Contenido

[1.1. Crear cuenta GitHub – Crear un repositorio remoto 2](#_Toc175688389)

[1.2. Herramientas y Funcionalidades de GitHub (Servidor en remoto) 3](#_Toc175688390)

[1.2.1. Repositorio de Ejemplo de Git en Entornos de Desarrollo: 3](#_Toc175688391)

[1.2.2. Fork, Star, Pin y Watch 3](#_Toc175688392)

[1.2.3. Repositorios y Commits 5](#_Toc175688393)

[1.2.4. Ramas y Pull Requests 6](#_Toc175688394)

[2. CONEXIÓN Y MANEJO DE GIT DE LOCAL A REMOTO 9](#_Toc175688395)

[2.1. Crear una copia local del repositorio remoto 12](#_Toc175688396)

[2.2. Trabajar con ramas (branches) 15](#_Toc175688397)

[2.2.1. Subir la rama al repositorio remoto 16](#_Toc175688398)

[2.2.2. Provocando un conflicto 16](#_Toc175688399)

[2.2.3. Resolver el conflicto 17](#_Toc175688400)

[2.3. Trabajar con etiquetas (tags) 19](#_Toc175688401)

1. GIT EN REMOTO - GITHUB

Para poder colaborar en cualquier proyecto Git, necesitas saber cómo gestionar repositorios remotos. Los repositorios remotos son versiones de tu proyecto que están hospedadas en Internet o en cualquier otra red. Los repositorios remotos van a permitir compartir desde o hacia nuestro repositorio. Las ubicaciones remotas son generalmente servidores locales, una máquina de un equipo de trabajo o bien un almacén de repositorios en la nube como ***GitLab*** o ***GitHub***.

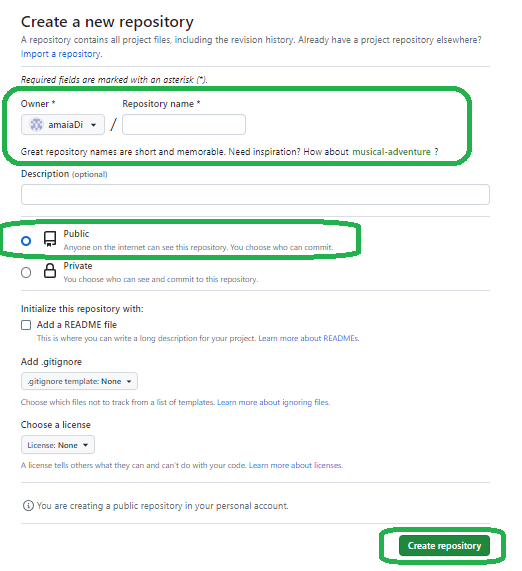


# Crear cuenta GitHub – Crear un repositorio remoto

En este apartado crearemos una cuenta en [GitHub](https://github.com/join). Seleccionaremos un plan Free que se deberá verificar por email.

Lo primero que haremos será crear un repositorio en la cuenta que acabamos de crear. Este ***repositorio*** puede contener todo tipo de archivos, y se suele utilizar para albergar **un solo proyecto**.

Para ***crear el repositorio*** hay que buscar la opción para crear el repositorio en la ventana de GitHub. Le asignamos un nombre y lo marcamos como ***public*** para que sea accesible para todo el mundo. En este caso al repositorio se le ha asignado el nombre de ***gitrepositorio***, y la cuenta es ***arama2020***.



# Herramientas y Funcionalidades de GitHub (Servidor en remoto)

### Repositorio de Ejemplo de Git en Entornos de Desarrollo:

Existen repositorios utilizados con frecuencia en cursos o formaciones sobre Git y desarrollo colaborativo. Estos repositorios pueden ayudar a entender mejor cómo se hace un manejo completo de Git en proyectos de gran tamaño con equipos de trabajo grandes.

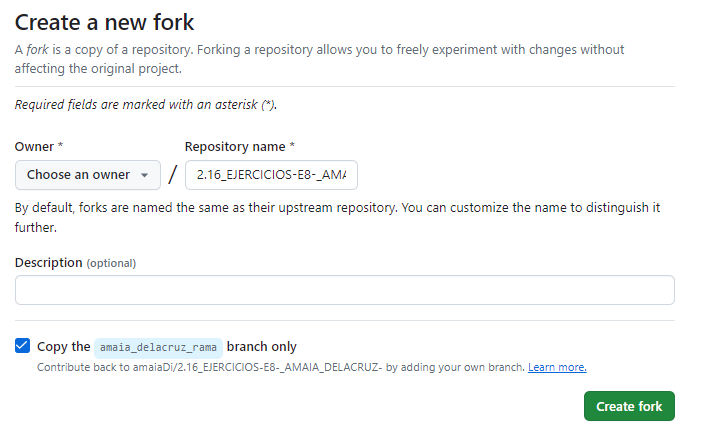
**GitHub Training Kit**: Este repositorio proporciona una serie de ejemplos y prácticas para aprender Git y GitHub. Incluye guías paso a paso y ejemplos prácticos de colaboración utilizando ramas, pull requests y otros aspectos fundamentales de Git.

* Repositorio: [github/training-kit](https://github.com/github/training-kit)

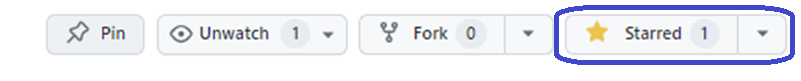
### Fork, Star, Pin y Watch

* **Fork:** **Crear una copia de un repositorio en tu cuenta de GitHub. Permite trabajar en proyectos sin afectar el repositorio original. Es útil para contribuir a proyectos open source o proyectos que son de otros.**

****

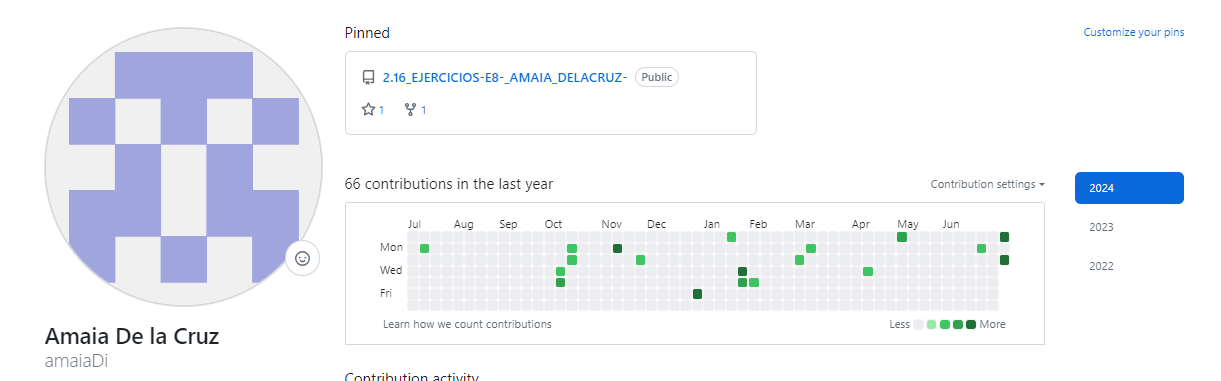
****

* **Star:** **Marcar un repositorio como favorito. Ayuda a los usuarios a encontrar y seguir proyectos interesantes o útiles.**

