GIT: COMANDOS BÁSICOS

1r DAW
Jose Manuel Barrera Arroyo

GIT: NAVEGAR ENTRE SNAPSHOTS

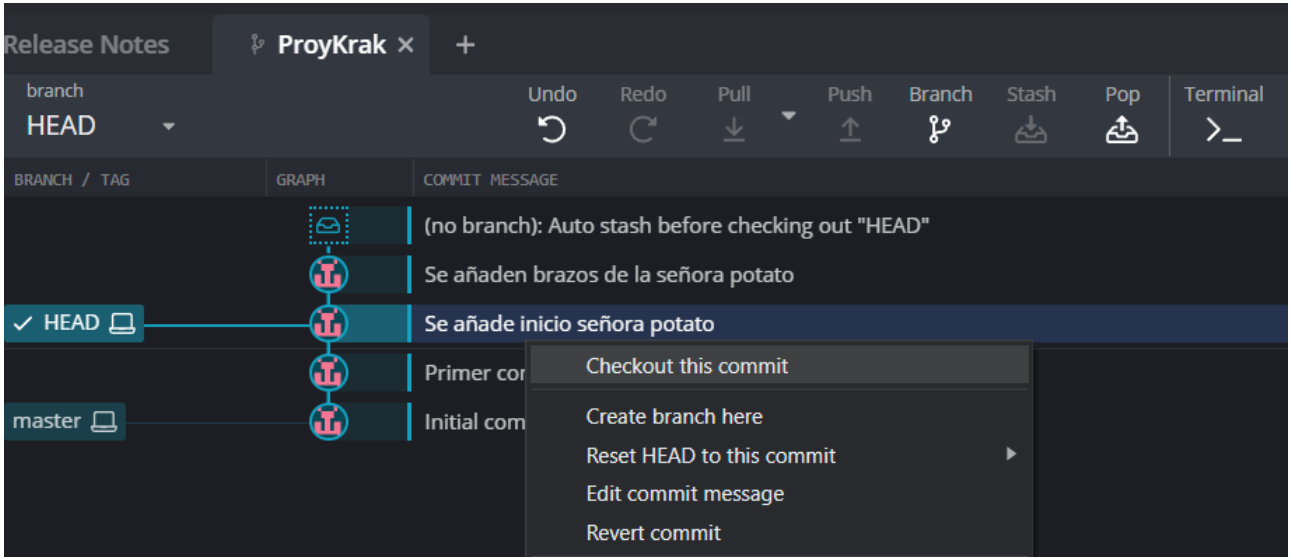
- git checkout hashdondequierasir

- Marcas vuelves al estado de la snopshot definida por el hash.

- OJO! Si no has guardados los cambios desde la última snapshot (haciendo otra), puedes perder lo que estés haciendo.

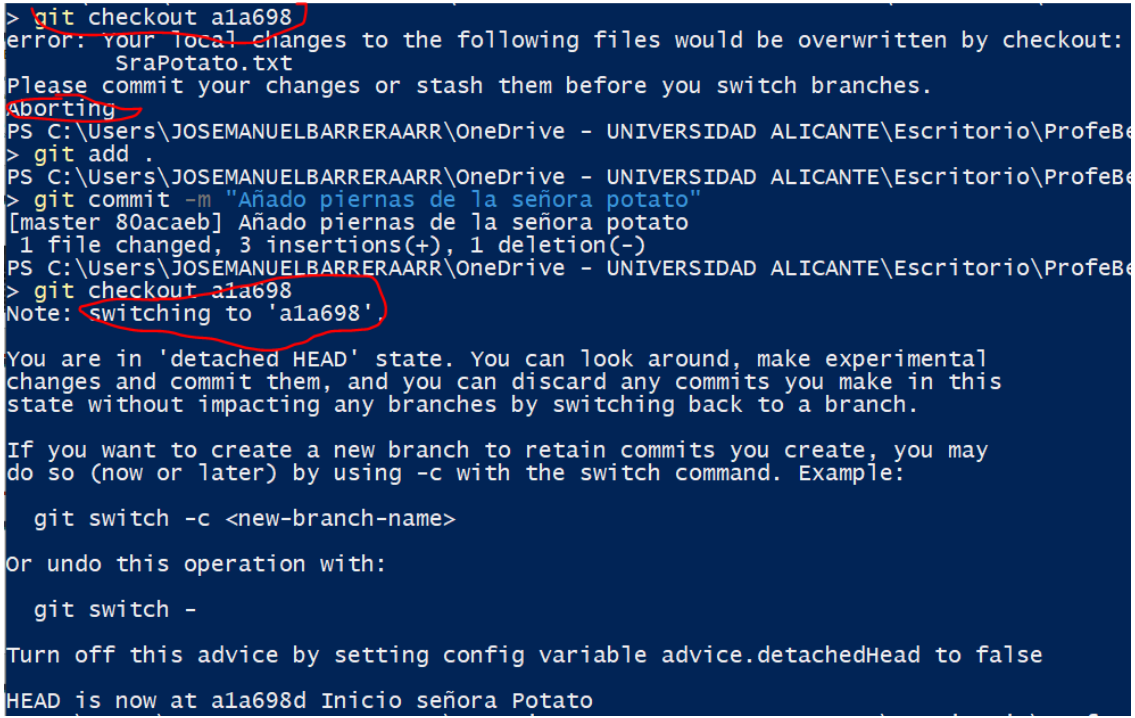
GIT: NAVEGAR ENTRE SNAPSHOTS

- Git Kraken



GIT: NAVEGAR ENTRE SNAPSHOTS

- Git Bash
  - El trabajo parece que se ha perdido, pero está guardado



GIT: CONCEPTOS DE NAVEGACIÓN

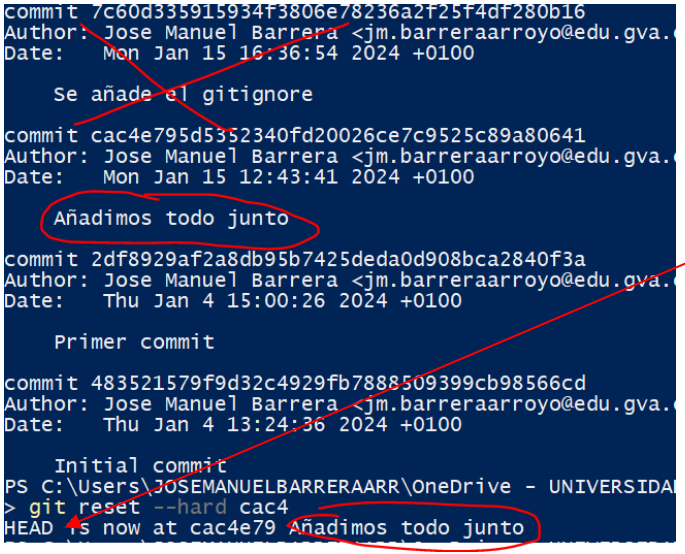
- HEAD:
  - Posición en la que te encuentras (normalmente, coincide con la posición más adelantada), pero si hago un checkout hacia atrás, el HEAD estará ahora en una posición atrasada.
  - Es el puntero que me dice donde estoy.

