# Fundamentos GIT

## Inicialzar git

git init // Crea el repositorio local y crea la carpeta .git

git add . //añadir toda la carpeta o archivo especifico al stage

git commit -m “Primer commit” // tomar foto commit -m “mensaje”

git status // archivos en rojo, archivos que han recibido modificación brinda información de la rama en la que se trabaja

git config --global user.name “David”

git config --global user.email “[david@gmail.com](mailto:david@gmail.com)”

git config --globlal -e // revisar archivo config

git checkout -- . // recuperar todo

git log // revisar los commits realizados

## Diferentes formas de agregar un archivo al escenario

git add \*. png // agrega todos los archivos terminados en .png

git reset \*.xml // quitar del stage todos los archivos .xml

git add ”\*.txt” // Agrega todos los txt de TODO el proyecto

git add –all //Agrega todos los archivos -A

git add <lista de archivos> // Agrega los archivos que listemos

git add pdfs/\*.pdf // Agrega todos los archivos PDF dentro de la carpeta pdfs

git add pdfs/ // Agrega todos los archivos dentro de la carpeta pdfs

## Otras formas de revisar el log y los cambios desde el ultimo commit.

git log --oneline // Muestra los commits en una sola línea.

git log -- oneline --decorate --all --graph//

* -- all // toda la información referente a las ramas
* --decorate --graph // Mejor visualización

git status -s -b //silent solo aparece los archivos modificadosy cuales al stage

* -s //silent
* -b //en que Branch estamos trabajando

## Creando alias para nuestros comandos

git config –global alias.lg “log –oneline –decorate –all -graph” //lg alias “comandos”

git config --global alias.s "status -s -b" // -sb = -s -b = status Branch

git config --global -l //revisar archivo .gitconfig

## Creando alias para nuestros comandos

git config --global alias.s “status -s -b”

git config --globlal alias.lg ”log --oneline --decorate --all --graph”

## Diferencias entre commits y restauración de archivos

git diff //revisar cambios en los archivos desde el ultimo commit

git diff --staged // Revisar los cambios de los archivos del stage

git checkout -- README.md // revertir los cambios

## Actualizar mensaje del commit y revertir commits

git commit -am “Readme actualz” // Agregar archivos al commit y al stage al mismo tiempo

git commit --amend -m “Actualizamos el Readme” // Corregir el mensaje del commit

git reset --soft HEAD^ // soft reset. HEAD apunta al commit ultimo commit que se hizo. HEAD^ aputa antes del ultimo commit realizado.

## Preparando un repositorio para viajes en el tiempo

git commit //abre editor para agregar mensajes multilínea

:w // escribir archivo

:q //Salir del archivo

## Viajes en el tiempo, resets y reflog

git reset --soft asdfas // ir a commit con dirección asdfas

git reset --mixed 05d8a11 // (comando por defecto) los cambios en los commits siguientes fueron quitados del stage, pero aun tienen sus modificaciones.

git reset --hard 05d8a11 //quiero llegar a un commit determinado y borrar todo los commits que se hicieron después.

git reflog // muestra todos los movimientos que se han hecho con commit

## Cambiar el nombre y eliminar archivos mediante git

git mv “destruir el mundo.txt” salvar-mundo .txt // (move) mover archivo y cambiar nombra

git rm salvar-mundo.txt // eliminar archivos

## Cambiar el nombre y eliminar archivos fuera de git

Git add -u // actualizar git

## Ignorando archivos que no deseamos

Crear archivo “.gitignore” y adentro poner los archivos, carpetas o extensiones que se desea que sean ignoradas. Este archivo “.gitignore” si debe ser parte de los commits.

# Ramas, uniones, conflictos y tags

## Merge, fast-forward

git branch rama-villanos // crea una nueva rama llamada villanos

git branch // me dice cuales son las ramas y donde estoy trabajando

git checkout rama-villanos //me muevo a la rama villanos

git diff rama-villanos master // comparar la rama villanos con la rama master (pueden ser indicadores o commits id)

git merge rama-villanos // desde la rama master se une la rama villanos e indica por cual método se unió (Fast-forward)

Como buena practica es aconsejable borrar las ramas que ya no se están usando.

git Branch -d rama-villanos //borra la rama deseada.

