**Git Essential Training**

Contenido

[Glosario 2](#_Toc2682234)

[Links útiles 2](#_Toc2682235)

[Capítulo 1. ¿Qué es Git? 3](#_Toc2682236)

[Capítulo 2. Configuración 3](#_Toc2682237)

[Capítulo 3. Comenzando 4](#_Toc2682238)

[Capítulo 4. Conceptos y Arquitectura. 6](#_Toc2682239)

[Capítulo 5. Hacer cambios a los archivos 7](#_Toc2682240)

[Capítulo 7. Deshacer Cambios 8](#_Toc2682241)

[Capítulo 8. Ignorar archivos 14](#_Toc2682242)

[Capítulo 9. Navegar el árbol de los commits. 16](#_Toc2682243)

[Capítulo 10. Branching 21](#_Toc2682244)

[Capítuo 11. Merging Branches 25](#_Toc2682245)

[Capítulo 12. Escondiendo (stashing) cambios 29](#_Toc2682246)

[Capítulo 13. Repositorios remotos 32](#_Toc2682247)

[Capítulo 14. Herramientas 38](#_Toc2682248)

[…………………………………Learning Git and Github 39](#_Toc2682249)

[Capítulo 1. Empezando 39](#_Toc2682250)

[Capítulo 2. Trabajando con proyectos de GitHub 40](#_Toc2682251)

# Glosario

* **~ (tilde en inglés):** representa el usuario en un directorio en el bash de GIT.

# Links útiles

# Capítulo 1. ¿Qué es Git?

Git es un software que monitorea los cambios que se hacen a directorios o archivos.

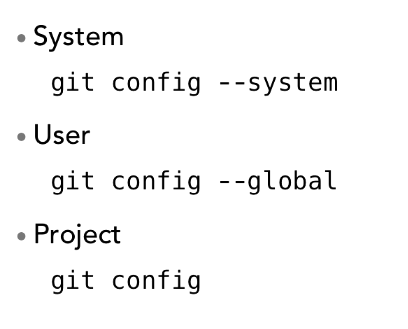
* Especialmente cambios de textos
* Versión 1, versión 2, versión 3.

Se le conoce como un VCS (Version control system) o SCM (Source Code Management)

# Capítulo 2. Configuración

Podemos guardar configuraciones a:

* *System*
* *User*
* *Project*



* Para ver las configuraciones se puede hacer: ***git config—list***
* Para cambiar el editor de texto predeterminado:

**$ git config—global core.editor “notepad.exe -Wl1”**

Auto completion en Git

Ya lo posee en Windows, pero Mac y Unix si lo necesita.

Usando Git Help

* ***Git help <command> =*** Nos muestra cómo usar GIT. Si se usa la parte del comando (sin los <>), da más información sobre el comando.

# Capítulo 3. Comenzando

Iniciar un repositorio

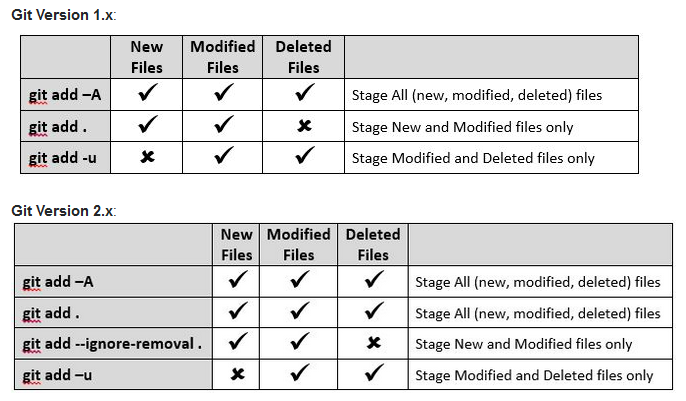
* ***Git init:*** Para inicializar un repositorio y lograr monitorear todo el proyecto desde donde se inicializa.

Ver que hay en la carpeta de GIT

Generalmente, este es el espacio de trabajo de GIT así que no se debería hacer muchas cosas ahí. Sin embargo, existe una carpeta que si se puede utilizar es el archivo ***config***a nivel del proyecto.

Primer commit

* Hacer cambios



git add -A is equivalent to git add—all

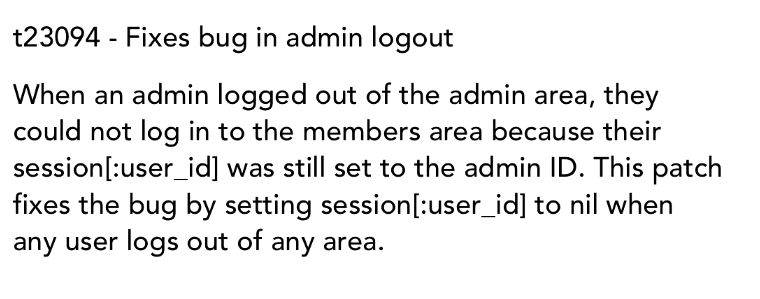
git add -u is equivalent to git add—update

* Añadir los cambios
* Commit los cambios en el repositorio con un mensaje

Mensajes en un Commit

* Resumen de 50 caracteres
* Una línea blanca
* Añadir información con una descripción más detallada
* Describir el *commit* en el tiempo del presente.

Ejemplo de un buen *commit:*



Ver el LOG de los commits

Para ver el log de todos los commits de un proyecto, simplemente se digita el siguiente comando: ***git log***.

