

Git & GitHub

Sobre mi



- **Manuel Landín Gómez**
- Antiguo alumno del IES Teis
- Desarrollo de Aplicaciones Multiplataforma
- Línea Cloud Native Gradient



kubernetes



Sobre el taller



Sobre el taller



Sobre el taller

Que no esperar

- Dominar Git al acabar el taller.
- Dominar GitHub al acabar el taller.

Que esperar

- Entender a alto nivel qué es Git y qué es GitHub y su diferencia.
- Realizar operaciones básicas con Git.
- Vincular un repositorio local a un remoto en GitHub.
- Entender a alto nivel el concepto de rama y su utilidad.
- Realizar operaciones básicas en GitHub.
- Comentar algunos patrones típicos.

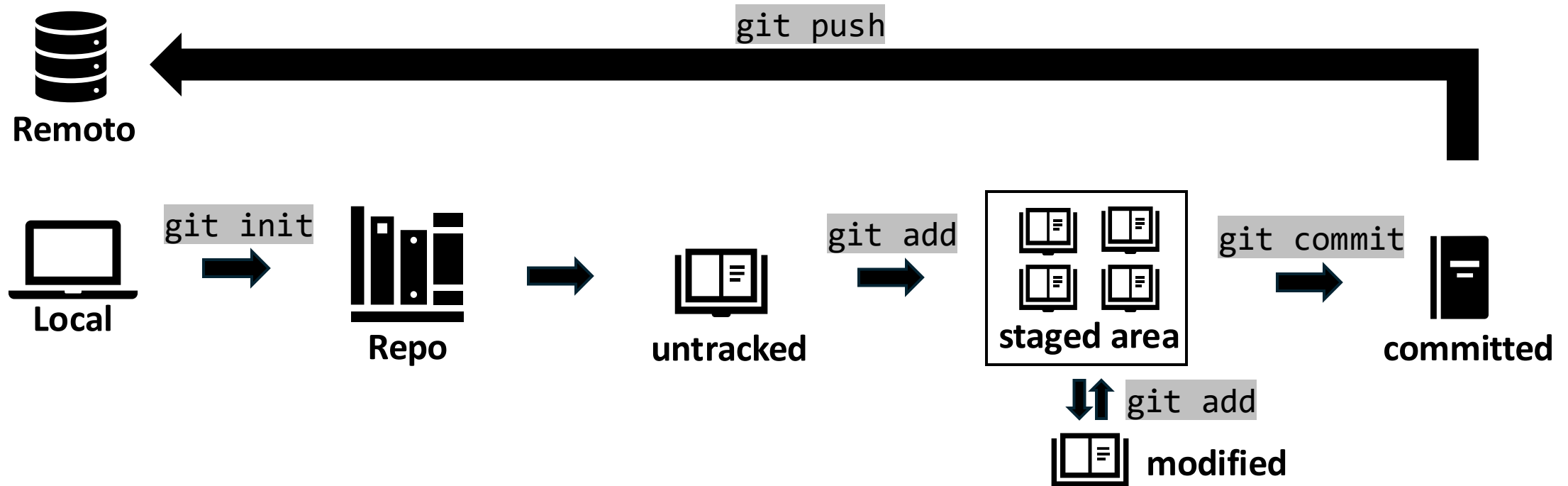
Observaciones

- Diapositivas con mucho texto. Es intencionado.
- Es un taller. *Hands on*.

¿Qué es git?

Git es un sistema de control de versiones distribuido que permite a los desarrolladores gestionar el historial de cambios de un **proyecto**.

Fue creado en 2005 por Linus Torvalds, para resolver problemas de **control de versiones** y **colaboración**.



¿Qué es git?

Para usar Git debemos tener el binario en nuestro sistema.

Podemos descargarlo a través de algún gestor de paquetes o desde la página web <https://git-scm.com>

Antes de empezar a usar Git configuremos nuestro perfil para el registro de *commits*.

Para ello almacenamos nuestro **nombre** e **email**:

```
git config --global user.name <"name">
```

```
git config --global user.email <"email">
```

Además, es habitual que la rama de trabajo por defecto se denomine **main**:

```
git config --global init.defaultBranch main
```

¿Qué es git?

Un **repositorio** es un espacio donde Git almacena el historial de cambios de un **proyecto**.

Podemos **crear un nuevo repositorio** asociado a un proyecto con el comando `git init <path>`, donde `<path>` será la ruta al directorio del proyecto. Si ya nos encontramos en dicho directorio, podemos usar simplemente `git init`.

Al usar `git init`, Git creará un directorio `.git` en la propia ruta del proyecto. Este directorio será el lugar en el sistema de ficheros de nuestra máquina donde Git almacenará el historial de cambios.

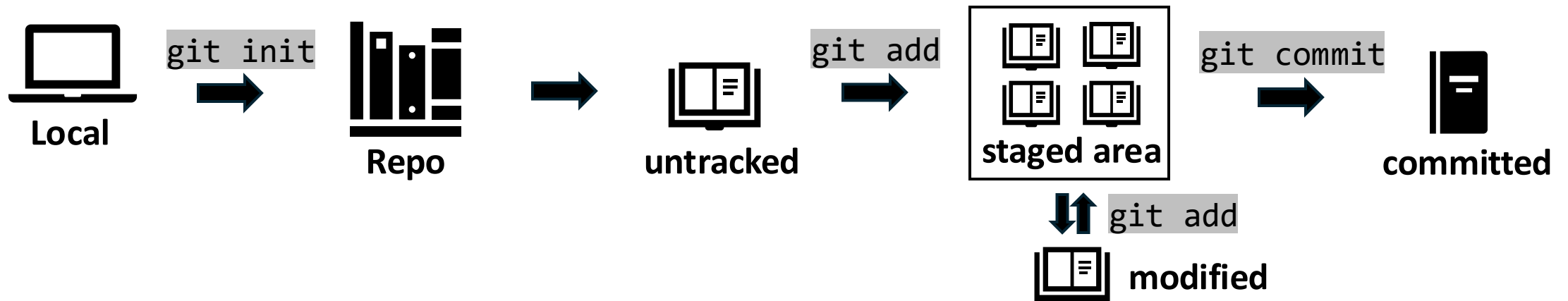


¿Qué es git?

Una vez ejecutado `git init`, se inicia el repositorio y todos los ficheros del proyecto pasan a estado **untracked**. Esto quiere decir que Git es consciente de que existen, pero no los considera preparados para confirmar. Podemos visualizar esto con `git status`.

Para poder confirmar un fichero, éste debe estar en el área de preparación (**staged area**). Podemos mover cualquier fichero al área de preparación con el comando `git add <file>`. Si modificamos un fichero del **staged area**, éste saldrá del área y tendremos que volver a añadirlo con `git add <file>`.

Para confirmar todos los ficheros del área de preparación usamos `git commit -m <"message">`. Los ficheros confirmados se guardan en el historial del repositorio como una *snapshot*.



¿Qué es git?

A medida que vayamos confirmando ficheros, iremos incrementando el número de *snapshots* de nuestro repositorio local. Esto nos permitirá llevar un **control de versiones** de nuestro proyecto.

