UT3 - Actividad 06 – Ramas y merge

Contenido

[1 Ramas uniones y conflictos 2](#_Toc124153061)

[1.1 Tipos de merge 2](#_Toc124153062)

[1.2 Merge Fast-forward 3](#_Toc124153063)

[1.3 Merge unión automática 9](#_Toc124153064)

[1.4 Merges manuales (con conflictos) 13](#_Toc124153065)

**A continuación, completa este manual con tus capturas tras la ejecución de los comandos, sustituyendo cuando sea necesario las imágenes por las tuyas.**

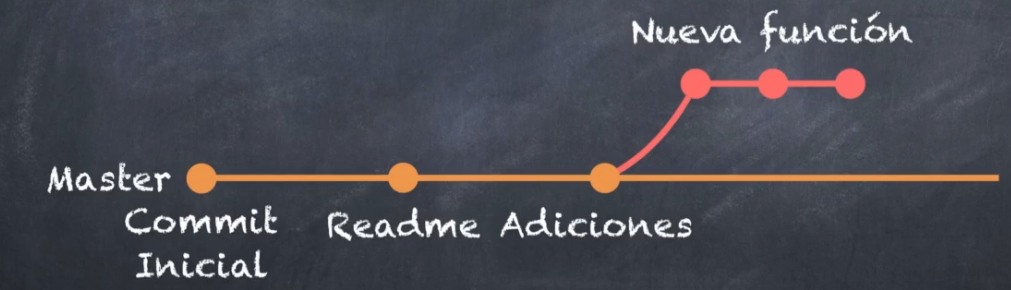
# Ramas uniones y conflictos

Una rama de un proyecto es una copia en la que empezamos a trabajar y a realizar modificaciones y commits. Desde el punto de vista de git podemos ver cada rama como un camino independiente de commits.

Hasta ahora hemos estado trabajando con sólo una rama que era la rama **master** que en un momento dado vimos como renombrar a **main** por ser más políticamente correcto:



Crear una nueva rama se puede ver visualmente como una bifurcación en el camino actual:



En un momento dado puede que queramos que una rama secundaria se integre en la principal a esto se le llama **unión** o **merge:**

****

## Tipos de merge

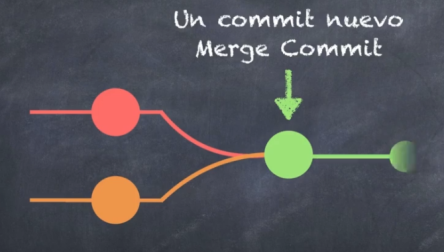
* **Fast-forward**: este tipo de unión se realiza cuando no ha habido cambios en la rama principal y por lo tanto se pueden unir sin problema las dos ramas simplemente agregando los commits de la rama secundaria a la principal:



* **Uniones automáticas**: este tipo de unión se realiza hay cambios en la rama principal pero que no entran en conflicto con los cambios realizados en la rama secundaria, por lo cual git puede realizar la unión de forma automática:



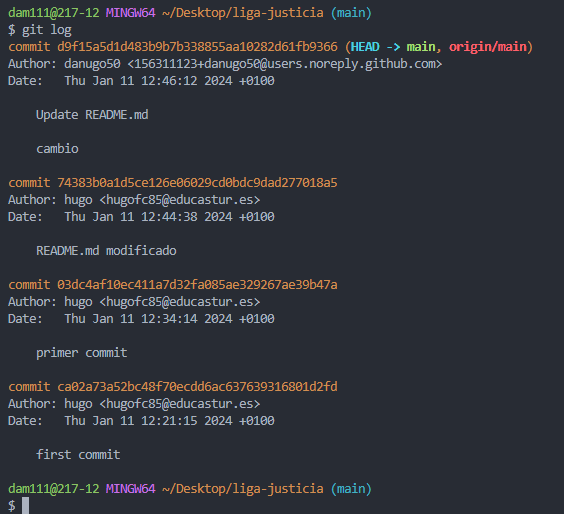
* **Manuales**: en este caso hay modificaciones en la rama principal y en la secundaria que entran en conflicto, por lo que git no puede decidir por sí mismo como hacer la unión y lo que hace es preguntarnos como realizar la unión y a continuación realizar un commit que se denomina **Merge Commit**