GIT: NAVEGAR Y BORRAR

- Vas a una snapshot, y “casi borras” el resto hasta esa snapshot
- Se puede recuperar con un git reflog, pero por ahora no va a ser necesario

GIT: NAVEGAR Y BORRAR

- Git Bash
  - git reset --hard hashdondequiero ir



HEAD se reposiciona en la posición reseteada.

GIT: NAVEGAR Y BORRAR

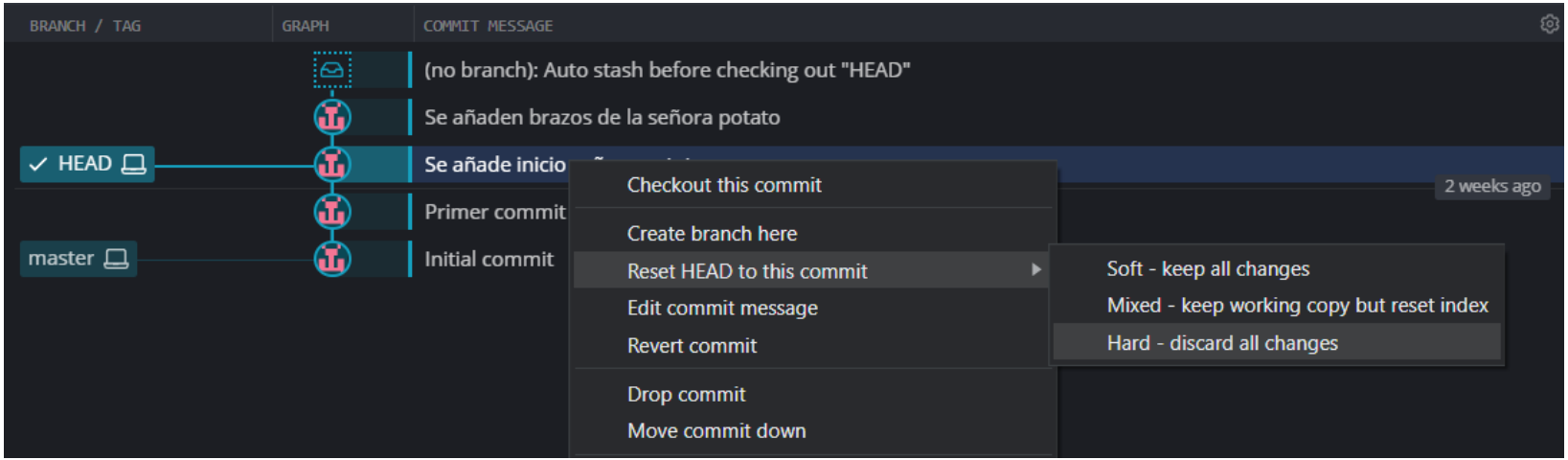
- Con otro “git reset --hard”, podríamos movernos hacia adelante otra vez, por lo estaríamos igual que al principio.

- Vale para moverte hacia adelante y hacia atrás.

- No obstante, el “git reset --hard” no es una herramienta muy usada.

GIT: NAVEGAR Y BORRAR

- GIT Kraken



GIT: ETIQUETADO (TAGGING)

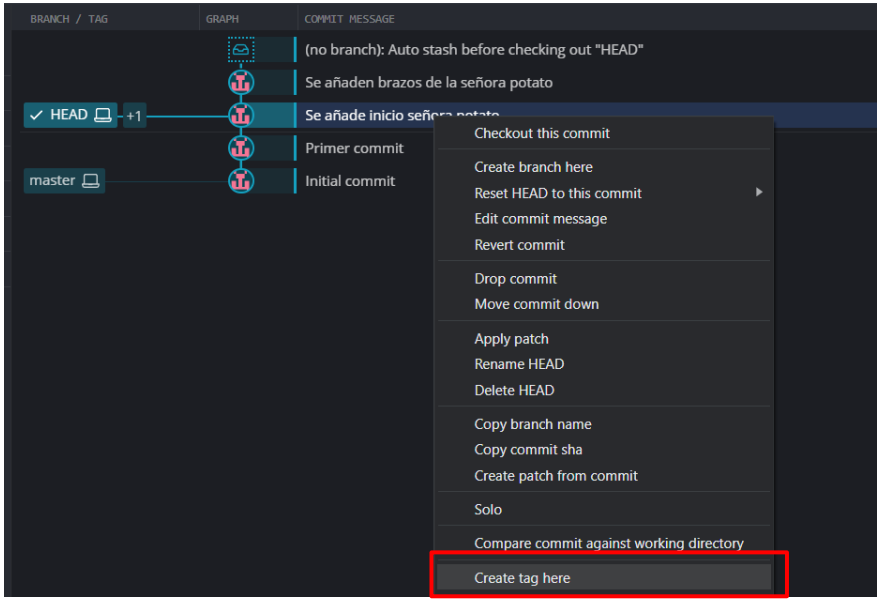
- Podemos crear etiquetas (o tags) en las distintas snapshots (o commits).

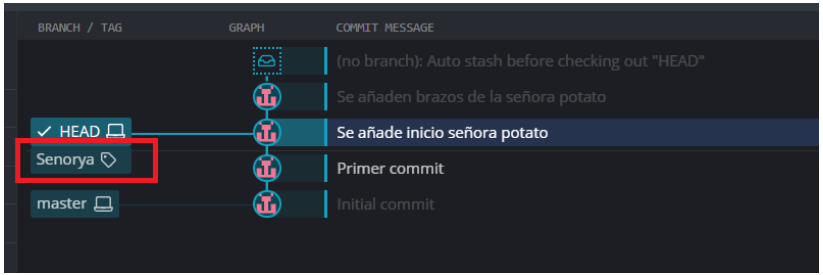
- Esto nos permite tener información adicional en commits realmente importantes (como releases).

- Además, nos facilita la navegación a los distintos tags, sin tener que sabernos el hash.

GIT: ETIQUETADO (TAGGING)

- GIT Kraken





GIT: ETIQUETADO (TAGGING)

