CURSO GIT

Para abrir una terminal de git donde lanzaremos los comandos del curso...

botón derecho y Git Bash here

git --version

comando de ayuda

git help

git help <comando>

**/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* configuración (es un oculto en tu usuario de windows .gitconfig \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

PARA ESTABLECER NOMBRE DE USUARIO

git config --global user.name "Superamoto"

PARA PONER EMAIL DE ENVIO POR DEFECTO

git config --global user.email "carlosaguirreperez@gmail.com"

PARA HACER QUE EL vim SEA EL EDITOR DE GIT.

git config --global core.editor "vim"

PARA VER LA CONFIGURACIÓN DEL GIT

git config --global -e

**/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* VIM \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

Editar

i

salir sin guardar

:q

salir guardando

:q!

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ABRIENDO NUESTRO PROYECTO EN ATOM \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

arrastramos la carpeta del proyecto a atom a la pestaña "untitled"

Para quitarlo botón derecho sobre el proyecto y remove project folder

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

Los archivos md son parecidos a los html. .Tienen un estándar de escritura

**/\*\*\*\*\*\*\*\*\*\*\*\*\* ALIAS DE COMANDOS \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

configuramos los alias de forma global (da igual la carpeta en la que estemos)

git config --global alias.lg "log --oneline --decorate --all --graph" **(git lg es lo que se escribe para usar el alias)**

git config --global alias.s "status -s -b"

git config --global alias.cfg "config --global -e"

git config --global alias.diffst "diff --staged"

git config --global -e NOS ABRE EL .gitconfig para ver todo lo que contiene (con los alias)

git config --global -l NOS MUESTRA LOS ALIAS CREADOS (mejor que abrir el .gitconfig no sea que lo toquemos y nos carguemos algo)

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*GITH BASH \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

ls -l muestra todo con detalle

ls -al muestra todo con detalle (incluído los ocultos)

Para CREAR UN REPOSITORIO LOCAL en la carpeta del proyecto (crea una carpeta oculta .git que es donde estará todo. NO TENEMOS QUE TOCAR NADA AHÍ DENTRO.

(LO GESTIONAN LOS COMANDOS). Este fichero lo podemos llevar a otro proyecto y es como tener todo el historial del mismo.

para inicializar un proyecto y que se cree un repositorio git. (Se creará un oculto .git en la carpeta donde se ejecutó el comando siguiente:)

git init

para ver lo que hay que git sabe que existen en la carpeta pero que GIT aún no trabaja (SALEN EN rojo los que no se hayan añadido al "stage" con el add)

git status (muestra detallado todo)

git status -s (muestra sólo lo imprescindible)

git status -s -b (muestra lo que hay por subir y la rama actual)

git status -sb (lo mismo que antes pero uniendo los parámetros s y b)

para agregar todos los archivos para que esté pendiente de los cambios (LOS SUBE AL STAGE o ESCENARIO, que es donde está todo lo que será un snapshot futuro...)

git add . git (para todo)

git add -A (añade el resto de archivos)

git add index.html (añade al escenario el archivo index.html)

git add \*.png (añade todos los archivos png)

git add css/ (añade toda la carpeta css)

git add "\*.txt" (añade todos los txt del proyecto estén en la carpeta que estén)

git add \*.txt (añade todos los txt de la carpeta actual)

git add --all (agrega todo lo que está pendiente)

git add <lista de archivos> (pones una lista)

git add pdfs/\*.pdf (agrega vertodo lo que hay en una carpet--a y con una extensión)

git add -u (actualizar el stage. Lo utilizó cuando renombró un archivo y el git se pensaba que había borrado el anterior y creado uno nuevo)

Para quitar de los agregados un archivo (SACARLO DEL STAGE)

git reset <archivo> (de todas las maneras vistas con el add pero con el reset)

git reset HEAD README.md (QUITA DEL HEAD el archivo README.md) **(HEAD es el último commit de la rama actual)**

git status (ahora los vewmos en verde significa que git ya "controla" esos archivos. Formarán parte de un snapshot que vamos a hacer)

para hacer un snapshot o fotografía de cómo se encuentra el proyecto en un momento dado

git commit (Así a secas se nos abre un fichero PARA PONER EL COMMIT que podemos editar pulsando "a") para salir ESC, : wq (escribe y graba)

git commit -m "Primer commit" (le metemos un comentario o mensaje QUE SEA MUY DESCRIPTIVO DE LO QUE SE HA HECHO)

git commit -am "README.md actualz" (con -am hacemos add y commit al mismo tiempo)

para modificar el mensaje...