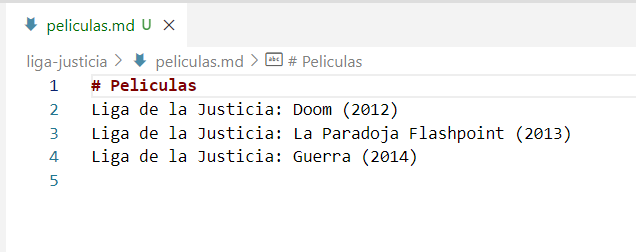
## Merge Fast-forward

Vamos a ver los commits que tiene el repositorio en este momento:

git log



Supongamos que estamos pensando en añadir al proyecto un apartado de peliculas, aunque no estamos seguros de si al final esa parte se quedará en el proyecto o no. De momento vamos a crear un archivo llamados peliculas.md con el texto que se ve en la imagen de más abajo:



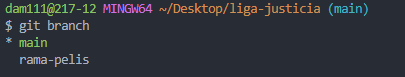
Vamos a crear una nueva rama que se llame rama-pelis:

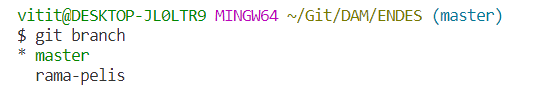
git branch rama-pelis



Ahora para ver las ramas existentes y en cual estamos ahora mismo ejecutamos el comando:

git branch

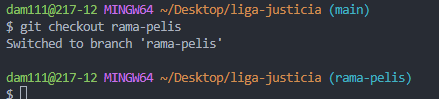




El asterisco y el color verde nos indica la rama sobre la que estamos actualmente.

Para cambiar a la nueva rama escribimos:

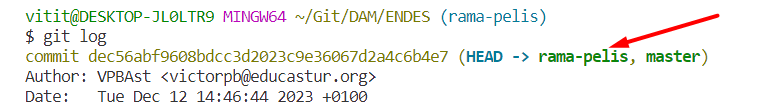
git checkout rama-pelis





Como vemos un mensaje nos indica que nos hemos movido a la rama rama-pelis

Si hacemos git log vemos lo siguiente:

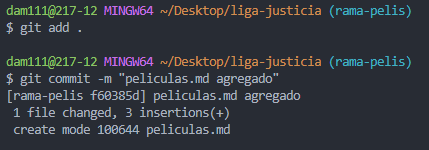


Vemos que la cabecera apunta a la rama rama-pelis y que la rama master está en el mismo punto.

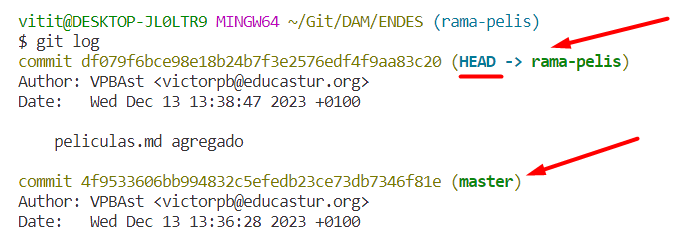
Agregamos todos los archivos al stage y hacemos commit:

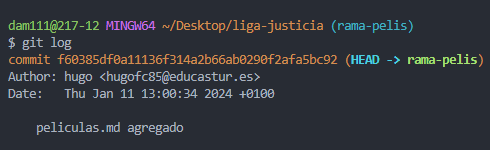
git add .

git commit -m “peliculas.md agregado”



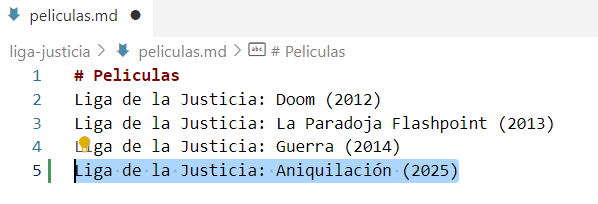
Si hacemos nuevamente git log:





Vemos que el **Head** apunta a la rama rama-pelis y que la rama master se ha quedado un commit atrás.

