

# Examen semana 4

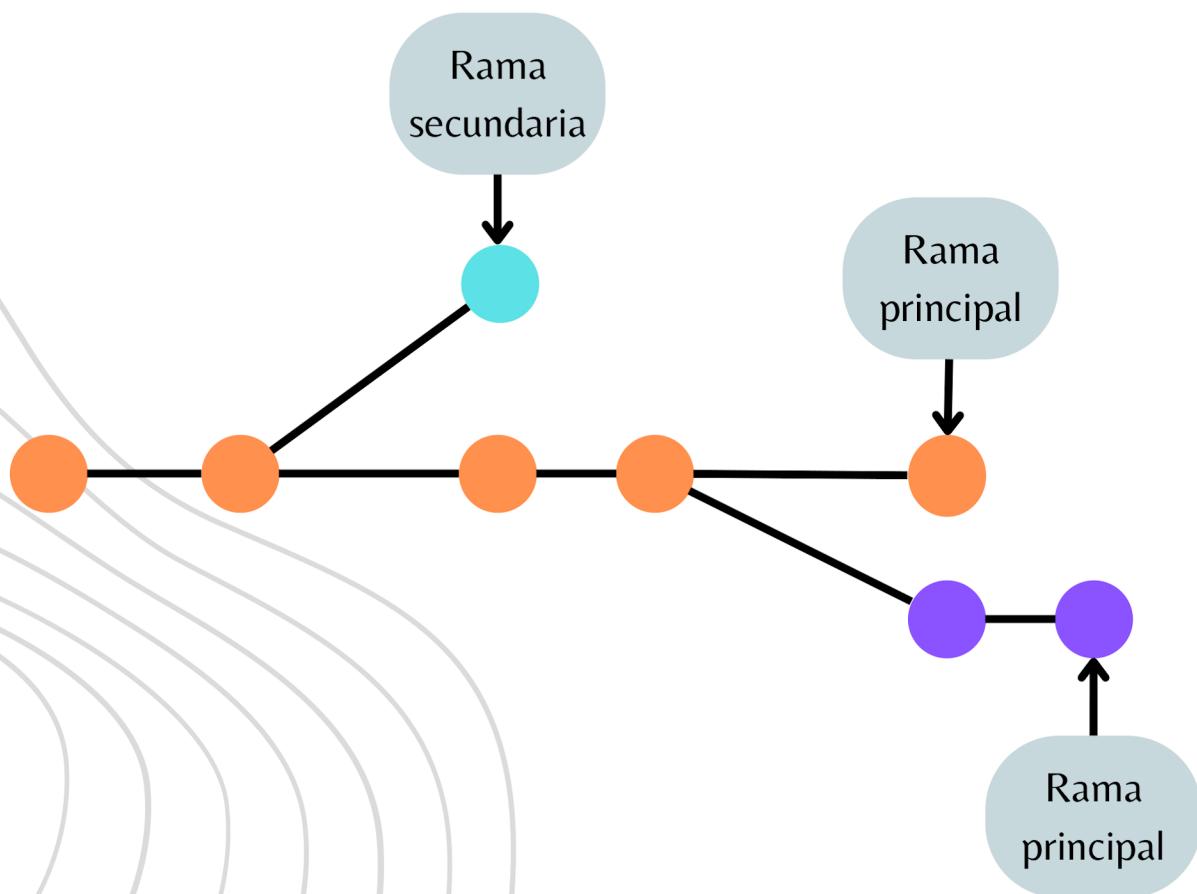
ACADEMIA JAVA  
XIDERAL

Miguel Angel Ortega  
Hernandez

# Trabajando con repositorios en Github

## G I T   B R A N C H

Una rama representa una línea independiente de desarrollo, la cuál, servirá para implementar mejoras, probar ideas y experimentar con el código sin afectar el proyecto principal.