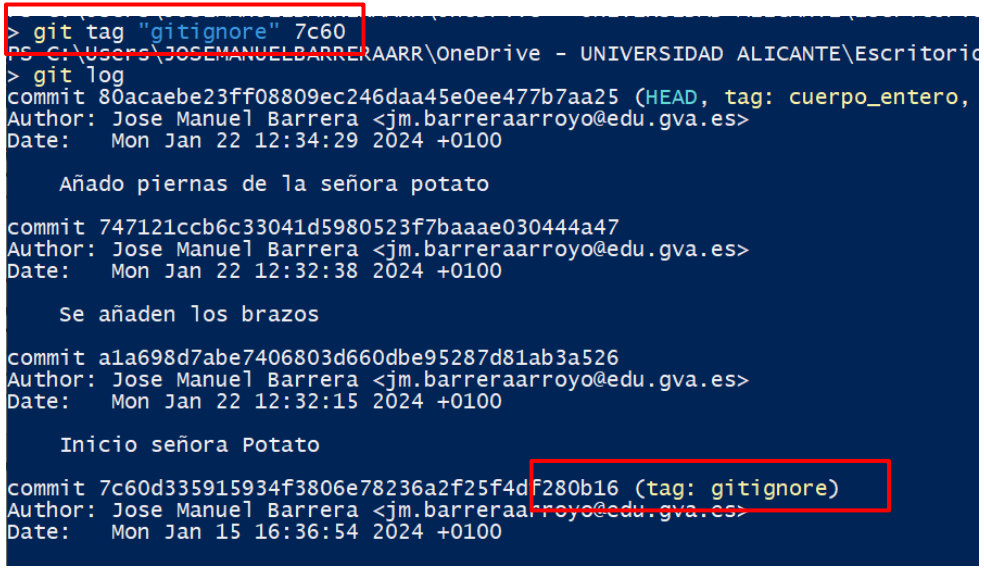
- GIT Bash
  - Si no pongo hash → Etiqueta a último commit
    - ✕ git tag “lo_que_sea”

```
> git tag "cuerpo_entero"
PS C:\Users\JOSEMANUELBARRERAARR\OneDrive - UNIVERSIDAD ALICANTE\Escritorio\ProfeBenido
> git tag
cuerpo_entero
PS C:\Users\JOSEMANUELBARRERAARR\OneDrive - UNIVERSIDAD ALICANTE\Escritorio\ProfeBenido
> git status
HEAD detached at 80acaeb
nothing to commit, working tree clean
PS C:\Users\JOSEMANUELBARRERAARR\OneDrive - UNIVERSIDAD ALICANTE\Escritorio\ProfeBenido
> git log
commit 80acaebe23ff08809ec246daa45e0ee477b7aa25 (HEAD, tag: cuerpo_entero, master)
Author: Jose Manuel Barrera <jm.barreraarroyo@edu.gva.es>
Date:   Mon Jan 22 12:34:29 2024 +0100

    Añado piernas de la señora potato
```

GIT: ETIQUETADO (TAGGING)

- GIT Bash
  - Puedo etiquetar un commit específico mediante hash
    - ✕ git tag “lo_que_sea” hash

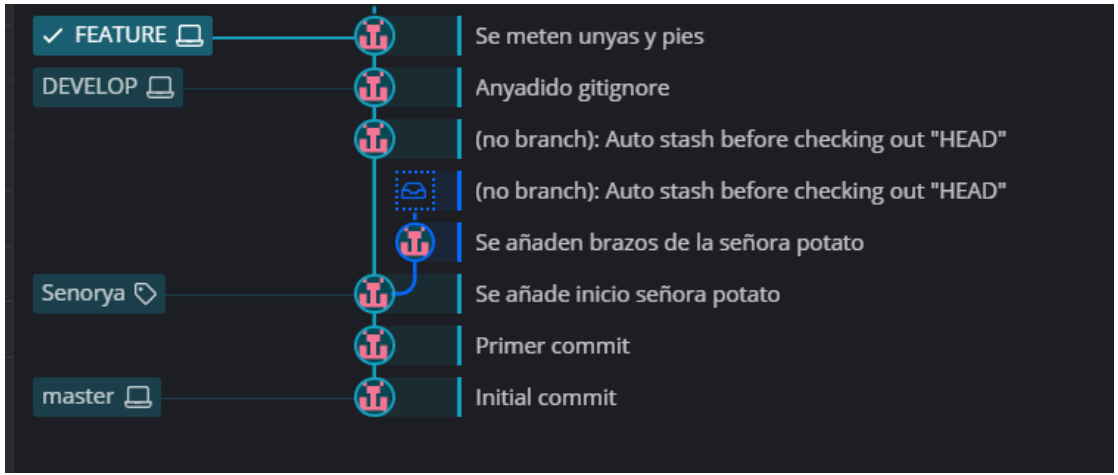


GIT: RAMAS

- A partir de ahora, vamos a empezar a trabajar con ramas.
- Una rama representa (normalmente) un flujo de trabajo distinto).
- MUY IMPORTANTE conocer la filosofía de ramas donde yo trabajo (Trunk Based Development, GIT Flow)

GIT: RAMAS

- Git kraken
  - Click derecho → Create branch here → Escribir nombre
    - ✕ Aquí se ha creado la rama “feature”



GIT: RAMAS

- GIT Bash
  - git branch nombrederama

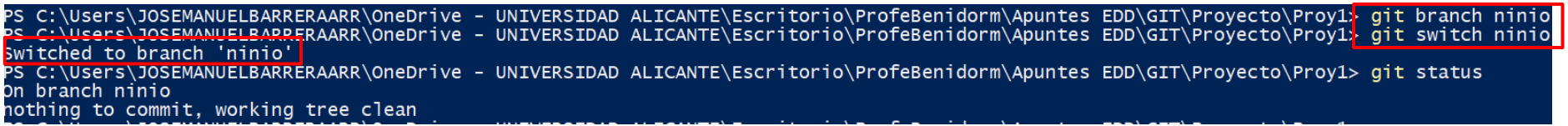
- Crea una rama, desde el punto donde nos encontramos (normalmente, la posición más adelantada de la rama)

  - git switch nombrederama

- Cambio a la rama “nombrederama”

