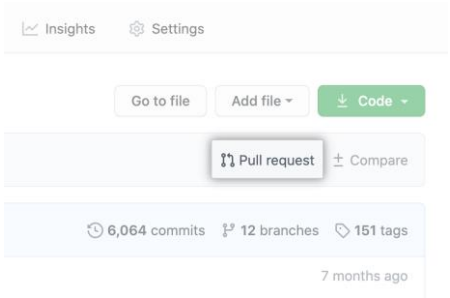
Explicación comandos avanzados GIT

Fork
Esta acción en git se refiere a la creación de un clone de un repositorio remoto en otro repositorio remoto propio de tal manera que puedas modificarlo, ya que es probable que no tengas los permisos de modificación del primer repositorio. De esta manera, puedes realizar cambios propios localmente y hacerles push a este nuevo repositorio. En el caso de GitHub para realizar el Fork debes entrar al repositorio que quieres clonar y encontrar el siguiente símbolo que permitirá crear en la cuenta propia una copia del repositorio.:

Pull Request
Generalmente los proyectos se trabajan entre varios desarrolladores por lo que un proyecto versionado tendrá varios contribuidores, pero debe tener un encargado de ir decidiendo como implementar esos cambios en la rama principal del proyecto. Por ejemplo para que tú hagas una funcionalidad a un proyecto es necesario que lo clones y crees una Branch propia en la cual trabajes, una vez que tus cambios estén listos haces *push* a esa nueva *branch* y haces un *pull request*, que básicamente es pedirle al encargado de monitorear la rama principal que revise tus cambios y haga merge con la rama principal. Es por eso que también se le conoce como *merge request*.

Comando después de hacer fork
```
$ git clone <Direccion HTTPS>
```

Luego se crea otra rama realizas cambios y haces *commit* y realizas un *push* al repositorio remoto. Cuando vuelvas a tu repositortio remoto presionas el botón *pull request.*



Rebase
Este se refiere al proceso de cambiar o combinar una serie de commits de una *branch* a otra. La diferencia con *merge* es que este último genera combina los *commits* de las dos *branches* y hace un *commit* a la branch principal. Rebase por su parte, quita los *commits* de la rama no principal, iguala los *commits* de ambas ramas y agregar al final los *commits* de la rama no principal. A partir de ahí se hace un merge, pero es un *merge fast forward*.

Crear una Branch dde cambio basada en main
```
$ git checkout -b newbranch main
```

Commit nuevo
```
$ git commit –a –m “Agregar cambio”
```

Esto tomara los *commits* en tu rama actual y los aplicara a la rama nueva

Crear una Branch dde cambio basada en main
```
$ git rebase <base>
```

Esto hare el rebase en la rama actual y la pasara en la base lo que es cualquier *commit* de referencia,

Stash
Es un comando que "congela" el estado en el que está el proyecto en el que estamos trabajando con cambios sin "commitear" y lo guarda en una pila provisional, para dar la posibilidad de recuperarlo más tarde. De esta manera el working tree queda limpiio y puedes cambiarte de reama sin problema, para después volver y seguir.

Comando
```
$ git stash
```

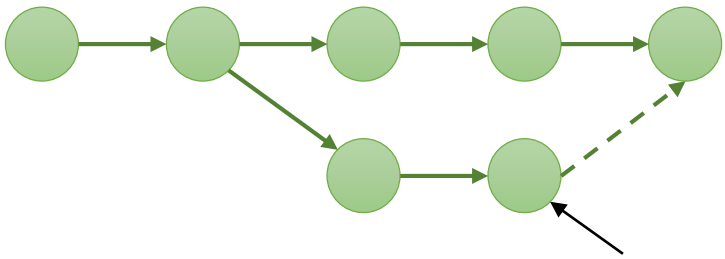
Esto dejara el estado en *"nothing to commit"* aunque lo anterior no se haya *commiteado*.

Clean
El comando sirve para deshacer sin embargo este comando trabaja con archivos que no se le han dado seguimiento.

Comando después de hacer fork
```
$ git clean -n>
```

La opción –n hará una prueba en caso de que no resulte forzar el borrado.

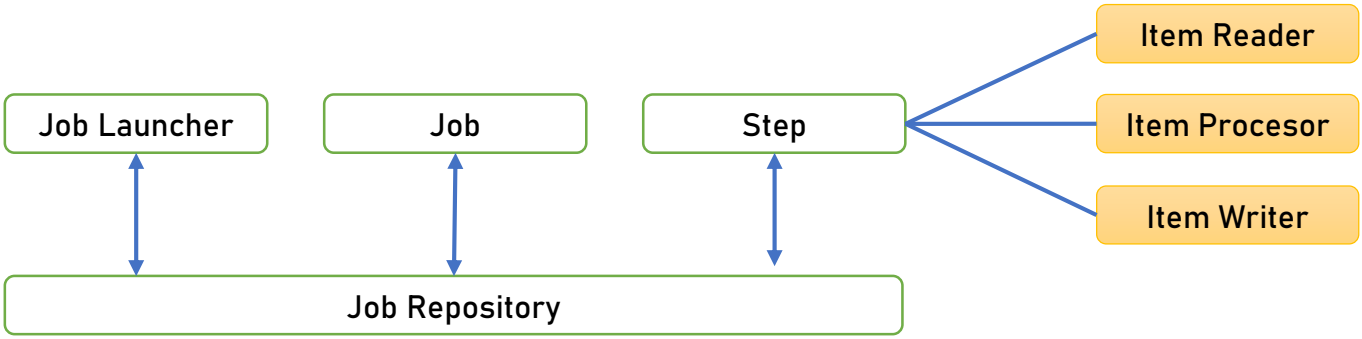
Cherry-pick
Elige ciertos *commits* de una rama y los lleva a otra. Esta función puede ser útil para deshacer cambios, por ejemplo una confirmación se puede aplicar por accidente en la rama equivocada y la puedes cambiar a la correcta. El riesgo del método es que se dupliquen los *commits* por lo que se debe hacer sólo en casos específicos.