git commit --amend -m "Actualizamos el README.md"

git commit --amend (SIN EL MENSAJE ABRIRÁ EL EDITOR VIM PARA MODIFICAR AHÍ MISMO EL ÚLTIMO COMMIT)

Para deshacer la subida a stage (UNSTAGE)

git reset HEAD <file>

git reset <file> (también a secas le funciona al menda…)

git reset HEAD <file>^ (para deshacer los cambios tras el último commit)

git rm –cached <file> (sale en el vídeo aunque él no lo menciona)

Para modificar UN COMMIT en concreto... es decir modificar archivos y meterlos en un commit que nos interese

git reset --soft HEAD (último commit)

git reset --soft HEAD^ (antes de ese último commit) NO LO ENTIENDO!!!

git reset --soft 29a3508 (es el id de una snapshot o commit)

Para movernos a un commit determinado

git reset --mixed 3fed1e1 (iría al comit de ese id) se perderían los commits posteriores pero no los archivos que están en amarillo en atom y con todo lo posterior

Para cargarnos todos los commits posterioes a uno ( y los cambios en los archivos) O VOLVER AL MISMO

git reset --hard 3fed1e1

otra vez hacemos git status y nos muestra "on branch master nothing to commit, working tree clean"

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Para deshacer lo cambiado DESDE EL último commit... (el git controla los archivos de la carpeta por eso parece que atom también lo reconoce y vuelve a aparecer nuestro trabajo tal y como estaba)

git checkout -- . (para todos los archivos) ES CON DOS GUIONES

git checkout -- README.md (para un archivo en concreto y esté o no en el stage (tras add))

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

PARA VER EL HISTORICO DE COMMITS

git log

git log --oneline (sólo una línea)

git log --oneline --decorate --all --graph (all muestra todo incluso otras ramas. Decorte y graph lo muestran de manera elegante)

PARA VER TODOS LOS MOVIMIENTOS QUE HEMOS HECHO (incluso cargarnos commits… )

git reflog

cuando hacemos git log nos aparece el HEAD (es el último commit que se hizo en la rama actual)

cuando ponemos git -- (dos guiones) es porque después viene una palabra (git --oneline)

cuando ponemos git - (un guión) es porque cada cosa que viene después es un parámetro (git -sae) donde sae serían tres parámetros (s, a y e)

PARA VER LAS DIFERENCIAS ENTRE EL COMMIT ANTERIOR Y EL MOMENTO ACTUAL

git diff (SIN HABER AÑADIDO AL STAGE EL ARCHIVO)

git diff --stage (HABIENDO AÑADIDO AL STAGE EL ARCHIVO)

PARA CAMBIAR UN NOMBRE DE UN ARCHIVO DESDE GITBASH

git mv destruir-mundo.txt salvar-mundo.txt

PARA ELIMINAR UN ARCHIVO

git rm salvar-mundo.txt

PARA TENER ARCHIVOS O CARPETAS QUE IGNORE EL GIT... (NO LOS ESTUDIE EN EL git status ...)

CREAMOS EL ARCHIVO **.gitignore** (EN LA CARPETA DEL PROYECTO) y dentro ponemos el nombre del archivo o ruta a ignorar por git (archivos o carpetas)

El .gitignore se sube al repositorio para que le proyecto lo tenga en cuenta.

**/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* RAMAS \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

**(HEAD es el último commit de la rama actual)**

merge

Al ejecutar el comando git merge la salida da una de las siguientes dependiendo de si encontró conflictos o no.

fast-forward cuando en la MASTER no ha habido modificaciones y se puede integrar otra rama fácilmente

uniones automáticas. En la MASTER hubo cambios pero no hay conflictos con la rama nueva

unión manual git no lo puede resolver de manera automática sino que nos lo propone manualmente. después hace un commit nuevo con todo

CREAR RAMA

git branch rama-villanos

CREAR RAMA Y CAMBIAR A ELLA EN UN SOLO PASO

git checkout -b rama-villanos

VER RAMAS (la verde es la actual)

git branch

CAMBIAR DE RAMA

git checkout rama-villanos ; git checkout master

VER DIFERENCIAS ENTRE RAMAS

git diff rama-villanos master

UNIR RAMAS (ESTANDO EN LA MASTER)

git merge rama-villanos

BORRAR UNA RAMA YA HABIENDO HECHO MERGE

git branch -d rama-villanos

Los TAGs hacen referencia a un commit y el estado del proyecto en ese mismo instante (HITOS IMPORTANTES... versión 1.0 del programa, etc)

para crear tags

git tag nombreDelTag

git tag -a v1.0.0 -m "versión 1.0.0" (a es como anotation. Esto es para el commit actual)

git tag -a v0.1.0 345d7de -m "Versión 0.1.0" (con el código de un commit determinado y así ponerle un tag)

para ver los tags

git tag

git show v1.0.0 (para ver el mensaje de un tag)

borrar un tag

git tag -d nombreDelTag

**/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* STASH \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

el stash es como una despensa cambios temporales posteriores al último commit y dejar el proyecto en ese commit

que no nos interesan tener en cuenta aún o que podemos dejar para más adelante.