GIT: RAMAS

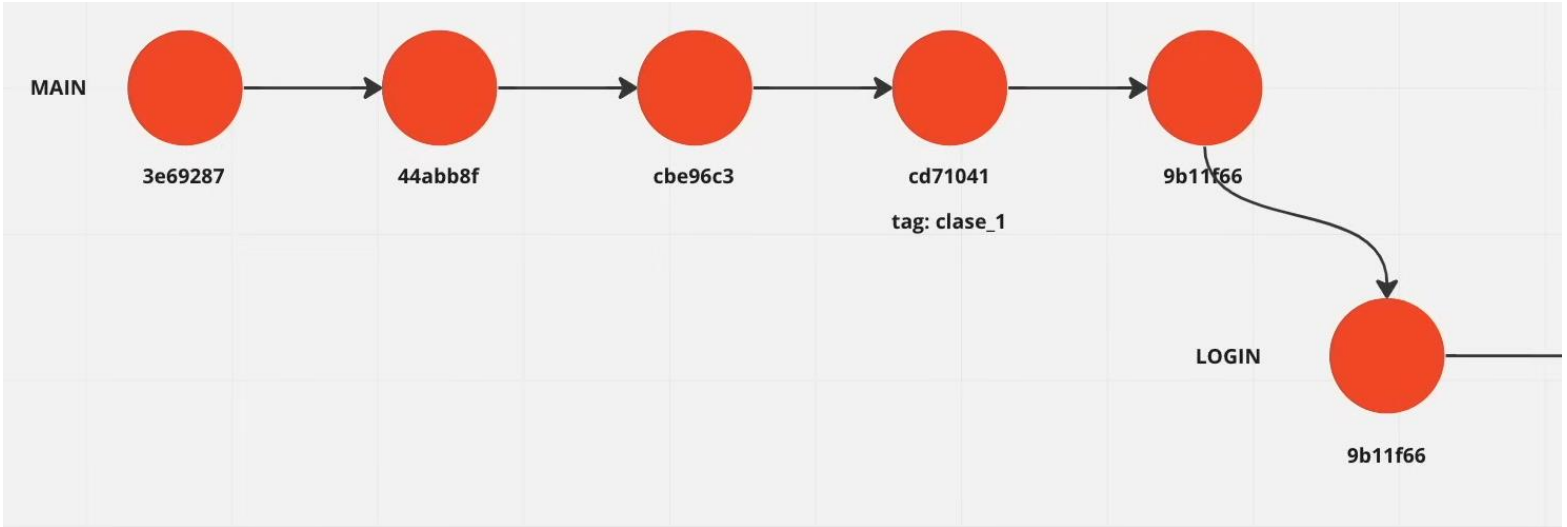
- GIT Bash



  - Para saber dónde estamos, git status

GIT: RAMAS

- Por ahora, solo se ve un flujo de datos

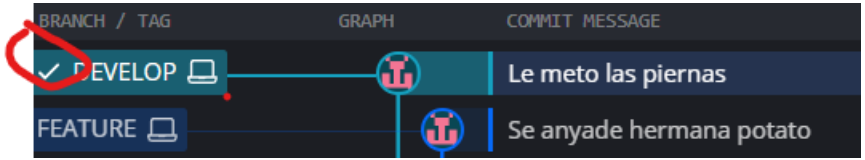


GIT: RAMAS

- Si en una rama se añade algo, y en la otra rama también, siguen flujos distintos

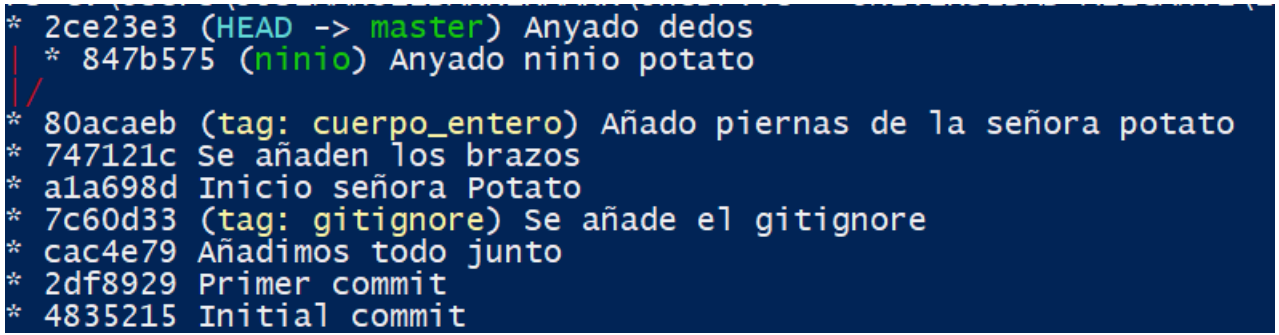
GIT: RAMAS

- GIT Kraken
  - Para alternar entre ramas, doble click a la rama (commit) que quiero ir.
  - El símbolo de check nos dice la posición en la que nos encontramos.
  - En el dibujo, se puede ver que estoy en al rama Develop



GIT: RAMAS

- GIT Bash
  - Hasta que no se hacen avances en las dos ramas, no se visualizan los cambios (al igual que en GIT Kraken).



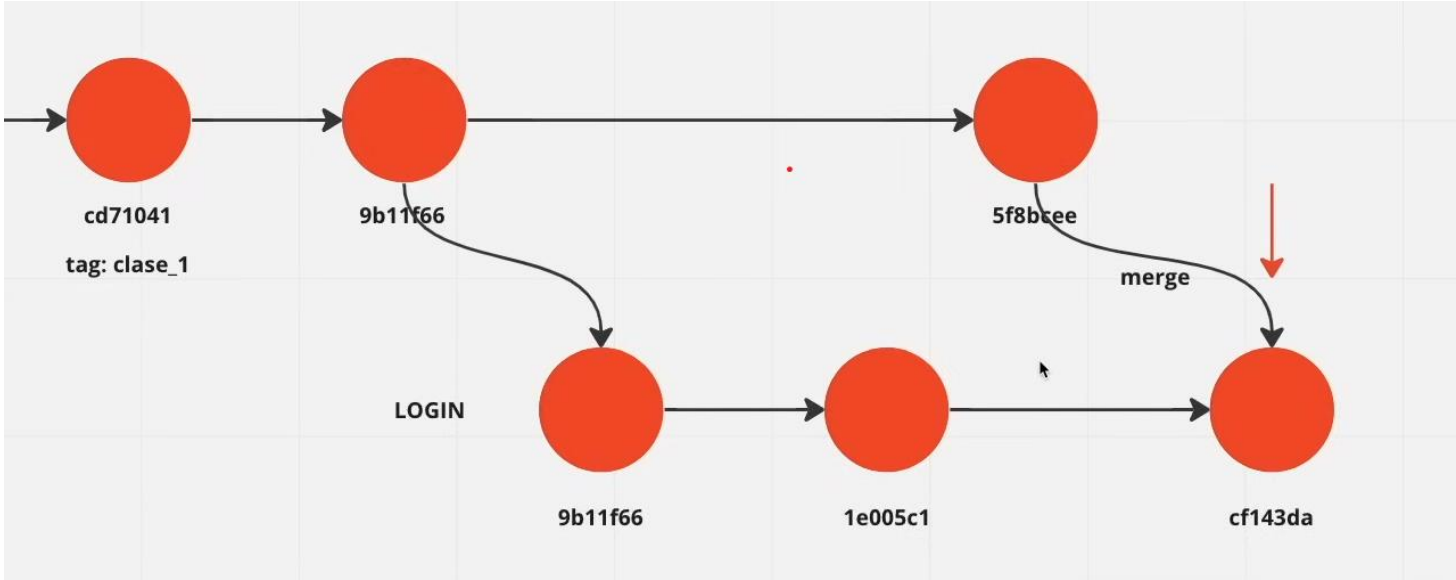
GIT: RAMAS

- Los commits en cada rama, aparecen adelantadas en función de los últimos envíos:
- Se puede ver en cada rama, quién ha enviado más tarde o más pronto.

GIT: MERGE

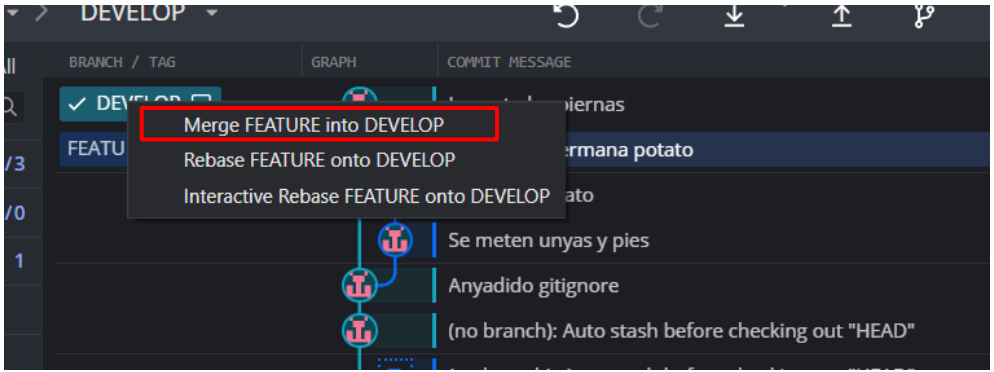
- El momento de la verdad, y cuando empiezan a haber problemas
- El comando “merge” se usa para juntar los elementos de una rama en otra rama.
- Normalmente, sirve para fusionar trabajos de varios equipos o desarrolladores.
- Suele dar conflictos.