**$ git log**

**commit 7aa195bd44ea0610796a1fcac8024c4a556f0621 (HEAD -> master)**

Author: Kevin Skoglund <[**someone@nowhere.com**](mailto:someone@nowhere.com)>

Date: Fri Jun 15 08:29:22 2012 -0400

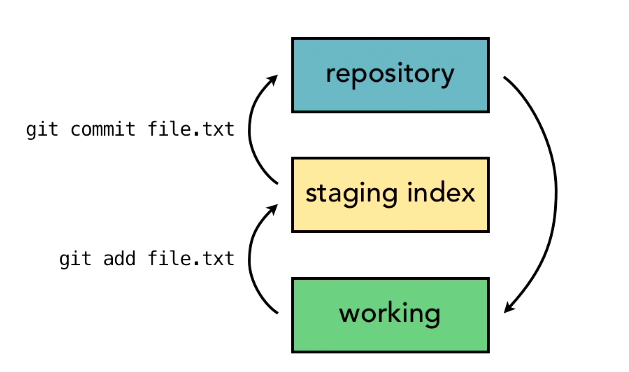
**Initial commit**

* Ver el número de commits indicados más recientes: ***git log -n #***.
* Ver desde una fecha específica hasta el día de hoy: ***git log—since =2012-06-14***.
* Ver hasta una fecha específica: ***git log—until =2012-06-14***.
* Ver cierto autor: ***git log—author=” Kevin”***.
* GREP: Global Regular Expression Search; buscar palabras dentro de los commits: ***git log—grep =” Init”***.

# Capítulo 4. Conceptos y Arquitectura.

Arquitectura de tres árboles – three-trees

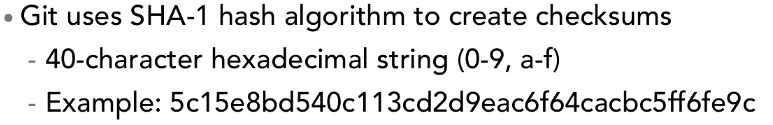
Existe el repositorio, el index de staging que es donde se guardan los cambios antes de hacer el *commit* y por último el espacio de trabajo.



Valores de HASH (SHA-1)

Cuando se añade un *commit* al repositorio, GIT genera un checksum para cada set de cambios para mantener la integridad de los datos.

Utiliza el algoritmo de SHA-1 que siempre generará un número que será de 40 caracteres hexadecimales.



Puntero de HEAD

Una variable de referencia llamada HEAD.

**Siempre apunta a la punta de la rama actual en nuestro repositorio, apunta al último commit.**

# Capítulo 5. Hacer cambios a los archivos

Añadir y Editar archivos

Importante comando para verificar que cambios se han realizado: ***git status***.

$ git status

On branch master

Untracked files:

(use “git add <file>...” to include in what will be committed)

second\_file.txt

third\_file.txt

nothing added to commit but untracked files present (use “git add” to track)

Ver los cambios que se han hecho (Directorio de trabajo).

Para ver cuáles han sido los cambios se puede hacer el comando: ***git diff (archivo)***.

$ git diff

diff—git a/first\_file.txt b/first\_file.txt

index 30f02a1..154f556 100644

--- a/first\_file.txt Es el archive (Archivo del repositorio REPRESENTA EL -)

+++ b/first\_file.txt (La nueva versión REPRESENTA EL +)

@@ -1 +1,3 @@

* This is the first file that I added to my project.

\ No newline at end of file

+This is the first file that I added to my project.

+

+It comes before all other files

\ No newline at end of file

Ver los cambios que se han hecho (Directorio de staging)

Para lograr ver los cambios comparando lo que está en el directorio de Staging y el repositorio se usa el siguiente comando:

**$ git diff—staged.**

Eliminar archivos

Se puede hacer como eliminando normalmente un archivo o con el comando de:

**$ git rm file\_to\_delete\_2.txt.**

**RM = remove.**

Mover y cambiar nombre de archivos

Se puede hacer como moviendo o cambiando el nombre de un archivo normalmente un archivo o con el comando de:

**$ git mv**



# Capítulo 7. Deshacer Cambios

Deshacer cambios en el directorio de trabajo

El comando de ***checkout*** puede lograr esto. Sin embargo, hay que tener en cuenta que este comando se usa para varios propósitos como trabajar con branches.

Lo que hace ***checkout*** es ir al repositorio, tomar la cosa con el nombre que se le asigno y hacer el directorio de trabajo verse así.

* Si se llama una Branch, pues se trae el Branch
* Si se llama un archivo, se baja el archivo.

Para evitar confusión si se quiere llamar una *Branch* o un archivo, se llama el archivo de la siguiente manera usando los guiones:

**$ git checkout—index.html**

Los guiones indican que no queremos otra Branch.

Quitar archivos del staging.

**$ git reset HEAD resources.html**

***Unstaged changes after reset:***

**M resources.html**

Simplemente, para bajar el archivo se usa el comando RESET. Esta instrucción aparece cada vez que se añaden archivos al staging para recordarnos como se hace.

Corregiendo commits

Solo se puede corregir el último Commit, es decir el que apunta HEAD.

**$ git commit—amend -m “Rearrange the items to bring on a trip”**

***[master 6449de7] Rearrange the items to bring on a trip***

**Date: Sat Dec 1 20:02:46 2018 -0500**

**1 file changed, 3 insertions(+), 3 deletions(-)**

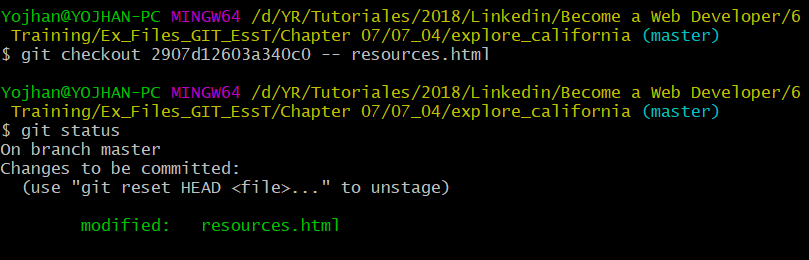
Cada vez que se hace esto, cambia el SHA del commit. Es por esto que solo se puede editar el último commit.

