visiting:
remote:     https://github.com/martin2745/cursoGit/pull/new/ramaDev
remote:
To github.com:martin2745/cursoGit.git
 * [new branch]      ramaDev > ramaDev
```

En la imagen anterior vemos cómo ejecuto los siguientes comandos:

```
git branch → Me permite ver las ramas existentes.  
git checkout -b ramaDev → Creamos y cambiamos de rama automáticamente.  
git add/git commit → Añadir y confirmar modificaciones.  
git show → Para ver las modificaciones.  
git push origin ramaDev → Me permite subir al repositorio la nueva rama con las modificaciones.
```

En este punto la ramaDev debería ser la rama donde se juntan las otras ramas de los desarrolladores de código, cuando se llega a una versión estable de código es el momento en que se hace el merge del código de la ramaDev con main.

Vamos a proceder a añadir a un nuevo colaborador a nuestro repositorio para simular un entorno de trabajo real. Para ello vamos a la zona de ajustes y colaboradores, desde ahí buscamos el usuario que queremos añadir para que pueda hacer modificaciones de código.

En el momento en que el usuario acepte nuestra invitación ya podrá actuar sobre nuestro código.

Vamos a suponer que este nuevo usuario ha desarrollado una nueva funcionalidad, este crea la pull request a revisar por el responsable del proyecto. El responsable verifica que la nueva funcionalidad o cambios son correctos y hace el merge desde la rama de la nueva funcionalidad a ramaDev. El resto de usuarios del proyecto, deben descargar estas modificaciones para poder integrarlas con las funcionalidades que están desarrollando en local. En la siguiente imagen puedo ver como el nuevo usuario ha creado la rama fix-name y ha realizado una modificación en el JS.

The screenshot shows a GitHub repository named 'cursoGit'. The 'fix-name' branch is selected. The commit history shows:

- macarracedo changed name (2069cc9, 14 hours ago) - 15 commits
- README.md (Añadido nuevos temas al README.md, 16 hours ago)
- codigo.js (changed name, 14 hours ago) - This commit is highlighted with a red underline.
- index.html (Mi primer commit, 2 days ago)

The 'Commit' view shows the diff for the 'codigo.js' file. The code changes are:

```

diff --git a/codigo.js b/codigo.js
@@ -8,7 +8,7 @@ if (isNaN(edad)) {
 8   8
 9   9   switch (true) {
10 10     case edad >= 0 && edad <= 12:
11 11     -     categoria = "neno";
12 12     +     categoria = "niño";
13 13     break;
14 14     case edad >= 13 && edad <= 18:

```

Como resultado de este proceso, el nuevo usuario genera la Pull Request con la finalidad de integrar los cambios de fix-name en ramaDev. Una Pull Request (PR) es una función clave en el sistema de control de versiones distribuido Git y en plataformas de alojamiento de proyectos como GitHub. La Pull Request facilita la colaboración entre desarrolladores al permitirles proponer cambios en un repositorio y solicitar que esos cambios se fusionen con la rama principal del proyecto. El flujo de funcionamiento de una Pull Request en GitHub:

- Clonar el Repositorio:** El desarrollador clona (descarga) su repositorio forked a su máquina local utilizando Git. Ahora, tienen una copia del proyecto en su propio entorno de desarrollo.
- Crear una Nueva Rama:** El desarrollador crea una nueva rama en su repositorio local para realizar los cambios. Esta rama es independiente de la rama principal y permite que el desarrollador trabaje en su característica o solución sin afectar la rama principal del proyecto.
- Realizar Cambios:** El desarrollador realiza los cambios necesarios en la nueva rama de su repositorio local.

4. **Enviar la Pull Request:** Una vez que los cambios están listos, el desarrollador envía una Pull Request al repositorio principal. Esto se hace a través de la interfaz web de GitHub. La Pull Request incluye detalles sobre los cambios realizados y proporciona una descripción del propósito de los cambios.
5. **Revisión del Código:** Los demás colaboradores del proyecto pueden revisar la Pull Request, realizar comentarios y discutir los cambios propuestos. La revisión facilita la colaboración y garantiza que los cambios cumplan los estándares del proyecto.
6. **Merge (Fusión):** Despues de la revisión y la aprobación, un mantenedor del proyecto puede fusionar la Pull Request en la rama principal. Esto incorpora los cambios propuestos al proyecto principal.

### Comparing changes

Choose two branches to see what's changed or to start a new pull request. If you need to, you can also [compare across forks](#) or [learn more about diff comparisons](#).

The screenshot shows a GitHub comparison interface. At the top, it says "base: ramaDev" and "compare: fix-name". A green checkmark indicates "Able to merge". Below this, there's a summary: "1 commit", "1 file changed", and "1 contributor". A list of commits on Nov 16, 2023, shows a single commit from "macarracedo" titled "changed name". The diff view shows the code for "codigo.js" with a change at line 11: "neno" is replaced by "ninho". The interface includes "Verified", "2069cc9", and "Create pull request" buttons.

The screenshot shows the GitHub pull request creation interface. At the top, it says "base: ramaDev" and "compare: fix-name". A green checkmark indicates "Able to merge". Below this, there's a title field containing "changed name" and a description field with the text "Se ha cambiado en el primer case del Switch la palabra \"neno\" por \"ninho\"." The interface includes "Write" and "Preview" tabs, rich text editing tools, and a "Create pull request" button.

Una vez creada la pull request se crea una apartado donde se puede mantener una conversación donde especificar cuestiones o modificaciones necesarias sobre el código. También se puede realizar el merge de los cambios y en caso de existir conflictos, estos han de solventarse antes de realizar el merge. Como se puede ver en la siguiente imagen Existe el botón de “Merge pull request” que permite integrar estas modificaciones en la rama ramaDev. Aunque existen tres opciones para realizar el merge como se indica en el desplegable, vamos a emplear la primera.

Add more commits by pushing to the [fix-name](#) branch on [martin2745/cursoGit](#).

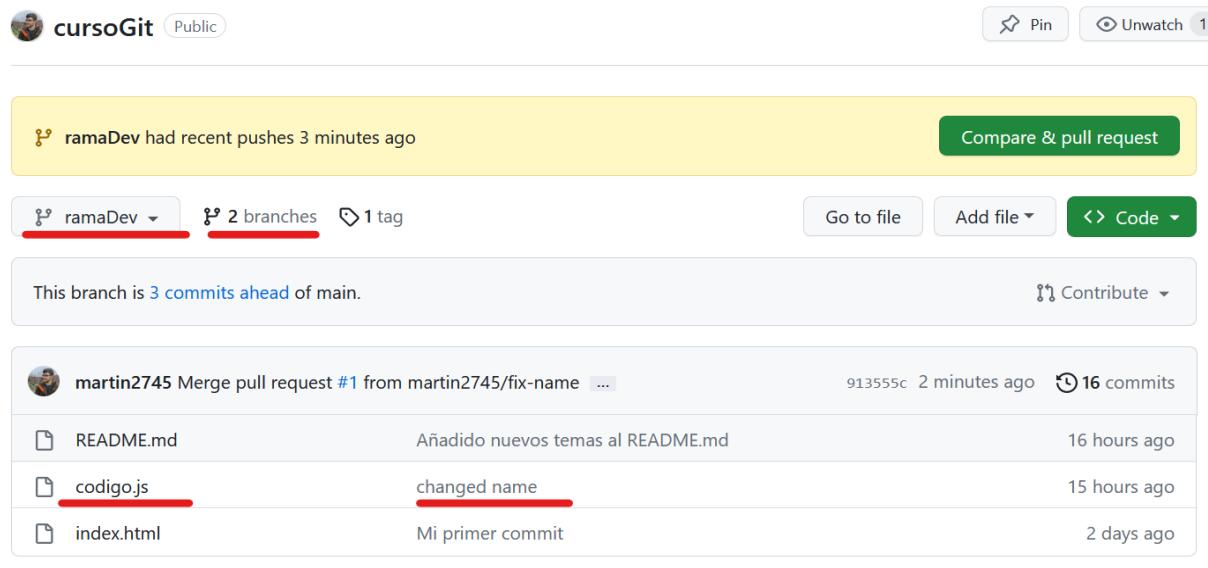
The screenshot shows a GitHub pull request interface. At the top, there's a message: "Add more commits by pushing to the [fix-name](#) branch on [martin2745/cursoGit](#)". Below this, there are two status messages: "Continuous integration has not been set up" (with a note about GitHub Actions) and "This branch has no conflicts with the base branch" (with a note about automatic merging). A prominent green button labeled "Merge pull request" is visible. A dropdown menu next to it lists three merge strategies: "Create a merge commit" (selected), "Squash and merge", and "Rebase and merge". To the right of the dropdown is a text editor area with standard rich text tools (bold, italic, etc.). At the bottom right are buttons for "Close pull request" and "Comment".

A continuación confirmaremos el merge y si lo deseamos podríamos eliminar la rama.

The screenshot shows a confirmation dialog box. It starts with the text "Merge pull request #1 from martin2745/fix-name" and "changed name". Below that is a note: "This commit will be authored by 80607504+martin2745@users.noreply.github.com". At the bottom are two buttons: "Confirm merge" (green) and "Cancel". In the background, there's a larger message box with a purple icon: "Pull request successfully merged and closed" followed by "You're all set—the [fix-name](#) branch can be safely deleted." There's also a "Delete branch" button.

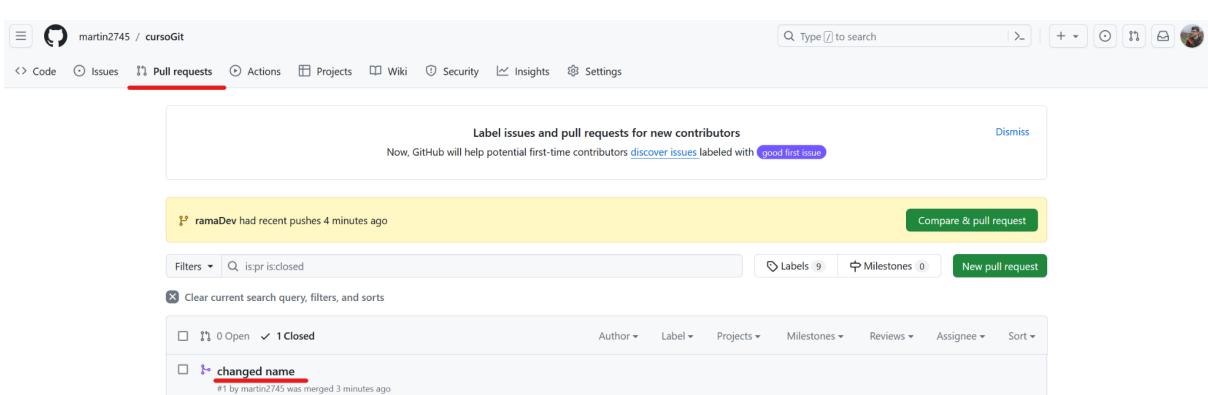
Una vez llegados hasta este punto los cambios se han integrado en la rama ramaDev y estos han de ser traídos del repositorio remoto al local de los usuarios que están trabajando en el proyecto.

A continuación vamos a ver que los cambios se han integrado en la ramaDev y también es posible ver dentro del apartado pull request aquellas PR creadas anteriormente o que todavía están sin resolver.



The screenshot shows a GitHub repository named 'cursoGit'. A yellow banner at the top indicates 'ramaDev had recent pushes 3 minutes ago'. Below it, a green button says 'Compare & pull request'. Underneath, there's a dropdown menu set to 'ramaDev' and a link to '2 branches'. A message states 'This branch is 3 commits ahead of main.' On the right, there are buttons for 'Go to file', 'Add file', and 'Code'. A red box highlights the 'ramaDev' dropdown. A message below says 'Contribute'.

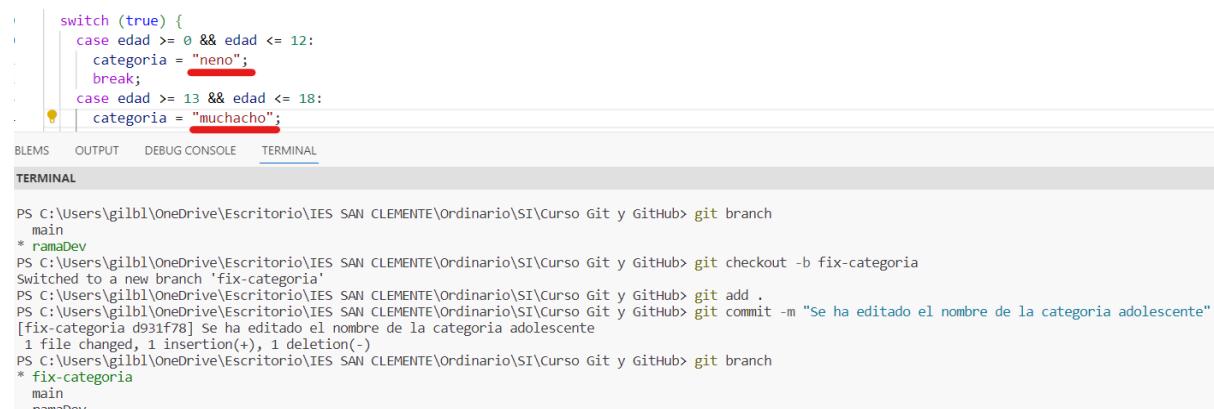
File	Commit Message	Time Ago
README.md	Añadido nuevos temas al README.md	16 hours ago
codigo.js	changed name	15 hours ago
index.html	Mi primer commit	2 days ago

The screenshot shows the 'Pull requests' tab in a GitHub repository. A yellow banner at the top indicates 'ramaDev had recent pushes 4 minutes ago'. Below it, a green button says 'Compare & pull request'. A search bar shows 'ispr is:closed'. A message encourages labeling issues and pull requests for new contributors. A red box highlights the 'ispr is:closed' filter. The list shows one merged pull request:

- #1 by martin2745 was merged 3 minutes ago

Supongamos ahora que estamos trabajando en otra funcionalidad diferente en local como qué queremos cambiar la categoría = "adolescente" por "muchacho" y ya hemos estado trabajando en estas modificaciones de código mientras se ha resuelto esta PR. Actualmente, el contenido de mi repositorio local está desactualizado por lo que tengo que actualizarlo antes de poder subir mi nueva funcionalidad.



The screenshot shows VSCode with a code editor and a terminal. The code editor has a file open with some JavaScript code. The terminal shows the following git commands:

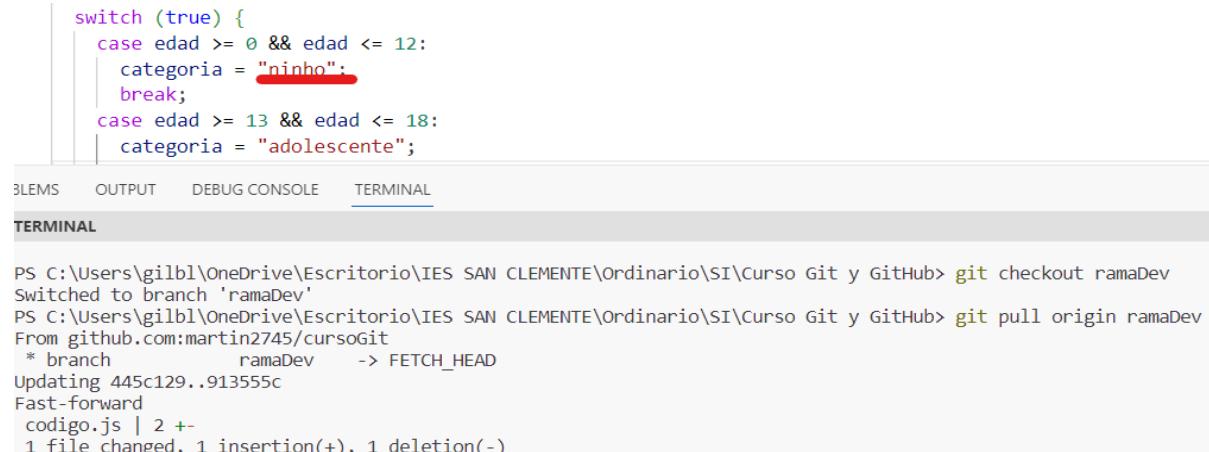
```

PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git branch
  main
* ramaDev
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git checkout -b fix-categoría
Switched to a new branch 'fix-categoría'
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git add .
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git commit -m "Se ha editado el nombre de la categoría adolescente"
[fix-categoría d931f78] Se ha editado el nombre de la categoría adolescente
  1 file changed, 1 insertion(+), 1 deletion(-)
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git branch
* fix-categoría
  main
    ramaDev

```

Para yo hacer este proceso es necesario:

- Situarnos en la rama ramaDev.
- Ejecutar el comando `git pull origin ramaDev` para poder actualizar el contenido local del proyecto.

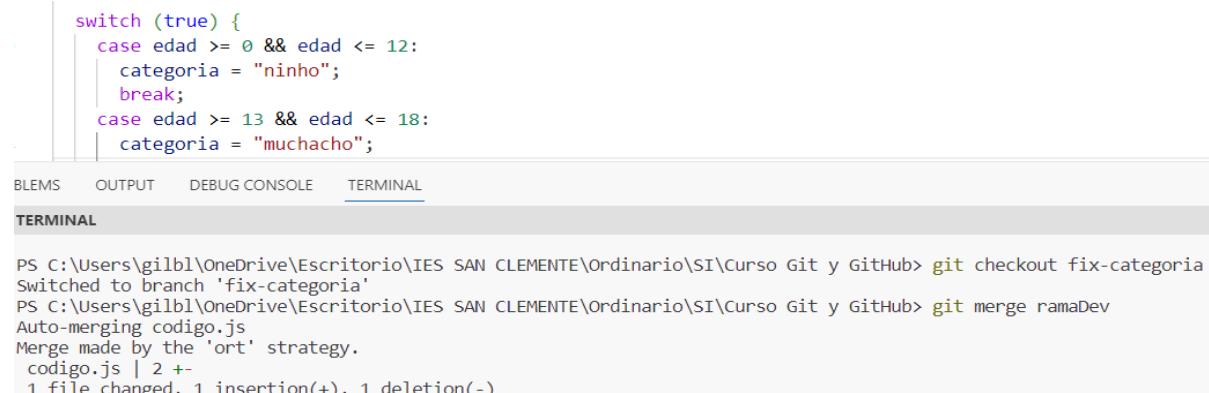


```
switch (true) {
  case edad >= 0 && edad <= 12:
    categoria = "ninho";
    break;
  case edad >= 13 && edad <= 18:
    categoria = "adolescente";
```

TERMINAL

```
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git checkout ramaDev
Switched to branch 'ramaDev'
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git pull origin ramaDev
From github.com:martin2745/cursoGit
 * branch            ramaDev      -> FETCH_HEAD
Updating 445c129..913555c
Fast-forward
  codigo.js | 2 ++
  1 file changed, 1 insertion(+), 1 deletion(-)
```

- Ejecutar el comando `git checkout otraRama` para situarnos en la rama de nuestra funcionalidad.
- Ejecutar el comando `git merge ramaDev` para actualizar el contenido de la rama donde estamos con el contenido de ramaDev realizando la nueva funcionalidad antes de subirla al repositorio remoto.