GIT: MERGE


cd71041
tag: clase_1
9b11f66
5f8bcee
merge
LOGIN
9b11f66
1e005c1
cf143da

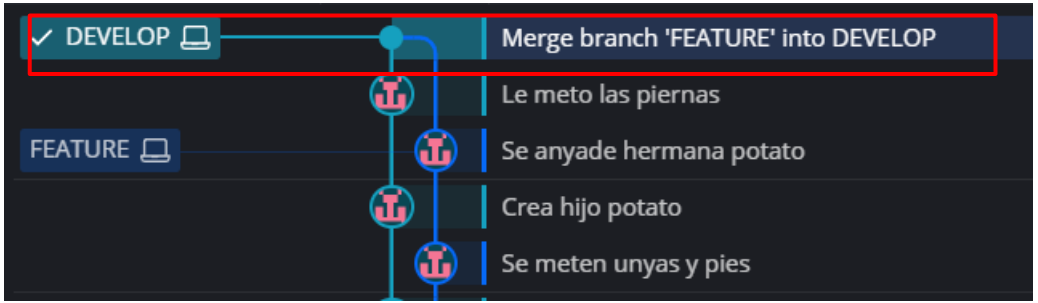
GIT: MERGE

- GIT Kraken
  - Se debe de arrastrar una rama encima de otra (click & drag), y seleccionar la opción de “merge”



GIT: MERGE

- GIT Kraken
  - En ese momento, las ramas pasan a estar unidas.



GIT: MERGE

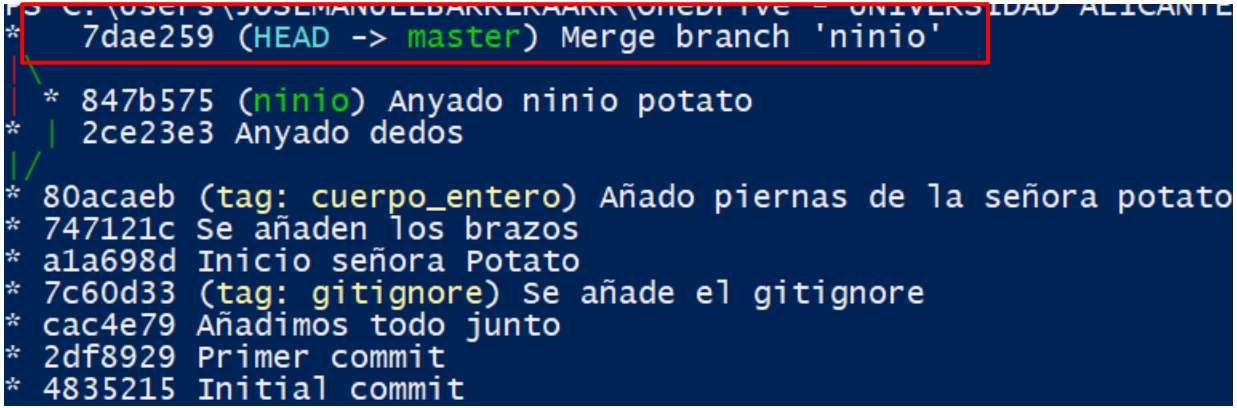
- GIT Bash
  - git merge ramaquequierotraer

- Ojo, muy importante. Se actúa DESDE la rama a la cuál voy a traer los cambios.

- No es lo mismo estar en MAIN y traer los cambios de DEVELOP, que estar en DEVELOP, y traer los cambios de MAIN.

GIT: MERGE

- GIT Bash

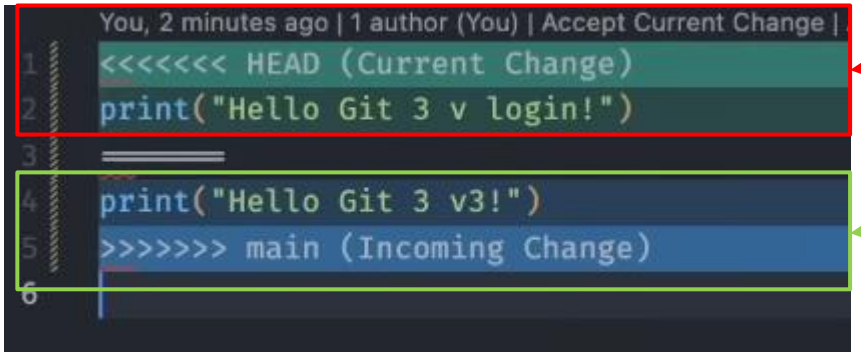


GIT: RESOLUCIÓN DE CONFLICTOS

- Los conflictos surgen cuando dos equipos han tocado la misma línea de código.
- Al hacer el merge, GIT no sabe cuál es la correcta.
- Filosofía de GIT: Os aclaráis vosotros, y luego me lo decís.
- La solución se hace en un nuevo commit, eligiendo la versión correcta PARA LA RAMA QUE YO ME ENCUENTRO.
- La rama de la que me he traído los cambios, no sufre ninguna modificación

GIT: RESOLUCIÓN DE CONFLICTOS

- El archivo con el conflicto visualiza las distintas opciones



Lo que tengo en la rama mía, en la actual (la rama login)

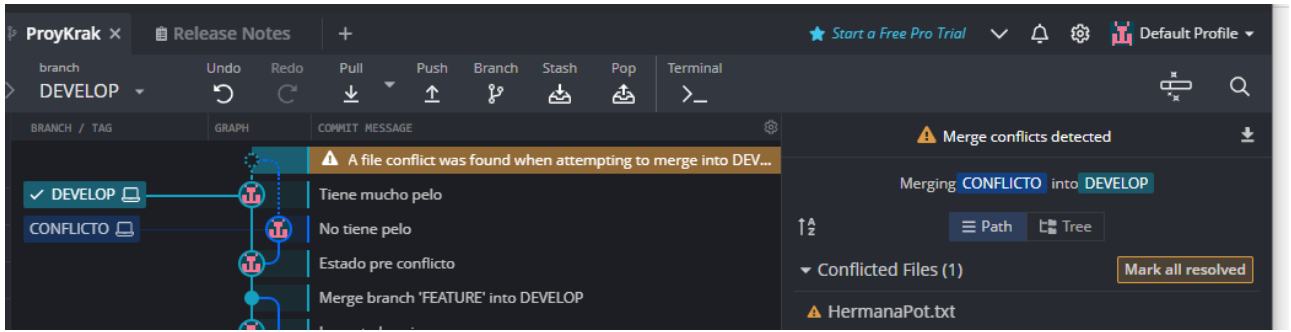
Lo que propone la rama que estoy importando (en este caso, la rama “main”)

GIT: RESOLUCIÓN DE CONFLICTOS

- GIT Kraken y GIT Bash.
  - Nos marca el conflicto.
  - Abrimos el archivo que genera el conflicto, y elegimos la opción correcta.
  - El archivo que que queda debe estar limpio: como debería de estar si no hubiese problemas (se elimina HEAD, el separador, y la info de la rama que genera el conflicto)

GIT: RESOLUCIÓN DE CONFLICTOS

- GIT Kraken y GIT Bash.