PARA CREAR UN STASH (stash = reserva)

git stash (crea un WIP(work in progress) y un index) Podemos tener varios stashes stash@{0} sería el primero

git stash save (lo mismo)

PARA RESTAURAR LOS CAMBIOS QUE SE HICIERON EN EL ULTIMO STASH

git stash apply (RESTAURA LA PRIMERA ENTRADA DEL STASH stash@{0})

git stash apply stash@{1} (para restaurar un stash determinado. Pero seguiran en el git stash list hasta que hagamos git stash drop)

PARA LISTAR LOS TRABAJOS EN PROGRESO QUE HAY

git stash list (WIP: WORK IN PROGRESS)

CUANDO tras eliminar el stash con el pop hay un conflicto y se resuelve... no se borra el stash del git stash list. En ese caso:

git stash drop (BORRA LA PRIMERA ENTRADA DEL STASH stash@{0})

git stash drop stash@{1} (PARA BORRAR UNA EN CONCRETO)

PARA ELIMINAR EL STASH

git stash pop (es como un git stash apply + git stash drop)

PARA GUARDAR TODO MENOS LO QUE HAY EN NEL STAGE (PREPARADOS PARA HACER UN COMMIT)

git stash save --keep-index

PARA GUARDAR TODOS LOS ARCHIVOS INCLUSO A LOS QUE GIT NO DA SEGUIMIENTO (QUE NO SE LES HA HECHO EL git add) (.gitignore)

git stash --include-untracked

para que el list dé más información

git stash list --stat

git show stash (da info del stash@{0})

git show stash@{1}

PARA METER MENSAJES EN LOS STASH

git stash save "Agregamos un mensaje" (por defecto para el stash{0})

PARA BORRAR TODO LO QUE HAY EN EL STASH A LO BURRO SIN MANERA DE RECUPERARLOS

git stash clear (¡¡CUIDADO!!) (por defecto para el stash{0})

**/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* REBASE \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

cuando en la rama master hacemos commits que nos pueden hacer falta en otra rama nueva, actualizamos el punto donde creamos ésta última para que los nuevos commits de la rama máster se le incluyan

git checkout rama-misiones (nos cambiamos a nuestra rama nueva)

git rebase master (hace copia temporal de los commits de la rama nueva, pasa el puntero de la misma al

último commit de la master -LO REBASA- y vuelve a traer los commits de la rama) VIDEO 6 stash-rebase

por supuesto también se puede hacer con ramas que salen de otras ramas

\*

|

\* \*

| |

| \*

|/

\*

master rama-misiones

\*

|

\*

/

\*

|

\*

|

|

|

\*

master rama-misiones

después hay que volver a la master y hacer un merge (que hará un fastforward) y drop de la rama-misiones, ya que el rebase no une la rama con la master

**rebase interactivo (vídeo 8 del git stash y git rebase)**

**git rebase -i HEAD~3** (LA i es para que sea interactivo . Apunta a tres commits antes del HEAD - podría ser otro hash de otro commit en vez del HEAD)

SE usa para ORDENAR COMMITS

PARA CORREGIR MENSAJES DE COMMITS

PARA UNIR COMMITS

PARA SEPARAR COMMITS (cuanod un commit es muy gordo y queremos fraccionarlo)

Cuando usamos éste comando se nos abre el editor con tantos hash de commits como el número HEAD~<NUMERO>

están en orden inverso, es decir, el commit más viejo arriba. Nos vienen unas pistas de parámetros.

En el editor para escribir pulsamos “a”.

s **SQUASH (apretujar)** Para unir dos commits en uno ponemos delante squash o s. (en el último y se unirá al que tenga inmediatamente encima dentro del editor)

al salir del editor con :wq se nos abre otro editor para meter un mensaje. Lo ponemos y en paz

r **REWORD (reelaborar)** Para cambiar el mensaje del commit usamos r o reword

e **EDIT** Para editar un commit. al hacer :wq nos salta otro interactivo para el mensaje y al hacer :wq nos avisa que

para terminar de editar tenemos que hacer git rebase --continue

mientras no hagamos ese --continue, al hacer git s no nos dice en qué rama estamos, como para recordarnos

que seguimos en el rebase. Se verá HEAD (no branch)

git reset HEAD^ (Para revertir los últimos cambios tras un commit)

:wq para salir guardando del editor