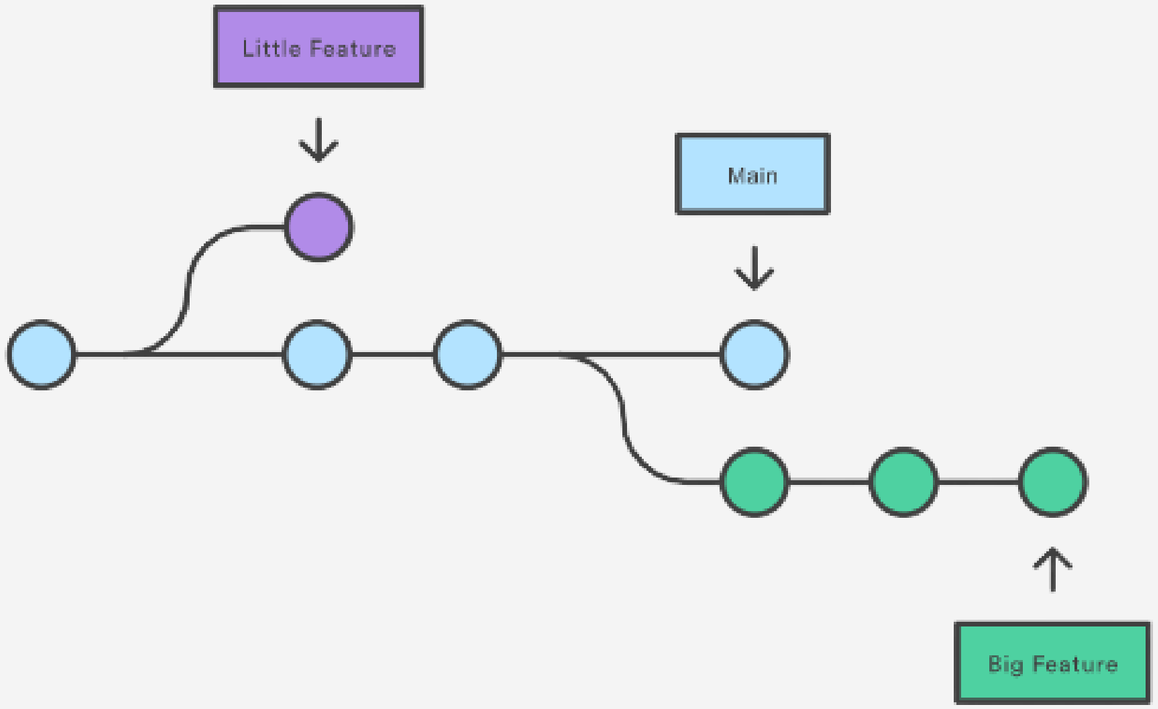
ARANXA GUADALUPE MARTÍNEZ OJEDA

Academia Java

Conceptos de GIT

BRANCHES

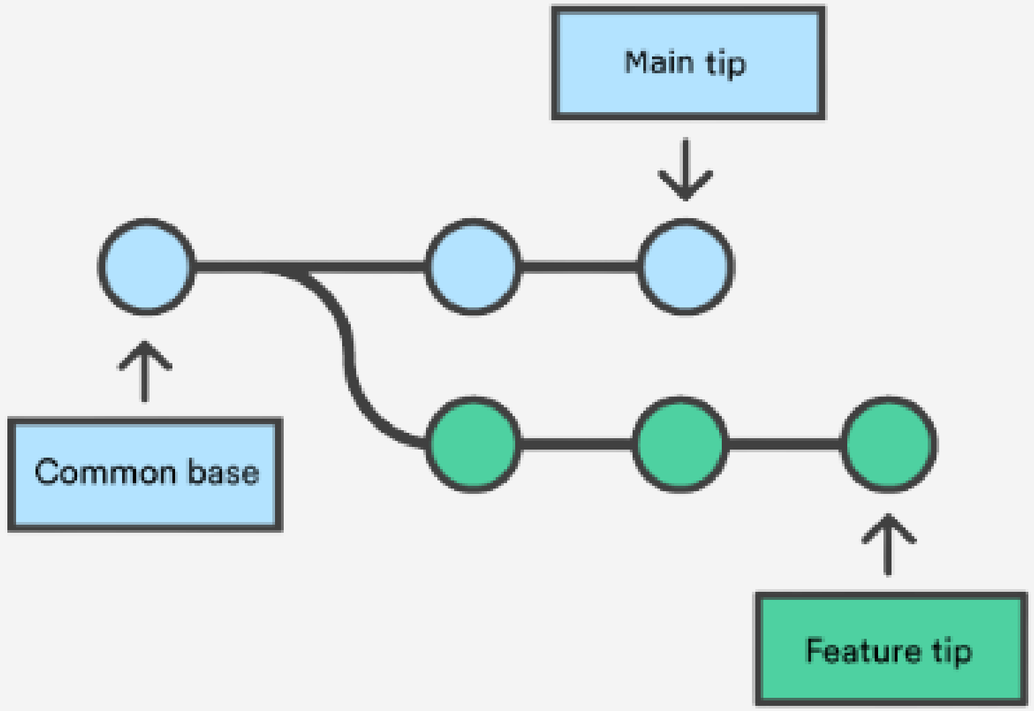


Las ramas son la forma de hacer cambios en nuestro proyecto sin afectar el flujo de trabajo de la rama principal.

Una rama representa una línea independiente de desarrollo. Las ramas sirven como una abstracción de los procesos de cambio, preparación y confirmación.
El diagrama representa un repositorio con 2 líneas de desarrollo aisladas, una para una función pequeña y otra para una función mas extensa.

> **git branch**: Nos muestra las ramas existentes
> **git branch <name>:** Crea una rama con el nombre asignado
> **git branch -d <name>:** Elimina la rama especificada
> **git checkout -b <new-branch>:** Se crea y nos mueve a la nueva rama

MERGE



El comando git merge fusionará cualquier cambio que se haya hecho en la base de código en una rama separada de tu rama actual como un nuevo commit.

Por ejemplo, si estás trabajando actualmente en una rama llamada dev y deseas fusionar los nuevos cambios que se hayan realizado en una rama llamada new-features, ejecutarías el siguiente comando:

git merge new-features

> **git merge <name>**: Fusiona las ramas
> **git checkout <name>:** Nos desplazamos a la rama especificada

CONFLICTS

Si dos ramas que tratamos de fusionar anteriormente han realizado cambios en la misma parte del proyecto. Git no sabrá cual de estas 2 versiones utilizar por lo que nos mandará un conflicto para poder resolverlos manualmente.

- **Git no puede iniciar la fusión**

Una fusión no podrá iniciarse cuando Git vea que hay cambios en el directorio de trabajo o en el área de preparación de tu proyecto actual. Como tal, Git no puede iniciar la fusión porque estos cambios pendientes podrían ser sobrescritos por las confirmaciones que estas fusionando