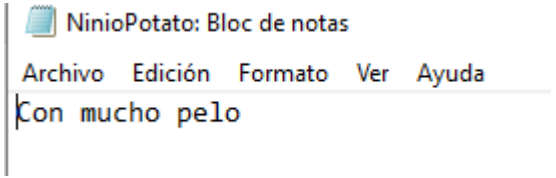

```
AzureAD+JOSEMANUELBARRERAARR@DESKTOP-UGDSMIK MINGW64 ~/Desktop/ProfeB
ntes EDD/GIT/Proyecto/Proy1 (master)
$ git merge ninio
Auto-merging NinioPotato.txt
CONFLICT (content): Merge conflict in NinioPotato.txt
Automatic merge failed; fix conflicts and then commit the result.
```

GIT: RESOLUCIÓN DE CONFLICTOS

- GIT Kraken y GIT Bash.

```
<<<<<<< HEAD
Totalmente calvo
=======
Con mucho pelo
>>>>>>> ninio
```

Archivo con el conflicto

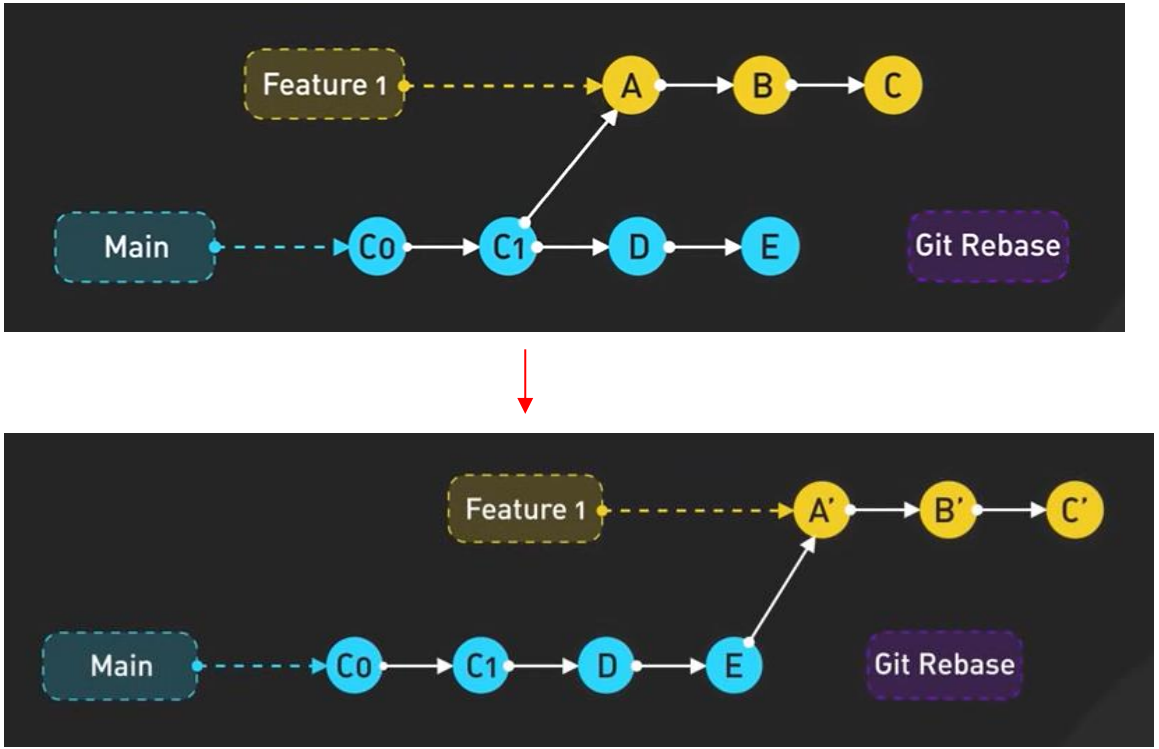


Archivo con el conflicto resuelto: hemos optado por el pelo

GIT: REBASE

- La otra manera de juntar dos que, es con el rebase.
- Es una alternativa a merge, con alguna particularidad.
- Mientras que con el git merge, se mantiene la rama de la que vengo, con rebase aparecen los commits de la rama externa en la rama donde pongo los cambios, y se borra la rama externa.

GIT: REBASE


Feature 1
A
B
C
Main
C0
C1
D
E
Git Rebase
Feature 1
A'
B'
C'
Main
C0
C1
D
E
Git Rebase

GIT: REBASE

- GIT Bash
  - git rebase ramaquequierotraer

- Ojo, muy importante. Se actúa DESDE la rama a la cuál voy a traer los cambios (igual que el Merge)

- Los commits de la ramaquequierotraer quedarán integrados en la rama donde actúo.

GIT: REBASE

```
$ git log --graph --all --oneline
* b98b951 (HEAD -> master) Commit en master
* 78c070a (origin/master) Update ArchivoRemoto
| * 249dd74 (testrebase) rebase3
| * 6fb9686 rebase2
| * 09e7641 Rebase1
|/
* a27f2ce Create ArchivoRemoto
* 8289ad3 Se anyade archivo local
```

```
$ git rebase testrebase
Successfully rebased and updated refs/heads/master.
```

```
$ git log --graph --all --oneline
* 3032473 (HEAD -> master) Commit en master
* bb6c5d5 Update ArchivoRemoto
* 249dd74 (testrebase) rebase3
* 6fb9686 rebase2
* 09e7641 Rebase1
| * 0cbdb2d (nuevonombre) Cambio a snapshot anterior
| | * 78c070a (origin/master) Update ArchivoRemoto
```

Actúo desde la rama master, y me traigo la rama testrebase, que quedará integrada en la master

GIT: REBASE

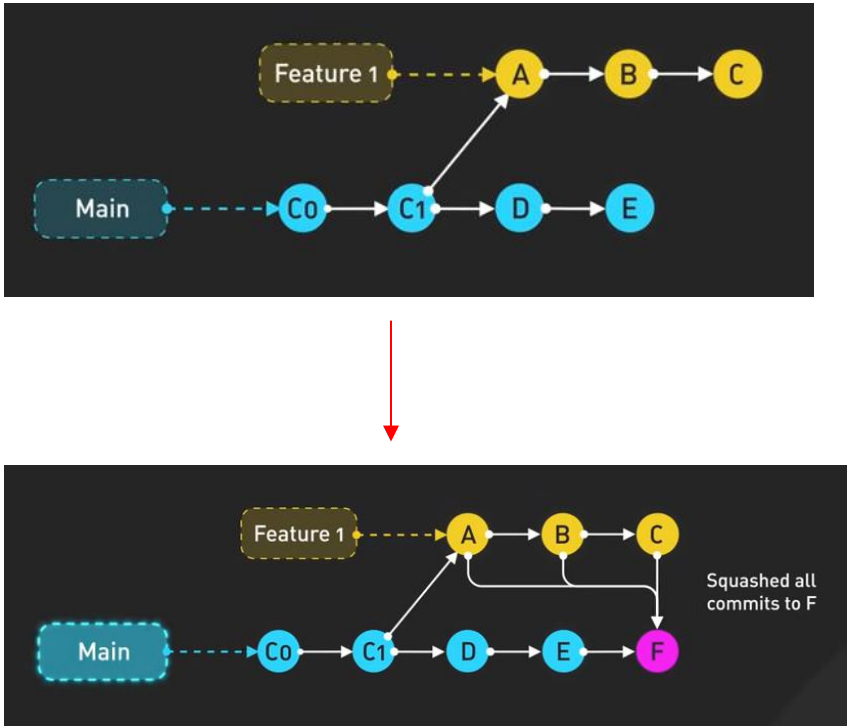
- VENTAJAS: Queda todo bonito, en una única secuencia (hay gente que lo prefiere). No se ven las distintas ramas.