## Merge automatico

git checkout -b rama-villano // Crea nueva rama e inmediatamente se mueve a esta.

Cuando se trabaja en dos ramas diferentes y las dos tienen cambios se ve de la siguiente manera.

\* 1078a9f (HEAD -> master) Borre a Marvel .

| \* aff66a7 (rama-villano) Agregamos a doomsday

| / .

\* 8957a2b Agregamos a Reverse flashmsdaym .

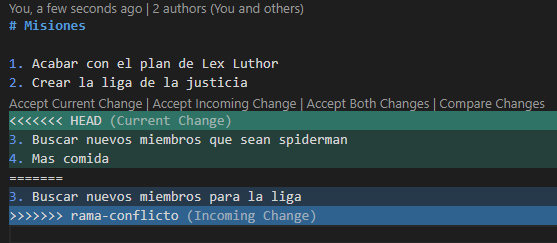
\* 9cee6cd Agregando villanos ...

## Merge Uniones con conflictos

Git merge rama-conflicto//



El archivo en conflicto se ve de la siguiente manera.



HEAD: El trabajo que nosotros hicimos

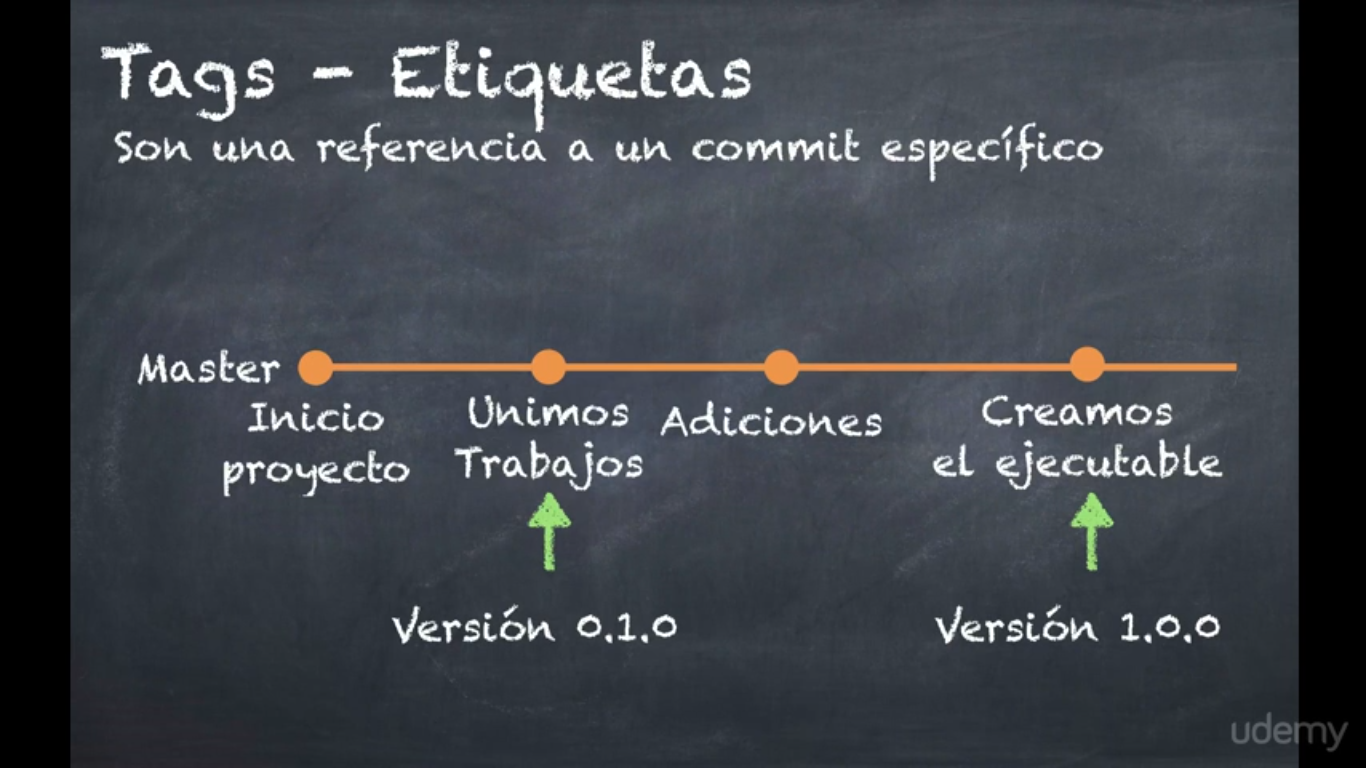
Rama-conflicto: el trabajo de alguien mas