Añadimos una nueva película al archivo:



Hacemos un nuevo commit con este cambio usando la forma que sube el archivo al escenario y hace el commit a la vez:

git commit -am “peliculas.md: Aniquilacion añadida”



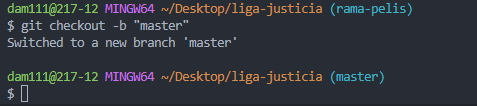
Hacemos nuevamente git lg:

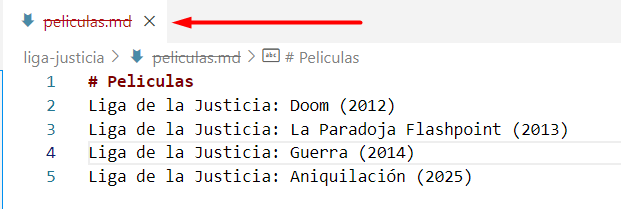


Vemos que la rama master se encuentra ya dos commits por detrás de la rama-pelis.

Si ahora cambiamos a la rama master:

git checkout master

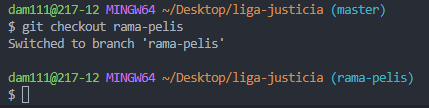




Vemos que el archivo pelis.md desaparece ya que en la rama master no está, y además visual studio code nos indica que la versión que teníamos abierta de pelis.md corresponde a un archivo borrado y por eso nos lo muestra en color rojo y tachado.

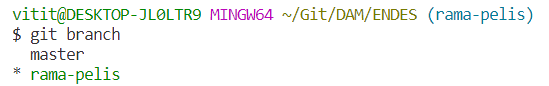
Si vuelvo a la rama rama-pelis el archivo volverá a aparecer:

git checkout rama-pelis



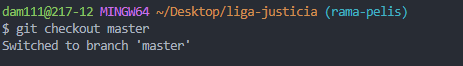
Ahora decidimos que los cambios implementados en esta funcionalidad de pelis son correctos y por lo tanto queremos integrarlos en la rama principal que es la rama master. Al hacer la unión de ramas hay que tener en cuenta que rama tenemos seleccionada, ya que la rama seleccionada recibirá los cambios de la otra rama.

Si hacemos git branch:



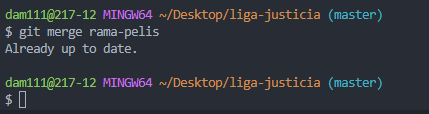
Vemos que ahora mismo está seleccionada la rama rama-pelis y de hacer la unión ahora los cambios de la rama master pasarían a la rama-pelis y lo que queremos es agregar los cambios de la rama rama-pelis en la rama master. Por lo tanto cambiamos de rama:

git checkout master



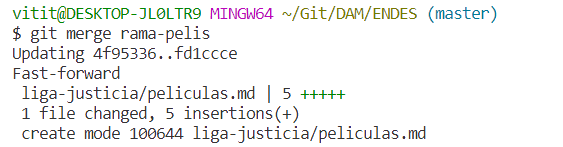
Hacemos la unión o merge:

git merge rama-pelis

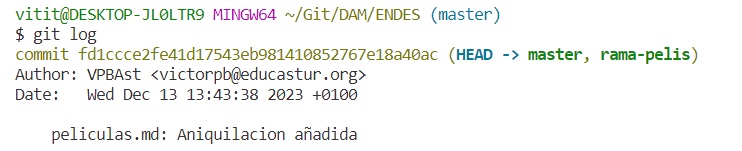


Vemos que se ha realizado un merge fast-forward que es la mejor opción porque implica que no ha habido conflictos y git ha podido hacer la unión sin ningún tipo de problemas.

También nos indica que al hacer el merge se incluyó el archivo md y sus modificaciones.