Podemos ver el historial de *commits* con el comando: `git log`




Podemos volver temporalmente a un *commit* anterior usando:
`git checkout <commit-hash>`

Podemos volver de manera **definitiva** usando:
`git reset --hard <commit-hash>`

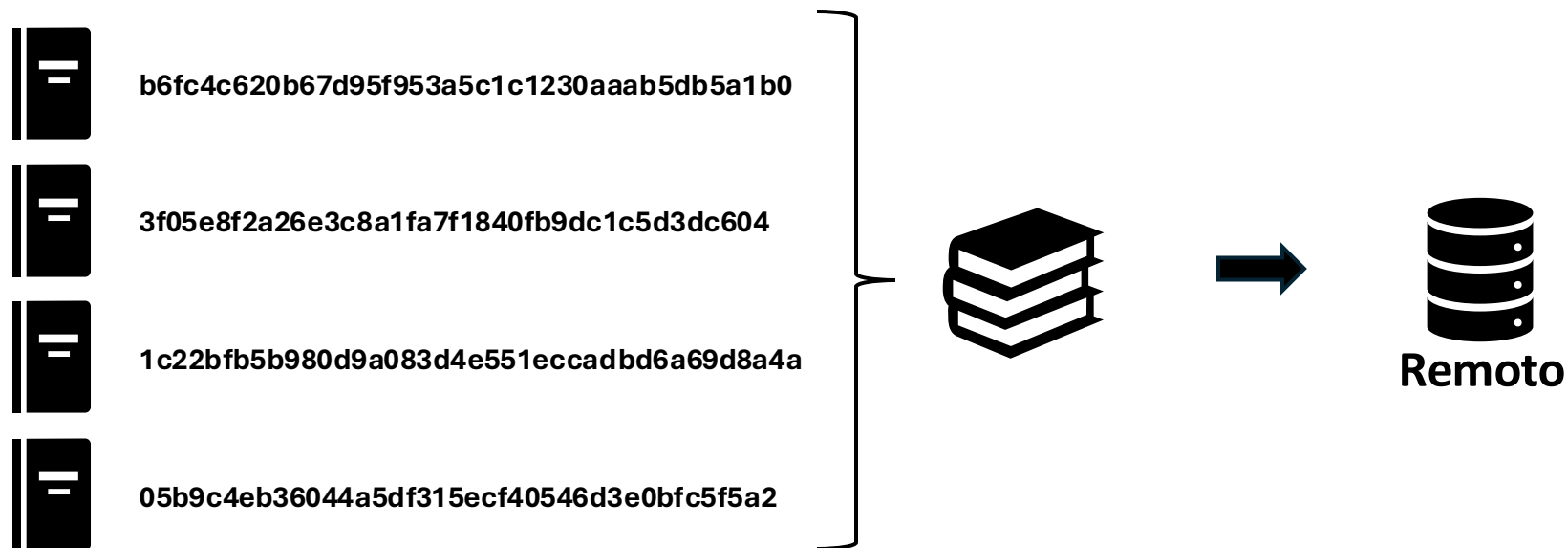
Podemos crear una nueva rama a partir de un *commit* anterior usando:
`git checkout -b <new-branch-name> <commit-hash>`

¿Qué es git?

En general es posible y conveniente trabajar con un **repositorio remoto**  .

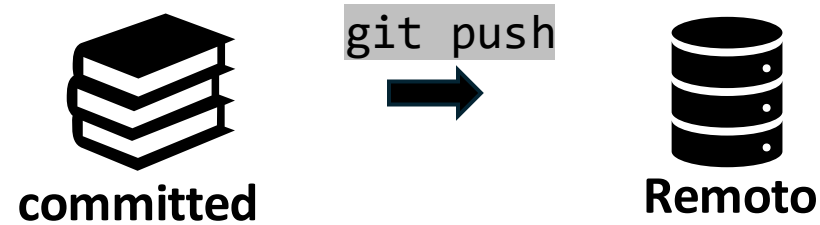
Un **repositorio remoto**  no es más que un servidor donde se almacena una copia de nuestro repositorio local.

Trabajar con un repositorio remoto es más seguro y permite la colaboración.



¿Qué es git?

Podemos enviar nuestros **ficheros confirmados** a un repositorio remoto usando el comando `git push`. Sin embargo, debemos configurar un servidor para que actúe como repositorio remoto de Git.

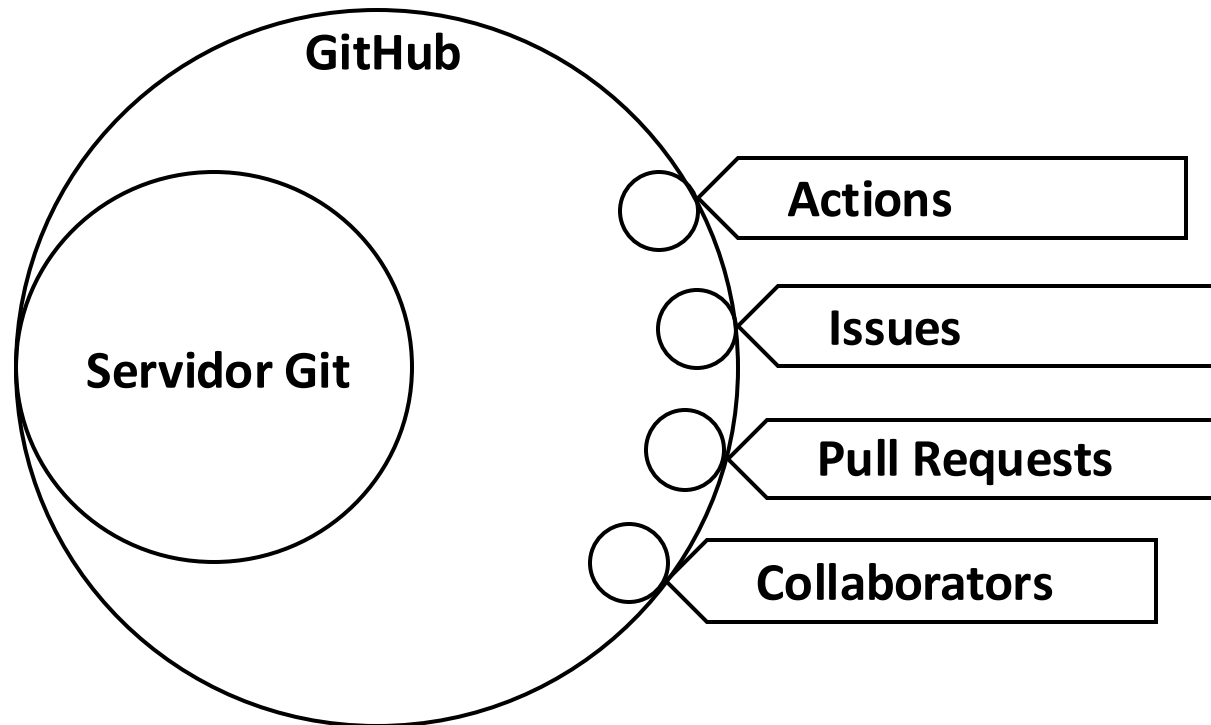


Una alternativa es usar una **solución de terceros** que nos evite el trabajo. **GitHub** es una de ellas.



¿Qué es GitHub?

GitHub es una **plataforma de repositorios Git** que además de actuar como servidor Git, incluye una gran variedad de herramientas y funcionalidades para facilitar la **colaboración**, la **gestión** de proyectos y la **automatización** del desarrollo de software.



¿Qué es GitHub?

Vamos a usar GitHub como nuestra plataforma de repositorios por defecto. Para esto, lo primero que tenemos que hacer es crear una cuenta en <https://github.com>

Ahora, debemos autenticar nuestra máquina local con nuestra cuenta de GitHub para interactuar con los repositorios remotos. Usaremos **SSH**, una shell segura basada en **clave pública** - **clave privada**, para la autenticación.

Generamos un par de claves con el comando: `ssh-keygen -t ed25519 -f ~/.ssh/github_key`
El comando genera una clave privada (`github_key`) y una clave pública (`github_key.pub`) en el directorio de claves `~/.ssh`. Accedemos a este directorio y copiamos el contenido de la clave pública.



`github_key`

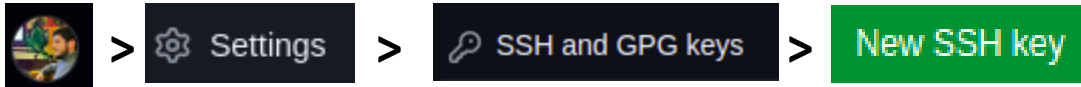


`github_key.pub`

ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQGC7XUu5KMii3VifmNp2rySVz3RAmq5MDINDcqxBKj9J46awc51G...

