Comandos para mover archivos entre los estados de Git:

* $ git init - Inicializa un repositorio local.
* $ git status: Nos permite ver el estado de todos nuestros archivos y carpetas.
* $ git add: Nos ayuda a mover archivos del Untracked o Unstaged al estado Staged. Podemos usar git nombre-del-archivo-o-carpeta para añadir archivos y carpetas individuales o git add -A para mover todos los archivos de nuestro proyecto (tanto Untrackeds como unstageds).
* $ git reset HEAD: Nos ayuda a sacar archivos del estado Staged para devolverlos a su estado anterior. Si los archivos venían de Unstaged, vuelven allí. Y lo mismo se venían de Untracked.
* $ git commit: Nos ayuda a mover archivos de Unstaged a Staged. Esta es una ocasión especial, los archivos han sido guardado o actualizados en el repositorio. Git nos pedirá que dejemos un mensaje para recordar los cambios que hicimos y podemos usar el argumento -m para escribirlo (git commit -m “mensaje”).
* $ git rm: Este comando necesita alguno de los siguientes argumentos para poder ejecutarse correctamente:
* $ git rm --cached: Mueve los archivos que le indiquemos al estado Untracked.
* $ git rm --force: Elimina los archivos de Git y del disco duro. Git guarda el registro de la existencia de los archivos, por lo que podremos recuperarlos si es necesario (pero debemos usar comandos más avanzados).
* $ git show nos muestra los cambios que han existido sobre un archivo y es muy útil para detectar cuándo se produjeron ciertos cambios, qué se rompió y cómo lo podemos solucionar. Pero podemos ser más detallados.
* $ git reset. Hay dos formas de usar git reset: con el argumento --hard, borrando toda la información que tengamos en el área de staging (y perdiendo todo para siempre). O, un poco más seguro, con el argumento --soft, que mantiene allí los archivos del área de staging para que podamos aplicar nuestros últimos cambios pero desde un commit anterior.
* $ git checkout + ID del commit nos permite viajar en el tiempo. Podemos volver a cualquier versión anterior de un archivo específico o incluso del proyecto entero. Esta también es la forma de crear ramas y movernos entre ellas. PELIGRO

Mis anotaciones

#To config the user

$ git config --global [user.name](http://user.name/) “johan7perez”  
$ git config --global user.email ["johan7perez@gmail.com](mailto:%22johan7perez@gmail.com)"

#To see the current config  
$ git config --list

#To see where the git configuration is stored  
$ git config --list --show-origin

#To start a new empty project with git  
$ git init

#To see the status of the current project  
$ git status

#To add a new file/modificated file to the staging area  
$ git add

#To remove a new file/modificated file from the staging area (cached, remove the file from the ram memory also)  
$ git rm --cached

#To see the history of a file  
$ git show

#To show the log of the commit or a file, stat is to see the changes at a bytes level  
$ git log [–stat]

#To confirm the changes that we have in the staging area  
$ git commit -m “Message”

#To see the differences of two files / if we do not pass any flag it will compare the current directory and the staging area  
$ git diff [commit file code] [another commit file code]

#To back in the time, if it’s hard will delete everithing made, but if it’s not, will treat the file as a modificated/untracked file  
$ git reset [commit code] [–hard]

#To see back in the pass and see the status of a file / for leave the file in the original state just use ‘git checkout master’  
$ git checkout [commit code] [file name]