Cargar versiones antiguas

Tomando en cuenta que se quiera recuperar el archivo como estaba en cierto commit, se puede hacer de la siguiente manera:

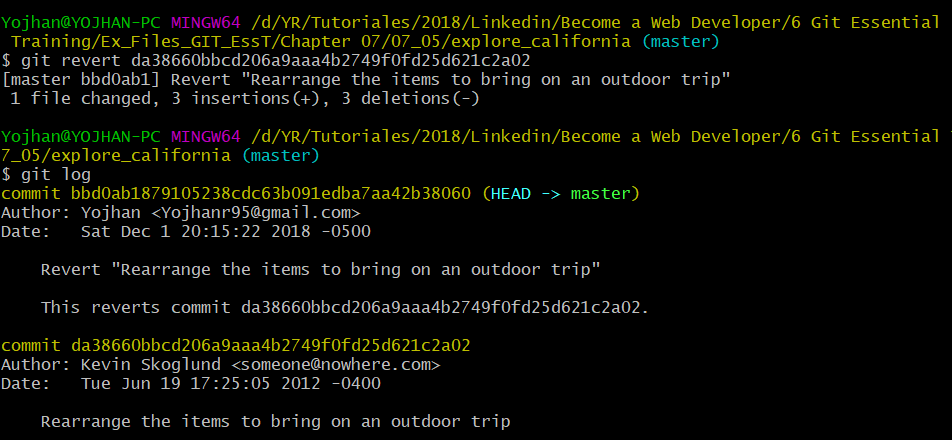
**$ git checkout 2907d12603a340c0 -- resources.html**

Indicando el SHA del commit, no necesariamente todo el SHA, doble guión para indicar que no se requiere la Branch y el archivo a recuperar. Esto, al traerlo lo pone en el **¡área de Staging!** Es decir que ya solo falta un commit para guardar los cambios.



Revertir un commit

Para revertir todo un commit se usa el comando ***revert*** usando el SHA del commit. Sin embargo, lo que hará es revertir todos los cambios que se habían hecho en el commit y es posible que haya conflictos adicionales que se resolverán con merging.



Usar RESET para deshacer commits

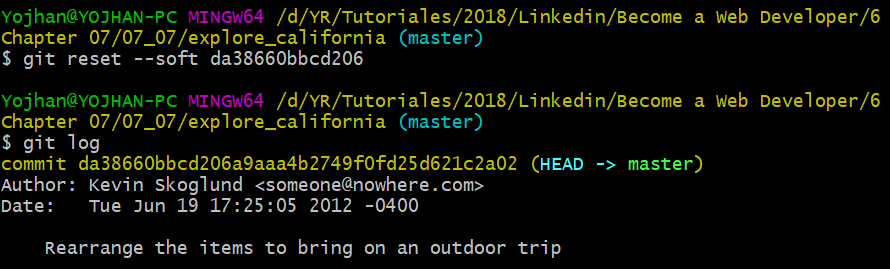
Nos permite deshacer varios commits. Hay que tener cuidado al usar este comando.

***Git reset***. Especificar donde el HEAD debería apuntar y lo que está después se elimina.

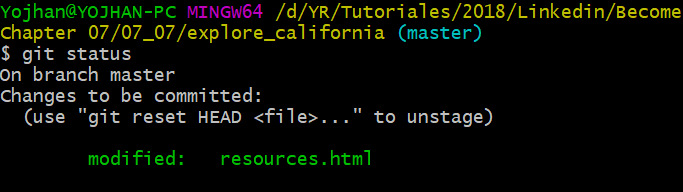
* soft: Mover el HEAD al ***commit*** especificado y no va a cambiar el ***staging index*** o el directorio de trabajo. Solo va mover el puntero. Es el más seguro que se puede hacer
* mixed: es el ***default***. Mueve el puntero al ***commit*** especificado y también el staging index para que se empareje con el repositorio. No cambia el directorio de trabajo.
* hard: Es el más destructivo. ¡Va a cambiar todo a donde se apunte! Todos los cambios después del commit donde se apunte se eliminarán.

Ejemplo SOFT RESET

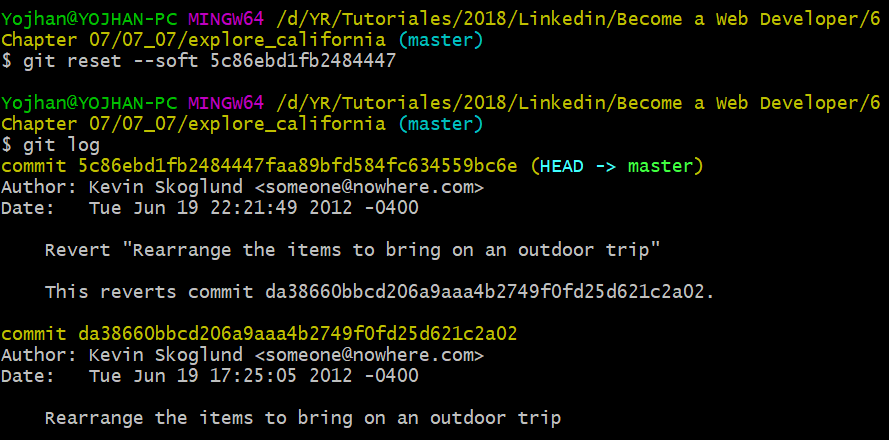
Se hace el cambio del puntero de la siguiente forma:



Y se puede observar que, en el **staging** y el **directorio de trabajo**, los cambios antiguos siguen intactos pendientes para hacer el commit.

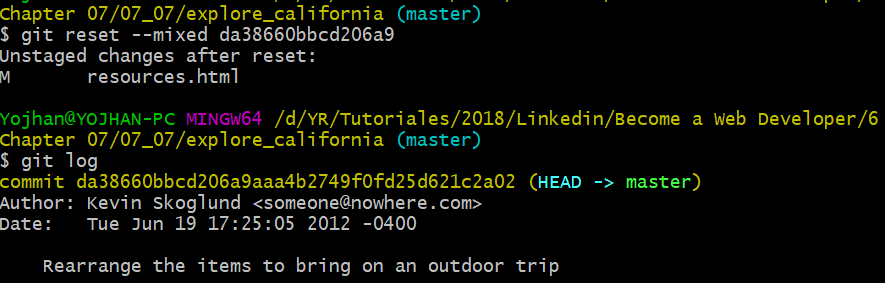


También se puede volver al commit que teníamos antes, siempre y cuando hayamos guardado el SHA anteriormente:

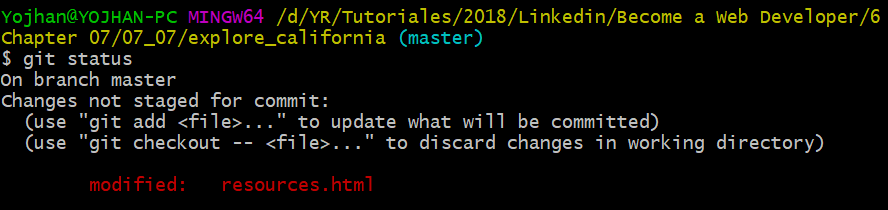


Ejemplo MIXED RESET

Trabaja igual que el anterior, solo que el staging index también va a cambiar para que quede igual con el repositorio, pero el directorio de trabajo sigue intacto. También nos recuerda a través del mensaje que deja, que no hay nada en el staging.