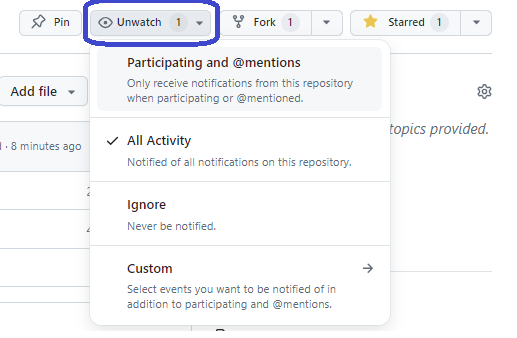
****

* **Pin: Sirve para fijar repositorios en tu perfil de GitHub. Muestra repositorios específicos en la parte superior de tu perfil para destacar tus proyectos más importantes o recientes.**

****

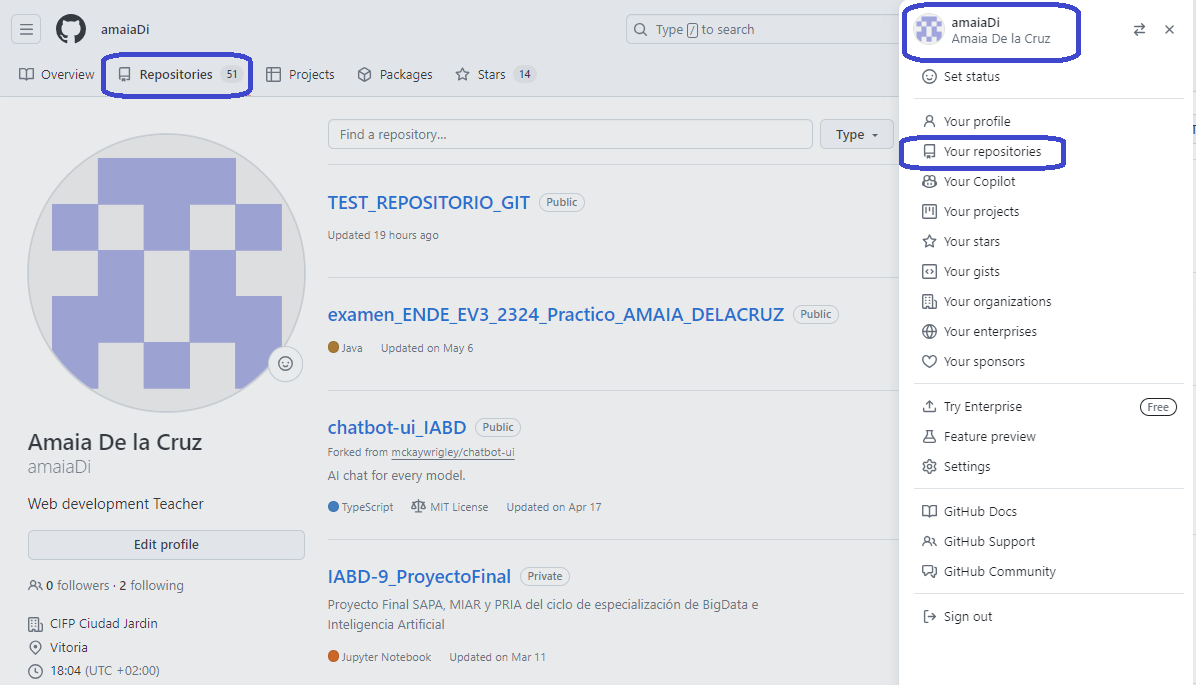
****

* **Watch: Seguir actualizaciones de un repositorio externo. Recibir notificaciones sobre nuevas actividades, como commits, issues, pull requests y releases.**