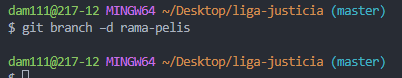
Si hacemos una vez más git log:



Vemos que después del merge las dos ramas están en el mismo punto

Como hemos terminado el desarrollo que estábamos haciendo en la rama rama-pelis y ya hemos unido su funcionalidad con la rama master, no necesitamos ya la rama rama-pelis. Vamos por lo tanto a borrar esta rama:

git branch –d rama-pelis



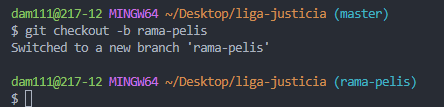
Si hubiera cambios en una rama que queremos borrar que no han sido unidos a otra rama, git nos avisará por si estamos borrando la rama por error. En ese caso podríamos forzar el borrado de la rama añadiendo el parámetro –f:

git branch –d rama-pelis -f

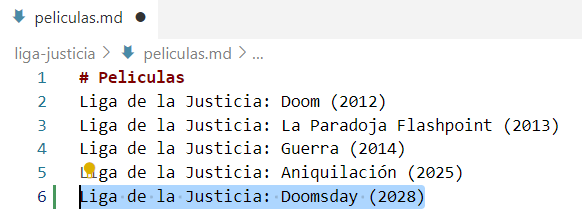
## Merge unión automática

Vamos a trabajar en unas modificaciones sobre el archivo de pelis y para ello vamos a crear una nueva rama. En el apartado anterior vimos como crear una rama y posteriormente movernos a ella usando por tanto dos comandos distintos. Vamos a ver ahora como crear una rama y movernos a la misma con un solo comando:

git checkout -b rama-pelis

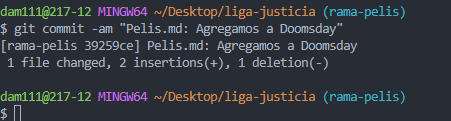


A continuación modificamos el archivo pelis.md y guardamos los cambios:

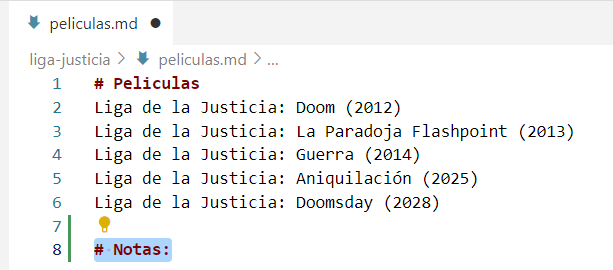


Hacemos commit de estos cambios:

git commit -am “Pelis.md: Agregamos a Doomsday”

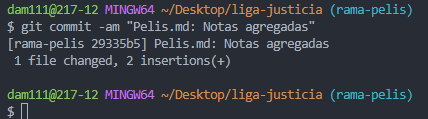


Hacemos otra modificación en el archivo pelis.md:



Y realizamos un nuevo commit:

git commit -am “Pelis.md: Notas agregadas”



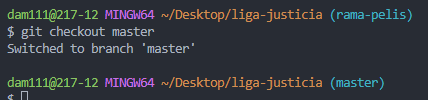
Hacemos git log:

Vemos que la rama master está dos commits por detrás de rama-pelis.

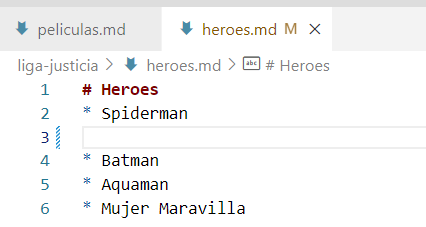
Ahora nos piden que hagamos un cambio en la rama master ya que hay un error en el archivo de heroes ya que que Daredevil es un héroe de Marvel y queremos tener sólo heroes del universo DC en ese archivo.

