ACADEMIA JAVA

XIDERAL: SEMANA 3

NOMBRE: JOSE MANUEL JIMENEZ HDZ

Pregunta 1: Git Advance commands (Pull Requet, Fork, Rebase, Stach, Clean, Cherry-pick, etc).

-git merge

Se utiliza para combinar dos ramas, normalmente de un historial bifurcado. El comando git merge permite integrar tu rama de desarrollo de una feature en otra rama (normalmente tu rama principal de desarrollo o tu rama master).

La fusión de una rama sobre otra se realiza sobre la rama actual. Es decir, la rama actual se fusiona con la segunda rama y la fusión queda en la rama actual. La segunda rama no se ve afectada y se mantiene.

-git merge [branch_name]

En algunas ocasiones, Git encuentra datos que han sido modificados en ambos historiales, y no puede combinarlos automáticamente. En este caso, se crea un conflicto y Git solicitará la intervención del usuario para poder continuar.

Una vez solucionado el conflicto, no hace falta realizar un nuevo commit, bastará con indicar a Git que continue con el merge:

-git merge --continue

Si por el contrario, no somos capaces de resolver el conflicto o necesitamos la ayuda de alguien para ver con qué nos quedamos y qué borramos y no dispones de esa persona en ese momento, puedes abortar el proceso:

-git merge --abort

Si todo va bien y el merge es satisfactorio, se mezclarán e intercalarán los commits de las dos ramas por fecha de creación del commit. Es decir, si hemos creado una rama para una nueva feature que partía de master, y master ha recibido más commits mientras terminábamos el desarrollo, cuando se fusiones las dos ramas, los commits de cada una de ellas aparecerán en orden de creación, pudiendo aparecer intercalados en el historial.

-git rebase

Parecido a merge, Rebase es una utilidad de Git que se utiliza para integrar cambios de una rama a otra. Merge es una fusion de ramas y Rebase se considera una reorganización de las mismas. Dispone de 2 modos principales: «manual» e «interactivo».

La razón principal para hacer un rebase e lugar de un merge es mantener un historial del proyecto lineal. Es decir, si mientras has trabajado en una nueva feature, alguien de tu equipo ha realizado otra feature y ya la ha integrado en master, cuando vayas a integrar tu parte, se van a intercalar los commits tuyos con los de tu compañero. Rebase lo que haría es poner tus commits a continuación de los de tu compañero, reorganizando el historial.

El Rebase de una rama sobre otra se realiza sobre la rama actual. Es decir, la rama actual se reorganiza con la segunda rama y queda en la rama actual. La segunda rama no se ve afectada y se mantiene. Adicionalmente, podemos indicar que el Rebase sea interactivo con el flag --i.

-git rebase [branch_name]

-git rebase [branch_name] --i

En modo interactivo, en lugar de mover de forma ciega todos los commits a la nueva base, la reorganización te da la oportunidad (se abre un editor) de alterar los commits individuales en el proceso. Eso te permite limpiar el historial eliminando, dividiendo y alterando una serie existente de commits. Es como git commit --amend a la máxima potencia.

-git stash

El comando git stash almacena temporalmente (o guarda en un stash (una pila)) los cambios que hayas efectuado en el código (y añadidos al index sin commitear) en el que estás trabajando para que puedas trabajar en otra cosa y, más tarde, regresar y retomar el trabajo donde lo habías dejado.

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

Almacena tus cambios en un stash

-git stash

Almacena tus cambios en un stash y añade un nombre descriptivo

-git stash save "Montando el entorno de test"

Lista todos los stash

-git stash list

Recuperar el último stash

-git stash pop

Recuperar un stash concreto

-git stash pop --index 2

Crear una rama con los cambios que tienes

-git stash branch F/branch-name

Eliminar los stashes

-git stash clear

Almacenar en un stash sólo algunos archivos

-git stash -p

git cherry-pick

Resumiendo, hacer un cherry-pick es elegir un commit de una rama y aplicarla a otra. Con un ejemplo quizá se entienda un poco mejor: imagina que empiezas a desarrollar una nueva funcionalidad y a mitad del desarrollo un compañero/a va a echarte una mano, pero necesita parte de lo que ya has ido avanzando, por ejemplo crear una estructura de datos.

La sintaxis es sencilla. Debemos obtener el hash del commit que queremos y a continuación situarnos sobre la rama donde queremos aplicarlo:

-git cherry-pick [commit_hash]

git blame

El comando git blame se usa para examinar el contenido de un archivo línea por línea y ver cuándo se ha modificado cada línea y quién es el autor de las modificaciones. Muchos IDEs muestran esta información.

