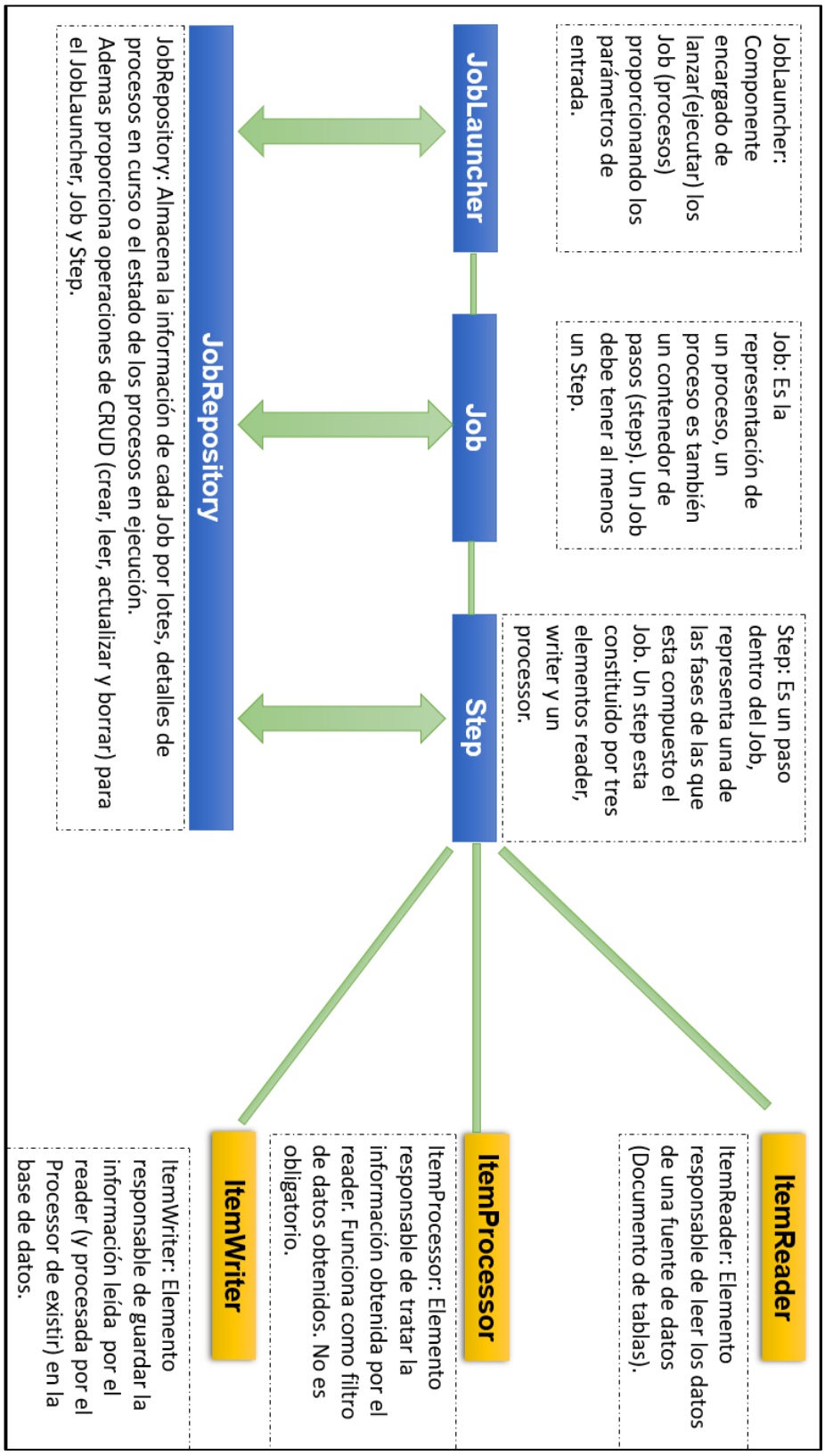
Javier de Jesús Pérez Santiago

Ultimo Examen Beca Xideral

1- **Cliente Rest: Crea un proyecto en Spring Boot que tenga la capacidad de generar vistas y consumir un servicio Rest.**
2- **Crea un Proyecto en Spring Batch**

3- Crea un diagrama de los componentes de Spring Batch y explícalo



Javier de Jesús Pérez Santiago

4- Explicar los comandos: Pull request, Fork, Stach, Rebase, Cherry-pick y Clean.

Git rebase

Es un comando utilizado para la reorganización de commits de dos ramas. Se asemeja a merge, sin embargo, merge solo combina los commit de las ramas sin reorganizar su historia.