Nos cambiamos a la rama master:

git checkout master

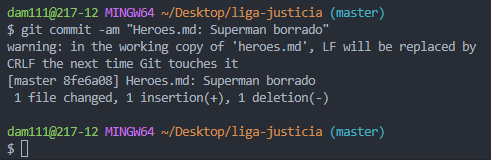


Borramos a Superman del archivo de heroes y guardamos los cambios:



Hacemos commit de este cambio:

git commit -am “Heroes.md: Superman borrado”

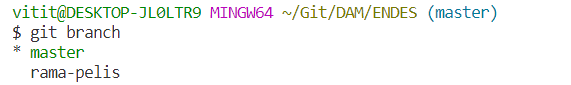


Si hacemos git log:

Vemos que la cabecera está en la rama master y que la rama rama-pelis está un commit por detrás. Además visualmente vemos que hay dos ramas.

Vamos a realizar la unión de estas dos ramas:

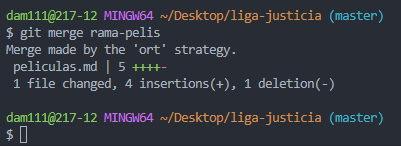
Queremos unir los cambios de rama-pelis a la rama master por lo que comprobamos si estamos en ella con git branch:

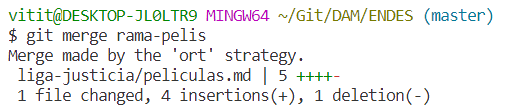


En esta ocasión ya nos encontrábamos en la rama master ya que estuvimos trabando en ella modificando el archivo de heroes.

Hacemos el merge:

git merge rama-pelis

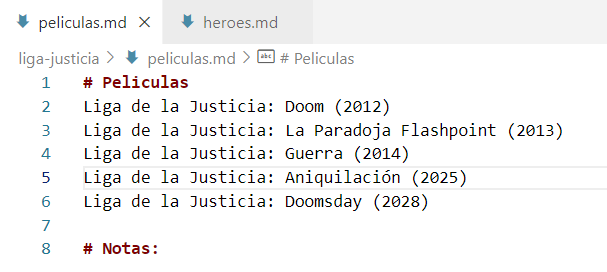




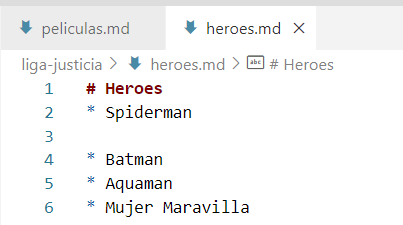
Vemos que git indica que ha realizado la unión usando la estrategia ORT por defecto y vemos que no hay conflictos y git ha podido hacer la unión sin preguntarnos nada.

**+ info sobre estrategias de “mergeo”** : <https://git-scm.com/docs/merge-strategies/>

Después de hacer la unión podemos ver que tenemos el archivo de pelis con los todas las pelis y el apartado de notas (editado desde rama-pelis):



Y que en el archivo héroes no tenemos a Superman (editado desde rama master):



Si hacemos una vez más git log:

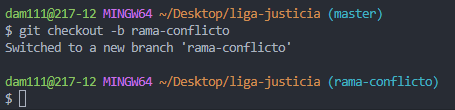


Vemos los dos caminos de commits y como en el ultimo merge se incorporaron los cambios de rama-pelis en master y el estado en el que se encuentran ambas ramas .

## Merges manuales (con conflictos)

Vamos a crear una nueva rama que se va a llamar rama-conflicto:

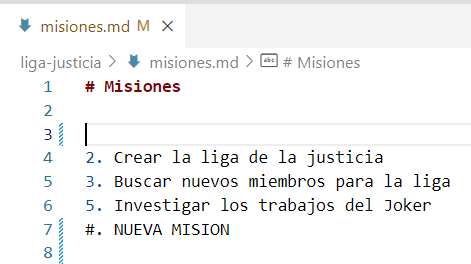
git checkout -b rama-conflicto





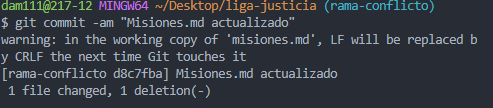
**Vamos a modificar el archivo misiones.md desde la rama-conflicto y desde la rama master y vamoa a intentar hacer un merge para generar un conflicto**

Modificamos el archivo desde la rama-conflicto. Por ejemplo quitamos la primera misión y añadimos una nueva.