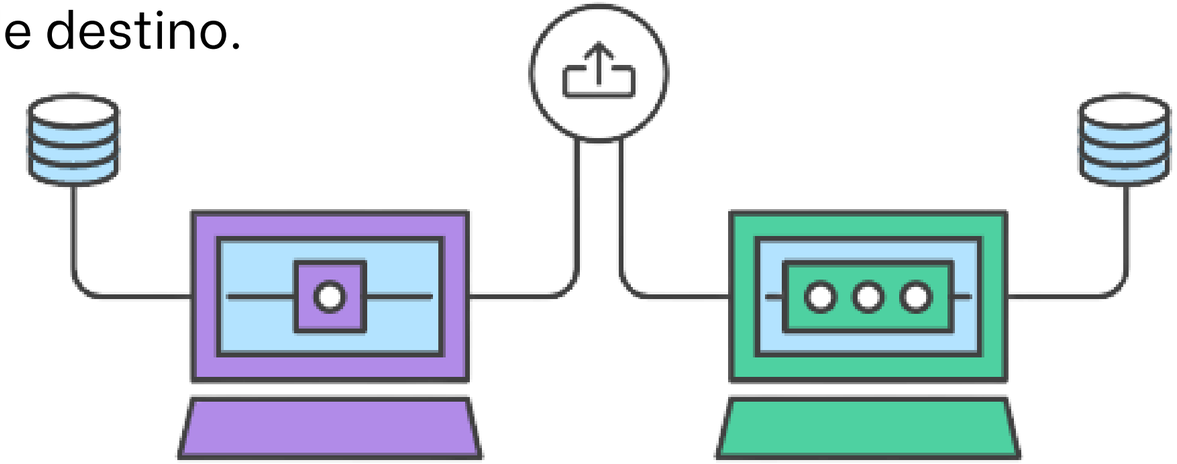
- **Git falla durante la fusión**

Una falla durante una fusión indica un conflicto entre la rama local actual y la rama que estas fusionando. Esto indica un conflicto con el código de otro desarrollador. Git hará todo lo posible para fusionar los archivos, pero dejará las cosas para que las resuelva manualmente en los archivos en conflicto.

PULL REQUEST

Las pull requests son una funcionalidad que facilita la colaboración entre desarrolladores. Esto sirve para verificar los cambios propuestos antes de integrarlos a un proyecto oficialmente.

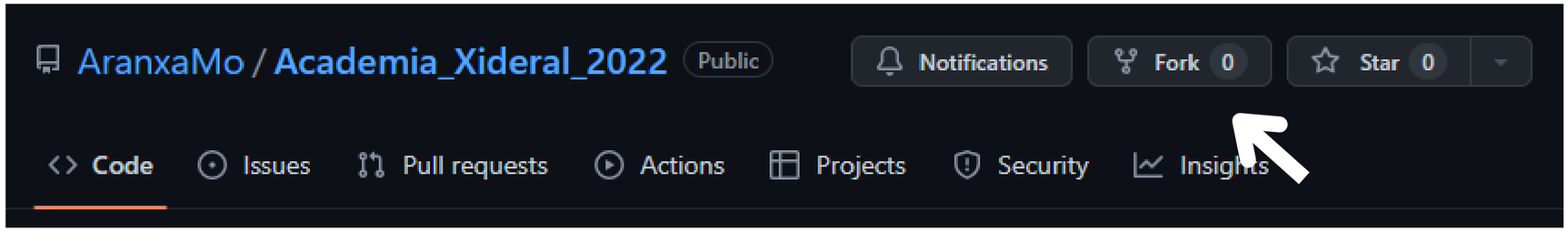
Cuando realizas una pull request, lo que haces es solicitar que otro desarrollador (por ejemplo, el mantenedor del proyecto) incorpore (o haga un pull) una rama de tu repositorio al suyo. Por tanto, para realizar esta solicitud, debes proporcionar la siguiente información: el repositorio de origen, la rama de origen, el repositorio de destino y la rama de destino.



FORK

Un fork es como una bifurcación del repositorio completo, tiene una historia en común, pero de repente se bifurca y pueden variar los cambios, ya que ambos proyectos podrán ser modificados en paralelo y para estar al día un colaborador tendrá que estar actualizando su fork con la información del original.