¿Qué es GitHub?

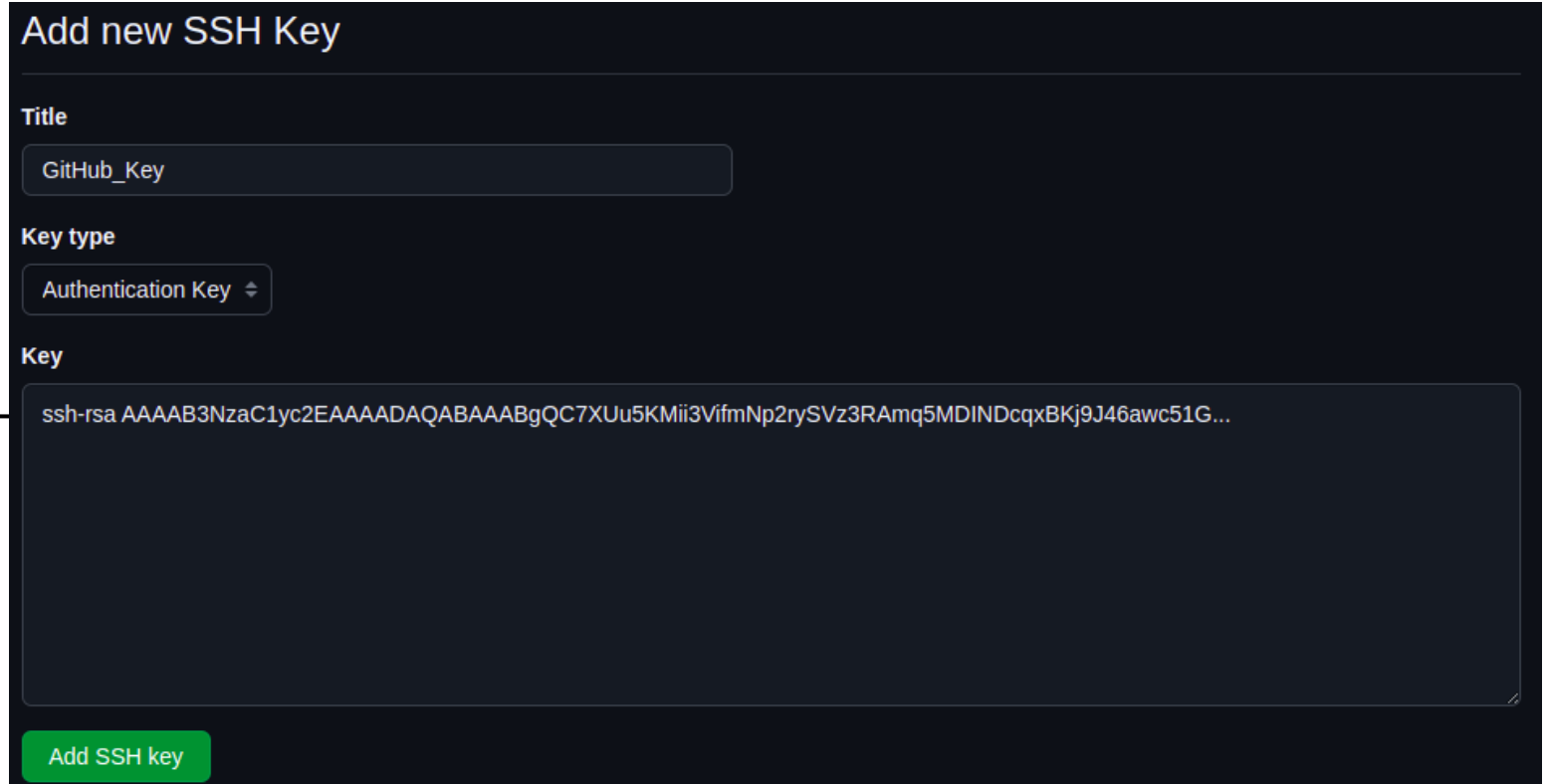
Ahora, accedemos a nuestra cuenta y navegamos:



Pegamos la clave pública en el campo Key y añadimos:



github_key.pub

A screenshot of the 'Add new SSH Key' form in a dark theme. The form has three sections: 'Title' with a text input containing 'GitHub_Key'; 'Key type' with a dropdown menu showing 'Authentication Key'; and 'Key' with a large text area containing a long SSH public key string: 'ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQGC7XUu5KMii3VifmNp2rySVz3RAmq5MDINDcqxBKj9J46awc51G...'. At the bottom right is a green 'Add SSH key' button.

Add new SSH Key

Title

GitHub_Key

Key type

Authentication Key

Key

ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQGC7XUu5KMii3VifmNp2rySVz3RAmq5MDINDcqxBKj9J46awc51G...

Add SSH key

¿Qué es GitHub?

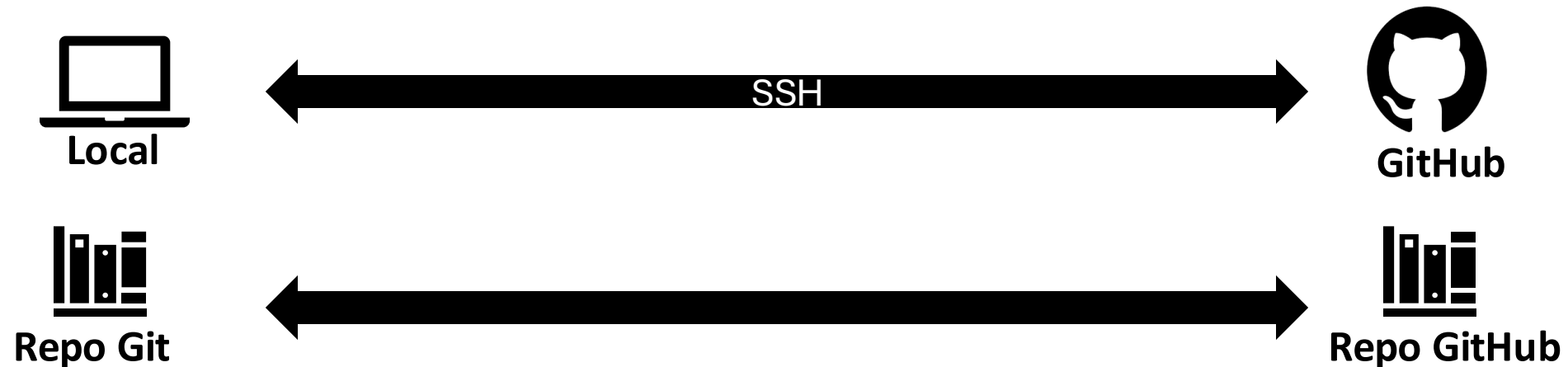
Ahora GitHub tiene nuestra clave pública. Cuando ejecutemos comandos contra el repositorio remoto como `git push`, Git invocará al cliente SSH para establecer una conexión segura con GitHub. El cliente SSH buscará la clave privada en el directorio `~/ .ssh` y se iniciará el proceso de validación. Todo el proceso de validación es transparente al usuario.




¿Qué es GitHub?

Llegados a este punto, ya podemos interactuar con nuestros repositorios. Sin embargo, debemos preparar el escenario del lado de GitHub para poder usar de manera efectiva comandos como `git push`.