```
switch (true) {
  case edad >= 0 && edad <= 12:
    categoria = "ninho";
    break;
  case edad >= 13 && edad <= 18:
    categoria = "muchacho";
```

TERMINAL

```
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git checkout fix-categoría
Switched to branch 'fix-categoría'
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git merge ramaDev
Auto-merging codigo.js
Merge made by the 'ort' strategy.
  codigo.js | 2 ++
  1 file changed, 1 insertion(+), 1 deletion(-)
```

- En este momento podemos hacer un `git push origin fix-categoría` y subir el contenido de nuestra rama al repositorio y el proceso volverá a comenzar.



This branch is 5 commits ahead of main.

Commit	Message	Time	Commits
 martin2745 Merge branch 'ramaDev' into fix-categoría	f3aa301 2 minutes ago	18	
 README.md	Añadido nuevos temas al README.md	16 hours ago	
 codigo.js	Merge branch 'ramaDev' into fix-categoría	2 minutes ago	
 index.html	Mi primer commit	2 days ago	

# README y .gitignore

El archivo **README** es un documento de texto generalmente en formato Markdown o en texto plano que se encuentra en la raíz de un proyecto de software. Su propósito principal es proporcionar información sobre el proyecto, su propósito, cómo instalarlo, cómo usarlo y cualquier otra información relevante que los colaboradores o usuarios del proyecto necesiten saber.

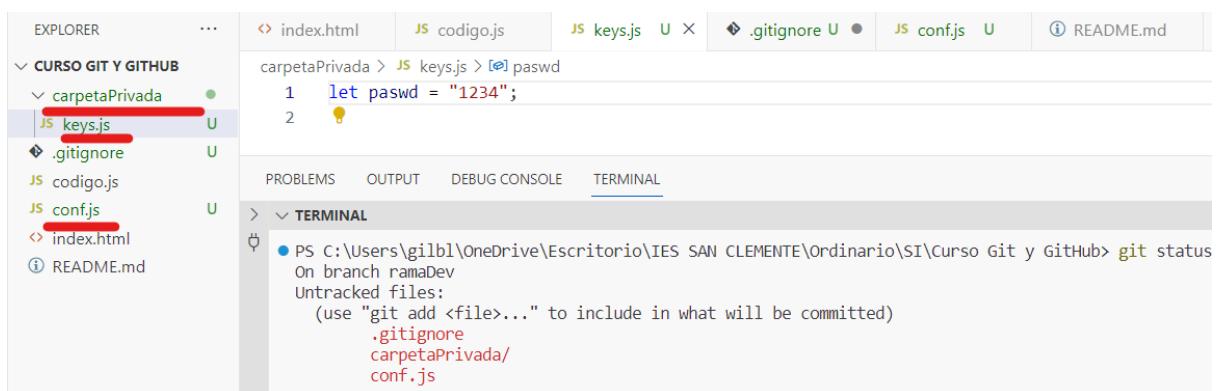
Un buen `README` es crucial para la documentación del proyecto y puede incluir secciones como:

- **Descripción del Proyecto:** Una breve descripción de lo que hace el proyecto.
  - **Instrucciones de Instalación:** Pasos detallados sobre cómo instalar y configurar el proyecto.
  - **Cómo Utilizar:** Información sobre cómo utilizar y ejecutar el proyecto.
  - **Estructura del Proyecto:** Una visión general de la estructura de carpetas y archivos del proyecto.
  - **Contribución:** Pautas para aquellos que deseen contribuir al proyecto.
  - **Licencia:** Información sobre la licencia bajo la cual se distribuye el proyecto.

El archivo `.gitignore` es utilizado para especificar archivos y carpetas que Git debe ignorar al rastrear cambios en un repositorio. Esto es útil para evitar que ciertos archivos generados automáticamente, archivos temporales, archivos de compilación y otros tipos de archivos no deseados sean incluidos en el control de versiones. Al definir reglas en el archivo `.gitignore`, puedes asegurarte de que estos archivos no se añadan accidentalmente al repositorio.

A continuación vamos a crear un `.gitignore` y ver su funcionamiento. En caso de que no exista un archivo `.gitignore` lo creamos, este archivo no tiene ningún tipo de extensión.

A mayores vamos a crear una un archivo conf.js y una carpeta que contiene un archivo llamado keys.js el cual no queremos que se comparta en nuestro repositorio.



Dentro del archivo `.gitignore` indicamos el contenido que no queremos que se adjunte al repositorio, el cual aparece en un color gris claro en el lado izquierdo una vez guardado. Si

ejecutamos un `git status` únicamente debería aparecer en rojo el archivo `.gitignore`.

```

EXPLORER            ...
CURSO GIT Y GITHUB
  carpetaprivada
    keys.js
    .gitignore
    codigo.js
    conf.js
  index.html
  README.md

PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL
> < TERMINAL
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git status
On branch ramaDev
Untracked files:
  (use "git add <file>..." to include in what will be committed)
    .gitignore

nothing added to commit but untracked files present (use "git add" to track)
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub>

```

Procedemos a subir nuestro nuevo contenido y podemos ver que en el repositorio remoto solo se ha añadido el archivo `.gitignore`.

	martin2745 Archivo .gitignore	5a397d7 1 minute ago	
	.gitignore	Archivo .gitignore	1 minute ago
	README.md	Añadido nuevos temas al README.md	17 hours ago
	codigo.js	Merge branch 'ramaDev' into fix-categoría	35 minutes ago
	index.html	Mi primer commit	3 days ago

Code	Blame	3 lines (3 loc) · 98 Bytes	Code 55% faster with GitHub Copilot	Raw	Copy	Download	Open
1 #Archivos y directorios que no queremos que se suban al repositorio remoto 2 /carpetaprivada 3 conf.js							

## MarkDown

Markdown es un lenguaje de marcado ligero diseñado para ser fácil de leer y fácil de escribir. Su sintaxis es bastante sencilla y se enfoca en ser legible incluso en su forma sin formato. Markdown se utiliza comúnmente para formatear texto en plataformas en línea como foros, wikis, y, entre otros, en archivos README de proyectos en GitHub.

En GitHub, los archivos README suelen estar escritos en Markdown, lo que facilita la creación de documentación legible y formateada para los proyectos. Además, GitHub proporciona funciones adicionales, como seguimiento de problemas, solicitudes de extracción y una interfaz de usuario intuitiva para gestionar proyectos de software.

La combinación de Markdown y GitHub proporciona una poderosa herramienta para la creación y gestión de contenido formateado de manera legible, facilitando la colaboración y el desarrollo de proyectos en línea.

Como extensiones para su manejo en VSCode se hace uso de las dos siguientes:



## markdownlint v0.52.0

David Anson [dlaa.me](#) | ⚡ 6,006,867 | ★★★★★ (73) | ❤ Sponsor

Markdown linting and style checking for Visual Studio Code

[Disable](#) [Uninstall](#) [Report Issue](#) [Sponsor](#)

This extension is enabled globally.

[DETAILS](#) [FEATURE CONTRIBUTIONS](#) [CHANGELOG](#) [RUNTIME STATUS](#)

## markdownlint

Markdown/CommonMark linting and style checking for Visual Studio Code

### Introduction

The [Markdown](#) markup language is designed to be easy to read, write, and understand. It succeeds - and its flexibility is both a benefit and a drawback. Many styles are possible, so formatting can be inconsistent. Some constructs don't work well in all parsers and should be avoided. For example, [here are some common/troublesome Markdown constructs](#).

[markdownlint](#) is an extension for the [Visual Studio Code editor](#) that includes a library of rules to encourage standards and consistency for Markdown files. It is powered by the [markdownlint library for Node.js](#) (which was inspired by [markdownlint for Ruby](#)). Linting is performed by the [markdownlint-cliv2 engine](#), which can be used in conjunction with this extension to provide command-line support for scripts and continuous integration scenarios. The [markdownlint-cliv2-action GitHub Action](#) uses the same engine and can be integrated with project workflows.



## Auto-Open Markdown Preview v0.0.4

hnw | ⚡ 445,998 | ★★★★★ (34)

Open Markdown preview automatically when opening a Markdown file

[Install](#) [Report Issue](#)

[DETAILS](#)

## Functionality

This VS Code extension automatically shows Markdown preview whenever you open new Markdown file. If you feel annoying to type "Ctrl+K V" or "⌘+K V" (preview side-by-side) many times, this extension helps you.

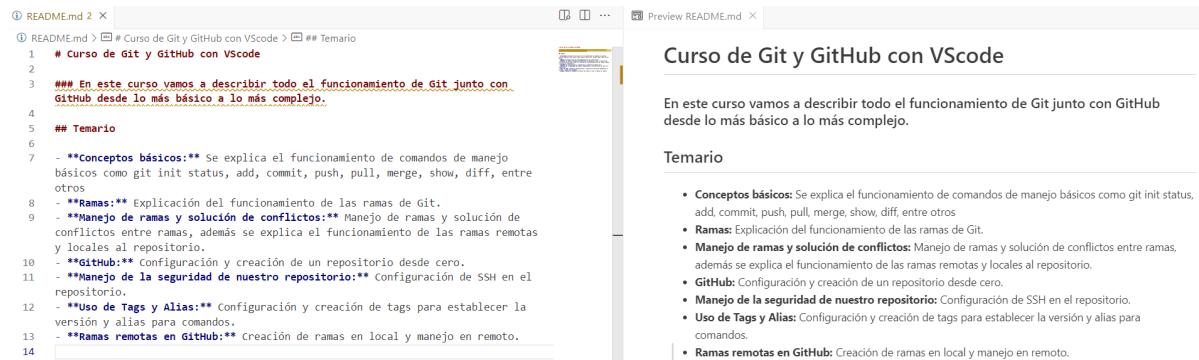
## Changelog

### version 0.0.4(2017/03/04)

- Support VSCode 1.10.0+

### version 0.0.3(2016/07/12)

- Support VSCode 1.3.0+



```

1 README.md 2 ×
① README.md > # Curso de Git y GitHub con Vscode > ## Temario
2
3 ## En este curso vamos a describir todo el funcionamiento de Git junto con GitHub desde lo más básico a lo más complejo.
4
5 ## Temario
6
7 - *Conceptos básicos: Se explica el funcionamiento de comandos de manejo básicos como git init status, add, commit, push, pull, merge, show, diff, entre otros.
8 - *Ramas: Explicación del funcionamiento de las ramas de Git.
9 - *Manejo de ramas y solución de conflictos: Manejo de ramas y solución de conflictos entre ramas, además se explica el funcionamiento de las ramas remotas y locales al repositorio.
10 - *GitHub: Configuración y creación de un repositorio desde cero.
11 - *Manejo de la seguridad de nuestro repositorio: Configuración de SSH en el repositorio.
12 - *Uso de Tags y Alias: Configuración y creación de tags para establecer la versión y alias para comandos.
13 - *Ramas remotas en GitHub: Creación de ramas en local y manejo en remoto.
14

```

# H1

## H2

### H3

#### H4

##### H5

###### H6

&lt;!--Itálicas--&gt;

Soy un texto normal

**\*\*Soy un texto Bold\*\***Soy un texto en itálica~~Texto tachado~~

\\*Para ver los caracteres especiales\\*

&lt;!--Dividir--&gt;

---

---

&lt;!--Links--&gt;

[Youtube] (<https://www.youtube.com/>)

&lt;!--Listas desordenadas--&gt;

- L1
- L2
  - L2.1

- L2.1.1
- L3

<!--Listas ordenadas-->

1. L1
2. L2
3. L3

<!--Código de programación-->

El texto de programación o comandos se añaden así `git branchh`

O así