Para reorganizar las commit:

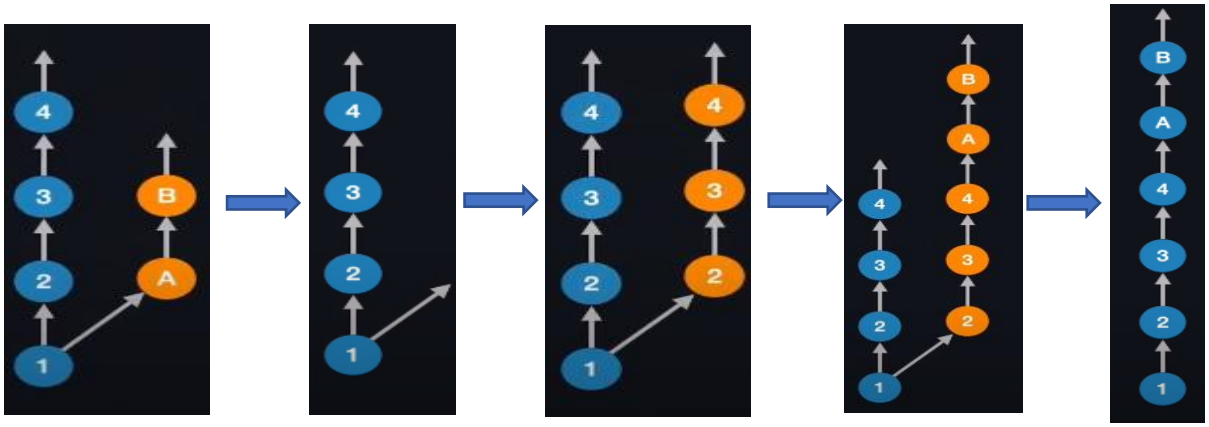
1) Tenemos una rama master (círculos azules) y una rama feature (círculos amarillos).

2) Quita los commit de la rama feature

3) Coloca los commit de la rama master

4) Devuelve hasta el final los commit de la rama feature

5) Por último, se hace un merge en la rama master, causando un fast forward que es un merge que no necesita un commit.



Para hacer un rebase:

Se observan los comandos en el archivo de texto: Git comandos/Git Rebase/Rebase.txt

Primero tenemos una rama master en la que hemos creado dos archivos html, cada uno fue añadido al índice con “git add” y guardado en la base de la rama master con “git commit”.

Hasta este punto tenemos “dos commit”, usamos “git checkout -b feature” para crear una rama clon de la rama master que se llama feature, esta contiene los dos commit de la rama master. Regresando a la rama master con “git checkout” creamos otro archivo html, lo añadimos con “git add” y lo guardamos con “git commit”.

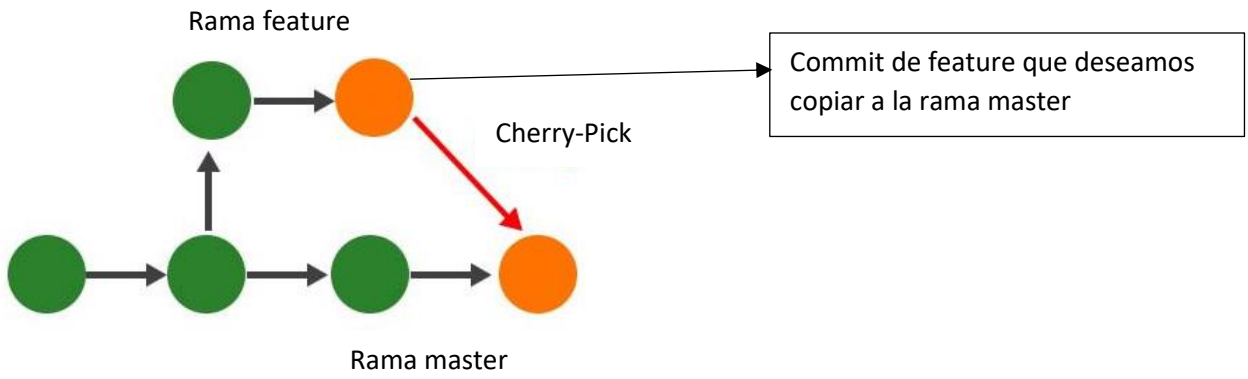
Regresamos a la rama feature con “checkout”, crearemos dos archivos html, para cada uno se hizo un “git add” y se les creo su propio “commit”, ahora tenemos dos commit en la rama feature y en la rama master tenemos 3.

En la rama feature ejecutamos “git rebase” y observamos que sucedió con “git log --oneline --graph”, observando que se colocaron los commit de la rama master al inicio de la historia de la rama feature y al final los commit que ya tenía.

Regresamos a la rama master, utilizamos “git merge” causando un merge fast-forward que ya no me pide un commit, además de colocar en la rama master los commit de la rama feature hasta el final.