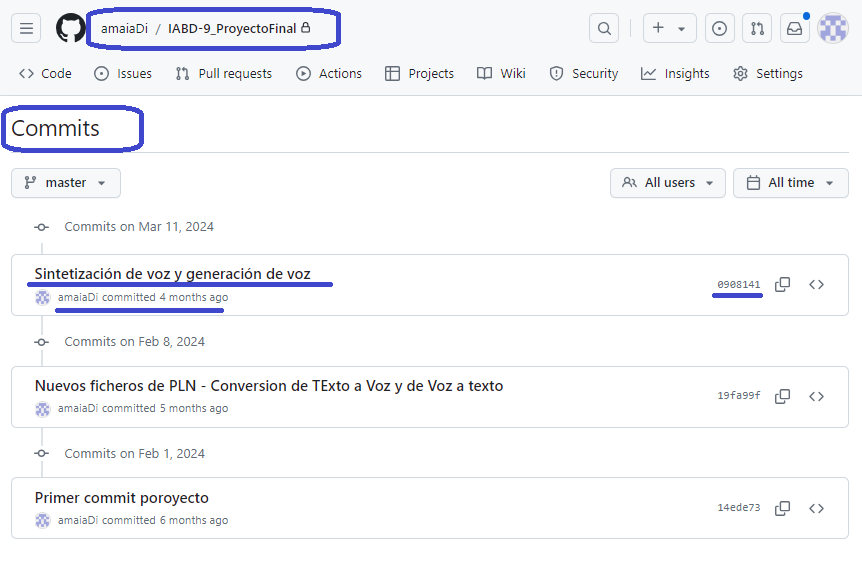
****

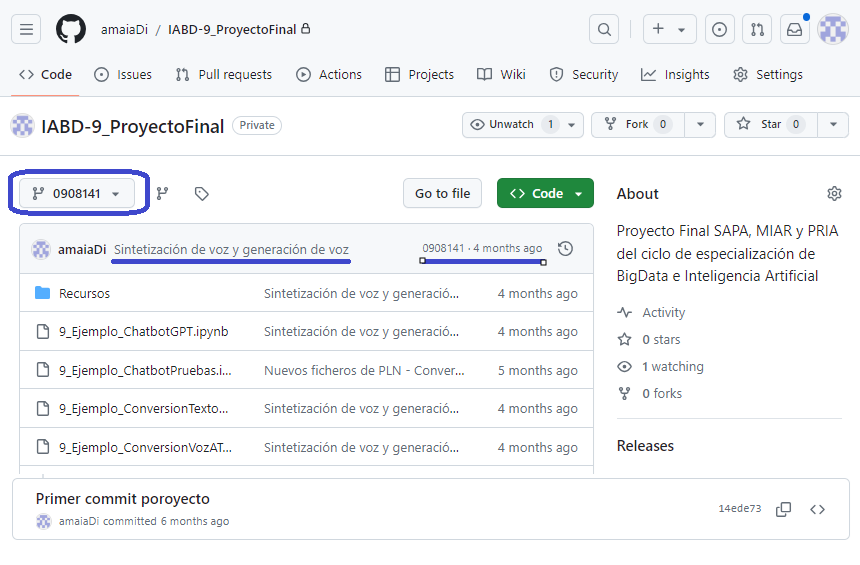
### Repositorios y Commits

* **Repositorio:** Es el lugar donde se almacena todo tu proyecto, incluyendo el código fuente, documentación, archivos de configuración y el historial de cambios.



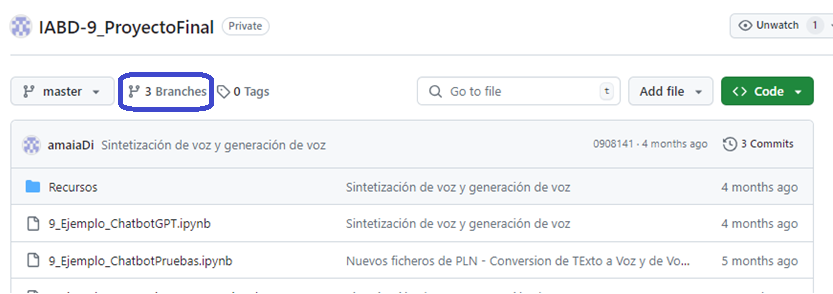
* **Commits:** Cada commit es como una instantánea del proyecto en un punto específico del tiempo. Es importante escribir mensajes descriptivos para cada commit para que otros puedan entender qué cambios se hicieron y por qué.