```
```  
git branch nuevaRama  
git checkout nuevaRama  
```
```

Y para código de un lenguaje concreto así

```
```java  
public class Persona{  
    private string nombre;  
    private int edad;  
    ...  
}  
```
```

```
```php  
$miVariable = "Hola";  
echo("DEBUGGER");  
var_dump($miVariable);  
exit;  
```
```

<!--Imagenes-->

![Logo Git y GitHub] (.\gitGithub.png)

<!--Tablas-->

|               |    |
|---------------|----|
| Nombre   Edad |    |
| -----   ----  |    |
| Martín        | 23 |
| Manuel        | 22 |

# Git Rebasing

En Git, tanto "merge" como "rebase" son herramientas utilizadas para combinar ramas, pero lo hacen de manera diferente y tienen sus propias ventajas y desventajas. Aquí hay una explicación de cada uno:

## Merge:

- **Proceso:** Combina los cambios de dos ramas en un nuevo commit de fusión.
- **Ventajas:**
  - Preserva la historia original de ambas ramas.
  - Es más fácil de entender visualmente.
- **Desventajas:**
  - Puede generar commits de fusión adicionales, lo que puede resultar en una historia de Git más desordenada.
  - La línea de tiempo del proyecto puede volverse más complicada si hay muchas fusiones.
- **Ejemplo de uso:**

```
git checkout rama-destino
git merge rama-origen
```

## Rebase:

- **Proceso:** Reescribe el historial de la rama actual moviendo o aplicando los cambios de otra rama, commit por commit, sobre la punta de la rama actual. Cambia la base de la rama actual a otro commit.
- **Ventajas:**
  - Mantiene una línea de tiempo más lineal y limpia, sin commits de fusión.
- **Desventajas:**
  - Puede resultar en la pérdida de contexto si se rebasean commits ya compartidos y visibles por otros colaboradores.
  - No se recomienda rebase en ramas que se comparten públicamente.
- **Ejemplo de uso:**

```
git checkout rama-origen
git rebase rama-destino
```

Vamos a ver un proceso de generación de código que va a concluir con un `git rebase` y una subida de información a nuestro repositorio con la nueva información. El proceso va a ser el siguiente:

- Vamos a crear dos ramas:
  - Rama rebase hija de ramaDev.
  - Rama rebaseB hija de rebase.
- Vamos a hacer un commit en la rama rebase (este commit es posterior a la creación de la rama rebaseB).
- Vamos a crear un commit en la rama rebaseB.

```
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git branch
  main
  ramaDev
* rebase
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git branch rebaseB
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git branch
  main
  ramaDev
* rebase
  rebaseB
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git add .
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git commit -m "Edición rebase A"
[rebase 4be4e97] Edición rebase A
  1 file changed, 1 insertion(+), 1 deletion(-)
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git checkout rebaseB
Switched to branch 'rebaseB'
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git add .
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git commit -m "Edición rebase B 1"
[rebaseB df51284] Edición rebase B 1
  1 file changed, 1 insertion(+), 1 deletion(-)
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git add .
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git commit -m "Edición rebase B 2"
```

- Vamos a ver el histórico de commits de rebase.

```
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git log
commit 4be4e9744dcef57a93791568ce8e2d1ad697cf5a (HEAD -> rebase)
Author: MARTÍN GIL BLANCO <gilblancomartin@gmail.com>
Date:   Fri Nov 17 15:20:31 2023 +0100

  Edición rebase A

commit 5a397d7014fc6993215d35c9d3f60629a6595dbc (origin/ramaDev, ramaDev)
Author: MARTÍN GIL BLANCO <gilblancomartin@gmail.com>
Date:   Fri Nov 17 06:42:32 2023 +0100

  Archivo .gitignore
```

- Vamos a ver el histórico de commits de rebaseB.

```
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git log
commit 43bad44fc228767d19ce7b82c59a69b8500a972d (HEAD -> rebaseB)
Author: MARTÍN GIL BLANCO <gilblancomartin@gmail.com>
Date:   Fri Nov 17 15:21:36 2023 +0100

  Edición rebase B 2

commit df51284f5139fbe014304b424b706e809361d9a1
Author: MARTÍN GIL BLANCO <gilblancomartin@gmail.com>
Date:   Fri Nov 17 15:21:12 2023 +0100

  Edición rebase B 1

commit 5a397d7014fc6993215d35c9d3f60629a6595dbc (origin/ramaDev, ramaDev)
Author: MARTÍN GIL BLANCO <gilblancomartin@gmail.com>
Date:   Fri Nov 17 06:42:32 2023 +0100

  Archivo .gitignore
```

Estando en la rama rebase hacemos el `git rebase rebaseB` para integrar los commits de la rama rebaseB antes que el commit de la rama rebase. Esto es así por lógica de funcionamiento de rebase.

A continuación mostramos el histórico de commits de la rama rebase una vez ejecutado el comando.

```
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git rebase rebaseB
Successfully rebased and updated refs/heads/rebaseB.
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git log
commit ef9bab1a2111169a64627f660a07655961e2a063 (HEAD -> rebase)
Author: MARTÍN GIL BLANCO <gilblancmartin@gmail.com>
Date:   Fri Nov 17 15:20:31 2023 +0100
```

Edición rebase A

```
commit 43bad44fc228767d19ce7b82c59a69b8500a972d (rebaseB)
Author: MARTÍN GIL BLANCO <gilblancmartin@gmail.com>
Date:   Fri Nov 17 15:21:36 2023 +0100
```

Edición rebase B 2

```
commit df51284f5139fbe014304b424b706e809361d9a1
Author: MARTÍN GIL BLANCO <gilblancmartin@gmail.com>
Date:   Fri Nov 17 15:21:12 2023 +0100
```

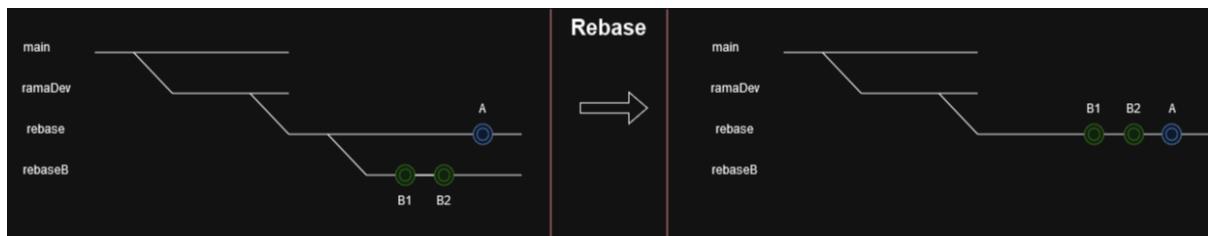
Edición rebase B 1

```
commit 5a397d7014fc6993215d35c9d3f60629a6595dbc (origin/ramaDev, ramaDev)
Author: MARTÍN GIL BLANCO <gilblancmartin@gmail.com>
Date:   Fri Nov 17 06:42:32 2023 +0100
```

Archivo .gitignore

```
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git log --all --graph --decorate --oneline
* ef9bab1 (HEAD -> rebase) Edición rebase A
* 43bad44 (rebaseB) Edición rebase B 2
* df51284 Edición rebase B 1
* 5a397d7 (origin/ramaDev, ramaDev) Archivo .gitignore
```

El escenario que se ha producido es el siguiente.



Como explicación podemos decir que creamos la rama rebase y en ella ejecutamos un commit posterior a la creación de la rama rebaseB. Al no tener el commit A la rama rebaseB, al hacer el rebase se juntan todos los commits en orden en una única rama.

Esto es muy positivo para no tener un flujo complejo al tener varias ramas y sus respectivos commits de mergeo pero falsea lo que ha ocurrido de verdad en el histórico de confirmaciones.

|  |  |         |  |  |  |
|--|--|---------|--|--|--|
| Merge pull request #3 from martin2745/rebase ... |  | 424f749 |  |  |  |
| martin2745 committed 39 minutes ago              |  |         |  |  |  |
| Edición rebase A                                 |  |         |  |  |  |
| martin2745 committed 42 minutes ago              |  |         |  |  |  |
| Edición rebase B 2                               |  |         |  |  |  |
| martin2745 committed 45 minutes ago              |  |         |  |  |  |
| Edición rebase B 1                               |  |         |  |  |  |
| martin2745 committed 45 minutes ago              |  |         |  |  |  |

## Git Stash

Cuando estás trabajando en una rama y necesitas cambiar rápidamente a otra rama, pero tienes cambios sin confirmar en tu rama actual, Git Stash es útil. Stash te permite guardar temporalmente tus cambios sin confirmar en una pila (stash), de modo que puedes cambiar de rama sin tener que comprometer esos cambios. Como comandos principales tenemos:

```
git stash save "Mensaje descriptivo" → Guardar cambios en el stash
git stash list → Listar stashes
git stash apply stash@{0} → Aplicar cambios del stash
git stash drop stash@{0} → Eliminar un stash específico
```

Vamos a editar el archivo `codigo.js` en una nueva rama (`ramaStash`) y guardar las modificaciones en el espacio de stash. Esto nos va a permitir cambiar a la rama `ramaDev`.