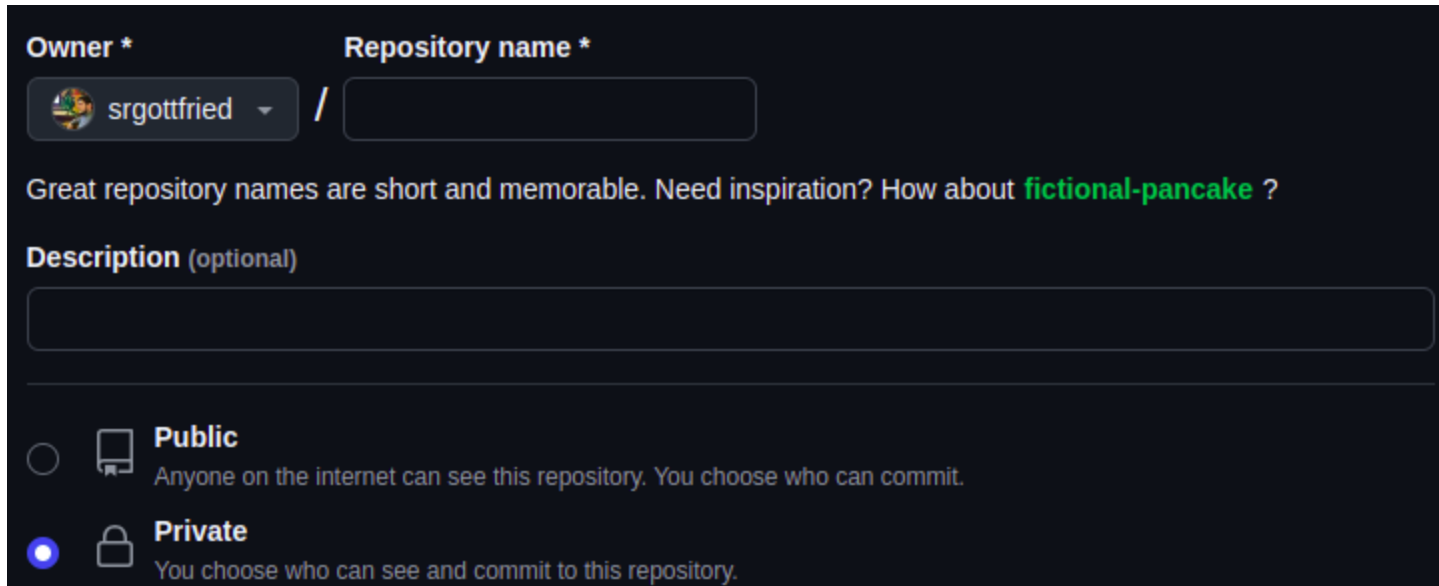
Y es que GitHub maneja el concepto de **repositorio** de manera independiente a Git. Para acabar este proceso de integración, tendremos que crear un nuevo repositorio en GitHub y vincularlo a nuestro repositorio de Git.




¿Qué es GitHub?

Crear un repositorio desde GitHub es un proceso sencillo. Desde el Dashboard localizamos el botón  Elegimos un nombre para nuestro repositorio y su visibilidad.

Luego creamos el repositorio 


A dark-themed form for creating a new repository on GitHub. It includes fields for "Owner" (a dropdown menu showing "srgottfried"), "Repository name" (an empty text box), and "Description (optional)" (a large text area). Below these fields are two radio button options: "Public" (with a monitor icon) and "Private" (with a lock icon). The "Private" option is selected, indicated by a blue dot. A hint text above the name field suggests "fictional-pancake" as an example.


Owner * Repository name *

 srgottfried /

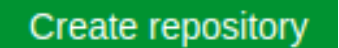
Great repository names are short and memorable. Need inspiration? How about **fictional-pancake** ?

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.



A green rectangular button with the text "Create repository" in white.

¿Qué es GitHub?

Ahora debemos **vincular el repositorio de GitHub con nuestro repositorio en local.**

Para ello asociamos la URL de nuestro repositorio al alias **origin** mediante el comando:

```
git remote add origin git@github.com:USER-NAME/PROJECT-NAME.git
```

Los repositorios ya se encuentran vinculados.



¿Qué es GitHub?

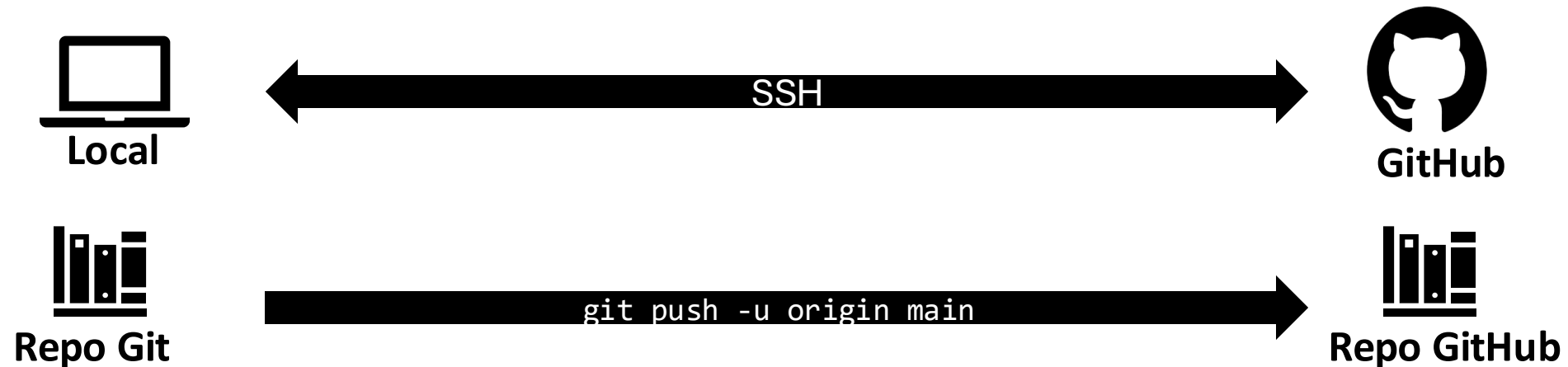
El último paso es **volcar nuestro repositorio local** al servidor remoto de GitHub.

Para ello renombramos la rama principal (por convención) a **main** si esta tuviese otro nombre:

```
git branch -M main
```

Empujamos todos los cambios en la rama **main** al repositorio remoto vinculado:

```
git push -u origin main
```



¿Qué es GitHub?

De manera similar, si un repositorio existe en el servidor remoto pero no en nuestro local, podemos traerlo mediante el comando:

```
git clone git@github.com:USER-NAME/PROJECT-NAME.git
```



¿Qué es GitHub?

En ocasiones, otros desarrolladores harán *push* a un repositorio remoto común de manera que nuestro repositorio local se quede atrasado respecto al remoto. Podemos actualizar nuestro repositorio local, descargando las novedades del remoto, usando el comando:

```
git fetch
```

o

```
git pull (git fetch + git merge)
```



¿Qué hemos conseguido?

1%

Llegados a este punto, hemos conseguido:

- Instalar Git en nuestro sistema y crear una cuenta de GitHub.
- Iniciar nuestro perfil en Git y crear nuestro primer repositorio Git.
- Autenticar nuestra máquina local en nuestra cuenta de GitHub.
- Crear un repositorio GitHub y vincularlo a nuestro repositorio Git.

¿Qué nos queda?