# COMANDOS PRINCIPALES AL TRABAJAR CON RAMAS

- Comando utilizado para crear una rama:

```
git branch <nombre-rama>
```

- Comando para cambiar de rama:

```
git checkout <nombre-rama>
```

- Comando para eliminar rmas:

```
git branch -d <nombre-rama>
```

- Listar todas las ramas del repositorio:

```
git branch
```

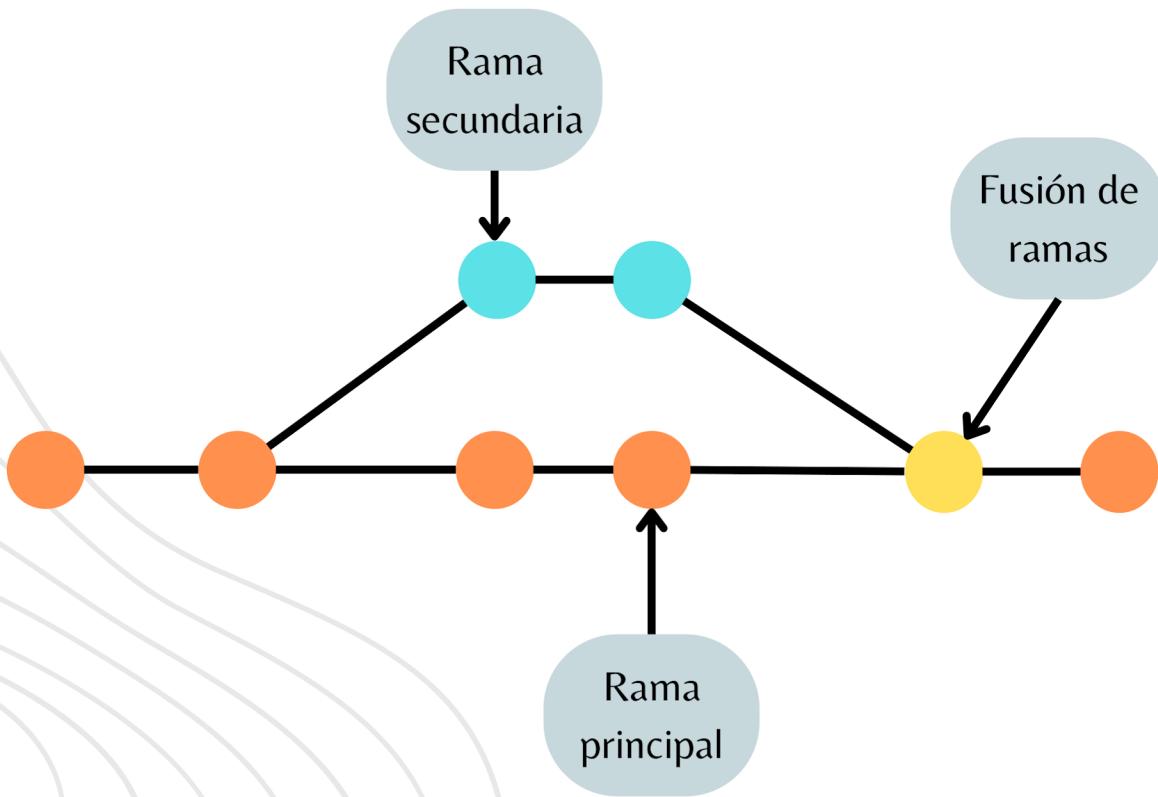
- Listar ramas fusionadas a la rama actual

```
git branch --merged
```

# GIT MERGE

una vez que se han realizado todos los cambios necesarios dentro de una rama y se confirma que son correctos, se deben de fusionar con la rama principal para integrarlas en una sola rama.

Al fusionar dos ramas, los cambios que se hayan realizado en la rama secundaria pasarán a integrarse a la rama principal.