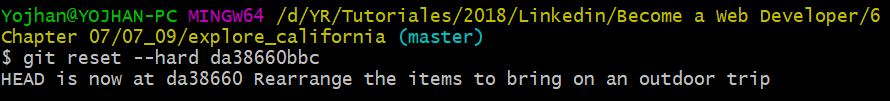
Y si verificamos el estado del proyecto, vemos que tenemos todo el trabajo realizado guardado.



Como en el anterior, se puede volver a el commit original que apuntaba HEAD haciendo el mismo comando: ***git reset—mixed SHA***.

Ejemplo HARD RESET

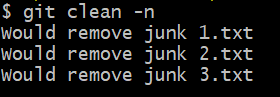
¡Elimina todo lo que hay después del commit a donde se apunte!



Todavía se puede mover al commit donde apuntaba el HEAD previamente para recuperar la información. ¡SIEMPRE Y CUANDO SEPAMOS EL SHA! Adicionalmente, es de manera temporal, no siempre estará disponible.

Remover archivos que no son rastreados (Tracked)

El siguiente comando sirve para quitar esos archivos no rastreados (eliminar) que por cierto motivo no se desea tener. Si se usa el ***-n***, se muestra los archivos que se removerían

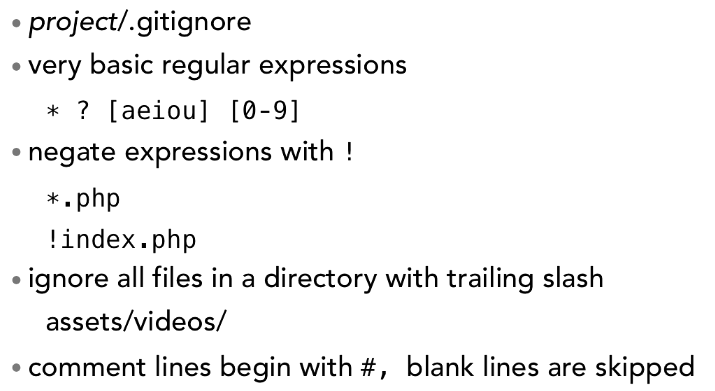


Para ejecutarlo correctamente se usa el ***-f***, para obligarlo a correr, o también se puede señalizar el archivo a eliminar como: ***git clean junk1.txt***.

# Capítulo 8. Ignorar archivos

Usando .gitignore

Sirve para decirle a **git** que archivos se pueden ignorar:



Se puede crear el archivo de. ***gitignore*** desde la consola.

¿Qué ignorar?

* Compilar el código fuente
* Packages y archivos comprimidos
* Logs y bases de datos
* Archivos del sistema operativo
* Archivos subidos por los usuarios.
* En Github muestra varios archivos que deben ser ignorados
* Articulo de ignorar archivos en github

<https://help.github.com/articles/ignoring-files/>

* Git ignores para ciertos lenguajes:

<https://github.com/github/gitignore>

Ignorar archivos globalmente

Sirve para ignorar en todos los repositorios, no únicamente en uno específico.

Es parte de la configuración de GIT **y es solo para el usuario**. Se debe crear primero el archivo en la carpeta del usuario y después se le dice a GIT sobe el archivo:

***Git config—global core.excludesfile ~/.gitignore\_global.***

Ignorar archivos que ya están siendo rastreados

Lo remueve del staging index sin quitarlo del repositorio. Así se puede para de rastrear el archivo:

***Git rm—cached tempfile2.txt.***

Rastrear directorios vacios

¡GIT no rastrea directorios vacíos! GIT solamente rastrea archivos.

Así que se debe usar un archivo así sea muy pequeño para mantener rastro de este directorio. Uno de los trucos más comunes es el siguiente:

***$ touch assets/pdfs/.gitkeep.***

# Capítulo 9. Navegar el árbol de los commits.

Referenciar commits:

En GIT la palabra: ***treeish*** significa algo que hace referencia parte del árbol. Ese “algo” puede variar ampliamente.

***Tree-ish***: en otro término, es una referencia a un commit porque ese commit referencia sucesivamente al árbol

* Usar el hash SHA-1
* Usar una versión corta del SHA-1
* Usar una rama
* Usar ascendencia o “linaje” de la siguiente manera para referirse a los commits padres, abuelos, etc.



Explorar el árbol

Se usa el comando: ***git ls-tree (tree-ish)***. Aquí se referencia la parte del árbol que queremos buscar y con el símbolo de ^, se mira el commit padre del que apuntamos

Yojhan@YOJHAN-PC MINGW64 /d/YR/Tutoriales/2018/Linkedin/Become a Web Developer/6 Git Essential Training/Ex\_Files\_GIT\_EssT/Chapter 09/09\_02/explore\_california (master)

$ git ls-tree master assets/

040000 tree 1ff91bf13a6d5461a85ad44f4cbacd77ee6c4917 assets/images

040000 tree ec4c1fc09292002d5389a24e4831a6d473dcf282 assets/javascripts

040000 tree d564d0bc3dd917926892c55e3706cc116d5b165e assets/pdfs

040000 tree 61324dc38b3c2856c5d96067000ca4bdafeb669a assets/stylesheets

Yojhan@YOJHAN-PC MINGW64 /d/YR/Tutoriales/2018/Linkedin/Become a Web Developer/6 Git Essential Training/Ex\_Files\_GIT\_EssT/Chapter 09/09\_02/explore\_california (master)