99%

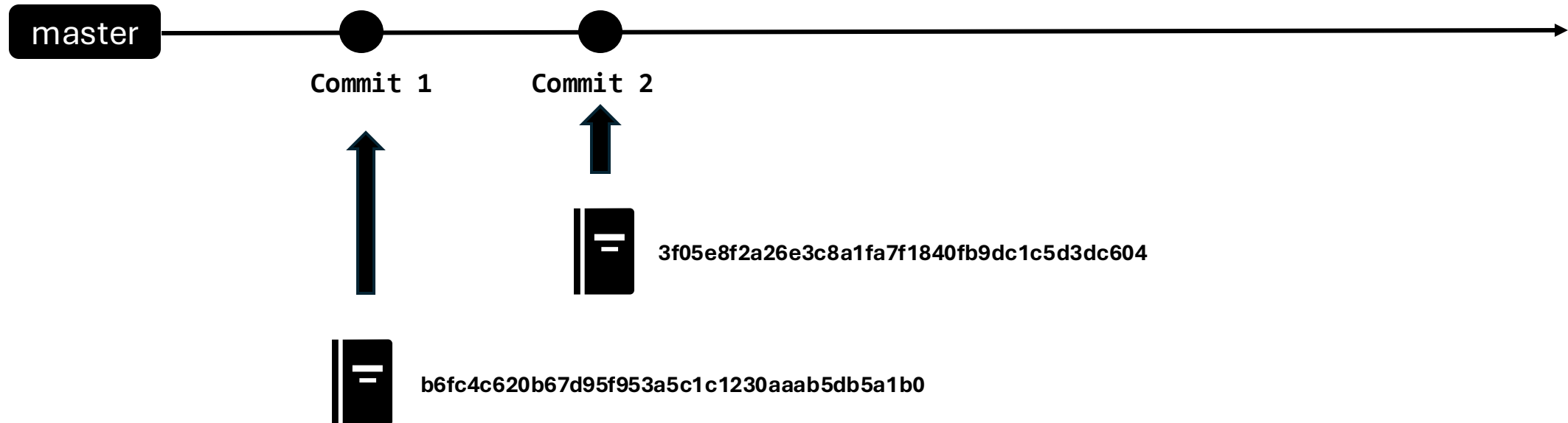
Nos queda mucho por ver. Intentaremos cubrir algunos puntos:

- Uso de ramas en Git.
- Uso de herramientas de GitHub.
- Patrones.

Uso de ramas en git

Uno de los elementos más potentes de Git son las **ramas** o **branches**. Podemos entender las ramas como líneas de tiempo paralelas de nuestro historial de versiones.

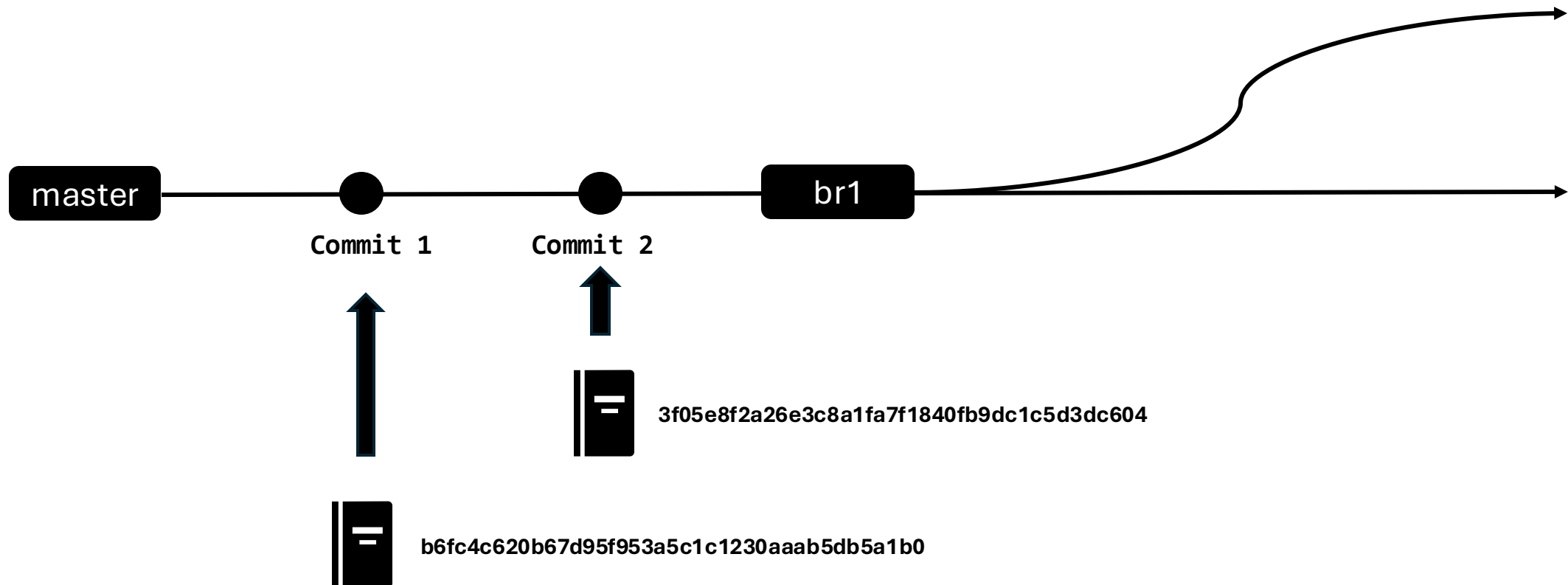
Por defecto, Git crea una rama **master** al realizar nuestro primer *commit*. Esta será nuestra línea de tiempo actual, y cualquier *commit* que hagamos en el futuro tendrá lugar aquí:



Uso de ramas en git

Sin embargo, es posible crear nuevas ramas usando el comando `git branch <name>`.

Por defecto, seguiremos viviendo en la rama master, pero podemos movernos entre las ramas existentes usando el comando `git checkout <name>`.

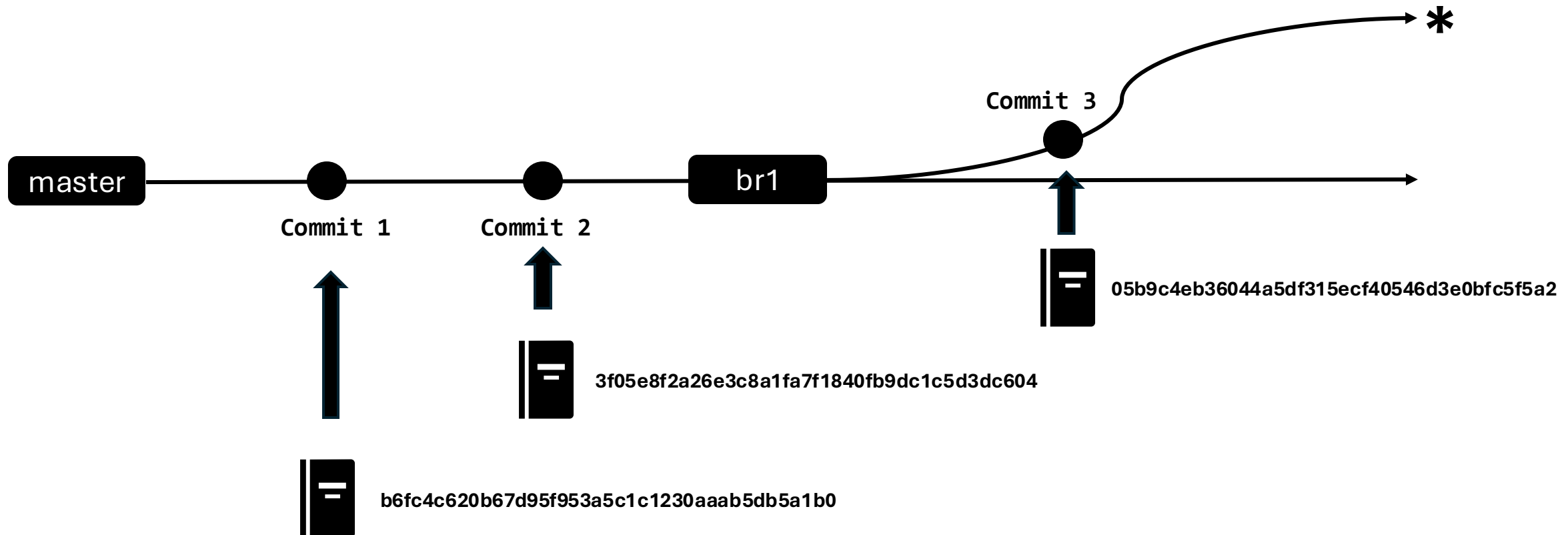


Uso de ramas en git

Sólo podemos estar en una rama a la vez. Para saber en qué rama estamos podemos consultarlo en el comando `git branch`. La rama actual estará marcada con un *****.