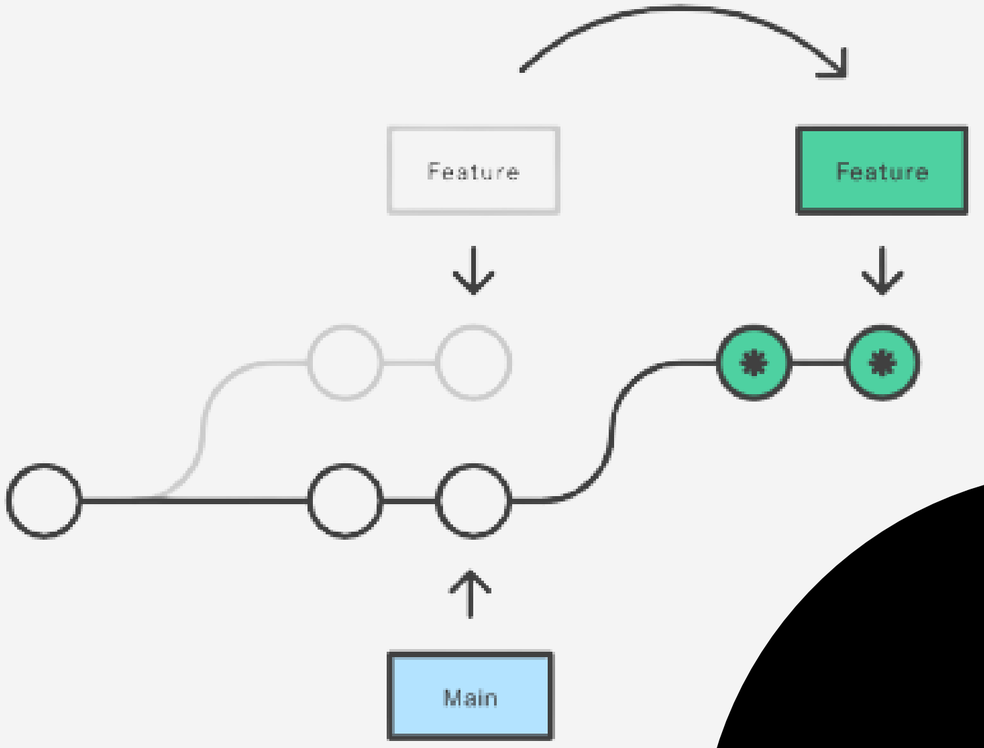
Al hacer un fork de un proyecto en GitHub, te conviertes en dueño del repositorio fork, puedes trabajar en éste con todos los permisos, pero es un repositorio completamente diferente que el original, teniendo alguna historia en común.
os forks son importantes porque es la manera en la que funciona el open source



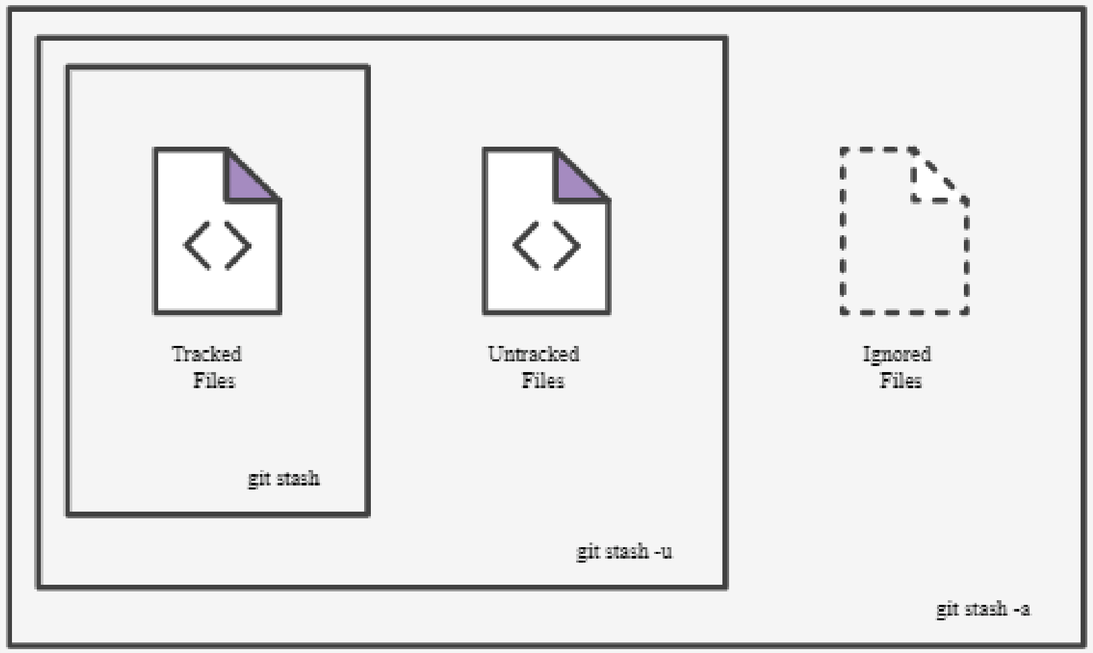
REBASE

Git Rebase se define como un comando de la plataforma de Git enfocado en integrar modificaciones de una rama a otra, a través de reorganizar o cambiar la base de una rama de *commit* a otra. Esto deriva en que parezca que se ha creado la rama desde un *commit* diferente.