Se requiere mantener el mensaje realizado por la rama en conflicto, pero también se requiere mas comida.

La forma de solucionar esto es quitando las etiquetas por nosotros mismos y corrigiendo el código a como mejor convenga.

## Tags – Etiquetas

Son una referencia a un commit específico en el tiempo



## Crear tags

git tag nombreTag // crea un tag con nombreTag

git tag // revisa todos los tags realizados.

git tag -d nombreTag // borrar tag

git tag -a v1.0.0 -m “Versión 1.0.0” // crear tag con mensaje -a (anotation) -m (message)

git tag -a v0.1.0 345d7de -m “Version alpha” // tag a un commit especifico con dirección de hush

git show v1.0.0 // Muestra información sobre el tag.

# Git Stash y Git Rebase – para realizar cambios de emergencia

## Stash

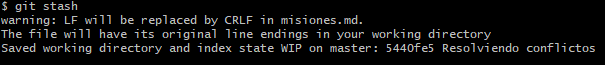
git stash //utilizar stash y restaura el ultimo commit

git stash sabe // utiliza stash y restaura el ultimo commit

Se ha salvado el directorio completamente y el index

WIP: Work in progress

Aparece la rama y el commit que se hizo cuando se hizo el stash



git stash list // Saber todo el work in progress que hay y la información del commit del stash

git stash pop // Extrae del stash todos los cambios pendientes y elimina el ultimo elemento del stash

## Conflictos con el Stash

Se resuelve como si fuera un conflicto de un merge, pero se debe tener en cuenta la eliminación del stash que se creo.



## Mas información sobre el Stash

Este comando toma archivos y los manda a un área temporal

git stash apply // Toma la ultima entrada en la lista del stash y lo restaura

git stash list // para ver las multiples entradas del stash

git stash apply stash@{1} // restaurar una entrada particular del stash

Cuando usamos el stash apply o en caso de conflictos las entradas en el list siguen ahí

git stash drop // borra la ultima entrada del stash

git stash drop stash@{0} // borra entrada del stash indicada

git stash pop // Restaura la ultima entrada del stash y la borra

### Otras formas del stash

git stash save --keep-index //guarda todos los archivos menos los que están en el stage, esto es porque estos archivos normalmente están listos para hacer un commit.

git stash save --include-untracked // Incluye todos los archivos, junto a los que git no les da seguimiento.

git stash list --stat // Aparece mas información de cada una de las entradas del stash