- INCONVENIENTES: Reduce la trazabilidad del commit (no se sabe cuando se han hecho los commits en la rama exportada)

GIT: SQUASH

- La tercera manera de juntar.
- Es una alternativa a merge, con alguna particularidad.
- Es igual que el rebase, pero te concentra todos los commits de la rama exportada en uno.
- Es más complicado. No lo vamos a practicar. Que sepáis que existe.

GIT: SQUASH

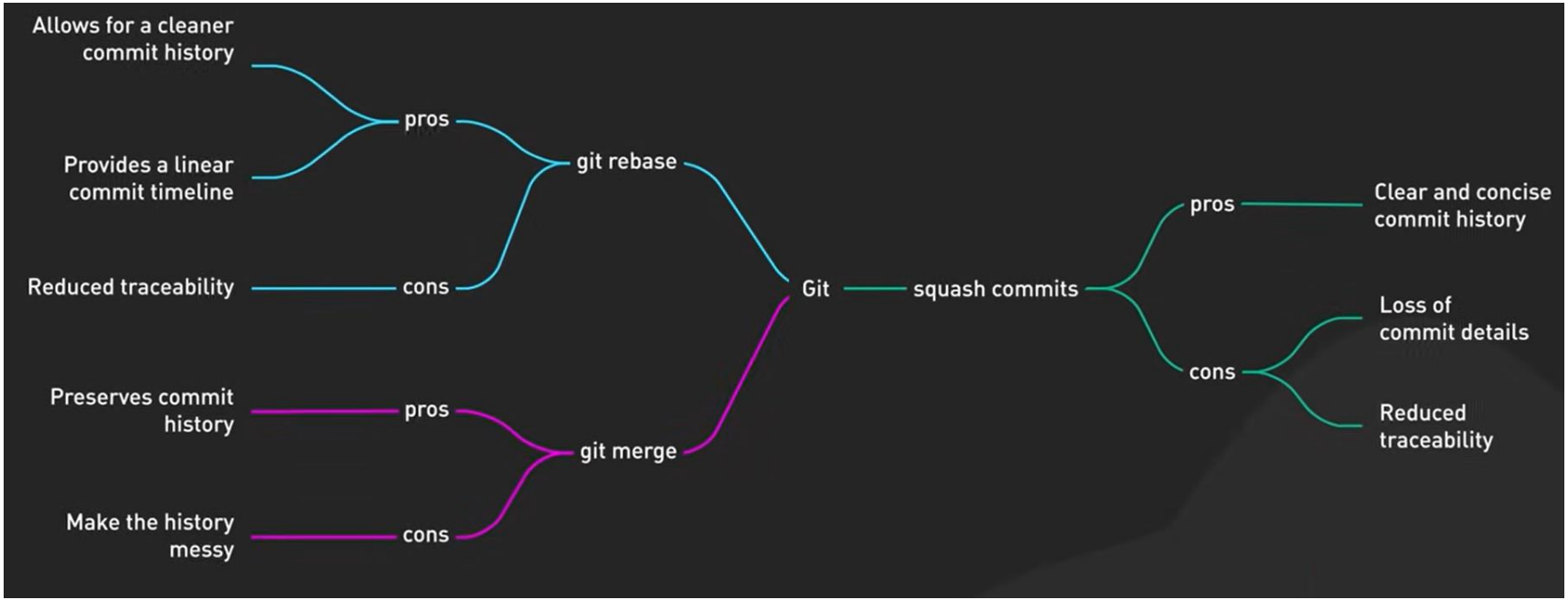

Feature 1
A
B
C
Main
C0
C1
D
E
Feature 1
A
B
C
Squashed all
commits to F
Main
C0
C1
D
E
F

GIT: SQUASH

- VENTAJAS: Más limpio que el rebase. Se queda todo el añadido de la rama exportada en un único commit. Se mantienen la trazabilidad en la rama feature

- INCONVENIENTES: Reduce la trazabilidad del commit (no se sabe cuando se han hecho los commits en la rama exportada)

GIT: SQUASH

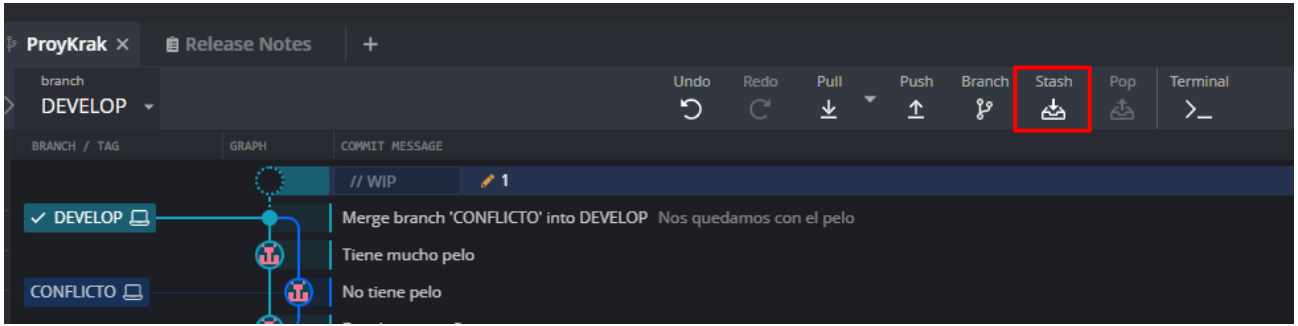

Allows for a cleaner commit history
Provides a linear commit timeline
pros
Reduced traceability
cons
git rebase
Preserves commit history
pros
Make the history messy
cons
git merge
Git
squash commits
pros
Clear and concise commit history
cons
Loss of commit details
Reduced traceability

GIT: STASH

- El stash es un commit temporal.
- Se usa para cuando no tengo algo acabado, y no quiero hacer un commit (porque los commits se hacen de cosas que funcionen) y me tengo que cambiar de rama, por alguna razón.
- Es un “lo dejo aparcado aquí un momento, y luego sigo”