**git init:** crea un área en memoria RAM que se llama staging en donde se agregan todos los cambios del proyecto y se crea un repositorio.  
º **git add:** una vez realizados los cambios, se agregan a staging con este comando y el archivo o proyecto pasa a vivir al staging.  
º **git commit -m**: agrega el archivo o proyecto a la carpeta repositorio llamada master.  
º **git config:** muestra todas las configuraciones que tiene git.  
º **git config --list:** nos permite ver toda la configuración por defecto de nuestro git.  
º \*\* git config --list --show --orgin: **permite ver en donde están las configuraciones guardadas.  
º git config --global: permite hacer cambios a nuestro git, como cambiar todos los usuarios.  
º git config --global**[**user.name**](http://user.name/)**nombreUser: permite cambiar el nombre del usuario.  
º git config --global user.email Email: permite cambiar el correo del usuario.  
º git log: muestra toda la historia del archivo o proyecto modificado, muestran los datos de quien lo modifico.  
º git show: muestra los cambios que hemos realizado al archivo.  
º git diff: permite comparar entre las versiones de los archivos modificados.  
º git reset (aquí va la cadena de la version) --hard: vuelve todo a la versión anterior.  
º git reset (aquí va la cadena de la version) --soft: vuelve todo a la versión anterior, pero lo que tenemos en staging sigue ahi.  
º git clone url: permite traer los datos desde un servidor remoto.  
º git push: envia la version lista al repositorio remoto.  
º** git fetch:\*\* nos permite traer una actualización después de clonarlo, pero no lo copea en los archivos. Para poder copiarlo en los archivos tenemos que fusionar la ultima versión que esta en el repositorio local con la versión actual.  
º **git merge:** nos permite hacer la fusión de las versiones de los archivos de las ramas de master.  
º **git pull:** copia el repositorio local, la bd de cambios y copia el directorio.  
**git commit -am:** automáticamente realiza un git add de los cambios pero solo funciona con archivos a los que ya anteriormente se les hizo add. PARA LOS ARCHIVOS NUEVOS NO SE UTILIZA ESTE COMANDO.

**CREAR RAMAS EN MASTER**  
1 º git branch nombreRama: crea la rama  
2 º git checkout nombreRama: este nos permite mover a la rama creada.

**FUSIONAR MASTER CON OTRA RAMA**  
Esto es ara fusionar los cambios de una rama a master.  
1º Si estamos en otra rama nos tenemos que mover a master con el comando **git checkout master.**  
2º **git merge nombreRama:** fusiona a master los cambios realizados en otra rama.

<h1>Curso de Git en Platzi</h1>

Comandos para mover archivos entre los estados de Git:

* $ git init - Inicializa un repositorio local.
* $ git status: Nos permite ver el estado de todos nuestros archivos y carpetas.
* $ git add: Nos ayuda a mover archivos del Untracked o Unstaged al estado Staged. Podemos usar git nombre-del-archivo-o-carpeta para añadir archivos y carpetas individuales o git add -A para mover todos los archivos de nuestro proyecto (tanto Untrackeds como unstageds).
* $ git reset HEAD: Nos ayuda a sacar archivos del estado Staged para devolverlos a su estado anterior. Si los archivos venían de Unstaged, vuelven allí. Y lo mismo se venían de Untracked.
* $ git commit: Nos ayuda a mover archivos de Unstaged a Staged. Esta es una ocasión especial, los archivos han sido guardado o actualizados en el repositorio. Git nos pedirá que dejemos un mensaje para recordar los cambios que hicimos y podemos usar el argumento -m para escribirlo (git commit -m “mensaje”).
* $ git rm: Este comando necesita alguno de los siguientes argumentos para poder ejecutarse correctamente:
* $ git rm --cached: Mueve los archivos que le indiquemos al estado Untracked.
* $ git rm --force: Elimina los archivos de Git y del disco duro. Git guarda el registro de la existencia de los archivos, por lo que podremos recuperarlos si es necesario (pero debemos usar comandos más avanzados).
* $ git show nos muestra los cambios que han existido sobre un archivo y es muy útil para detectar cuándo se produjeron ciertos cambios, qué se rompió y cómo lo podemos solucionar. Pero podemos ser más detallados.
* $ git reset. Hay dos formas de usar git reset: con el argumento --hard, borrando toda la información que tengamos en el área de staging (y perdiendo todo para siempre). O, un poco más seguro, con el argumento --soft, que mantiene allí los archivos del área de staging para que podamos aplicar nuestros últimos cambios pero desde un commit anterior.
* $ git checkout + ID del commit nos permite viajar en el tiempo. Podemos volver a cualquier versión anterior de un archivo específico o incluso del proyecto entero. Esta también es la forma de crear ramas y movernos entre ellas. PELIGRO puede ser peligroso
* $ git merge nos permite crear un nuevo commit con la combinación de dos ramas (la rama donde nos encontramos cuando ejecutamos el comando y la rama que indiquemos después del comando).