```
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git status
On branch ramaStash
Last command done (1 command done):
pick 1bf5a77 ERROR B
No commands remaining.
You are currently editing a commit while rebasing branch 'B' on '9e4c0aa'.
  (use "git commit --amend" to amend the current commit)
  (use "git rebase --continue" once you are satisfied with your changes)

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    modified:   codigo.js

PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git stash save "Guardo el mensaje de alert"
Saved working directory and index state On ramaStash: Guardo el mensaje de alert
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git stash list
stash@{0}: On ramaStash: Guardo el mensaje de alert

PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git checkout ramaDev
Switched to branch 'ramaDev'
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git pull origin ramaDev
From github.com:martin2745/cursoGit
 * branch            ramaDev    -> FETCH_HEAD
Already up to date.
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git checkout ramaStash
Switched to branch 'ramaStash'
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git status
On branch ramaStash
Last command done (1 command done):
  pick 1bf5a77 ERROR B
No commands remaining.
You are currently editing a commit while rebasing branch 'B' on '9e4c0aa'.
  (use "git commit --amend" to amend the current commit)
  (use "git rebase --continue" once you are satisfied with your changes)

nothing to commit, working tree clean
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git stash apply
On branch ramaStash
Last command done (1 command done):
  pick 1bf5a77 ERROR B
No commands remaining.
You are currently editing a commit while rebasing branch 'B' on '9e4c0aa'.
  (use "git commit --amend" to amend the current commit)
  (use "git rebase --continue" once you are satisfied with your changes)

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
    modified:   codigo.js

no changes added to commit (use "git add" and/or "git commit -a")
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git status
On branch ramaStash
Last command done (1 command done):
  pick 1bf5a77 ERROR B
No commands remaining.
You are currently editing a commit while rebasing branch 'B' on '9e4c0aa'.
  (use "git commit --amend" to amend the current commit)
  (use "git rebase --continue" once you are satisfied with your changes)

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
    modified:   codigo.js

no changes added to commit (use "git add" and/or "git commit -a")
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git stash list
stash@{0}: On ramaStash: Guardo el mensaje de alert
```

Es decir, con lo anterior puedo guardar en el stash una determinada información y cambiar de rama, una vez que ya pueda volver a mi rama puedo volver a aplicar las modificaciones almacenadas en el stash sobre mi staging area y de este modo volver al punto donde me encontraba en un primer momento.

```
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git stash drop
Dropped refs/stash@{0} (74e74d8022ae839fdf60ad1e40907b9be035af8)
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git stash list
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git status
On branch ramastash
Last command done (1 command done):
  pick 1bf5a77 ERROR B
No commands remaining.
You are currently editing a commit while rebasing branch 'B' on '9e4c0aa'.
  (use "git commit --amend" to amend the current commit)
  (use "git rebase --continue" once you are satisfied with your changes)

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
    modified:   codigo.js

no changes added to commit (use "git add" and/or "git commit -a")
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git stash save "Guardo el mensaje de alert"
Saved working directory and index state On ramastash: Guardo el mensaje de alert
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git stash list
stash@{0}: On ramastash: Guardo el mensaje de alert
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git status
On branch ramastash
Last command done (1 command done):
  pick 1bf5a77 ERROR B
No commands remaining.
You are currently editing a commit while rebasing branch 'B' on '9e4c0aa'.
  (use "git commit --amend" to amend the current commit)
  (use "git rebase --continue" once you are satisfied with your changes)

nothing to commit, working tree clean
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git stash pop
On branch ramastash
Last command done (1 command done):
  pick 1bf5a77 ERROR B
No commands remaining.
You are currently editing a commit while rebasing branch 'B' on '9e4c0aa'.
  (use "git commit --amend" to amend the current commit)
  (use "git rebase --continue" once you are satisfied with your changes)

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
    modified:   codigo.js

no changes added to commit (use "git add" and/or "git commit -a")
Dropped refs/stash@{0} (526a3a8c1f21ec6f115c3a19139ec6e65bd2c977)
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git stash list
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> █
```

También puedo eliminar el stash e incluso los comandos `git stash apply` y `git stash drop` se pueden simplificar en un único comando que sea `git stash pop`. Además de esto, mencionar que puedo tener varias modificaciones del stash y se almacenan como si fuera una pila.

```
if (isNaN(edad)) {
  alert(
    "Por favor, introduce un número válido para la edad. Editado en rama rebaseB"
  );
} else {
  let categoria;

  switch (true) {
    case edad >= 0 && edad <= 12:
      categoria = "ninho";
      break;
    case edad >= 13 && edad <= 18:
      categoria = "muchacho";
      break;
  }
}

BLEMS OUTPUT DEBUG CONSOLE TERMINAL
TERMINAL
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git stash list
stash@{0}: On ramastash: Guardo categoria como muchachito
stash@{1}: On ramastash: Guardo el mensaje de alert
```

```

if (isNaN(edad)) {
    alert(
        "Por favor, introduce un número válido para la edad. Editado en rama rebaseB"
    );
} else {
    let categoria;

    switch (true) {
        case edad >= 0 && edad <= 12:
            categoria = "ninho";
            break;
        case edad >= 13 && edad <= 18:
            categoria = "muchachito";
            break;
    }
}

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
TERMINAL

PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git stash pop
On branch ramastash
Last command done (1 command done):
pick 1bf5a77 ERROR B
No commands remaining.
You are currently editing a commit while rebasing branch 'B' on '9e4c0aa'.
(use "git commit --amend" to amend the current commit)
(use "git rebase --continue" once you are satisfied with your changes)

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
    modified:   codigo.js

no changes added to commit (use "git add" and/or "git commit -a")
Dropped refs/stash@{0} (12990321cf5f69cf5d88cdb698237d15d0601008)
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git stash list
stash@{0}: On ramastash: Guardo el mensaje de alert

```

## Git Amend

El comando `git commit --amend` es una herramienta poderosa en Git que permite realizar modificaciones en el commit más reciente. Este comando combina los cambios realizados en el área de trabajo con el commit anterior, creando así un nuevo commit. Es útil cuando se necesita ajustar el mensaje del commit, agregar archivos olvidados o corregir errores antes de compartir los cambios con otros.

Cuando ejecutas `git commit --amend`, se abre un editor de texto que te permite modificar el mensaje del commit anterior. Además, puedes añadir o quitar archivos del commit, así como realizar cambios en los archivos existentes. Una vez que guardas y cierras el editor, Git crea un nuevo commit con los cambios realizados y reemplaza el commit anterior.

Voy a crear un nuevo commit el cual voy a suponer que es incorrecto ya que tengo acciones sin terminar en el commit. En lugar de crear un nuevo commit con las modificaciones y que quede un registro en el historial de confirmaciones de ello, voy a hacer uso del comando `git commit --amend` para solucionar el problema.

```

PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git commit -m "Commit de amend"
[ramaDev d1818d7] Commit de amend
 1 file changed, 1 insertion(+), 1 deletion(-)
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git status
On branch ramaDev
Last command done (1 command done):
  pick 1bf5a77 ERROR B
No commands remaining.
You are currently editing a commit while rebasing branch 'B' on '9e4c0aa'.
  (use "git commit --amend" to amend the current commit)
  (use "git rebase --continue" once you are satisfied with your changes)

nothing to commit, working tree clean
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git show
commit d1818d769a3ec1753413a9b8968f35f2fb1e7232 (HEAD -> ramaDev)
Author: MARTÍN GIL BLANCO <gilblancmartin@gmail.com>
Date:   Fri Nov 17 18:23:34 2023 +0100

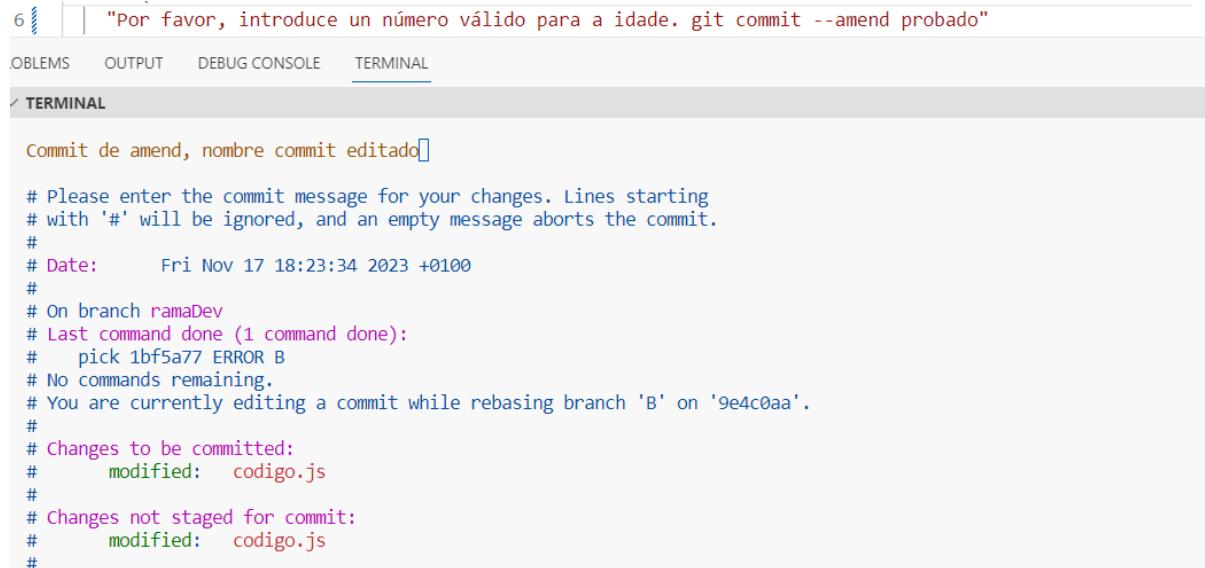
    Commit de amend

diff --git a/codigo.js b/codigo.js
index 709aae4..fb23297 100644
--- a/codigo.js
+++ b/codigo.js
@@ -3,7 +3,7 @@ edad = parseInt(edad);

if (isNaN(edad)) {
  alert(
-   "Por favor, introduce un número válido para la edad. Editado en rama rebaseB"
+   "Por favor, introduce un número válido para la edad. Borro el mensaje para probar amend"
  );
} else {
  let categoria;

```

Una vez somos conscientes del error, hacemos las modificaciones necesarias, ejecutamos el comando `git add .` y el comando `git commit --amend`. A continuación vemos un editor vi donde podemos editar el nombre del commit y ver qué archivos se van a añadir al commit.



The screenshot shows the VS Code interface with the terminal tab selected. The terminal window displays the following text:

```

6 | "Por favor, introduce un número válido para la edad. git commit --amend probado"
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
TERMINAL
Commit de amend, nombre commit editado

# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
#
# Date:      Fri Nov 17 18:23:34 2023 +0100
#
# On branch ramaDev
# Last command done (1 command done):
#   pick 1bf5a77 ERROR B
# No commands remaining.
# You are currently editing a commit while rebasing branch 'B' on '9e4c0aa'.
#
# Changes to be committed:
#   modified:  codigo.js
#
# Changes not staged for commit:
#   modified:  codigo.js
#

```

Una vez confirmadas las modificaciones podemos ver que el último commit ha sido modificado.



The screenshot shows the VS Code interface with the terminal tab selected. The terminal window displays the following command-line session:

```

git commit --amend probado
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git status
On branch ramaDev
Last command done (1 command done):
  pick 1bf5a77 ERROR B
No commands remaining.
You are currently editing a commit while rebasing branch 'B' on '9e4c0aa'.
  (use "git commit --amend" to amend the current commit)
  (use "git rebase --continue" once you are satisfied with your changes)

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
    modified:   codigo.js

no changes added to commit (use "git add" and/or "git commit -a")
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git commit --amend
[ramaDev 4a00e71] Commit de amend, nombre commit editado
  Date: Fri Nov 17 18:23:34 2023 +0100
  1 file changed, 1 insertion(+), 1 deletion(-)
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git log
commit 4a00e7139d58853f8ffd23903974d43faeff38c4 (HEAD -> ramaDev)
Author: MARTÍN GIL BLANCO <gilblancomartin@gmail.com>
Date:   Fri Nov 17 18:23:34 2023 +0100

  Commit de amend, nombre commit editado

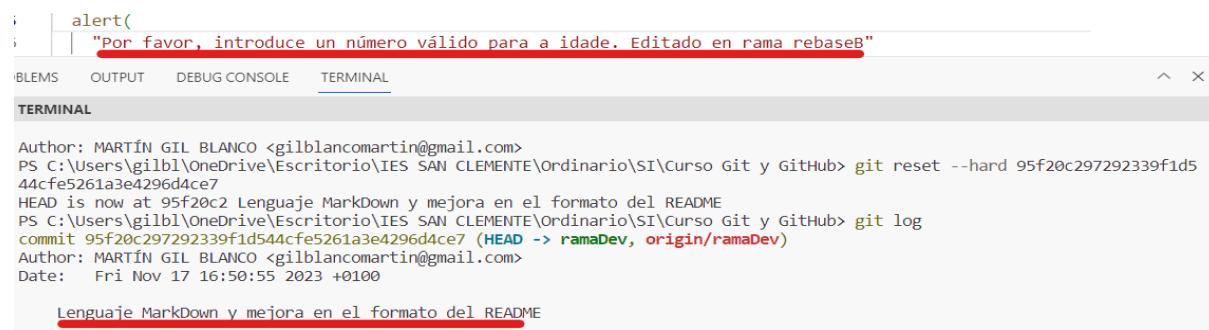
```

## Git Reset

`git reset` es un comando fundamental en Git que se utiliza para deshacer cambios en el árbol de trabajo y en el índice. Puede ser empleado de varias maneras, y una de las opciones comunes es `git reset HEAD <archivo>` que quita los cambios del archivo del área de preparación (index), pero mantiene los cambios en el árbol de trabajo. Si se utiliza con la opción `--hard`, `git reset` puede ser más potente, ya que restablece tanto el índice como el árbol de trabajo al commit especificado, eliminando permanentemente los cambios realizados después de ese commit. Esto es útil cuando se desea "reiniciar" el proyecto a un estado anterior.

Supongamos que el último commit que hicimos para probar el comando `git amend` no nos interesa y queremos volver al estado anterior. Para ello vamos a emplear el comando:

```
git reset --hard HASH_COMMIT_DESTINO
```



The screenshot shows the VS Code interface with the terminal tab selected. The terminal window displays the following command-line session:

```

alert()
Por favor, introduce un número válido para la idade. Editado en rama rebaseB
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git reset --hard 95f20c297292339f1d544cf5261a3e4296d4ce7
HEAD is now at 95f20c2 Lenguaje Markdown y mejora en el formato del README
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git log
commit 95f20c297292339f1d544cf5261a3e4296d4ce7 (HEAD -> ramaDev, origin/ramaDev)
Author: MARTÍN GIL BLANCO <gilblancomartin@gmail.com>
Date:   Fri Nov 17 16:50:55 2023 +0100

  Lenguaje Markdown y mejora en el formato del README

```

## Git Grep

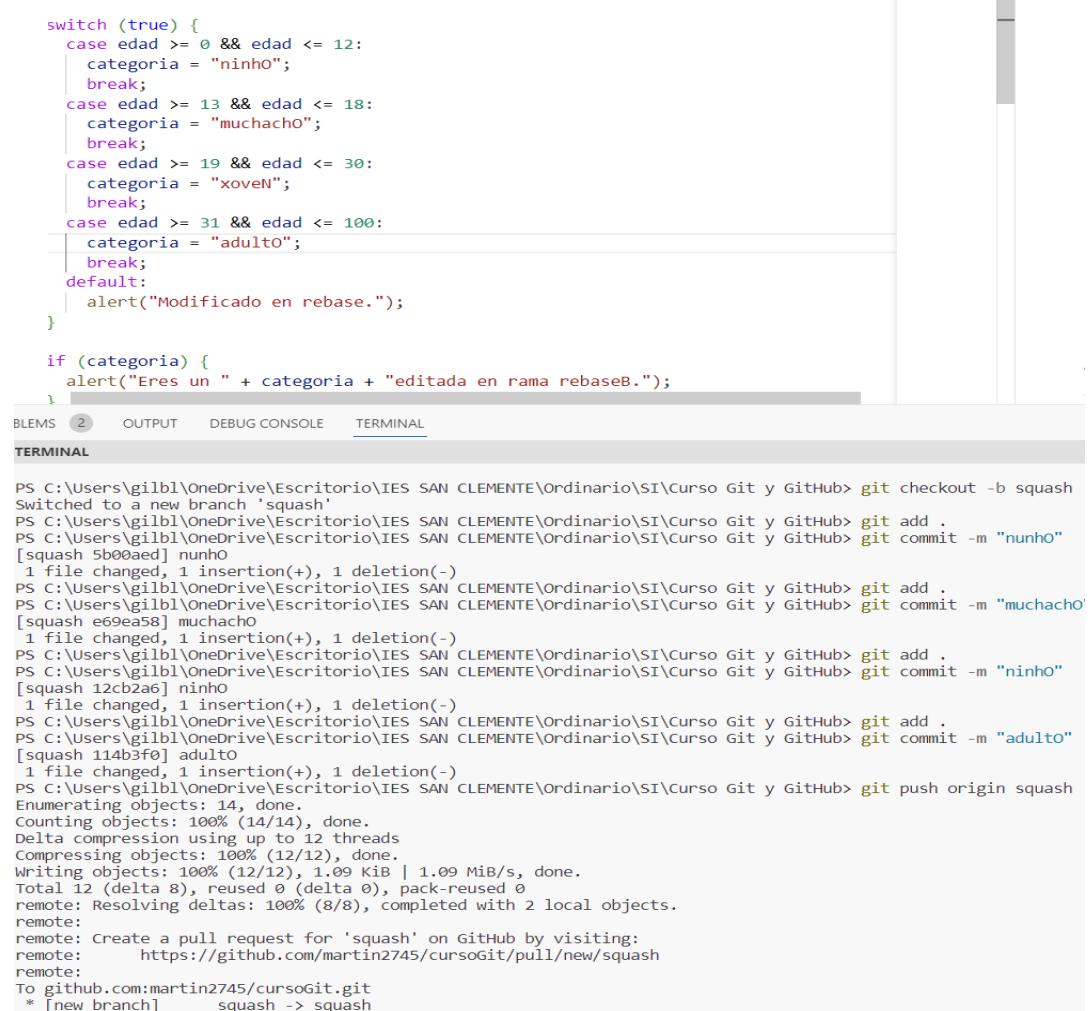
El comando `git grep` se utiliza para buscar patrones de texto en los archivos de tu repositorio. La sintaxis básica es la siguiente `git grep patron_busqueda`

```
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git grep -n let
codigo.js:1:let edad = prompt("Por favor, introduce a túa idade:");
codigo.js:9: let categoria;
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git log -S "Readme"
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git log -S "README"
commit 95f20c297292339f1d544cfe5261a3e4296d4ce7 (HEAD -> ramaDev, origin/ramaDev)
Author: MARTÍN GIL BLANCO <gilblancomartin@gmail.com>
Date:   Fri Nov 17 16:50:55 2023 +0100
```

Lenguaje MarkDown y mejora en el formato del README

## Git Squash

`git squash` es una técnica que permite combinar múltiples commits relacionados en un solo commit antes de enviarlos al repositorio remoto, con el objetivo de mantener un historial de cambios más limpio y comprensible. Utilizando comandos como `git rebase`, esta técnica reduce la cantidad de commits en el historial, facilitando la revisión y comprensión del progreso del proyecto.



The screenshot shows the VS Code interface. The code editor has the following code:

```
switch (true) {
  case edad >= 0 && edad <= 12:
    categoria = "ninho";
    break;
  case edad >= 13 && edad <= 18:
    categoria = "muchacho";
    break;
  case edad >= 19 && edad <= 30:
    categoria = "xoven";
    break;
  case edad >= 31 && edad <= 100:
    categoria = "adulto";
    break;
  default:
    alert("Modificado en rebase.");
}

if (categoria) {
  alert("Eres un " + categoria + " editada en rama rebaseB.");
}
```

The terminal window shows the following command sequence:

```
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git checkout -b squash
Switched to a new branch 'squash'
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git add .
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git commit -m "nunho"
[squash 5b00aed] nunho
 1 file changed, 1 insertion(+), 1 deletion(-)
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git add .
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git commit -m "muchacho"
[squash e69ea58] muchacho
 1 file changed, 1 insertion(+), 1 deletion(-)
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git add .
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git commit -m "ninho"
[squash 12cb2a6] ninho
 1 file changed, 1 insertion(+), 1 deletion(-)
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git add .
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git commit -m "adulto"
[squash 114b3f0] adulto
 1 file changed, 1 insertion(+), 1 deletion(-)
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git push origin squash
Enumerating objects: 14, done.
Counting objects: 100% (14/14), done.
Delta compression using up to 12 threads
Compressing objects: 100% (12/12), done.
Writing objects: 100% (12/12), 1.09 KiB | 1.09 MiB/s, done.
Total 12 (delta 8), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (8/8), completed with 2 local objects.
remote:
remote: Create a pull request for 'squash' on GitHub by visiting:
remote:     https://github.com/martin2745/cursoGit/pull/new/squash
remote:
To github.com:martin2745/cursoGit.git
 * [new branch]      squash -> squash
```

Lo suyo sería juntar todos estos commits -en unos solo para que nuestro árbol de confirmaciones sea más legible. Para ello tenemos que realizar el siguiente comando:

```
git rebase -i HEAD~4
```

Con ello lo que conseguimos es juntar los 4 commits desde la posición del HEAD tal y como se nos indica en la siguiente documentación.