$ git ls-tree master^ assets/

040000 tree 1ff91bf13a6d5461a85ad44f4cbacd77ee6c4917 assets/images

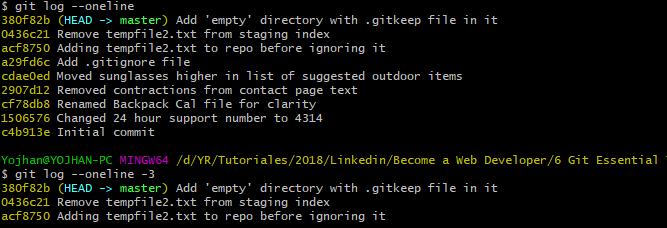
040000 tree ec4c1fc09292002d5389a24e4831a6d473dcf282 assets/javascripts

040000 tree 61324dc38b3c2856c5d96067000ca4bdafeb669a assets/stylesheets

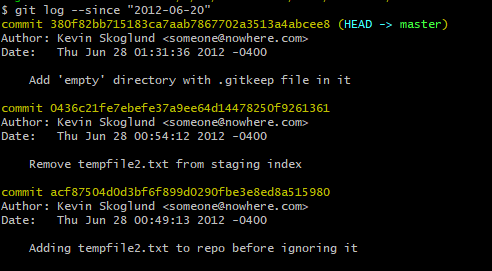
También cómo se puede observar, los resultados arrojados muestran que es un *tree* o un *blob*. *Blob* simplemente significa un archivo. El código que arroja, si es un *tree*, también puede ser usado para explorar el árbol general

Otras funciones de GIT LOG

***Git log—oneline***: Comprime la forma como muestra los resultados. Asi mismo si se le añade un número solo muestra los últimos 3 en este caso.



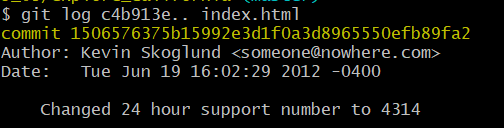
Asímismo, con fechas usando—since or—after o también—unitl o – before.



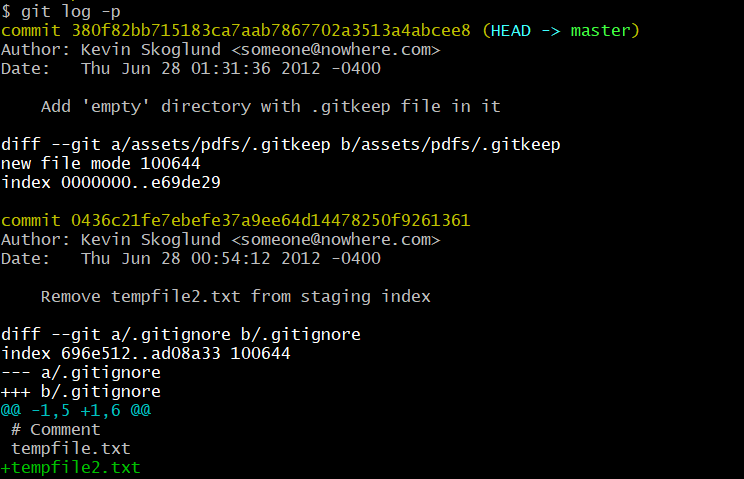
¡También puede ser como autor! ***--author***.

GREP: Global Regular Expression Search; buscar palabras dentro de los commits: ***git log—grep =” Init”***.

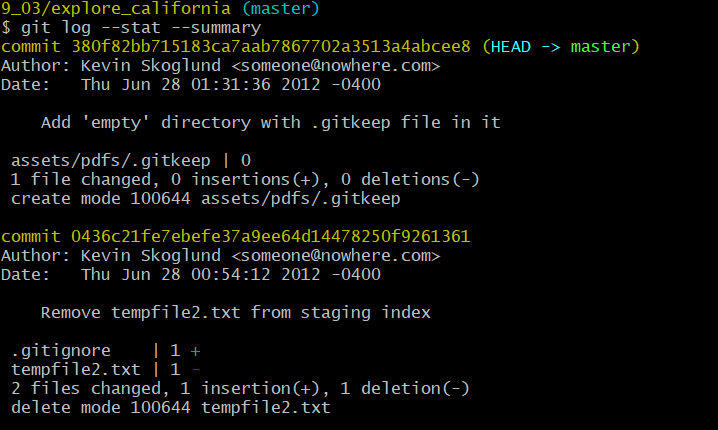
* Ver los commits que han afectado un archivo desde un commit en adelante. En este caso el commit indicado es el primero



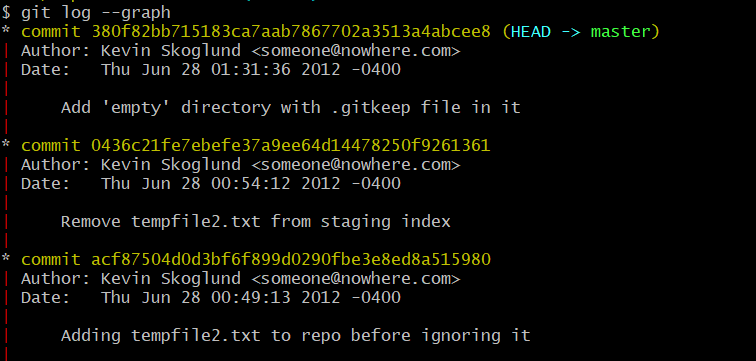
* Con el atributo de -p, muestra varios datos de lo que cambio en cada commit