Los *commits* realizados a partir de ahora se harán sólo contra la rama en la que nos encontremos.

Los ficheros de nuestro proyecto se corresponderán a las versiones de la rama en la que nos encontremos.

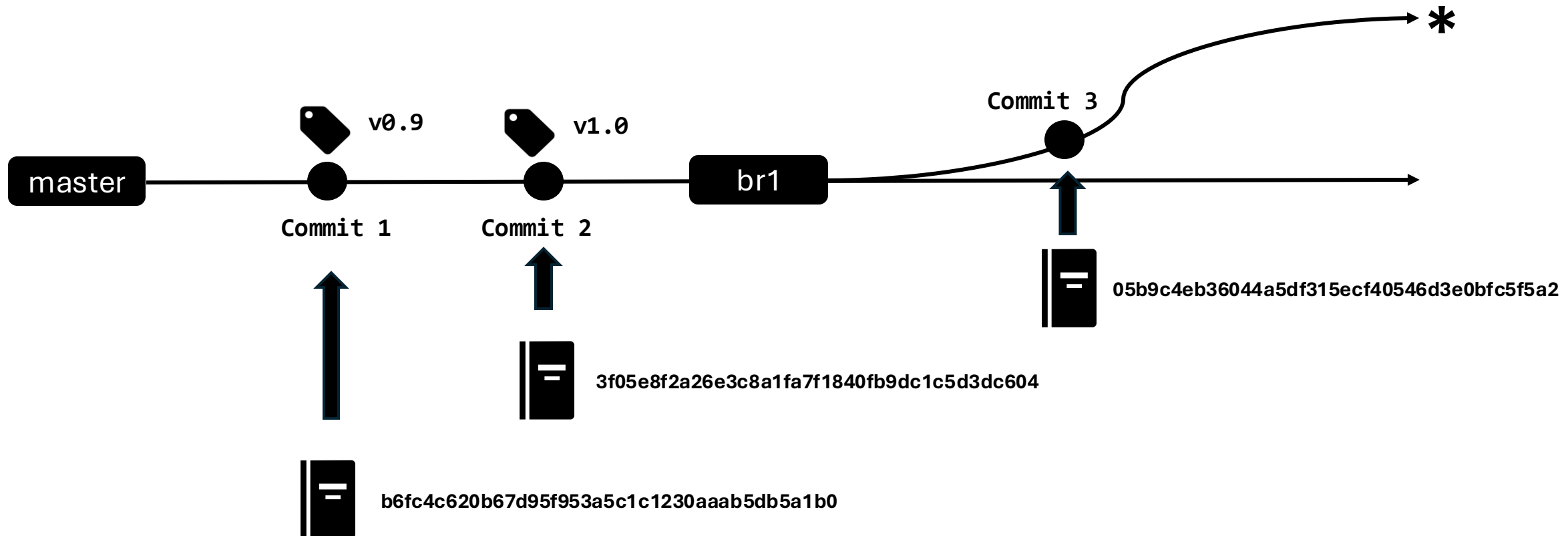


Uso de ramas en git

También podemos crear **etiquetas** o **tags** en *commits* concretos del desarrollo con el objetivo de marcar algún hito. Podemos hacer esto a través del comando `git tag <"tag-name">`.

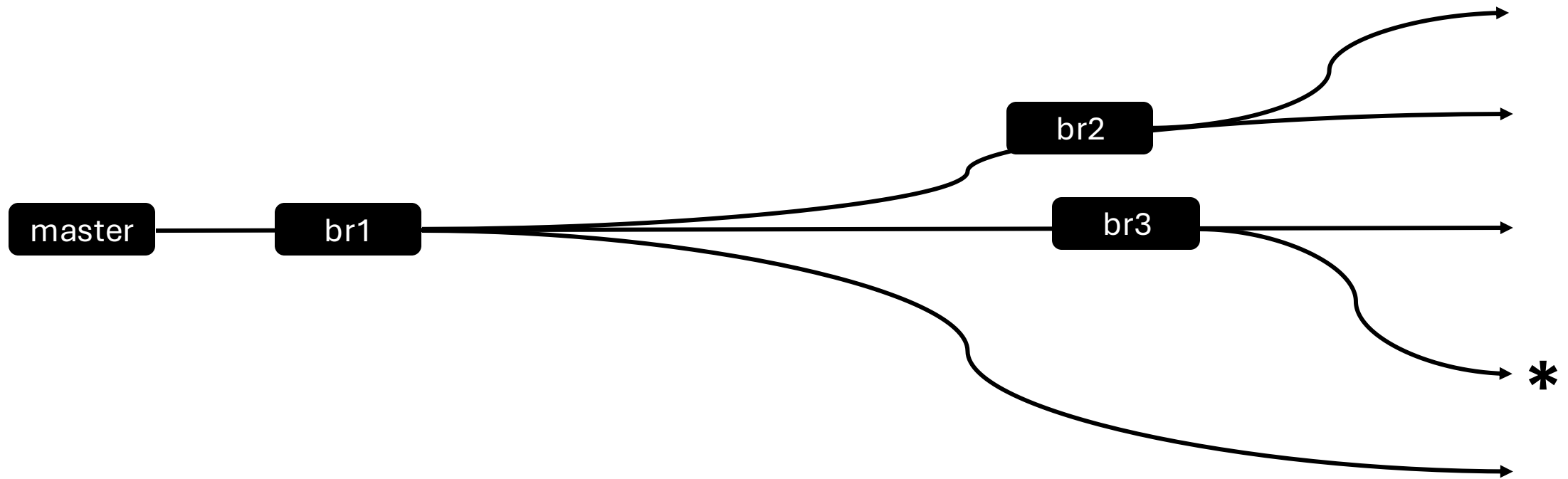
Estas etiquetas quedan vinculadas al *commit* actual.

Es posible enviar estas etiquetas al repositorio remoto con `git push --tags`.



Uso de ramas en git

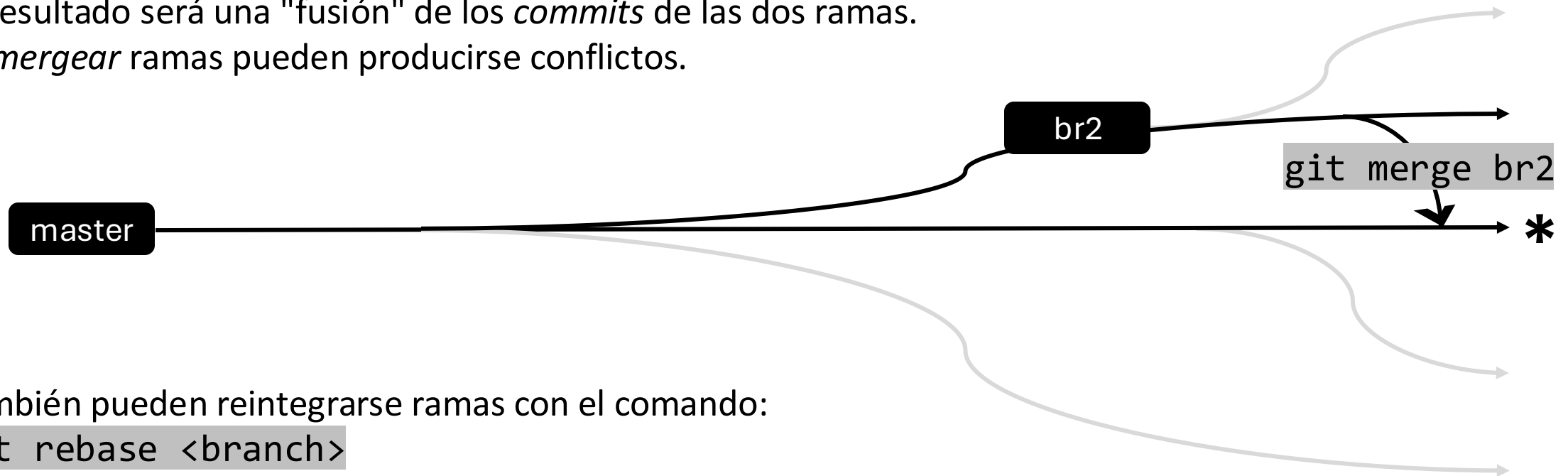
Es posible crear tantas ramas como queramos, tanto a partir de la rama principal como de ramas secundarias:



Uso de ramas en git

Es posible reintegrar una rama en otra. Para ello, nos posicionamos en la rama sobre la que vamos a integrar, y usamos el comando `git merge <branch>`, donde `<branch>` será la rama integrada.