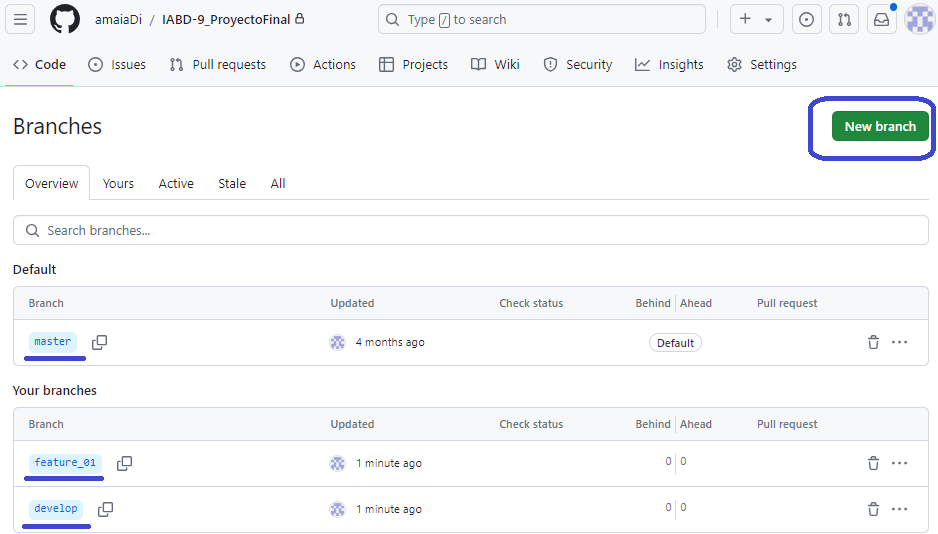




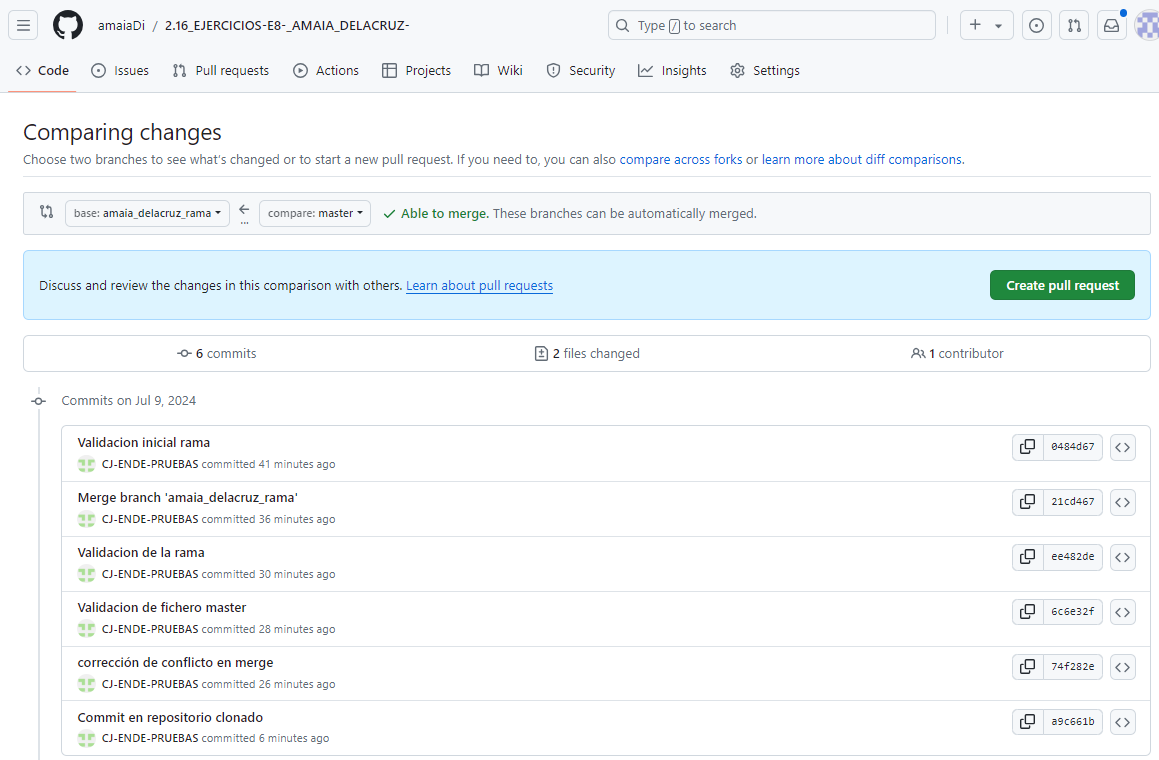
### Ramas y Pull Requests

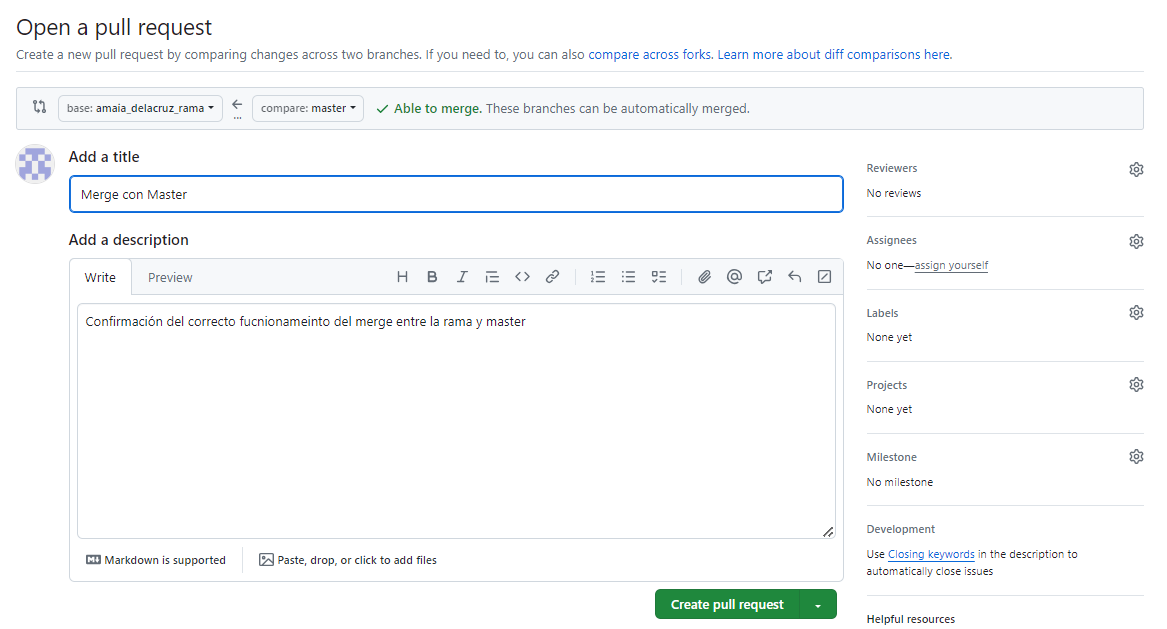
* **Ramas:** Las ramas principales en base al gitflow son master y develop. Se trata de ramas sobre las que no se debería hacer pruebas. Para ello, se usan ramas para trabajar en nuevas características o corregir errores sin afectar la rama principal (main o master y develop). Esto permite a cada miembro del equipo trabajar en paralelo sin conflictos.

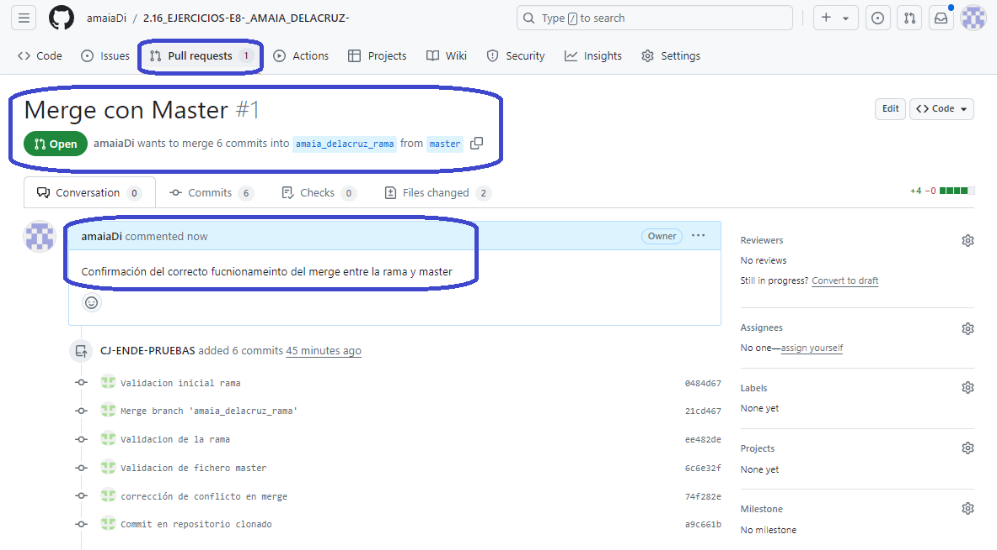


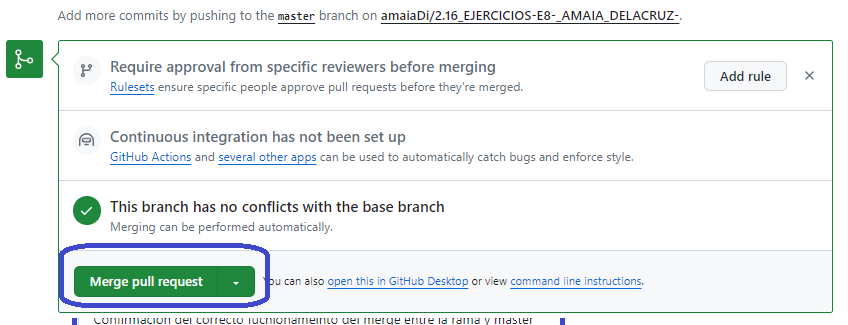


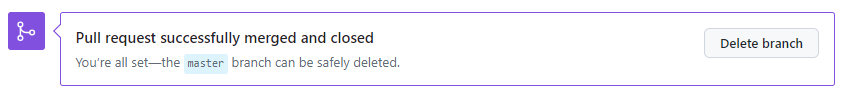
* **Pull Requests:** Una vez que termines de trabajar en tu rama, abre una pull request para fusionar tus cambios con la rama principal. Esto permite que otros miembros del equipo revisen tu código y discutan los cambios antes de que sean integrados.