* También es posible usar ***git log –stat –summary.***

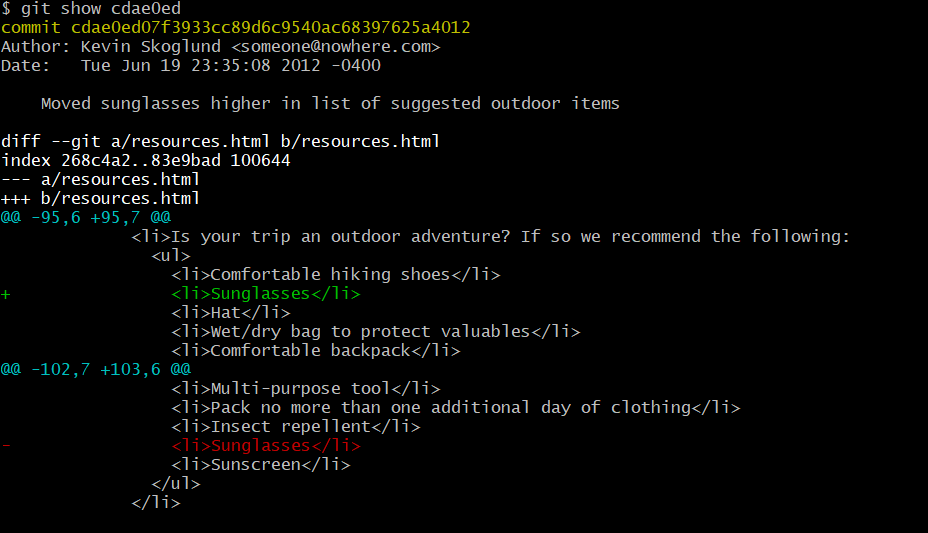


* Git log graph muestra el gráfico con ramas.



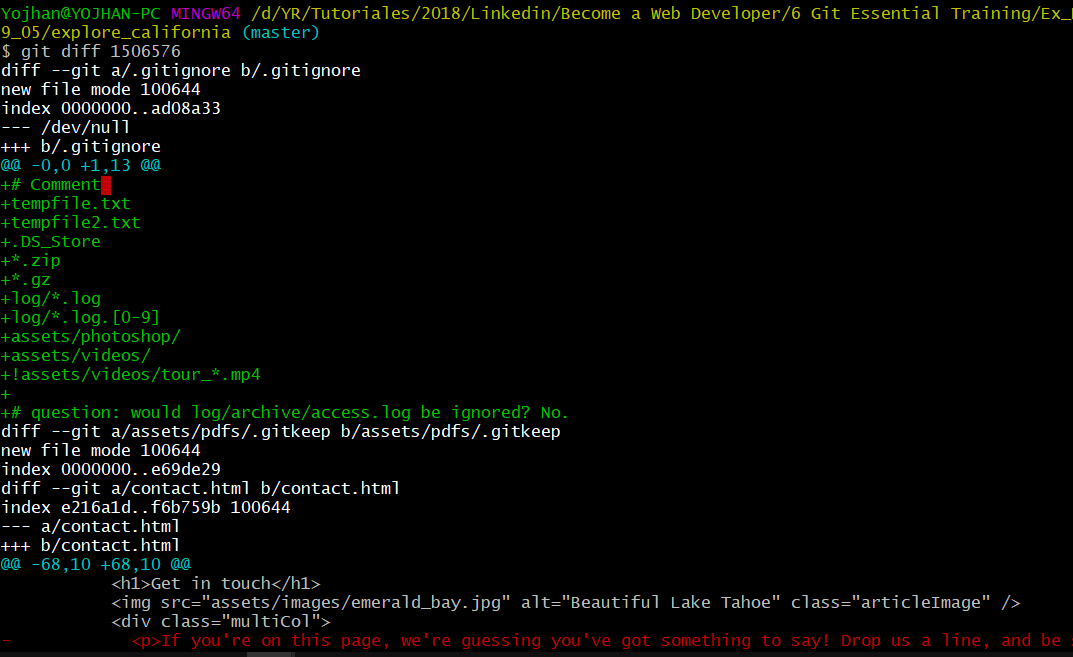
Ver un commit en especifico

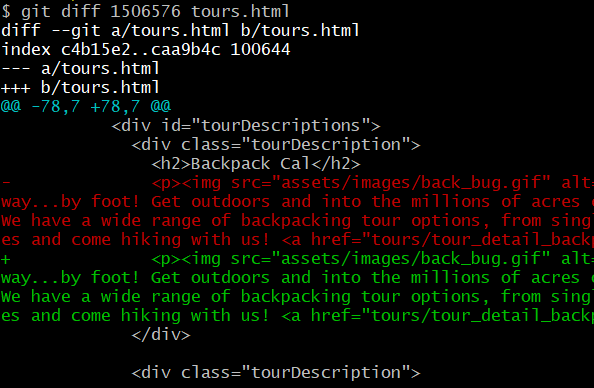
Se hace de la siguiente manera donde muestra detalles del commit y lo que cambió: ***git show commit***



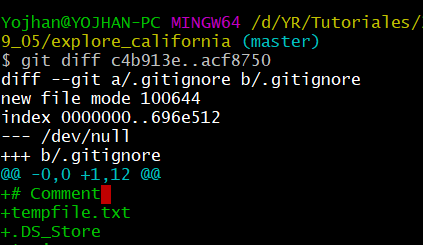
Comparar dos commits

En realidad, se comparan dos directorios y ver que ha cambiado entre los dos.

* Se usa ***git diff*** para comparar el directorio de trabajo con el staging index.
* Se usa ***git diff—staged*** para comparar el staging index con el repositorio
* Si se usa ***git diff SHA(commit)*** mostrará todos los cambios que se han hecho desde ese commit hasta el actual directorio de trabajo.
* Se puede ser más especifico comparando un archivo desde el commit elegido hasta la fecha actual en el directorio de trabajo.



* No tiene que ser unicamente con el directorio actual de trabajo, también se puede con otro commit. Se usa **(..)** para separar los dos commits a comparar.