El resultado será una "fusión" de los *commits* de las dos ramas.
Al *mergear* ramas pueden producirse conflictos.



También pueden reintegrarse ramas con el comando:
`git rebase <branch>`

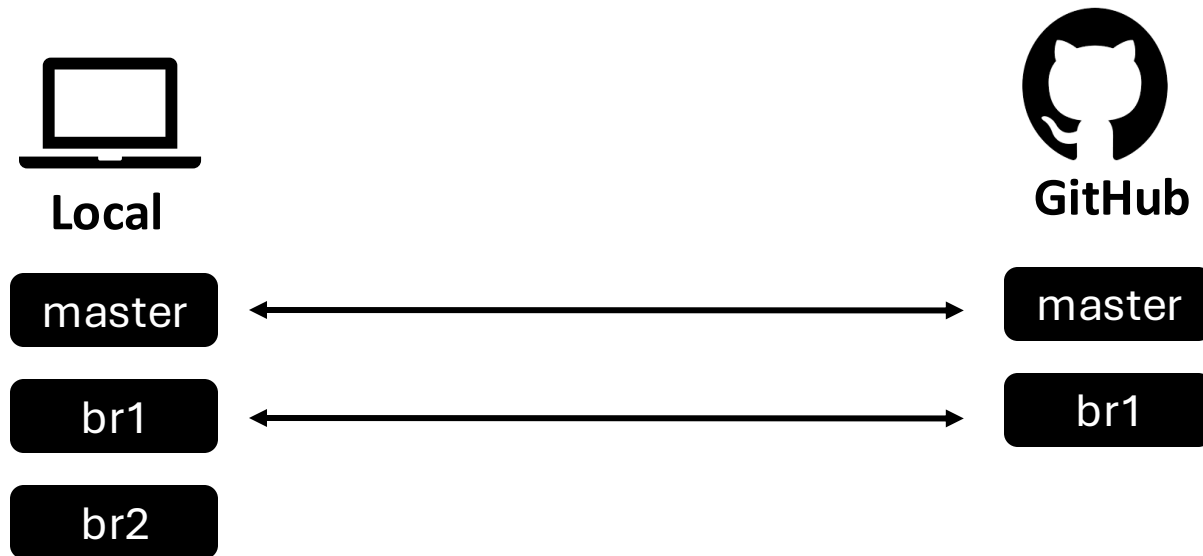
La diferencia entre *merge* y *rebase* está en cómo se muestran los *commits* en la rama resultante.

Uso de ramas en git

Es posible enviar nuevas ramas a nuestro repositorio de GitHub. Para enviar una nueva rama a nuestro repositorio debemos vincularla con el remoto manualmente:

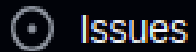
```
git branch --set-upstream-to=origin/<BRANCH-NAME>
```

Una vez vinculada, podremos operar sobre esta rama con normalidad.

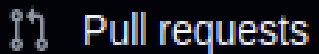


Uso de herramientas de GitHub

GitHub cuenta con muchas opciones y herramientas para la gestión de repositorios, el trabajo colaborativo o la automatización. Vamos a centrarnos en dos especialmente importantes:



Las **issues** son propuestas que podemos hacer sobre un repositorio en relación a un tema determinado, como un bug o una mejora del código. Una issue inicia una conversación asincrónica y puede evolucionar en la creación de una rama y una posterior pull request.



Las **pull requests (PR)** son propuestas de cambios en un repositorio de código. Permite a un desarrollador particular notificar al propietario del repositorio que ha realizado cambios en una rama y desea integrarla con otra rama del proyecto.

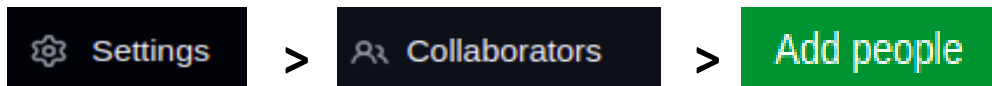
Uso de herramientas de GitHub

La colaboración es uno de los fuertes de GitHub.

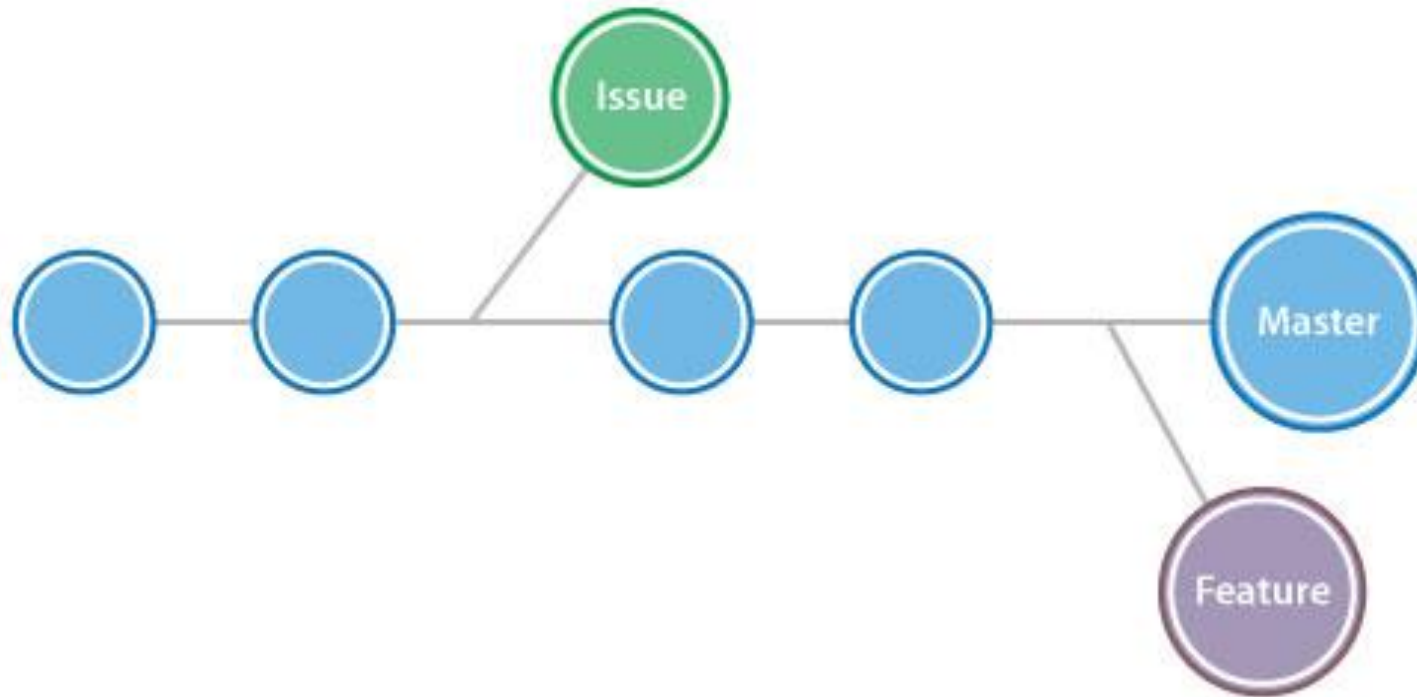
Es posible colaborar en repositorios a través de **Issues** y **Pull Requests**. También es posible añadir colaboradores de manera explícita, gestionando sus permisos de manera fina. Esto es especialmente útil en repositorios privados.