Codigo del commit: "abcde"

Sol se necesita abcde commit
```
$ git cherry-pick abcde
```

Diagrama y explicación de Spring Batch



Spring Batch es un *framework* para el procesamiento por lotes para Java. Esto quiere decir que permite procesar grandes volúmenes de datos. Esto se logra porque lo va haciendo en lotes o en "pedazos".

Job Repository → Su responsabilidad es la persistencia de los datos relativos a los procesos, es decir, qué procesos están en curso o cuál es el estado de las ejecuciones.

Job Launcher → El componente que se encarga de lanzar los procesos, suministra los parámetros de entrada solicitados por los mismos procesos. Está compuesto de Jobs.

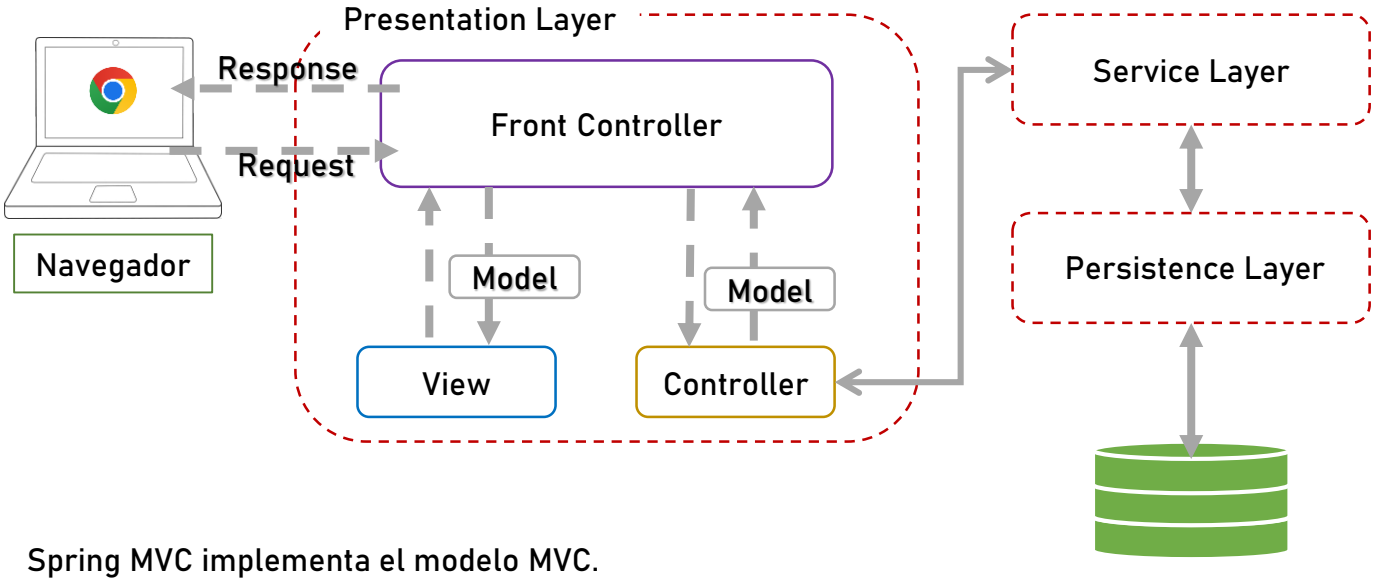
Job → La representación de un proceso. Un job puede contener uno o varios pasos, *steps*.

Job → La representación de un proceso. Un job puede contener uno o varios pasos, *steps.*

Step → Es un elemento independiente dentro de un job que representa una de las fases de un proceso. Los steps pueden estar compuestos de un *Reader*, un *Processor* y un *Writer*.

ItemReader → Es el encargado de la lectura por bloques. Esta lectura puede ser de una base de datos o ficheros como csv, xml, json etc.
ItemProcesor → Se encarga de transformar lo que se leyó.
ItemWriter → Este elemento hace lo opuesto al reader, se encarga de la escritura o la inserción en base de datos o ficheros.

Explicar cómo implementa Spring el modelo MVC



Spring MVC implementa el modelo MVC.
Es un patrón de arquitectura.

Presentation Layer
Es aquí dónde se va a ver el modelo MVC. Y se va a conectar con las otras Layers.

Front Controller
Es el *dispatcher*. Ocupa un rol importante en este modelo. Todo pasa por aquí, es el punto de entrada y salida, pero nunca se toca por el desarrollador es el responsable de la lógica de presentación.

Todos los *request* que vienen en la URL se conectan desde donde ahí se indica. También contiene indicado dónde el *path* dónde están las propiedades de la conexión a base de datos.

Model
Aquí no se conecta directamente a la base de datos, sino que es el "transporte en el que viajan los datos". Sólo se usa, no se crea porque es proporcionado por Spring. Es el componente que lleva los datos.

Controller
Antes era el único punto de entrada y salida aunque no siempre se cumple. En este modelo se cumple con el front controller y los controllers que se van a conectar con la capa de servicios.

View
Este continua con su responsabilidad de manejar la vista o la presentación de los datos por eso aquí se encontrarían las jsp.

Service Layer

Responsable de la lógica de negocio. Por eso en este espacio se definen los métodos importantes. En este espacio se obtienen los datos de la capa de persistencia y se los envía al *controller*

Persistence Layer

Responsable de conectarse a la Base de Datos.