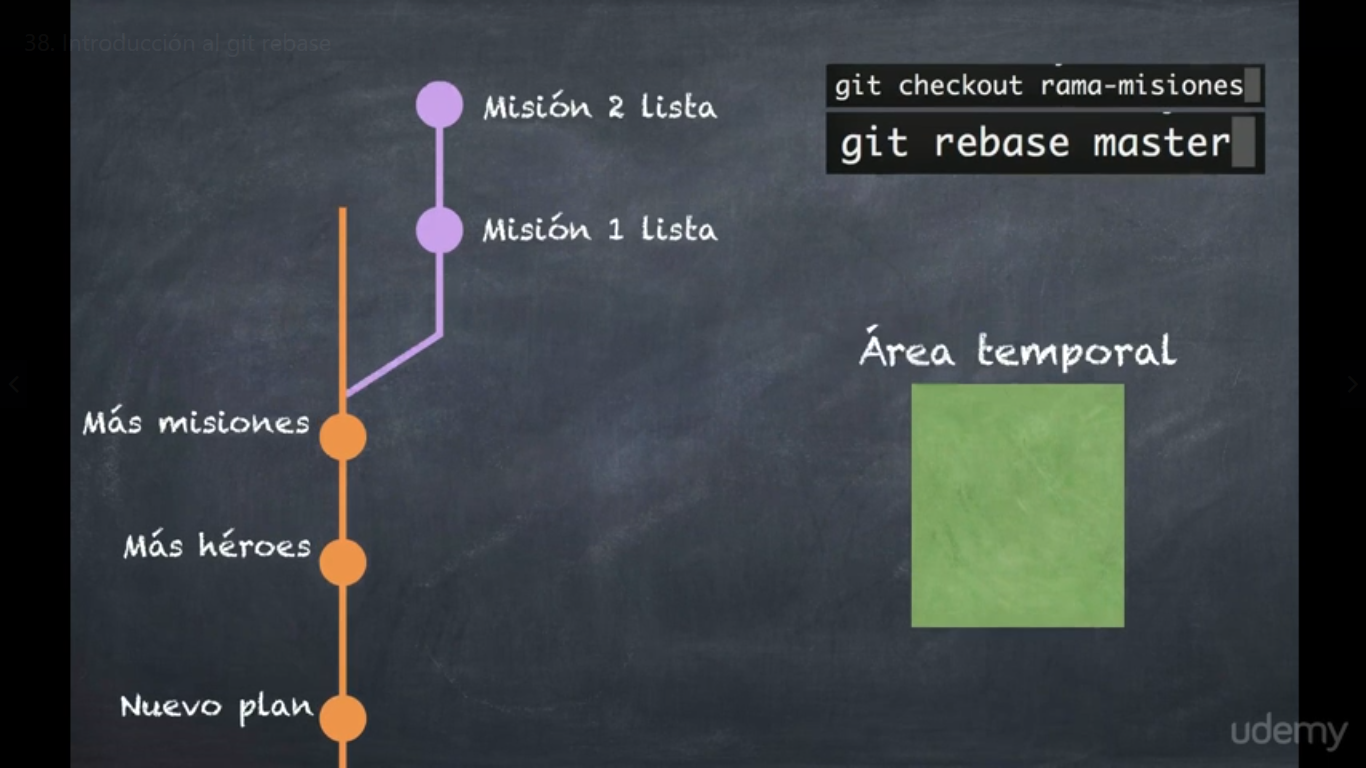
git show stash // nos informa lo que se hace en cada stash