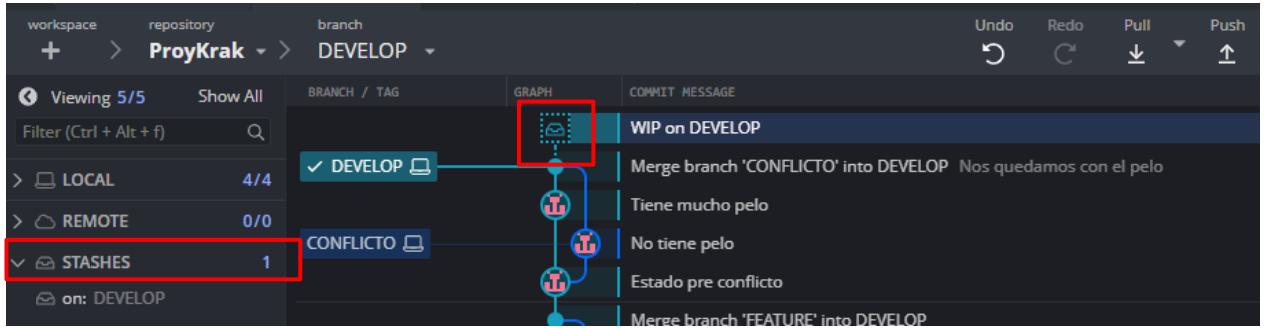
GIT: STASH

- GIT Kraken
  - Hay un botón específico para ello



GIT: STASH

- GIT Kraken
  - Una vez he hecho el stash, me lo marca, y ya puedo cambiar de ramas sin problemas.
  - Aparece un icono de stash en la barra de la izquierda.



GIT: STASH

- GIT Kraken
  - Una vez haya acabado con el stash (y ya no necesite guardarlo temporalmente, porque he hecho commit), lo puedo eliminar con el Delete Stash

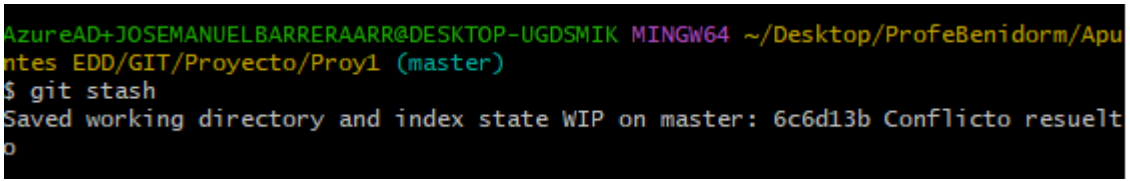
GIT: STASH

- GIT Bash
  - Si intento hacer un cambio de rama, sin tener los cambios guardados, no me dejará: me avisa que o commit, o stash

```
AzureAD+JOSEMANUELBARRERAARR@DESKTOP-UGDSMIK MINGW64 ~/Desktop/ProfeBenidorm/Apu
ntes EDD/GIT/Proyecto/Proy1 (master)
$ git checkout cac4
error: Your local changes to the following files would be overwritten by checkou
t:
        CuerpoEntero.txt
        NinioPotato.txt
Please commit your changes or stash them before you switch branches.
Aborting
```

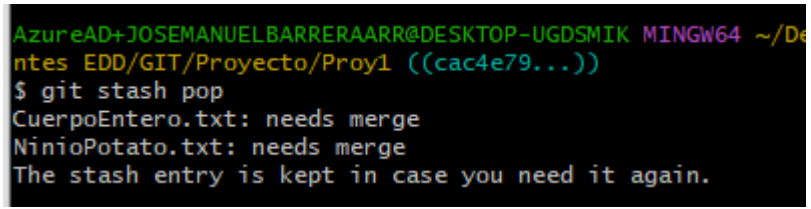
GIT: STASH

- GIT Bash
  - git stash
    - Para hacer el stash, y guardarlo temporalmente (con su hash)



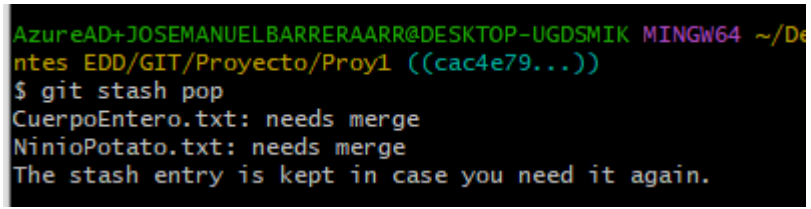
GIT: STASH

- GIT Bash
  - git stash pop
    - Cuando haya acabado, vuelvo al commit que tiene el Stash, y ya puedo “popearlo”.



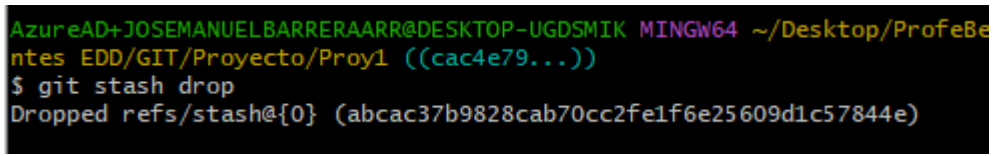
GIT: STASH

- GIT Bash
  - git stash pop
    - Cuando haya acabado, vuelvo al commit que tiene el Stash, y ya puedo “popearlo”.



GIT: STASH

- GIT Bash
  - git stash drop
    - Para dropear (eliminar) el stash



GIT: BORRADO DE RAMAS

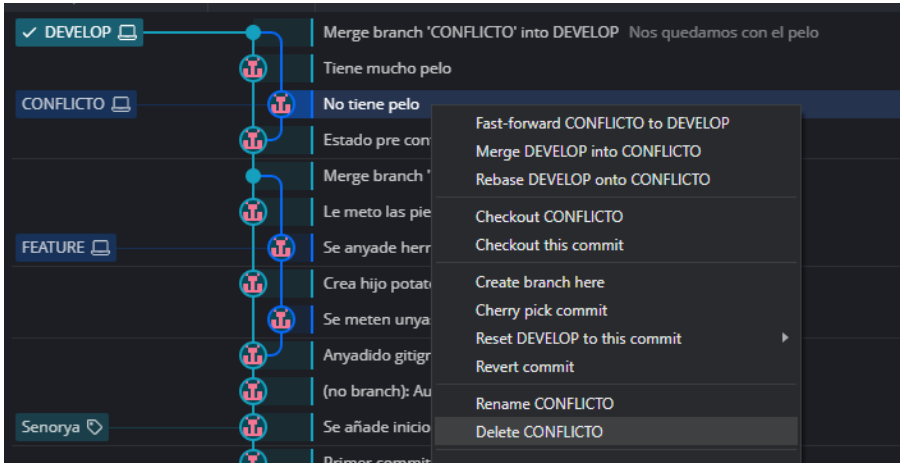
- Cuando se ha acabado una fase de desarrollo, lo normal es “borrar” la rama.

- Esto se hace una vez se haya integrado una rama en otra rama más importante (como meter develop en main).

- El borrado de ramas facilita la visualización de los árboles, pero las ramas siguen existiendo y se podría acceder a ellas (si hiciese falta, pero la idea es que no).

GIT: BORRADO DE RAMAS

- GIT Kraken.
  - Click derecho en la rama → Delete rama
  - Avisa de que va a destruirse una rama.



GIT: BORRADO DE RAMAS

- GIT Kraken.
  - Click derecho en la rama → Delete rama
  - Avisa de que va a destruirse una rama.