Principales

Repositorio sin REDME

$ git init - Inicializa un repositorio local

$ git remote add origin (url)

$ git status: Nos permite ver el estado de todos nuestros archivos y carpetas

$ git add: Nos ayuda a mover archivos del Untracked o Unstaged al estado Staged

$ git commit: Nos ayuda a mover archivos de Unstaged a Staged. Esta es una ocasión especial, los archivos han sido guardado o actualizados en el repositorio. Git nos pedirá que dejemos un mensaje para recordar los cambios que hicimos y podemos usar el argumento -m para escribirlo (git commit -m “mensaje”)

$ git push (origin) (master) (Sube Cambios)

Repositorio con REDME

$ git init - Inicializa un repositorio local

$ git remote add origin (url)

$ git remote -v(es verval)

$ git pull origin master --allow-unrelated-histories fusiona lo que viene con lo que tengo

$ git pull origin master (Descarga cambios)

$ git status

$ git commit

$ git push (origin) (master) (Sube Cambios)

Otros

$ git log muestra información y historial de los commits

$ git diff: permite comparar entre las versiones de los archivos modificados

$ git reset HEAD: Nos ayuda a sacar archivos del estado Staged para devolverlos a su estado anterior. Si los archivos venían de Unstaged, vuelven allí. Y lo mismo se venían de Untracked

$ git reset nombredelcomit --hard vuelve al estado del commit deseado borra lo nuevo para volver al ultimo commit.

$ git rm --cached: Mueve los archivos que le indiquemos al estado Untracked

$ git rm --force: Elimina los archivos de Git y del disco duro. Git guarda el registro de la existencia de los archivos, por lo que podremos recuperarlos si es necesario (pero debemos usar comandos más avanzados)

$ git show nos muestra los cambios que han existido sobre un archivo y es muy útil para detectar cuándo se produjeron ciertos cambios, qué se rompió y cómo lo podemos solucionar. Pero podemos ser más detallados

$ git checkout + ID del commit nos permite viajar en el tiempo. Podemos volver a cualquier versión anterior de un archivo específico o incluso del proyecto entero. Esta también es la forma de crear ramas y movernos entre ellas. PELIGRO puede ser peligroso

$ git merge nos permite crear un nuevo commit con la combinación de dos ramas (la rama donde nos encontramos cuando ejecutamos el comando y la rama que indiquemos después del comando)

CREAR RAMAS EN MASTER  
1 º $ git branch nombreRama: crea la rama  
2 º $ git checkout nombreRama: este nos permite mover a la rama creada.

$ git commit -am “Mensaje” Para crear commits dentro de la rama

FUSIONAR MASTER CON OTRA RAMA  
Esto es ara fusionar los cambios de una rama a master.  
1º Si estamos en otra rama nos tenemos que mover a master con el comando git checkout master.  
2º git merge nombreRama: fusiona a master los cambios realizados en otra rama.

Si da error se deben hacer los cambios luego se hace un comit se suben los cambios y luego se vuleve a hacer el merge.

En gitlab

Se debe dirigir a la carpeta donde se quiera clonar

git clone url

Se dirige a la carpeta creada por git

git add .

git commit -m “Comit texto”

git push -u origin master

Para clonar un proyecto compartido

Se va a la carpeta donde se quiera guardar

git clone url

cd nombreproyecto

git checkout develop

git pull

Llaves públicas y privadas

ssh-keygen -t rsa -b 4096 -C “[youremail@example.com](mailto:youremail@example.com)”

luego se le asigna una contraseña

Posterior se verifica que el archivo .ssh se creo

Se copia la llave publica

Se dirige a la consola

Se intro duce el comando eval $(ssh-agent - s) que verifica que el servidor ssh este activo

ssh-add ~/.ssh/id\_rsa Agregar la llave al sistema o servidor