Se recomienda que esta funcionalidad solo debe de realizarse en repositorios locales.
Usamos el comando:
git rebase <base>



STASH



> **git stash save** “Mensaje opcional“: Guarda los cambios y revierte el directorio a como se veía en el ultimo commit
> **git stash list:** Devuelve una lista de las capturas guardadas en el stash
> **git stash apply <stash name>:** Aplica los cambios y deja una copia en el stash
> **git stash pop <stash name>:** Aplica los cambios y elimina los archivos del stash

Git tiene un área especial llamada “stash“ donde podemos almacenar temporalmente una captura de un cambio sin enviarlos al repositorio para que puedas trabajar en otra cosa y, más tarde, regresar y volver a aplicar los cambios más tarde.
Está area esta separada del direcorio de trabajo (working directory), del área de preparación (staging area) o del repositorio.

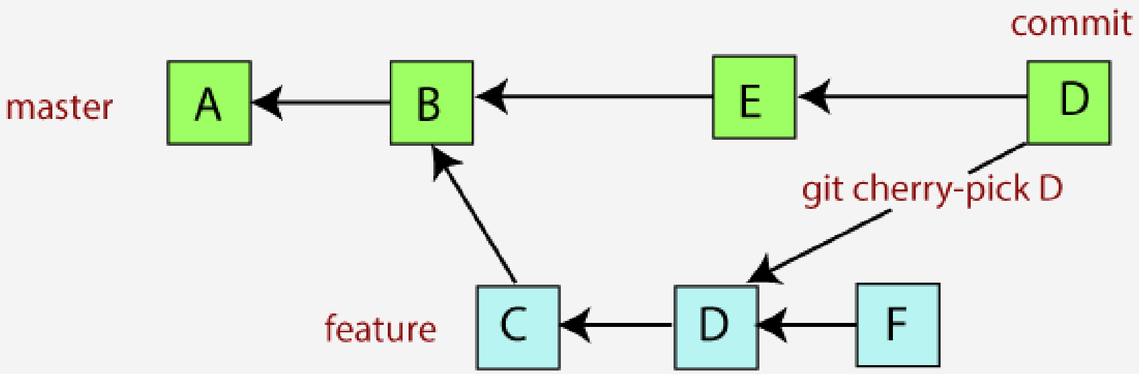
Guardar los cambios en stashes resulta práctico si tienes que cambiar rápidamente de contexto y ponerte con otra cosa, pero estás en medio de un cambio en el código y no lo tienes todo listo para confirmar los cambios.

CLEAN

El comando clean actúa en archivos sin seguimiento, este tipo de archivos son aquellos que se encuentran en el directorio de trabajo, pero que aún no se han añadido al índice de seguimiento de repositorio con el comando add.
La ejecución del comando predeterminado puede producir un error. La configuración global de Git obliga a usar la opción force con el comando para que sea efectivo. Se trata de un importante mecanismo de seguridad ya que este comando no se puede deshacer.

> **git clean:** Elimina los archivos que no están bajo el control de versiones. Junto con este comando existen ciertos flags:
> **-n:** No remueve nada, solo muestra los archivos que se van a eliminar
> **-f:** Elimina los archivos que no se encuentran versionados
> **-d:** Le indica a Git que también se quieren eliminar los directorios sin seguimiento
> **-x:** Indica a Git que también se eliminaran los archivos ignorados.
> **git clean --dry-run:** Nos ayuda a saber que archivos vamos a borrar

CHERRY PICK



> **git cherry-pick <hash code>:** Tomamos el commit que nos interese de acuerdo a su hash code.

Hay que tener en cuenta que cherry-pick crea un nuevo commit, por lo que antes de usarlo no debemos de tener ningún archivo modificado que no haya sido incluido en un commit.

Git Cherry-pick es un comando que permite tomar uno o varios commits de otra rama o branch sin tener que hacer un merge completo. Así, gracias a cherry-pick, podríamos aplicar los commits relacionados con nuestra funcionalidad en la rama master sin necesidad de hacer un merge.

Es considerado una mala práctica ya que cuando hagamos el merge de todas las ramas con fixes y nuevos features nos vamos a tomar con muchos conflictos que habrá que resolver