# CONEXIÓN Y MANEJO DE GIT DE LOCAL A REMOTO

* **Configurar usuario de GitHub en local**

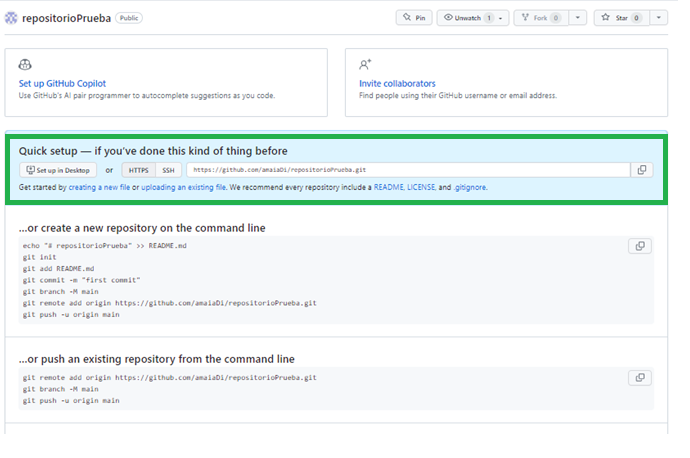
Usaremos un nuevo usuario para copiar el repositorio remoto al repositorio local, añadir un fichero nuevo y validar los cambios en el remoto.

**$ git config –-global user.name “Usu2 Name”**

* **Conectar con el Repositorio Remoto desde local**

Una vez creado el repositorio remoto en GitHub, la herramienta nos indica cuál es la URL del repositorio y cuáles son los pasos para conectarse a él.

**$ git remote add origin https://github.com/*nombre\_cuenta*/*nombre\_repositorio***



* **Verificar los remotos configurados**

Después de añadir un remoto, es una buena práctica verificar que se ha añadido correctamente con el comando **git remote -v .**Esto listará todos los remotos configurados para tu repositorio junto con sus URLs.

**$ git remote -v**

* **Flujos de Trabajo con Repositorios Remotos**

Con los comandos **git push** subiremos los commit del repositorio local al remoto, y con **git fetch y git pull** descargaremos los commits del repositorio remoto al local. La diferencia radica en que **git fetch** descarga los datos del repositorio remoto, como los commits, las referencias (branches y tags), y las actualizaciones. Sin embargo, no fusiona automáticamente estos cambios en la rama local en la que estás trabajando. El comando **git pull** es una combinación de **git fetch** y **git merge**. Este comando se utiliza para obtener los cambios del repositorio remoto y fusionarlos automáticamente con tu rama local.

Al trabajar con repositorios remotos, el repositorio local se identifica como ***master*** y el remoto como ***origin/master***.

Creamos en local una carpeta nueva con nombre r***epositorio3***, abrimos ***Git Bash***, y creamos el repositorio Git con ***git init***. Añadimos dos archivos y validamos dos veces, así tendremos dos commits. Visualizamos el log para ver cómo nos queda.

**$ git init**

**$ echo “Fichero repo1” > repo1.txt**

**$ echo “Fichero repo2” > repo2.txt**

**$ git add .**

**$ git commit –m “Creo repo1 y repo2 paso1”**

**$ echo “Nuevo dato repo1” >> repo1.txt**

**$ echo “Nuevo dato repo2” >> repo2.txt**

**$ git add .**

**$ git commit –m “Nueva linea en repo1 y repo2 paso2”**

**$ git status**

**$ ls –la**

**$ git log –-oneline**

81fad39 (HEAD -> master) Nueva linea en repo1 y repo2 paso2

884f9ed Creo repo1 y repo2 paso1

Después nos conectamos al repositorio remoto y subimos lo que está validado en el área de commit, el master local (HEAD -> master).

El área commit local se envía al repositorio remoto utilizando **git push origin master**. El primer parámetro, **origin**, es el nombre amigable del repositorio remoto, para poder utilizar este nombre primero hay que crearlo con **git remote add <nombre\_remoto> <url del repositorio remoto>**. El segundo parámetro, **master**, es el nombre de la rama. Al lanzar el comando ***push***, va a pedir conexión con el repositorio remoto, hay que indicarle los datos de login.

Una vez finalizado podemos ver la configuración de los entornos con el comando **git remote -v.** Para poder eliminar las configuraciones creadas desde el repositorio local a los distintos repositories remotos, una vez que se haya identificado el nombre de la conexión remota que se desea eliminar, se utilizará el comando **git remote remove <nombre\_remoto>.**

**$ git remote add origin https://github.com/amaiaDi/TEST\_REPOSITORIO\_GIT.git**

**$ git push origin master**

Enumerating objects: 8, done.

Counting objects: 100% (8/8), done.

Delta compression using up to 12 threads

Compressing objects: 100% (4/4), done.

Writing objects: 100% (8/8), 643 bytes | 643.00 KiB/s, done.