Al hacer la fusión de ramas se pueden dar dos resultados diferentes:

- Fast-Forward: La fusión se hace automática, no hay conflictos por resolver.
- Manual Merge: La fusión hay que hacerla manual, para resolver conflictos de duplicación de contenido.

# ¿COMO SE REALIZA UNA FUSIÓN DE RAMAS?

## Fast-forward

- Principalmente, debemos de situarnos en la rama en la que quedará el contenido fusionado

```
git checkout <rama-principal>
```

- Después se debe ejecutar el comando para fusionar la rama secundaria.

```
git merge <rama-secundaria>
```

## Manual Merge (conflicts)

Cuando se realiza una fusión de ramas en las que se ha modificado el mismo archivo y específicamente las mismas líneas de código en ambas ramas, podemos tener conflictos al hacer la fusión, ya que, git no sabrá cuál es la versión que debe utilizar.

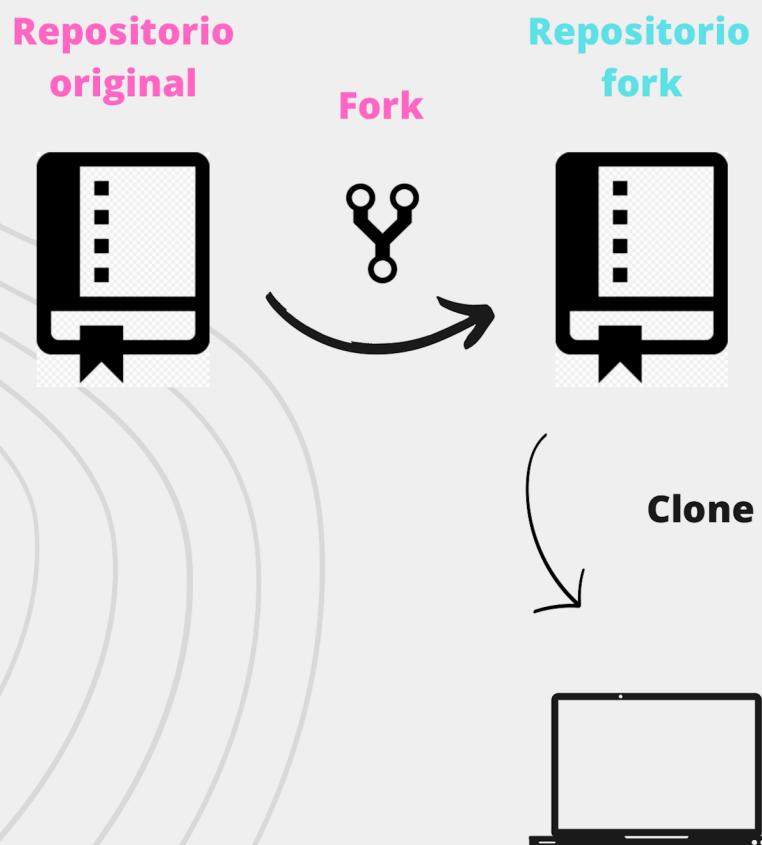
Cuando esto ocurre, git se detendrá antes de la confirmación de la fusión y creará un archivo temporal que contiene ambas versiones, al ocurrir esto, deberemos seleccionar manualmente las porciones de código con las que nos quedaremos y posteriormente realizar un commit.

# Comandos avanzados de Git

## FORK

Cuando un repositorio es subido a Github, es posible que este pueda ser copiado por otras personas, esto nos permitirá realizar modificaciones evitando trabajar directamente en el repositorio original.

La función fork, nos permite realizar una copia exacta de un repositorio que se encuentra en Github, esto permitirá clonar el repositorio que ha realizado el Fork y realizar modificaciones sin afectar el proyecto original.