Git Cherry-pick

Comando de git utilizado para copiar un commit de alguna rama e insertarlo en otra rama.



Como ejecutarlo:

Los comandos se encuentran en el archivo de texto: Git comandos/Git Cherry-pick /Git Cherry-pick.txt

Empezamos en una rama master donde hemos creado dos archivos html, cada uno fue añadido al índice con “git add” y guardado en la base de la rama master con “git commit”.

Creamos una rama nueva con “git checkout -b feature”, en esta rama creamos un archivo html, lo añadimos al índice con “git add” y lo guardamos en la base del repositorio con “git commit -m “commit para master””.

Usamos el comando “git log –oneline” para visualizar los commit de la rama feature y ver su short SHA o también conocido como identificador corto del commit y copiamos este identificador (en este caso el Short Sha del commit con el comentario “commit para master”).

Regresamos a la rama master con “git checkout”, ejecutamos “git cherry-pick <Identificador Corto>” para insertarle una copia del commit de feature “commit para master”.

Con el comando “git log –oneline” observamos que se agrego el commit de feature en master.

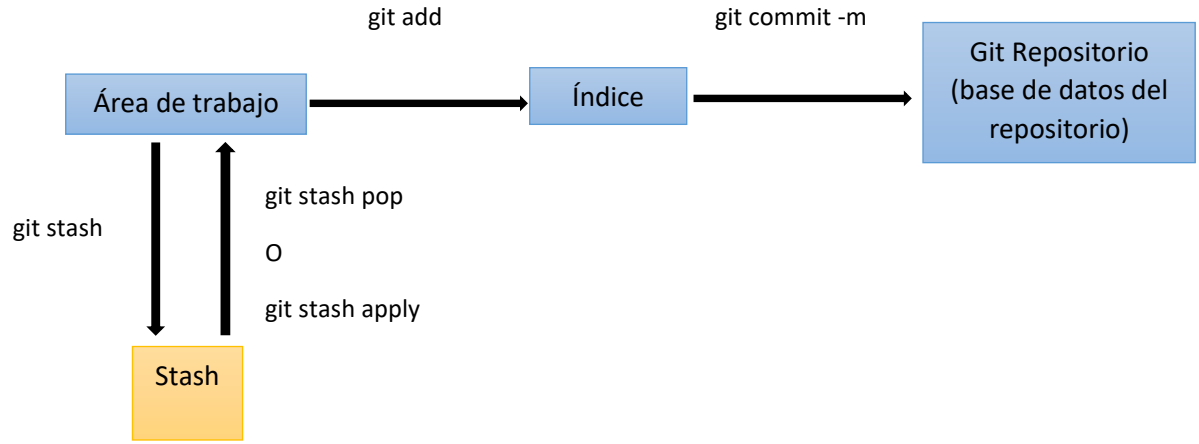
Javier de Jesús Pérez Santiago

Git stash

Comando para almacenar cambios temporales locales en Git.

Git tiene un área que se llama stash, donde puede almacenar temporalmente los cambios sin necesidad de guardarlos en la base de datos del repositorio. Esta área esta aparte del área de trabajo, el índice y la base del repositorio.

Nota: Solo guardara cambios en el stash mientras el archivo modificado ya se encuentre en el historial del repositorio.



Como hacer Git stash:

Los comandos se encuentran en el archivo de texto: Git comandos/Git stash/Git stash.txt

Ubicándonos en la rama master, esta contiene un archivo txt con el nombre “archivo” que ya se encuentra en la historia de este repositorio.

Realizamos un cambio con “notepad archivo.txt” y guardamos, para hacer el stash se ejecuta “git stash save “hice un stash””, esto mandara una copia del archivo a la zona stash, con “git stash list” podemos observar una lista de stash que se tienen en la zona stash con su identificador en este caso “stash{0}” para el stash que se acaba de crear.

Con “git stash show stash@{0}” se observan con mas detalles que tipos de cambios se guardaron en el stash.