Total 8 (delta 0), reused 0 (delta 0), pack-reused 0

To https://github.com/amaiaDi/TEST\_REPOSITORIO\_GIT.git

\* [new branch] master -> master

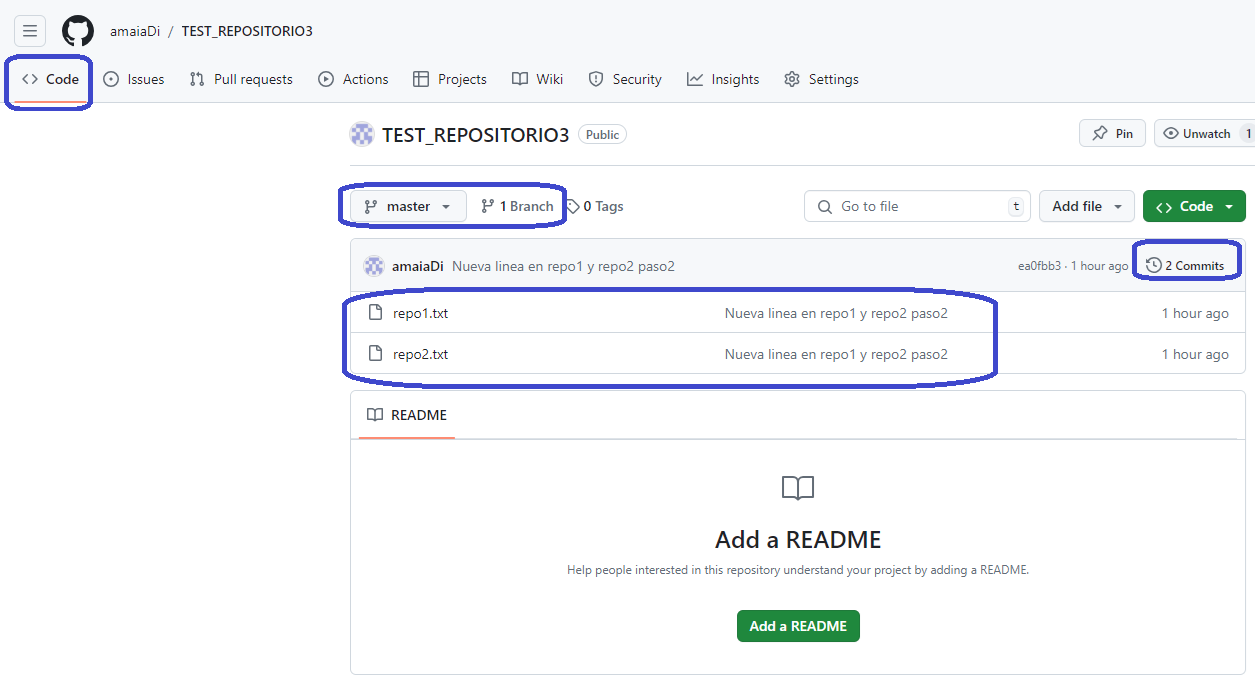
**$ git remote -v**

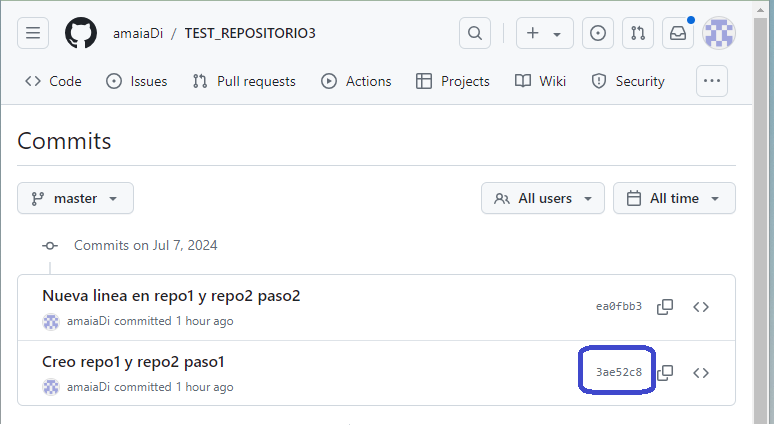
origin https://github.com/amaiaDi/TEST\_REPOSITORIO\_GIT.git (fetch)

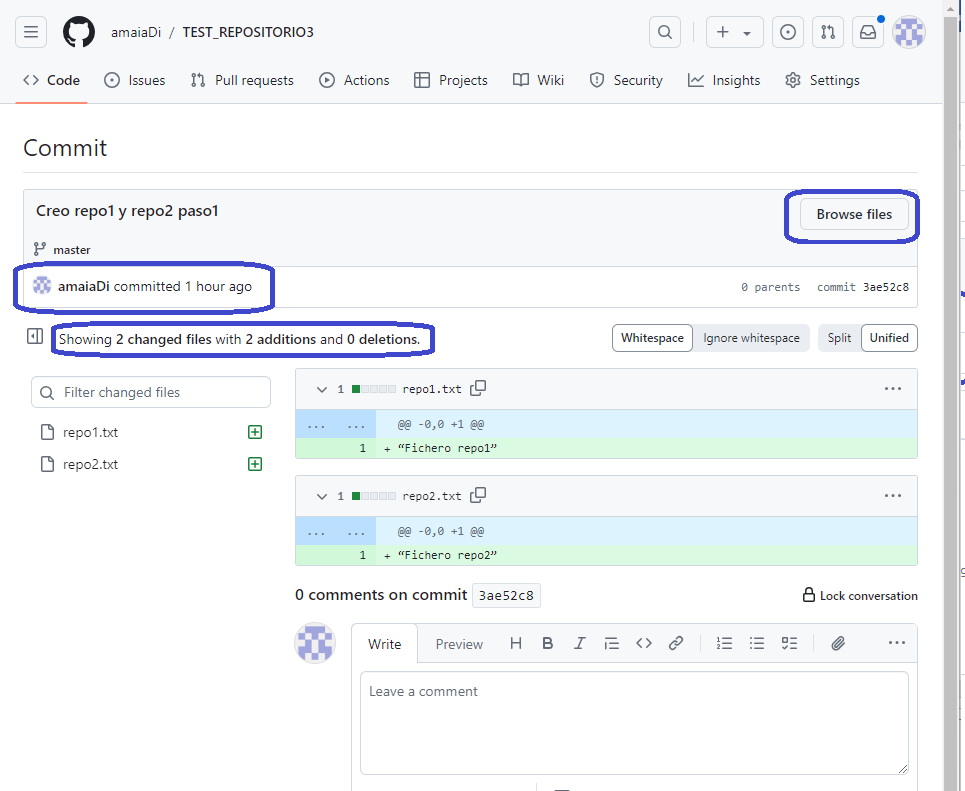
origin https://github.com/amaiaDi/TEST\_REPOSITORIO\_GIT.git (push)

**$ git remote remove origin**

En el repositorio de GitHub se verá la rama ***master***, con los dos commits, y el usuario que hizo los commits.







# Crear una copia local del repositorio remoto

Formas de traer el contenido del repositorio remoto:

* **Crear copia con checkout**

Creamos una nueva carpeta con el nombre ***repositorio4*** y entramos en ***Git Bash***. Después creamos el repositorio con **git init**, vinculamos el repositorio local con el remoto, nos traemos el repositorio remoto con **git fetch origin master** al repositorio local y descargamos el contenido del repositorio al área de trabajo con **git checkout origin/master**.

**$ git init**

**$ git remote add origin** **https://github.com/amaiaDi/TEST\_REPOSITORIO\_GIT.git**

**$ git fetch origin master**

remote: Enumerating objects: 8, done.

remote: Counting objects: 100% (8/8), done.

remote: Compressing objects: 100% (4/4), done.

remote: Total 8 (delta 0), reused 8 (delta 0), pack-reused 0

Unpacking objects: 100% (8/8), 623 bytes | 21.00 KiB/s, done.