Opciones adicionales para usar con DIFF

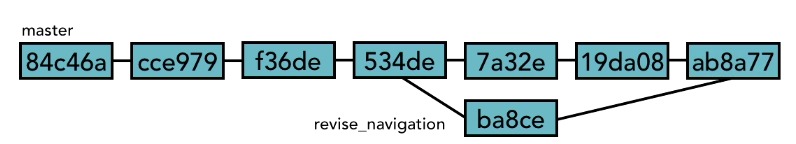
* Igualmente al hacer este procedimiento se pueden usar – stat y –summary
* -b (ignora el cambio de espacios)
* -w (ignorar todos los espacios)

# Capítulo 10. Branching

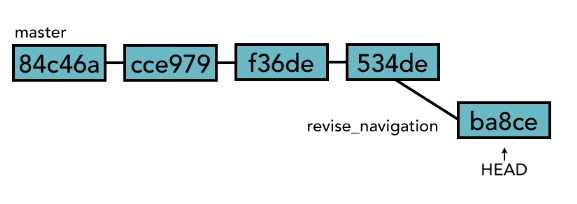
Las ramas son baratas: No cuestan muchos dolores de cabeza, fácil de trabajar con ellos.

Permiten trabajar en ideas para ver si funciona o no en una rama. Si no sirve se puede eliminar, pero si sirve se puede unir a la rama master.

Como se puede apreciar en el esquema, una vez se complete la aprobación para hacer *merging* del commit en la rama *revise\_navigation,* GIT hará una fusión y lo pondra en un nuevo commit en la rama *master.*



Cuando se hace algún commit en una rama, el puntero HEAD pasa a este si se está usando la rama:



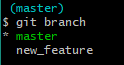
Crear branches

***Git branch nombre***



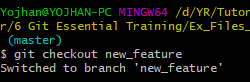
Ver branches

Si hacemos el comando ***git branch*** nos muestra las ramas que existen. El asterisco representa la rama en la que estamos trabajando.

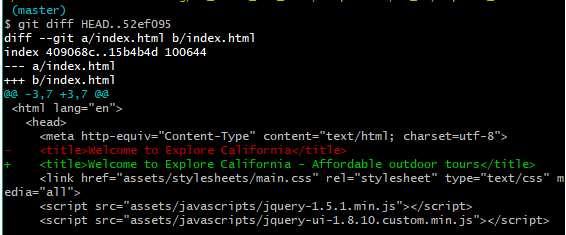


Cambiar de branches

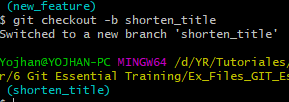
Simplemente con el ***checkout*** se puede cambiar de ramas.



Y una vez se hagan commits, es posible incluso compararlos con los commits de master una vez se cambie de nuevo a esta rama, como se muestra a continuación:

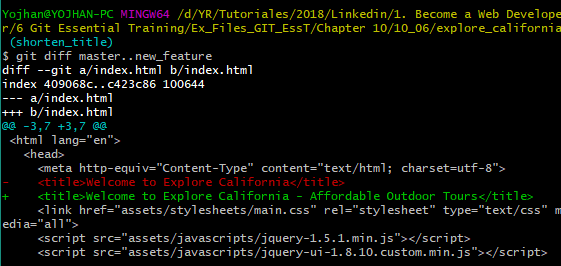


Crear y cambiar Branches al mismo tiempo



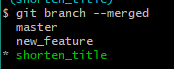
Comparar Ramas

Se puede lograr con: git diff branch..branch.



Para saber que ramas están incluidas en la rama que escojamos:

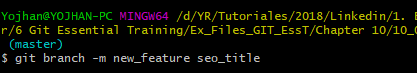
***Git branch –merged.***



Acá muestra las ramas que tiene.

Cambiar nombre de una Branch

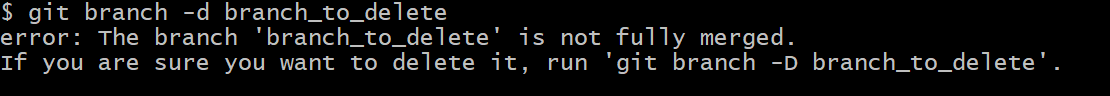
* ***m:*** Mover. También puede ser –move

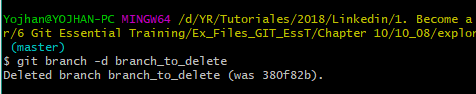


Eliminar una rama

* ***d:*** Eliminar. También puede ser –delete

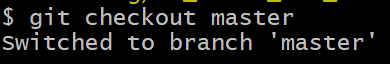
Si hay commits dentro de esta Branch, GIT no dejará eliminarla sin que estos sean *merged* a alguna rama. Como se ve a continuación:



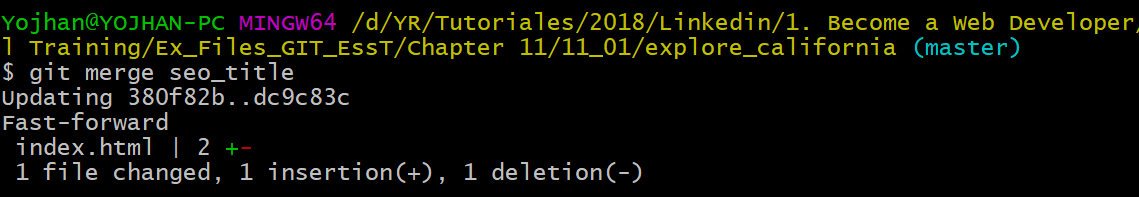


# Capítuo 11. Merging Branches

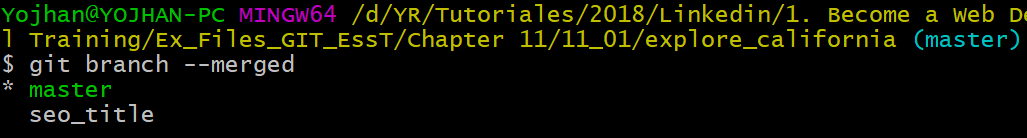
* Primero debemos hacer ***checkout*** en la rama que va recibir el *merge.* En este caso ***master*** va a recibir los cambios desde la rama ***seo\_title***



* Solo basta con hacer ***git merge rama.***



* El comando ***git branch –merged*** muestra las ramas que han sido merged en la rama en la que estemos:

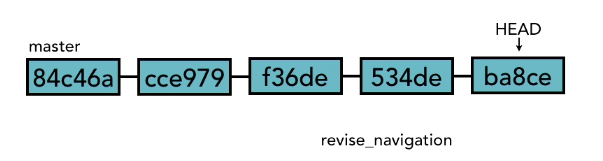


* Ya se puede eliminar, si necesario, la rama anterior.