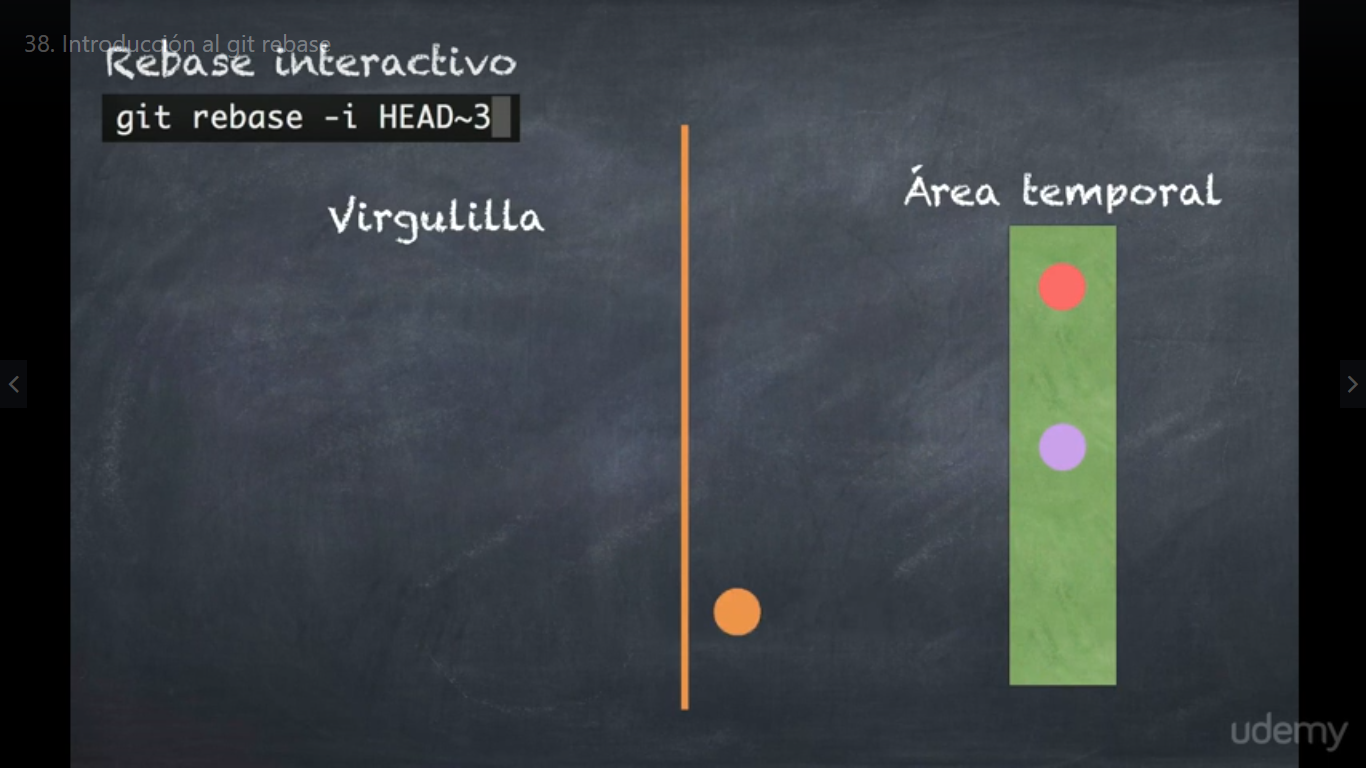
git show stash@{1} // acceder a la información de un stash en específico.

git stash save “mensaje” // Agregar mensajes a los stash

git stash clear // Borra todas las entradas del stash

## Introduccion al git rebase





Usos Rebase interactivo

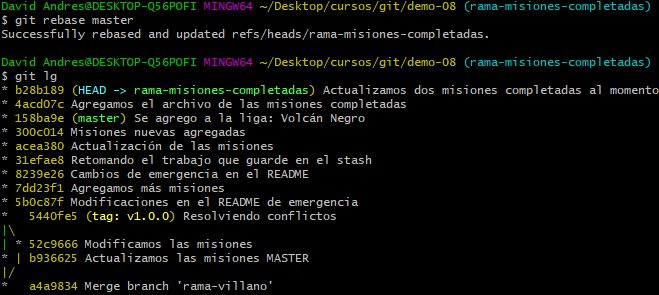
* Ordenar commits
* Corregir mensajes de los commits
* Unir commits
* Separar commits

## Rebase – Actualizando la rama

El rebase es utilizado normalmente para evitar conflictos en los merge.