```

pick 5b00aed nunho
s e69ea58 muchacho
s 12cb2a6 ninho
s 114b3f0 adulto

# Rebase 0b61e0c..114b3f0 onto 0b61e0c (4 commands)
#
# Commands:
# p, pick <commit> = use commit
# r, reword <commit> = use commit, but edit the commit message
# e, edit <commit> = use commit, but stop for amending
# s, squash <commit> = use commit, but meld into previous commit
# f, fixup [-C | -c] <commit> = like "squash" but keep only the previous
# commit's log message, unless -C is used, in which case
# keep only this commit's message; -c is same as -C but
# opens the editor
# x, exec <command> = run command (the rest of the line) using shell
# b, break = stop here (continue rebase later with 'git rebase --continue')
# d, drop <commit> = remove commit
# l, label <label> = label current HEAD with a name
# t, reset <label> = reset HEAD to a label
# m, merge [-C <commit> | -c <commit>] <label> [<oneline>]
# .       create a merge commit using the original merge commit's
# .       message (or the oneline, if no original merge commit was

# This is a combination of 4 commits.
# This is the 1st commit message:

nunho

# This is the commit message #2:

muchacho

# This is the commit message #3:

ninho

# This is the commit message #4:

adulto

# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
#
# Date:      Sat Nov 18 19:59:00 2023 +0100
#
# interactive rebase in progress; onto 0b61e0c
# Last commands done (4 commands done):
#   squash 12cb2a6 ninho
#   squash 114b3f0 adulto
# No commands remaining.
# You are currently rebasing branch 'squash' on '0b61e0c'.
#
# Changes to be committed:
#   modified:  codigo.js

PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\Curso Git y GitHub> git rebase -i HEAD~4
[detached HEAD 60b3a30] nunho
Date: Sat Nov 18 19:59:00 2023 +0100
1 file changed, 4 insertions(+), 4 deletions(-)
Successfully rebased and updated refs/heads/squash.

```

#### Commit

```

nunho
muchacho
ninho
adulto

# squash
martin2745 committed 3 minutes ago

1 parent 0b61e0c commit 60b3a30

Showing 1 changed file with 4 additions and 4 deletions.

```

```

diff --git a/codigo.js b/codigo.js
@@ -10,16 +10,16 @@ if (!isNah(edad)) {
    switch (true) {
      case edad == 0 && edad <= 12:
        categoria = "ninho";
        break;
      case edad >= 13 && edad <= 18:
        categoria = "muchacho";
        break;
      case edad >= 19 && edad <= 30:
        categoria = "xoven";
        break;
      case edad >= 31 && edad <= 100:
        categoria = "adulto";
        break;
      default:
        alert("Modificado en rebase.");
    }
}

```

# Issues y Milestones

Los "issues" de GitHub son un componente fundamental para la colaboración efectiva en proyectos de desarrollo de software. Estos actúan como puntos de entrada centralizados para discusiones, seguimiento de tareas y reportes de problemas dentro de un repositorio. Cada issue se inicia con un título descriptivo que resume el problema o la tarea, seguido de una sección de descripción que proporciona detalles adicionales. Esta descripción puede incluir pasos para reproducir un error, especificaciones para una nueva característica o cualquier información relevante para comprender y abordar el asunto en cuestión.

Los issues no solo sirven como un medio de comunicación estructurado, sino que también ofrecen características adicionales. Pueden etiquetarse para categorización, asignarse a miembros específicos del equipo para indicar responsabilidades, y se pueden vincular a otros issues o pull requests para establecer relaciones contextuales. Los comentarios permiten discusiones continuas y pueden incluir actualizaciones sobre el progreso de la resolución. Además, los issues están integrados con los commits, lo que significa que pueden cerrarse automáticamente mediante commits específicos, proporcionando una forma eficiente de rastrear el estado de una tarea o problema a lo largo del tiempo. En conjunto, los issues en GitHub constituyen una herramienta valiosa para la organización, seguimiento y resolución colaborativa de problemas en el desarrollo de software. Podemos crear un nuevo Issue en nuestro proyecto en la segunda opción, entre code y pull request.

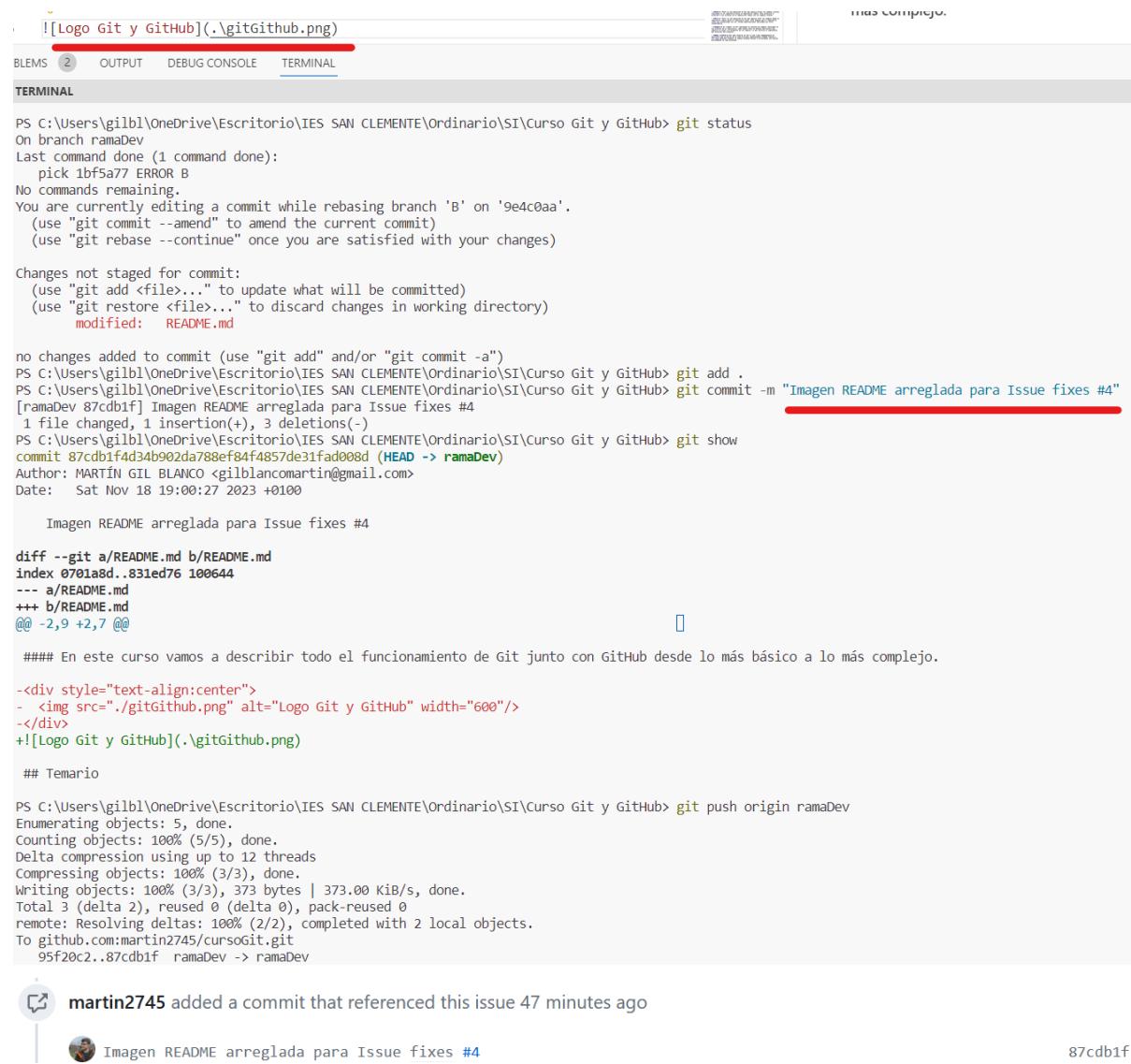
The screenshot shows the GitHub interface for creating a new issue. At the top, there's a placeholder for a profile picture and a button labeled 'Add a title'. Below it is a text input field with the placeholder 'Cambiar el tamaño de la imagen del README.md'. Underneath is a section titled 'Add a description' with two tabs: 'Write' (selected) and 'Preview'. The 'Write' tab contains a rich text editor toolbar with various icons for bold, italic, underline, etc. Below the toolbar is a text area containing the Markdown text: 'La imagen del README.md no tiene el tamaño correcto, sería necesario editar el archivo MarkDown para corregirlo.' At the bottom left of the text area, there's a note: 'Styling with Markdown is supported'. On the right side of the text area is a green 'Submit new issue' button.

## Cambiar el tamaño de la imagen del README.md #4

[Open](#) martin2745 opened this issue now · 0 comments

The screenshot shows the comment section for issue #4. A user named 'martin2745' has commented 'commented now'. The comment text is: 'La imagen del README.md no tiene el tamaño correcto, sería necesario editar el archivo MarkDown para corregirlo.' There are three small circular icons at the bottom left of the comment box: a smiley face, a reply arrow, and a link icon. At the top right of the comment box, there are 'Owner' and '...' buttons.

Una cuestión muy interesante de los issues de GitHub es la posibilidad de cerrar un issue automáticamente referenciando a un commit concreto como veremos a continuación. Fíjemonos en que el Issue tiene el identificador #4 por lo que podemos hacer una modificación de nuestro código y escribir un commit terminado en **fixes #4** como identificador de esta incidencia. También hay que mencionar que en este momento se cerrará automáticamente el Issue y se indicará el identificador del commit que cerró el problema.