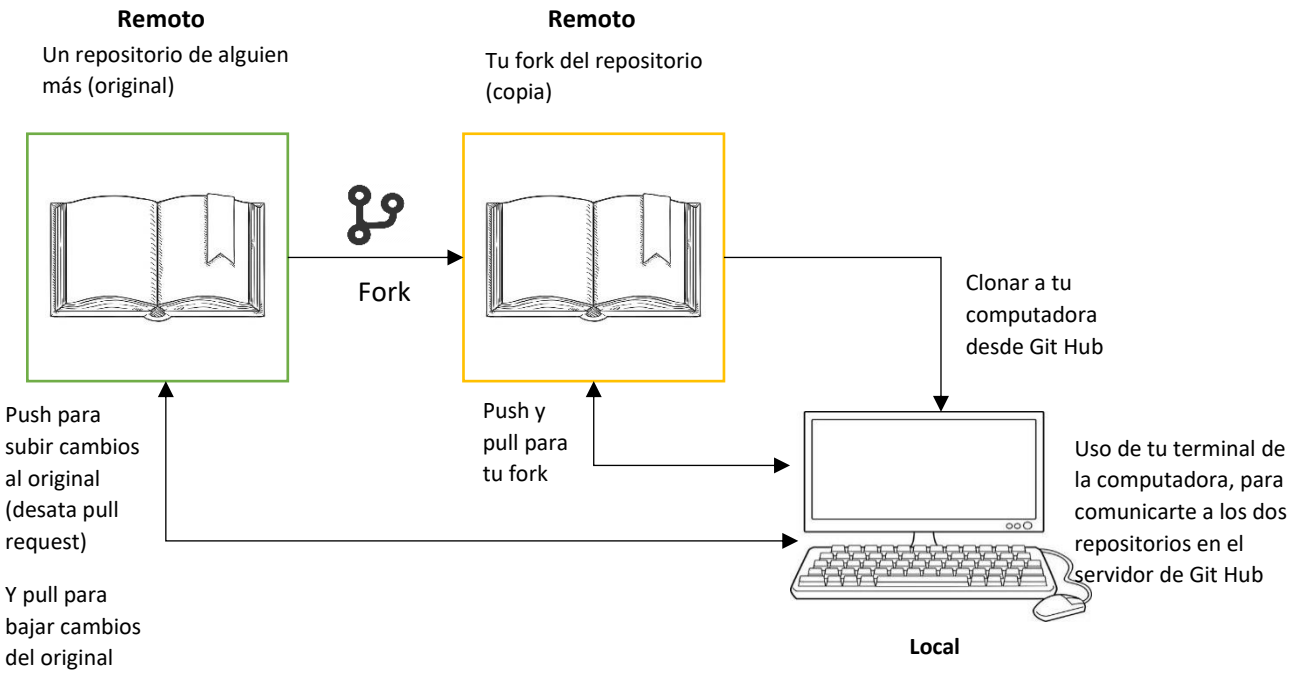
Ahora podemos hacer dos cosas para guardar estos cambios en nuestro repositorio:

Utilizamos “git stash apply stash@{0}” para aplicar los cambios en el repositorio y dejar una copia del archivo en el stash o podemos usar “git stash apply pop stash@{0}”, para aplicar los cambios en el repositorio y eliminar los archivos del stash.

En ambos casos al utilizar estos comandos, necesitamos usar un “git add” y un “git commit” para guardar los cambios en la base de datos del repositorio.

Si se necesita borrar un stash sin aplicar los cambios en el repositorio tenemos que ejecutar “git stash drop stash@{0}” o para borrar todos los stash utilizamos “git stash clear”.

Fork y pull request



Un Fork: es una copia en remoto de un repositorio Git de quien sea en nuestra cuenta de GitHub, en esta copia podemos realizar acciones sin restricción como pull y push de tus cambios realizados. Esta copia sigue conectada al repositorio original, por lo que es capaz de bajar los cambios que hayan hecho en el repositorio original y añadírselos a sí misma.

Contribuciones:

Git pull request: Nosotros podemos bajar de manera local el repositorio original, y podemos hacer cambios de manera local sin restricción. Por otro lado, si nos encontramos en un proyecto colaborativo o queremos dar una aportación a un proyecto en el Git Hub, ya sea para mejorarlo o para corregir un error, podemos hacer un push con esa actualización al repositorio original.

Sin embargo, los cambios no se verán reflejados hasta que alguno de los administradores de ese repositorio acepte esos cambios, a esta solicitud para agregar cambios se le conoce como pull request.

Este permitirá que los administradores acepten los cambios o los rechacen, también Git Hub permite que entre colaboradores y administradores se comuniquen y ofrecer un comentario de porque deberían ser aceptados o de porque fueron rechazados estos cambios.

Git clean

Es un comando utilizado para la eliminación de archivos, carpetas y archivos ignorados por el git ignore que no han sido añadidos al índice con “git add” o con “git commit”. Necesario para que git no nos obligue a añadirlos a la historia del repositorio.

Para ejecutar git clean:

Creamos un archivo nuevo con touch y una carpeta con mkdir.

Si usamos git status, git nos alertara que este archivo y la carpeta no han sido añadidos al directorio.

Con “git clean -n” nos mostrara los archivos que no están añadidos con “git add” o en la base de repositorio con “git commit”

Para borrar usamos:

Git clean -df: borrara archivos y carpetas sin seguimiento.

Git clean -dx –forcé: borrara archivos ignorados por gitignore.