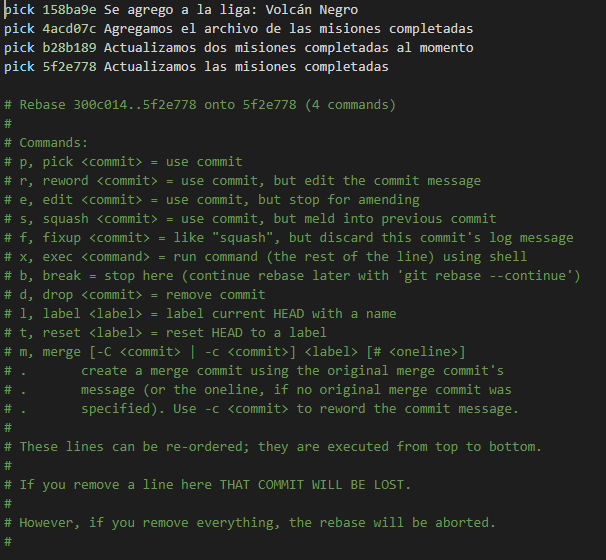
## 

git rebase master // mover la rama en la que estamos adelante del master para posteriormente hacer un merge por fast-forward.



## Utilizando el Squash

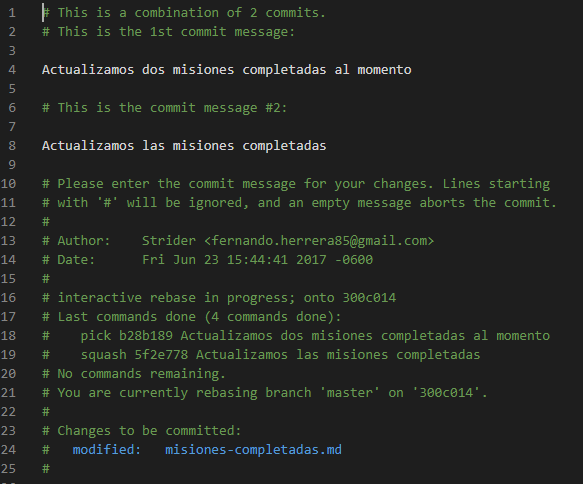
git rebase -i HEAD~4 // -i (rebase interactivo) 4 (toma los últimos 4 commits)



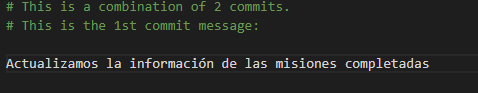
Con la siguiente modificación se unen los dos últimos commits realizados



Esto genera otro archivo para modificar el nombre de los commits



Al modificarlos queda de la siguiente manera.



## Rebase Rework

git rebase -i HEAD~1 // -i(interactivo) solo quiero un commit antes del head

No se debe actualizar directamente en el archivo abierto. Se debe modificar la palabra pick por la palabra reword. Para Posteriormente abrir otro archivo y sobre este archivo si modificar el nombre del commit. Al final es importante seguir el comando git checout master para verificar la rama en la que se esta trabajando.

## Rebase Edit

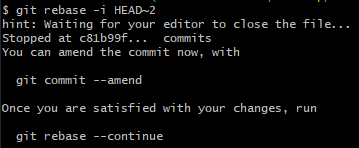
Se utiliza para separar commits

git rebase -i HEAD~2 // Se van a modificar los dos últimos commits.



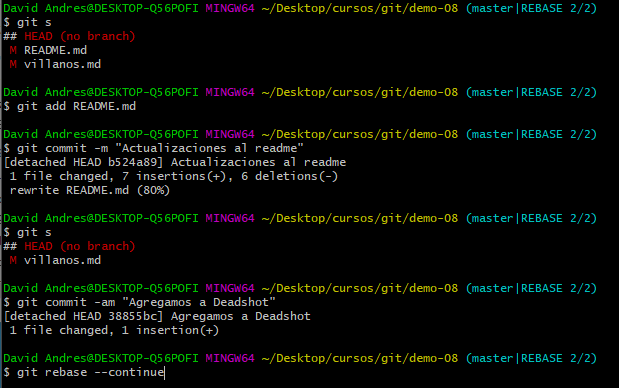
Se utiliza la palabra edid en el archivo que fue abierto por el comando git rebase

Esto produce el siguiente mensaje.



git reset HEAD^ // Revierte los cambios del ultimo commit sin destruirlos.

Se realizan las modificaciones necesarias y la respectiva separación de commits



git rebase –continue // Para terminar el rebase y poder trabajar normalmente de nuevo.