Sólo actúa sobre ficheros individuales, por lo que hay que indicar el archivo sobre el que queremos obtener información. El formato de salida de git blame se puede modificar con varias opciones de línea de comandos.

git blame [filename]

Restringir la salida a in intervalo de líneas (ej: de la 1 a la 15)

-git blame -L 1,15 [filename]

Para mostrar el e-mail del usuario que hizo las modificaciones en lugar de su nombre de usuario

-git blame -e [filename]

Ignorar los cambios en los espacios en blanco y tabulaciones

-git blame -w [filename]

Detectar las líneas que se han copiado o movido dentro del mismo archivo

-git blame -M [filename]

Detectar las líneas que se han movido o copiado desde otros archivos

-git blame -C [filename]

git bisect

El comando git bisect nos ayudará a encontrar bugs en nuestro código. A través del control de versiones podemos volver hasta un punto anterior donde sepamos que nuestro código funciona.

-git bisect start

-git bisect bad

-git bisect good [commit_hash_or_tag_version]

-git bisect reset

Git sabrá el número de commits que se han producido desde la última versión «buena» hasta el commit que hemos marcado como malo. Lo que hará es ir acotando, por ejemplo, si hay 100 commits se situará sobre el commit de en medio (el 50) y te preguntará si tu código funciona o no.

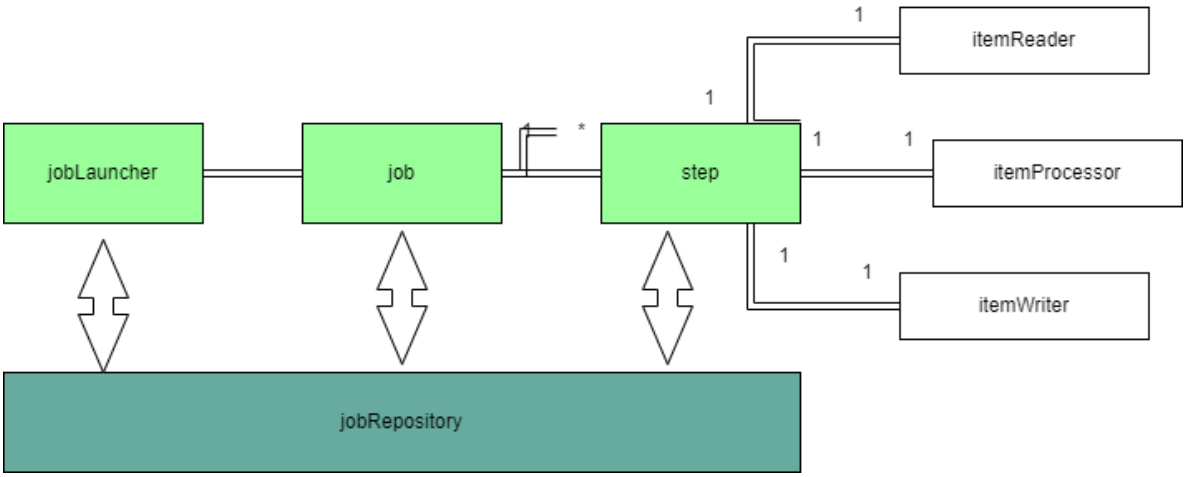
Si el código funciona deberás escribir git bisect good, en caso contrario git bisect bad. Git ahora entre los 50 commits donde sabe que existe el problema, volverá a acotar al medio (el 25) y volverá a preguntar. Así hasta dar con el commit en cuestión donde se introdujo el bug.

Pregunta 2: Explica y diagramar Spring batch.

¿Qué es Spring Batch?