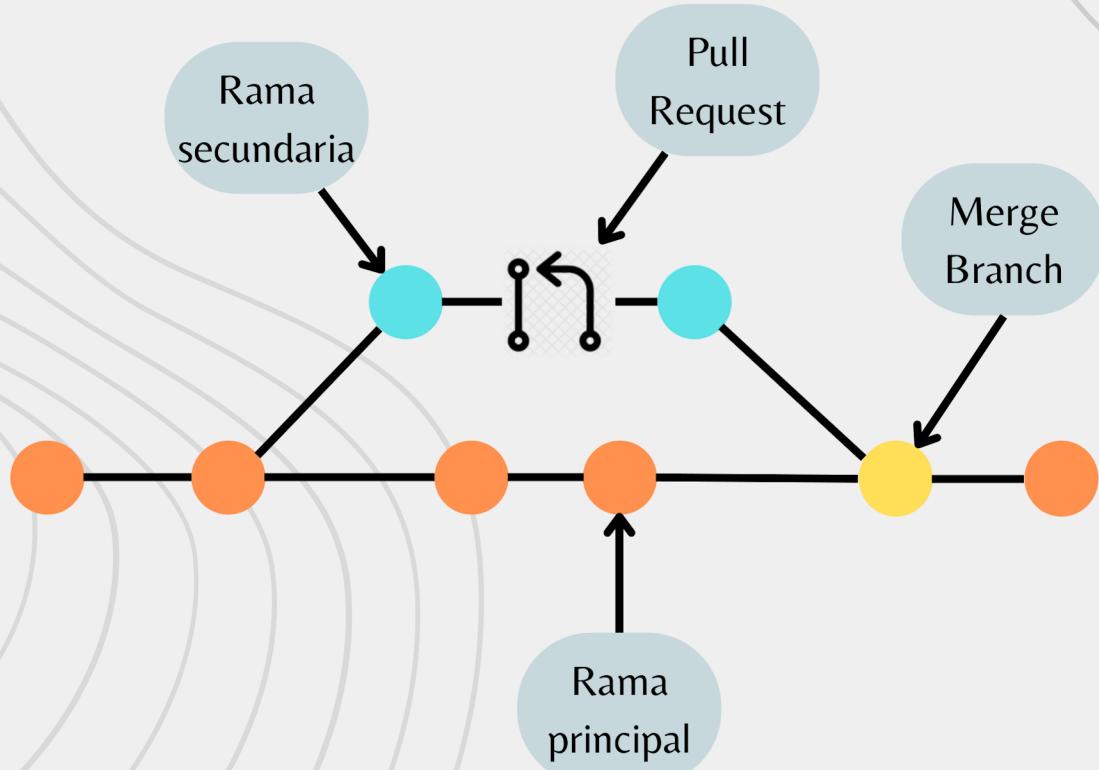


# PULL REQUEST

Cuando varios desarrolladores se encuentran trabajando en distintas partes del proyecto al mismo tiempo, en algún momento será necesario integrar todos los cambios al proyecto original.

La opción Pull Request es una solicitud que se realiza desde una cuenta que ha realizado un Fork a un proyecto original, esto con la intención de integrar las nuevas funcionalidades y fusionarlas con la rama original.

El administrador del proyecto original recibirá la solicitud de la petición y podrá revisar los cambios que se desean integrar, añadir comentarios y discutir los cambios propuestos, si se decide que los cambios son correctos, entonces podrá aceptar la solicitud y los cambios se integrarán al proyecto original.