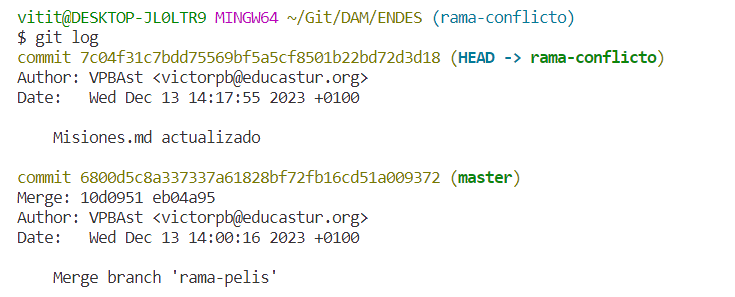
Hacemos commit de los cambios realizados en el archivo misiones.md:

git commit -am “Misiones.md actualizado”



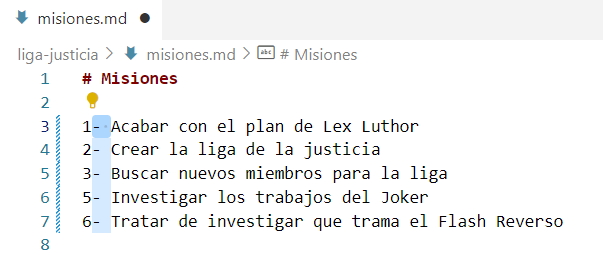
Miramos el estado actual:

git log



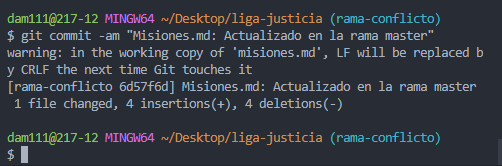
Ahora regresamos a la rama master y realizar cambios en el mismo archivo misiones.md:

(Por ejemplo añadimos guiones en vez de puntos **en todas las líneas**)



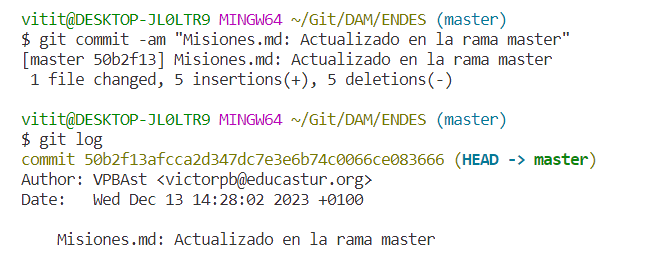
Hacemos commit de estos cambios:

git commit -am “Misiones.md: Actualizado en la rama master”



Comprobamos el estado actual:

git log



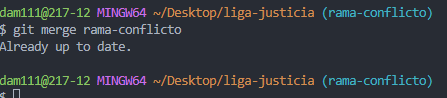
Ahora mismo tenemos dos ramas que tienen el mismo archivo modificado, con modificaciones diferentes en cada una de las ramas.

Vamos a realizar la unión o merge. Nos aseguramos que estamos en la rama master ya que queremos traer los cambios de la rama rama-conflicto a la rama master:



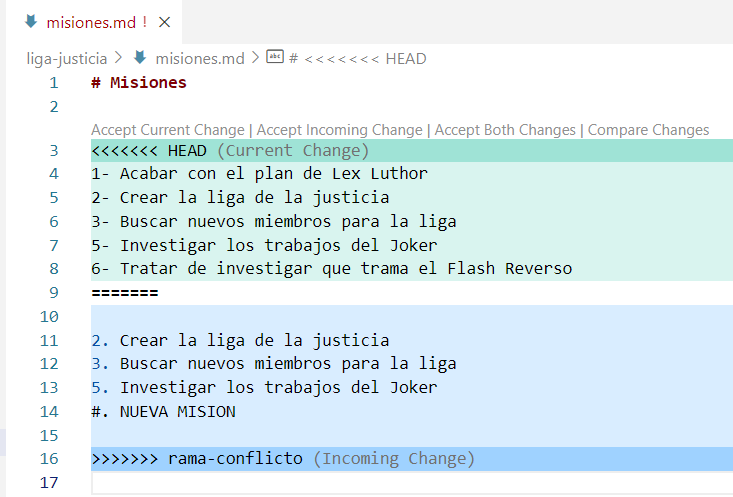
Realizamos el merge:

git merge rama-conflicto



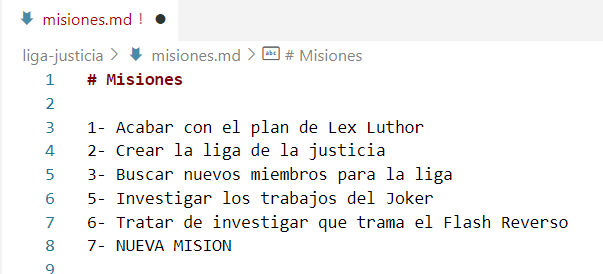
Git nos muestra un error. Nos dice que ha intentado hacer el merge automático (Auto-merging) pero que no es posible realizarlo porque hay conflictos que hay que resolver de manera manual.

Si abrimos el archivo misiones.md veremos algo así (si no nos aparece algo parecido puede deberse a que lo teníamos abierto con anterioridad, en ese caso debemos cerrarlo y volverlo a abrirlo):



Vemos las dos versiones del archivo. En color verde nos indica que es la versión actual ya que es la de la rama en la que estamos. En color azul nos marca la versión de la rama rama-conflicto y nos dice que es la versión entrante (incoming).

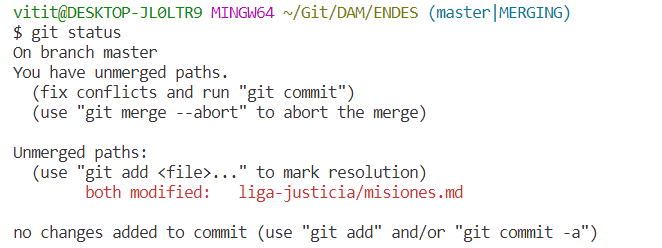
La forma de resolver el conflicto de forma manual es comparar visualmente ambas versiones e ir editando el archivo hasta dejar la versión que queramos. Por ejemplo podemos dejar una mezcla de las dos versiones recuperando todas las misiones incluida la borrada y nueva y con guiones :



Guardamos los cambios.

Si ahora hacemos git status

Git status



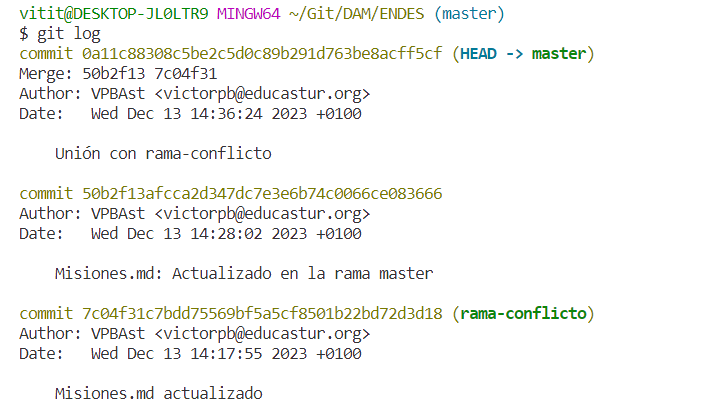
Vemos que git nos marca la rama como MERGGING (mergeo pendiente) y el conflicto en la unión de las dos versiones del fichero con dos ramas (both modified) Vemos que git nos da más información y nos dice que tenemos un conflicto que tenemos que resolver y hacer luego commit. También nos dice que podemos deshacer el merge usando git merge –abort

Como ya hemos arreglado el conflicto de forma manual editando el fichero con las dos versiones sólo nos falta hacer el commit:

git commit -am “Unión con rama-conflicto”



Y si miramos el estado actual:



Vemos que ya tenemos el commit en el que hemos unido la rama master con la rama rama-conflicto.

Como ya no necesitamos la rama rama-conflicto la borramos:

git branch -d rama-conflicto