Los **Collaborators** en GitHub son personas a las que se les otorga acceso a un repositorio para colaborar en su desarrollo. Al añadir colaboradores, puedes asignarles permisos específicos para que puedan clonar, editar y enviar cambios al repositorio.

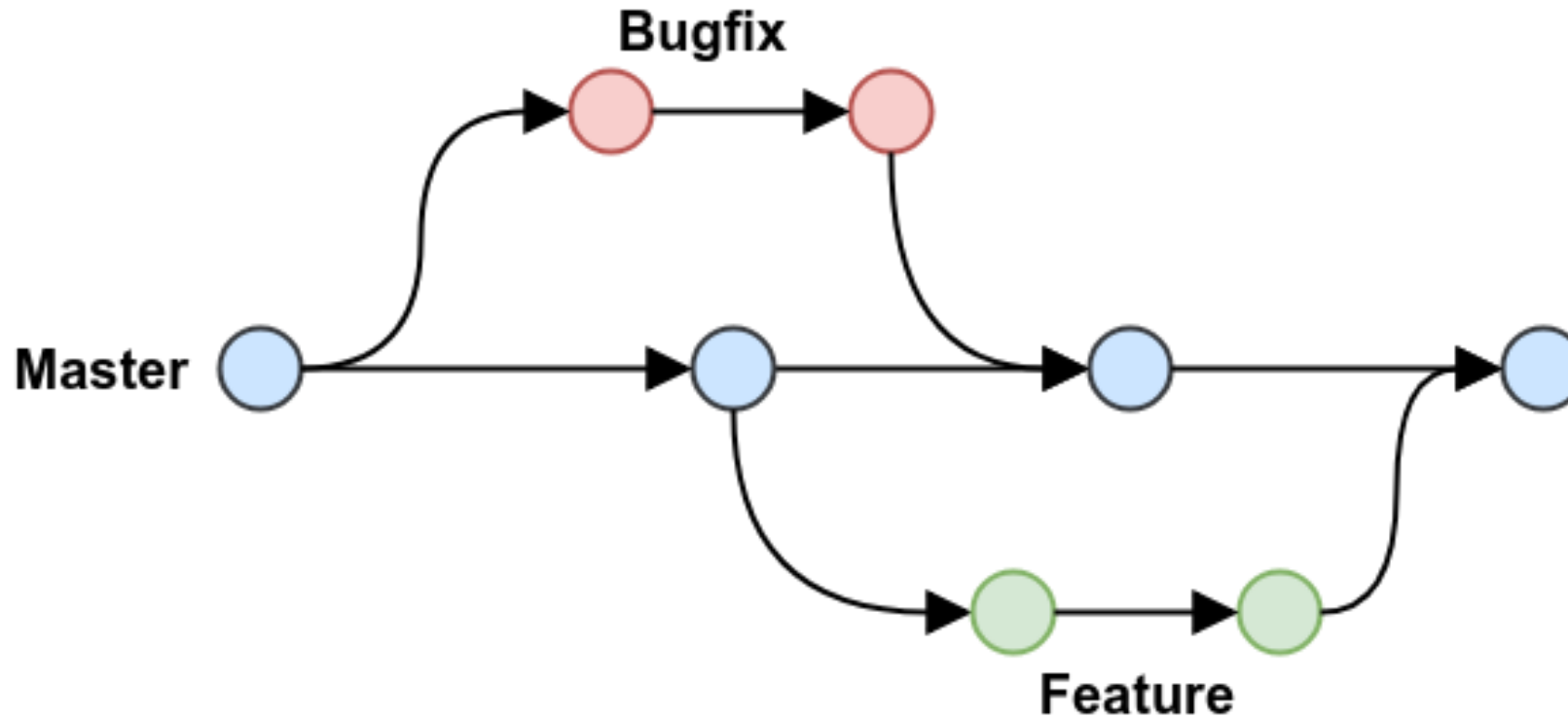
Podemos añadir un colaborador para un repositorio navegando desde ese repositorio:



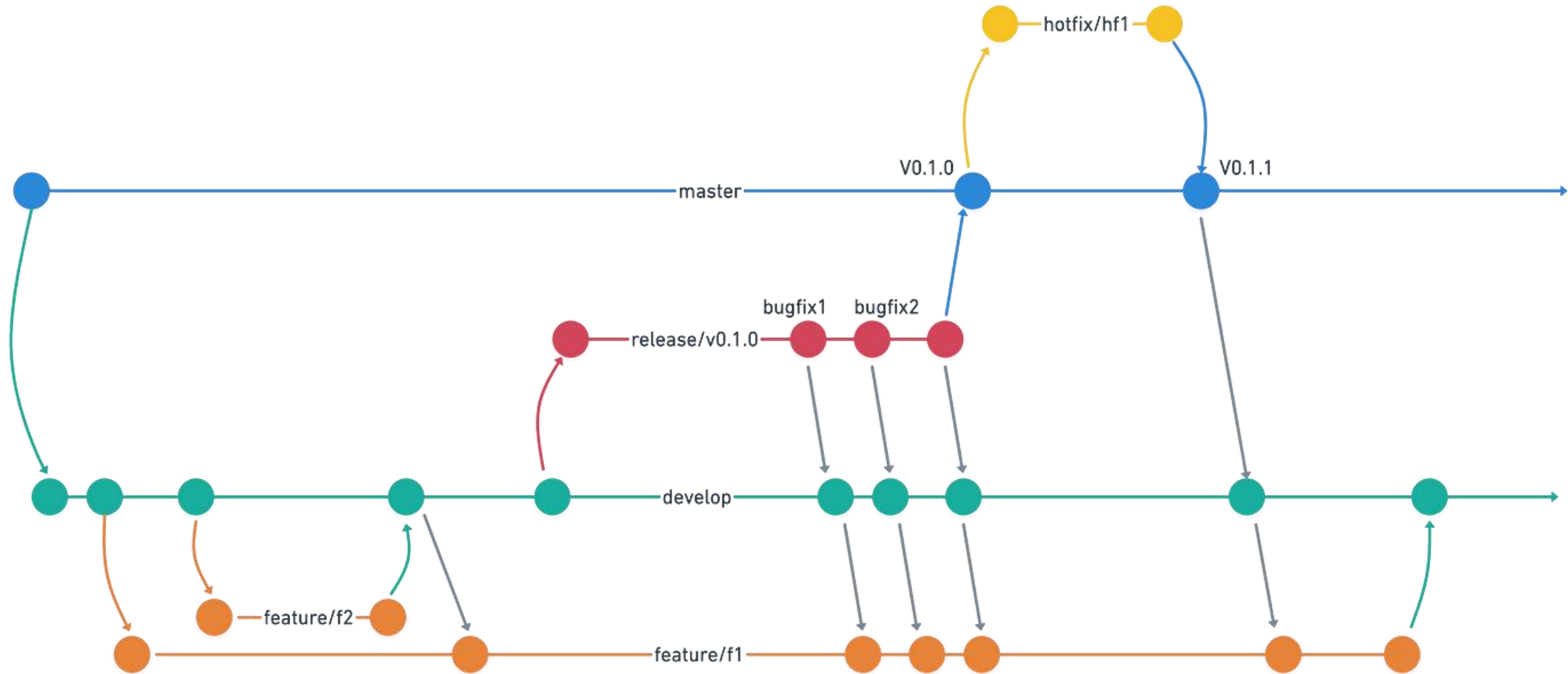
Patrones – Single branch



Patrones – Feature branches



Patrones – Git Flow



Recursos

Git - <https://git-scm.com/doc>

Git Cheat Sheet - <https://education.github.com/git-cheat-sheet-education.pdf>

GitHub - <https://docs.github.com>

GitHub Flow - <https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow>

Contacto

LinkedIn - www.linkedin.com/in/manuellandin

Email - manuel.landin@outlook.es