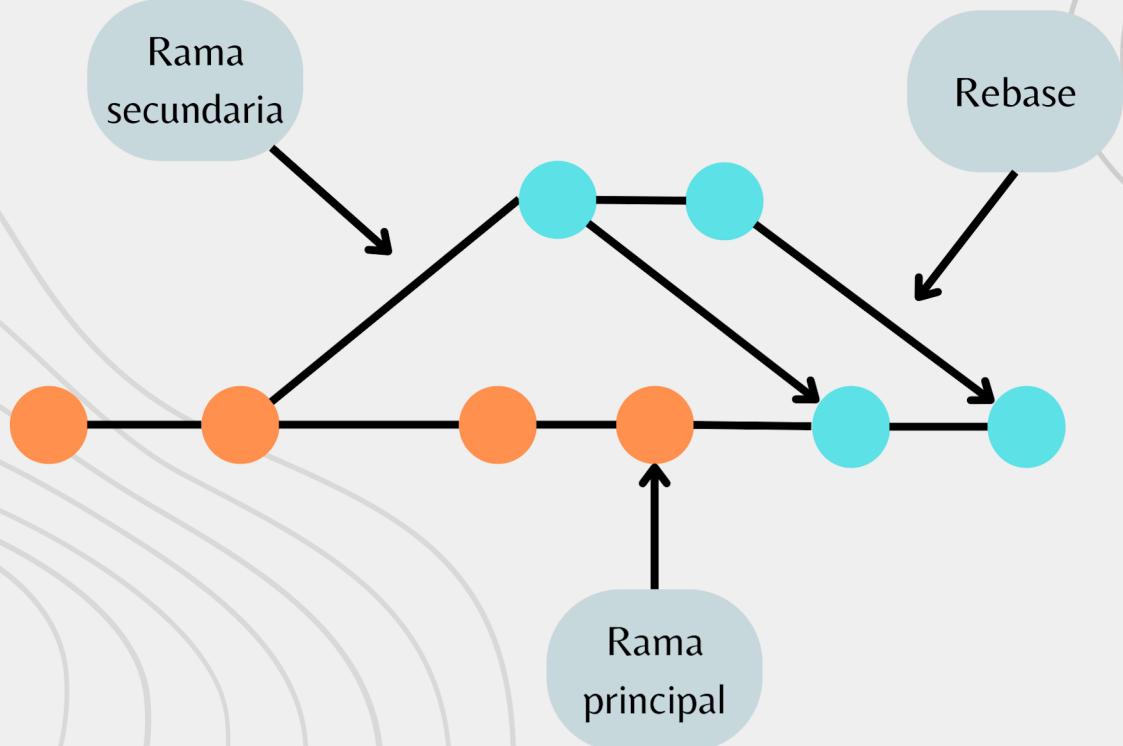


# REBASE

El comando rebase es utilizado para mover en su totalidad una rama por delante de otra rama, esto quiere decir que la rama secundaria se fusionará con la rama principal, pero se mantendrá todo el historial de commits que se hayan realizado en la rama secundaria.

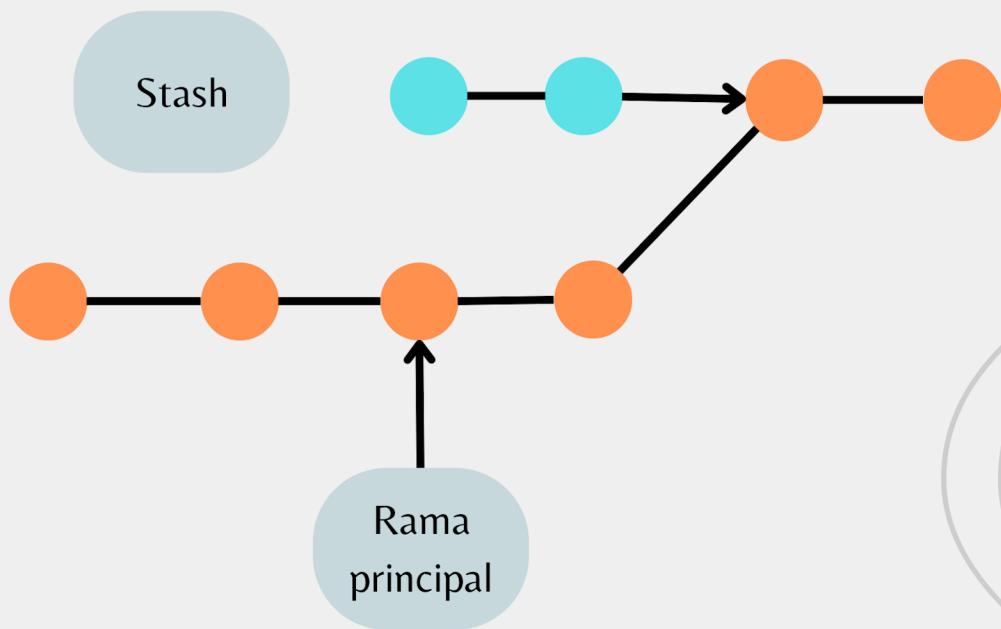
- Para hacer un rebase, primero tenemos que situarnos en la rama en la que queremos hacer el rebase y luego ejecutar el siguiente comando:

```
git rebase <rama-principal>
```