```

! [Logo Git y GitHub](./github.png) más complejo.
BLEMS 2 OUTPUT DEBUG CONSOLE TERMINAL
TERMINAL
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\curso Git y GitHub> git status
On branch ramaDev
Last command done (1 command done):
  pick 1bf5a77 ERROR B
No commands remaining.
You are currently editing a commit while rebasing branch 'B' on '9e4c0aa'.
  (use "git commit --amend" to amend the current commit)
  (use "git rebase --continue" once you are satisfied with your changes)

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
    modified: README.md

no changes added to commit (use "git add" and/or "git commit -a")
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\curso Git y GitHub> git add .
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\curso Git y GitHub> git commit -m "Imagen README arreglada para Issue fixes #4"
[ramaDev 87cdb1f] Imagen README arreglada para Issue fixes #4
  1 file changed, 1 insertion(+), 3 deletions(-)
PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\curso Git y GitHub> git show
commit 87cdb1f4d34b902da788ef84f4857de31fad008d (HEAD -> ramaDev)
Author: MARTÍN GIL BLANCO <gilblancmartin@gmail.com>
Date:   Sat Nov 18 19:00:27 2023 +0100

 Imagen README arreglada para Issue fixes #4

diff --git a/README.md b/README.md
index 0701a8d..831ed76 100644
--- a/README.md
+++ b/README.md
@@ -2,9 +2,7 @@
#### En este curso vamos a describir todo el funcionamiento de Git junto con GitHub desde lo más básico a lo más complejo.

<div style="text-align:center">
- 
- </div>
+! [Logo Git y GitHub](./github.png)

## Temario

PS C:\Users\gilbl\OneDrive\Escritorio\IES SAN CLEMENTE\Ordinario\SI\curso Git y GitHub> git push origin ramaDev
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 12 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 373 bytes | 373.00 KiB/s, done.
Total 3 (delta 2), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
To github.com:martin2745/cursoGit.git
  95f20c2..87cdb1f ramaDev -> ramaDev

```

 martin2745 added a commit that referenced this issue 47 minutes ago

 Imagen README arreglada para Issue fixes #4 87cdb1f

En GitHub, los "labels" (etiquetas) son una forma de categorizar y organizar los "issues" (problemas) en un repositorio. Los labels son pequeñas etiquetas con colores asociados que se pueden aplicar a los issues para indicar información adicional sobre ellos. Estos labels son personalizables y permiten a los colaboradores del proyecto clasificar y filtrar los issues de manera más efectiva.

The screenshot shows the GitHub 'Labels' page for a repository. At the top, there are tabs for 'Labels' (selected), 'Milestones', and a search bar. A green 'New label' button is in the top right. Below, a table lists 9 labels:

| 9 labels                      |  | Sort ▾                                |
|-------------------------------|--|---------------------------------------|
| <code>bug</code>              | Something isn't working                    | <code>Edit</code> <code>Delete</code> |
| <code>documentation</code>    | Improvements or additions to documentation | <code>Edit</code> <code>Delete</code> |
| <code>duplicate</code>        | This issue or pull request already exists  | <code>Edit</code> <code>Delete</code> |
| <code>enhancement</code>      | New feature or request                     | <code>Edit</code> <code>Delete</code> |
| <code>good first issue</code> | Good for newcomers                         | <code>Edit</code> <code>Delete</code> |
| <code>help wanted</code>      | Extra attention is needed                  | <code>Edit</code> <code>Delete</code> |
| <code>invalid</code>          | This doesn't seem right                    | <code>Edit</code> <code>Delete</code> |
| <code>question</code>         | Further information is requested           | <code>Edit</code> <code>Delete</code> |
| <code>wontfix</code>          | This will not be worked on                 | <code>Edit</code> <code>Delete</code> |

## [DevTools Bug]: hook parsing fails while using functional component + redux or alternative #27724

The screenshot shows a GitHub issue thread for issue #27724. A comment from user `zeroxx1986` is highlighted, dated yesterday. The comment includes:

- Website or app:** <https://github.com/reduxjs/redux-template/tree/master/packages/cra-template-redux-typescript>
- Repro steps:**

Before the step-by-step approach, please note, this is not specific to the cra-template-redux-typescript repo mentioned above, nor to redux itself, as it can be easily reproduced by integrating redux (or even other alternative, like zustand) to any react-based project where you use functional components together with hooks.

  - set up a bare template repo: `npx create-react-app my-app --template redux-typescript`
  - `cd my-app`
  - `npm start`
  - Observe the `Counter` component in devtools -> notice hook parsing failed.
  - go to `src/features/counter/counter.tsx` and stub out the two lines which wire redux into the component (easier to stub out than to delete them and all reference in the JSX):

```
// const count = useAppSelector(selectCount); // stubbed out
// const dispatch = useAppDispatch(); // stubbed out
const count = 1;
const dispatch = (x:any) => null;
```

  - save & reload (or let HMR do it)
  - Observe the `Counter` component in devtools -> notice hook parsing works properly.

On the right side of the comment, there are sections for Assignees (No one assigned), Labels (Component: Developer Tools, Status: Unconfirmed, Type: Bug), Projects (None yet), Milestone (No milestone), Development (No branches or pull requests), Notifications (Customize, Subscribe, You're not receiving notifications from this thread.), and 1 participant (with a profile picture).

Los "milestones" (hitos) en GitHub son una forma de organizar y dar seguimiento al progreso de un conjunto específico de issues o pull requests en un repositorio. Los milestones son útiles para agrupar tareas relacionadas y establecer objetivos a nivel de proyecto. Aquí hay algunas características clave sobre los milestones en GitHub.

Desde GitHub podemos crear un nuevo Milestone para una fecha determinada y asociar Issues a un Milestone concreto. Según se vayan completando varios Issues la barra de progreso del Milestone se irá incrementando hasta completarse por completo.

The screenshot shows the GitHub interface for managing milestones. At the top, there are tabs for 'Labels' and 'Milestones', with 'Milestones' being the active tab. A green button labeled 'New milestone' is visible in the top right. Below the tabs, there are filters for 'Open' (1) and 'Closed' (0) issues. The main section displays a single milestone titled 'Milestone de prueba'. It includes a progress bar indicating '50% complete' with '1 open' and '1 closed' issues. There are links to 'Edit', 'Close', and 'Delete' the milestone. Below the milestone title, a note says 'Juntamos los dos Issues para una prueba de creación de un milestone'.

También se pueden asociar commits a Issues y entre varios Issues a través de comentarios creados por los usuarios.

This screenshot shows a GitHub issue thread. On the left, a commit from 'martin2745' is shown with the message 'Cambiar el tamaño de la imagen del README.md #4'. A note below it says 'La imagen del README.md no tiene el tamaño correcto, sería necesario editar el archiv...'. In the center, a comment from 'martin2745' is displayed with the message 'Este Issue se solucionó en el Issue #4'. On the right, a reply from the same user says 'You are assigned to and opened this issue'. The bottom right corner of the screenshot has buttons for 'Owner', 'Author', and '...'. The entire screenshot is framed by a light blue border.

## Wiki

La función de wiki en GitHub permite a los usuarios colaborar en la creación y mantenimiento de documentación asociada a un repositorio. Una wiki es una colección de páginas web editables que contienen información sobre el proyecto. Estas páginas pueden incluir instrucciones de instalación, guías de contribución, información sobre la arquitectura del software, tutoriales y cualquier otro contenido relevante para el proyecto.

The screenshot shows the GitHub repository wiki page for 'cursoGit'. The top navigation bar includes 'Code', 'Issues', 'Pull requests', 'Actions', 'Projects', 'Wiki' (which is highlighted in red), 'Security', 'Insights', and 'Settings'. A search bar at the top right says 'Type [ ] to search'. The main content area is titled 'Wiki del repositorio' and shows a recent edit by 'MARTÍN GIL BLANCO' made 7 minutes ago with 2 revisions. Below the title, there's a section titled 'Explicación de que es un wiki' containing the text: 'La función de wiki en GitHub permite a los usuarios colaborar en la creación y mantenimiento de documentación asociada a un repositorio. Una wiki es una colección de páginas web editables que contienen información sobre el proyecto. Estas páginas pueden incluir instrucciones de instalación, guías de contribución, información sobre la arquitectura del software, tutoriales y cualquier otro contenido relevante para el proyecto.' There are two bullet points at the bottom: '• Git' and '• GitHub'. To the right of the main content, there's a sidebar titled 'Pages' with a list of pages: 'Home del repositorio' and 'Wiki del repositorio', with the latter having a sub-section titled 'Explicación de que es un wiki'.

# Configura tu perfil de GitHub

A continuación se adjunta un README de presentación de un perfil en GitHub. Para poder crear esta presentación del perfil es necesario crear un repositorio público con el mismo nombre que el usuario de GitHub con un README y adjuntar el siguiente código editado.

```
### ¡Hola a todos! Mi nombre es X 🙋

### Estudios:

- Estudio 1
- Estudio 2
- Estudio 3

### Lenguajes y Herramientas que utilizo:

**Generales:**  

<br />







<br />

### Contactame!:

[][linkedin]
<br />

[linkedin]: https://www.linkedin.com/in/tu-linkedin
```