From https://github.com/amaiaDi/TEST\_REPOSITORIO\_GIT

\* branch master -> FETCH\_HEAD

**\* [new branch] master -> origin/master**

**$ git checkout origin/master**

HEAD is now at ea0fbb3 Nueva linea en repo1 y repo2 paso2

**$ git status**

HEAD detached at origin/master

nothing to commit, working tree clean

Si vemos el estado del área de trabajo (***git status***), no nos dice que estamos en la rama master, sino que nos dice que ***HEAD***, la rama master del local está separada de ***origin/master (HEAD detached at origin/master)*** la rama master del remoto. (HEAD detached at origin/master: Esto significa que no estás en una rama local como master, sino que estás en un estado "desprendido" (detached). En este estado, HEAD (el apuntador a la revisión actual en la que estás trabajando) está apuntando directamente a un commit específico, en este caso, al commit que es la punta de origin/master, la rama master del repositorio remoto.) Si observamos el log, vemos que a ***HEAD*** le falta el apuntador ***master*** local:

**$ git log --oneline**

73daf15 (HEAD, origin/master) Nueva linea en repo1 y 2 paso2

Caa0b10 Creo repo1 y repo2 paso1

alumno@w10virtual MINGW64 /c/repositio4 ((73daf15...))

Para restablecer la rama principal (***master***) utilizamos el comando **get checkout master**.

**$ git checkout master**

**$ git log --oneline**

73daf15 (HEAD -> master, origin/master) Nueva linea en repo1 y 2 paso2

Caa0b10 Creo repo1 y repo2 paso1

alumno@w10virtual MINGW64 /c/repositio4 ((master))

En este momento el local y el remoto estarían en la misma instantánea, sincronizados: (HEAD -> master, origin/master).

Con el comando **git branch -v -a** podemos ver la rama local en la que estamos, y la rama remota a la que estamos conectados.

* **Clonando el repositorio remoto**

Al clonar el repositorio no tenemos que hacer todos los pasos que se hicieron anteriormente.

Para la prueba creamos una carpeta, ***repositorio5***. Al añadir un punto al final del comando clone conseguimos que el repositorio se clone en la carpeta donde se lanza el comando., si no se creará una subcarpeta. Hacemos el clonado en la carpeta creada. Añadimos un archivo y lo subimos al remoto.

**$ git clone https://github.com/amaiaDi/TEST\_REPOSITORIO\_GIT.git .**

Cloning into '.'...

remote: Enumerating objects: 8, done.

remote: Counting objects: 100% (8/8), done.

remote: Compressing objects: 100% (4/4), done.

remote: Total 8 (delta 0), reused 8 (delta 0), pack-reused 0

Receiving objects: 100% (8/8), done.

**$ echo “Fichero repo3” > repo3.txt**

**$ git add .**

**$ git commit –m “Creo repo3 paso3”**

**$ git log –-oneline**

c87626e (HEAD -> master) Creo repo3 paso3

ea0fbb3 (origin/master, origin/HEAD) Nueva linea en repo1 y repo2 paso2

3ae52c8 Creo repo1 y repo2 paso1

**$ git push origin master**

* **Visualización de cambios en remoto**

Para visualizar el historial de commits en el repositorio remoto, puedes utilizar el comando **git log --remotes**. Este comando te mostrará el historial de commits de todas las ramas remotas que tu repositorio local conoce.

Este comando muestra el historial de commits de todas las ramas remotas que tu repositorio local está siguiendo. Esto incluye tanto las ramas principales como las ramas de características que otros desarrolladores pueden haber creado y subido al repositorio remoto.

**$ git log --remotes**

commit c87626ecd2f28e924b4b942629c37191ecf4dc74 (HEAD -> master, origin/master, origin/HEAD)

Author: Amaia De la cruz <amaia.di@icjardin.com>

Date: Sun Jul 7 22:30:34 2024 +0200

Creo repo3 paso3

commit ea0fbb3eb66448ade5f2ae711693f3fae12e0375

Author: Amaia De la cruz <amaia.di@icjardin.com>

Date: Sun Jul 7 20:42:29 2024 +0200

Nueva linea en repo1 y repo2 paso2

commit 3ae52c8d1d6061cb1c2a712a14795feb9d02172e

Author: Amaia De la cruz <amaia.di@icjardin.com>

Date: Sun Jul 7 20:40:33 2024 +0200

Creo repo1 y repo2 paso1

Si estás en una rama local, verás el historial de commits de esa rama y de todas las ramas remotas.

Puedes agregar opciones adicionales como **--oneline, --graph**, o **--decorate** para obtener una vista más detallada y visual del historial de commits remotos.

* **Comparación de diferencias entre ramas locales y remotas**

Para comparar las diferencias entre las ramas locales y remotas, especialmente útil antes de fusionar cambios o sincronizar tu trabajo con el repositorio remoto, puedes utilizar **git diff**.

Puedes comparar las diferencias entre tu rama local y cualquier rama remota utilizando el comando **git diff <rama-local> <rama-remota>**

**$ git diff master origin/master**

# Trabajar con ramas (branches)

Ya hemos visto que por defecto *Git* llama ***master*** a la ***rama principal*** del proyecto o del desarrollo. Cuando hablamos de ramificaciones, significa que cogemos la rama principal de desarrollo (***master***) y a partir de ahí continuamos trabajando añadiendo nuevas funciones, sin seguir la rama master.

Para realizar las pruebas con las ramas, el repositorio de prueba estará conectado al repositorio remoto, en este caso el ***repositorio4*** **(git remote add origin**  [**https://github.com/amaiaDi/TEST\_REPOSITORIO\_GIT.git**](https://github.com/arama2020/gitrepositorio)).

Visualizamos las ramas que tenemos con **git branch -v -a** (-a incluirá también las ramas remotas, y –v incluirá el mensaje commit del HEAD de la rama). También con **git log** se pueden ver las ramas.

# Subir la rama al repositorio remoto

Si queremos subir el contenido de una rama a una ubicación remota utilizamos **git push <nombreremoto> <nombrederama>**.

**$ git push origin master pruebas**

**$ git log --oneline**

Si comprobamos el repositorio en GitHub observaremos que ahora tiene dos ramas.

Una vez la rama se ha subido al repositorio remoto, la vamos a borrar del repositorio local, **git branch –d <nombrederama>.** La rama existirá en el repositorio remoto, pero no en el local.