# STASH

El comando stash permite almacenar temporalmente los cambios que se hayan realizado en el código y de esta manera poder trabajar en otra parte del proyecto. Es posible acceder a los cambios que son almacenados en el stash en cualquier momento que decidamos.



- Para almacenar cualquier parte de nuestro proyecto en el stash necesitamos ejecutar el siguiente comando:

```
$ git stash
```

- Para volver a aplicar los cambios de un stash podemos utilizar el siguiente comando:

```
$ git stash pop
```

- Para aplicar y seguir conservando los cambios en el stash aplicamos el siguiente comando:

```
$ git stash apply
```

## CLEAN

Cuando nos encontramos trabajando en un proyecto, podemos añadir archivos que no pertenecen a nuestro directorio de trabajo, estos archivos son los que aún no se han añadido con el comando "add".

El comando Clean removerá todos estos archivos a los que no se les ha dado seguimiento o fueron añadidos a nuestro flujo de trabajo.

- Para remover los archivos a los que no se les ha dado seguimiento podemos ejecutar el siguiente comando:

```
$ git cleean
```

- Con el siguiente comando podemos revisar los archivos a los que no se les ha dado seguimiento:

```
$ git cleean --dry-run
```

## CHERRY PICK

Este comando nos permitirá añadir Commits de una rama a otra sin la necesidad de hacer un Merge o un Rebase. De esta manera solo se añadirán funcionalidades específicas a la rama que queramos.

- Para realizar un Cherry Pick primero debemos situarnos en la rama en la que se aplicará el commit y luego ejecutar el siguiente:

```
$ git cherry-pick <hash del commit>
```

# Buenas prácticas API REST

## HTTP

Es un protocolo de transferencia que nos permite la transmisión de documentos a través de la red. Fue diseñado para la comunicación entre navegadores y servidores web.

Está basado en el clásico modelo cliente-servidor, en el cual un cliente realiza una petición a un servidor y espera hasta que se recibe una respuesta del mismo.

Cada petición es realizada por medio de una serie de verbos que sirven para identificar el tipo de petición que se va a realizar.

## PRINCIPALES VERBOS HTTP

Dentro de los principales verbos http podemos encontrar los siguientes

**GET**: es un verbo de solo lectura, se utiliza para transmitir los datos identificados por la URL.

**POST**: se utiliza para guardar la representación de un nuevo recurso.

**PUT**: es utilizado cuando se quiere actualizar un recurso registrado por la URL.

**DELETE**: se utiliza para eliminar un recurso en la URL registrada.

## EJEMPLOS EN LA CREACION DE URL'S

- Regresa una lista de usuarios

**GET** **/persons**

- Regresa un usuario al suministrar un identificador.

**GET** **/persons/{username}**

- Crea un nuevo usuario en la base de datos.

**POST** **/persons**

- Actualiza un usuario existente en la base de datos.

**PUT** **/persons/{username}**

- Elimina un usuario de la base de datos.

**DELETE** **/persons/{username}**