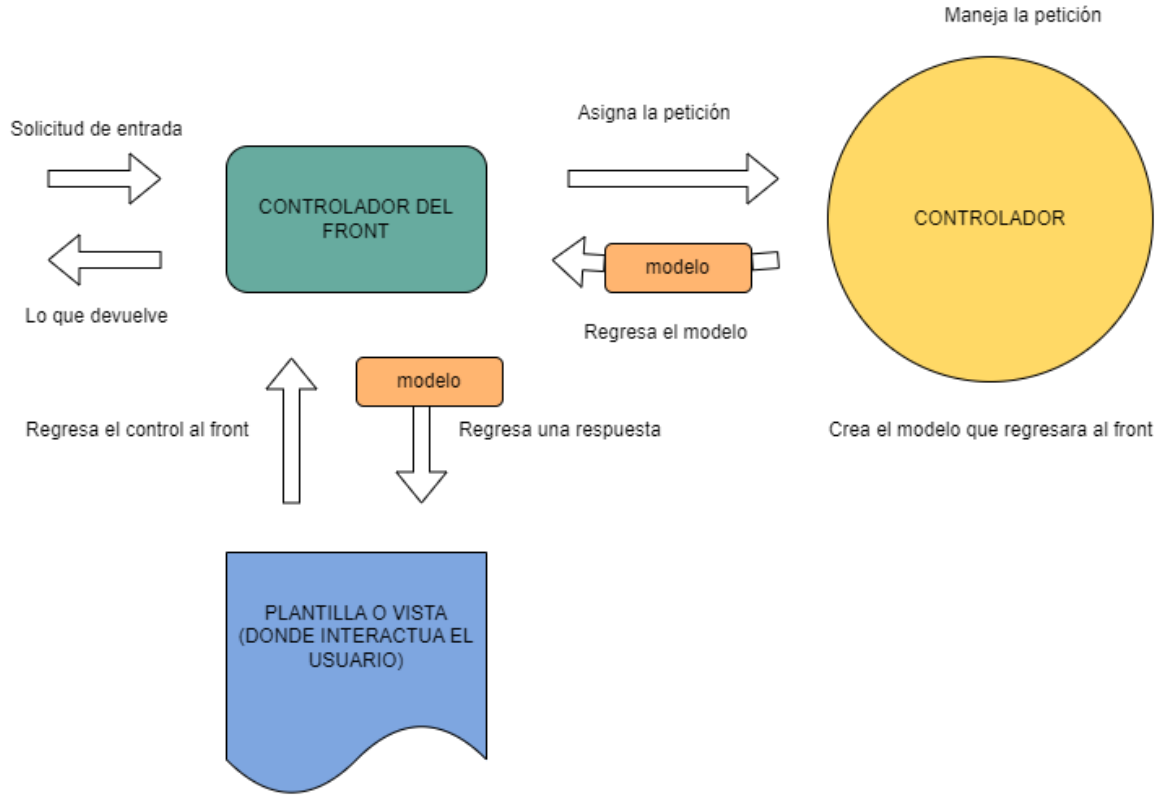
Spring Batch provee funcionalidades esenciales para el procesamiento de grandes volúmenes de registros, logs, transacciones, procesamiento de estadísticas, omisiones, reinicio de procesos, etc. Los procesos por lotes, o batch processing requieren normalmente grandes cantidades de recursos, spring batch nos permite extraer grandes cantidades de datos de diversos formatos, csv, XML, base de datos SQL y no SQL, para esto el framework nos provee de librerías específicas para cada formato, de igual manera nos permite depositar los datos en gestores de datos.

Mi explicación gráfica.



Veamos cómo funciona Spring Batch en términos muy generales. Los trabajos son los bloques de trabajo, los procesos que se ejecutan. Como se puede ver en el diagrama anterior, un trabajo puede tener uno o más pasos, cada uno de los cuales contiene un solo ItemReader, ItemProcessor y ItemWriter. Un trabajo debe ejecutarse o lanzarse desde un JobLauncher (el que usamos en las pruebas para probar, por ejemplo). Los metadatos del proceso en ejecución deben almacenarse en algún lugar, ese lugar es JobRepository.

Pregunta 3: Como usa SPRING el modelo MVC ---> Diagramar



Todas las solicitudes HTTP se enrutan a través del controlador frontal. En casi todos los frameworks MVC que siguen este patrón, el controlador frontal no es más que un servlet cuya implementación es propia del framework. Para Spring, la clase DispatcherServlet.

Los controladores frontales generalmente determinan a partir de la URL a qué controlador llamar para manejar la solicitud. Para ello se utiliza HandlerMapping.

Se llama al Controller, este ejecuta la lógica de negocio, obtiene los resultados y los devuelve al servlet, encapsulados en un objeto de tipo Model. Además, se devolverá el nombre lógico de la vista a mostrar (normalmente devolviendo una cadena, como en JSF).

El ViewResolver es responsable de averiguar el nombre físico de la vista que corresponde al nombre lógico del paso anterior.

Finalmente, el controlador frontal (DispatcherServlet) redirige la solicitud a la vista, que muestra el resultado de la acción realizada.