**$ git branch –d pruebas**

**$ git log –-oneline**

e5a3d40 (HEAD -> master, origin/pruebas, origin/master, origin/HEAD) Cambios en la rama pruebas de repo1.txt

a5599d6 Ficheros nuevos en la rama

fc11dc5 (origin/master, origin/HEAD) Creo repo3 paso3

73daf15 Nueva linea en repo1 y 2 paso2

caa0b10 Creo repo1 y repo2 paso1

Si quisiéramos descargar la rama del repositorio remoto al repositorio local:

**$ git fetch origin pruebas**

**$ git checkout pruebas**

# Provocando un conflicto

Tendremos un conflicto creando dos archivos con el mismo nombre en ramas diferentes, ***pruebas2*** y ***master***. Al intentar fusionar las dos ramas es cuando aparecerá el conflicto. Para ello creamos una nueva carpeta, ***repositorio5*** y clonamos el repositorio remoto. Después creamos un nueva rama, ***pruebas2***, añadimos un nuevo archivo con contenido y validamos. Nos movemos a la rama ***master*** y creamos el mismo fichero con otro contenido. Validamos el archivo y realizamos la fusión.

**$ git clone**  [**https://github.com/amaiaDi/TEST\_REPOSITORIO\_GIT.git**](https://github.com/arama2020/gitrepositorio) **.**

**$ git branch pruebas2**

**$ git checkout pruebas2**

**$ echo “Creo archivo en pruebas2” >> archivo.txt**

**$ echo “Otro texto” >> archivo.txt**

**$ git add archivo.txt**

**$ git commit –m “Creado archivo.txt en la rama pruebas2”**

**$ git log --oneline**

**$ git checkout master**

**$ echo “Creo en master” > archivo.txt**

**$ git add archivo.txt**

**$ git commit –m “Creado archivo.txt en master”**

**$ git log –oneline**

**$ git merge pruebas2 master**

CONFLICT (add/add): Merge conflict in archivo.txt

Auto-merging archivo.txt

Automatic merge failed; fix conflicts and then commit the result.

Al visualizar el estado del área de trabajo, ***git status***, git sugiere las acciones a realizar para resolver el conflicto.

**$ git status**

On branch master

Your branch is ahead of ‘origin/master’ by 1 commit.

(use “git push” to publish your local commits)

You have unmerged paths.

(fix conflicts and run “git commit”)

(use “git merge --abort” to abort the merge

Unmerged paths:

(use “git add <file>...” to mark resolution)

Both added: archivo.txt

no changes added to commit (use “git add” and/or “git commit –a”)

alumno@w10virtual MINGW64 /c/repositorio5 (master|MERGING)

# Resolver el conflicto

Cuando fusionamos ramas en Git y existe un conflicto en los cambios entre ellas, es necesario resolver estos conflictos antes de continuar con el proceso de fusión. Para resolver el conflicto hay que seguir las sugerencias de *Unmerged paths*. Pero antes hay que sincronizarse con el repositorio remoto, añadir y validar el archivo en conflicto y volver a sincronizarse.

Es recomendable asegurarse de que estamos sincronizados con los últimos cambios del repositorio remoto antes de fusionar ramas y resolver conflictos.

**$ git push origin master**

**$ git status**

**$ git log --oneline**

Al hacer ***merge*** tenemos un conflicto. Va a pedir que añadamos y validemos *archivo.txt*. Hasta que se resuelva el conflicto no podremos volver a la rama *pruebas2*.

**$ git merge pruebas2 master**

Abrimos el archivo archivo.txt en un editor de texto y encontramos marcas especiales (<<<<<<<, =======, >>>>>>>) que indican las diferencias entre las ramas. Decidimos qué cambios mantener y eliminamos las marcas de conflicto, dejando el contenido como deseamos que quede después de la fusión.

**$ cat archivo.txt**

<<<<<<< HEAD

Creo en master

=======

Creo archivo en pruebas2

Otro texto

>>>>>>> pruebas2

Una vez resuelto el conflicto, agregamos el archivo modificado y confirmamos los cambios con un mensaje descriptivo. De esta forma se comunica a GitHub que se ha resuelto el conflicto.

**$ git add archivo.txt**

**$ git commit –m “valido para solucionar conflicto”**

Una vez solucionado ya se puede cambiar de rama, ***git checkout pruebas2***. Al visualizar el contenido de archivo.txt veremos el contenido inicialmente guardado, el contenido fusionado se encuentra en la rama master. La rama pruebas2 no se ha subido al repositorio remoto, **git push origin master pruebas2**. Finalmente, después de haber solucionado el conflicto hay que sincronizar la rama master local con el master remoto, origin/master.

**$ git push origin master**

Git dispone de un comando para consultar toda la información de un fichero: las revisiones, el autor de las modificaciones, líneas modificadas, etc; **git blame <nombredefichero>**. Se puede filtrar el rango de filas sobre los que se quiere hacer la consulta gracias al atributo **–L <fila inicio>,<fila fin>**.

**$ git blame archivo.txt**

**$ git blame –L 4,6 archivo.txt**

# Trabajar con etiquetas (tags)

Las etiquetas en Git se utilizan para marcar puntos específicos en la historia del repositorio, típicamente para señalar versiones de lanzamiento importantes. Hay dos tipos principales de etiquetas en Git. Esta funcionalidad se utiliza, normalmente, para marcar versiones de lanzamiento.

El comando para mostrar las etiquetas es ***git tag***. Para crear una etiqueta se utiliza **git tag –a <nombreetiqueta> -m <mensajedeetiqueta>**.

**$ git tag –a v1.00 –m “Version 1.0”**

Se pueden listar todas las etiquetas que cumplen un patrón, por ejemplo, las que empiecen por v1:

**$ git tag –l ‘v1.\*’**

Hay dos tipos de etiquetas:

* **Anotadas**: se guardan en la base de datos de Git como objetos enteros. Contienen el nombre del etiquetador, el correo electrónico y fecha; tienen un mensaje asociado; y pueden estar firmadas y verificadas con GNU Privacy Guard (GPG). Normalmente se recomienda crear este tipo de etiquetas. La forma de crear una etiqueta anotada es la forma mostrada anteriormente, si no se especifica –m Git abrirá el editor de texto para crear el mensaje.
* **Ligeras:** se utilizan como etiquetas temporales y no tienen mensaje asociado.La forma de crearlos es **git tag <nombreetiqueta>**.

Para ver la información de una etiqueta hay que utilizar **git show <nombreetiqueta>**.

**$ git show v1.00**

Por defecto, el comando ***git push*** no transfiere etiquetas a servidores remotos. Para enviarlas hay que utilizar el comando **git push origin <nombreetiqueta>** o, en caso de tener muchas etiquetas, **git push origin --tags**.

**$ git push origin v1.00**