Dos tipos de *merging:* Fast-Forward y True Merge

* **FAST FORWARD MERGE**

Lo que pasa cuando GIT hace un *merge*, toma el punto que queremos *merge* y comienza al final de este y mira atrás todos los ancestros hasta el inicio. Si encuentra el puntero HEAD de master, hace un *FAST-FORWARDING*. No hay necesidad de hacer un nuevo *commit*



***Git merge—no—ff branch:*** Obliga a GIT a que no haga FF (Fast forward)

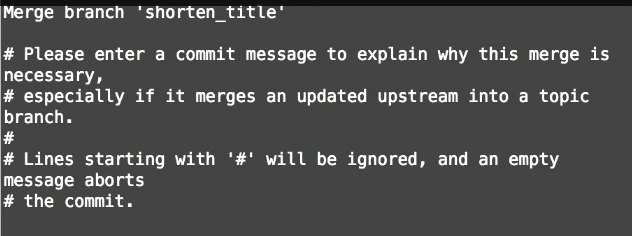
***Git merge—ff:*** Le indica a GIT que haga el *merging* **SOLO si** puede hacerlo con FF. Si no, aborte.

* **TRUE MERGE**

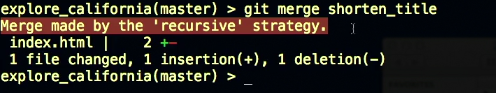
El proceso es el mismo:



**GIT** abre un editor para añadir el mensaje al commit que está creando (commit de *merging*)

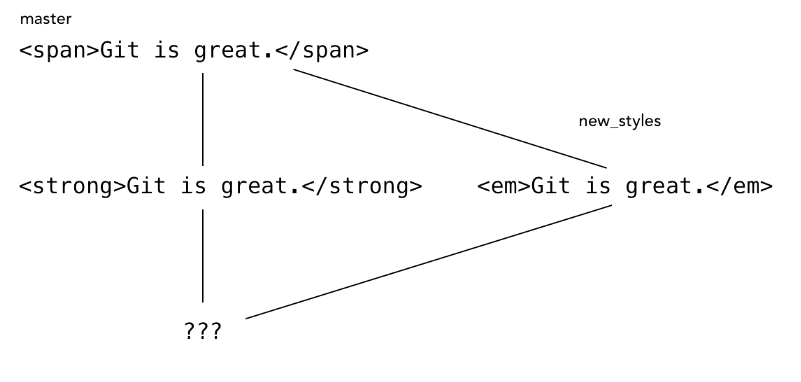


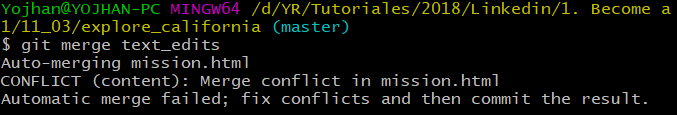
Una vez se acepte muestra que la estrategia utilizada es ‘recursiva’. Es porque no hay conflictos y GIT resolvió como unir las dos ramas.



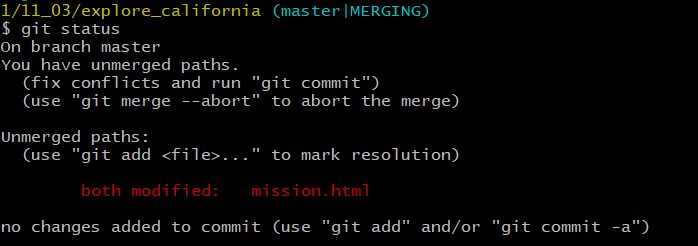
¡Merging Conflictos!

Representación de un conflicto. Es cuando GIT en realidad no sabe que hacer, porque no puede descifrar cual de las dos posibles elecciones deseamos.





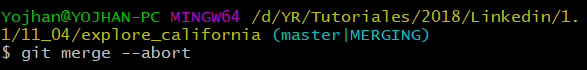
Entra en un estado de conflicto que se debe resolver. Como se observa, está en ***MASTER | MERGING***. Una vez se escojan los cambios deseados se puede hacer el commit.



¡Resolver conflictos de merging!

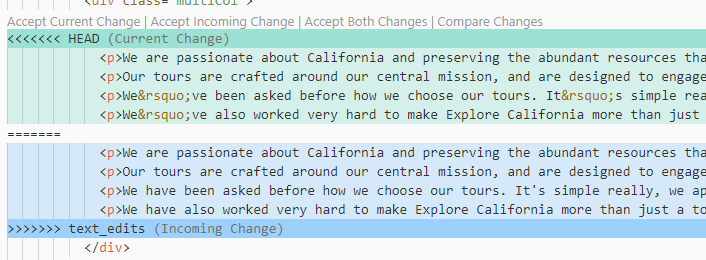
Se tienen tres opciones en que hacer para resolver estos conflictos:

* Abortar el *merging.*



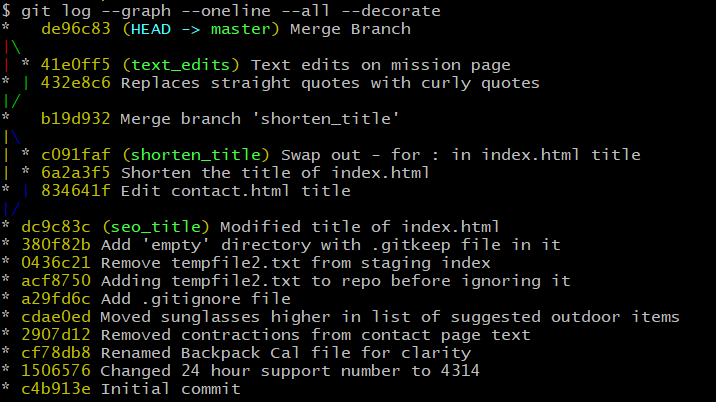
* Podemos resolver los conflictos manualmente. (Es el más común)

En VS Code, nos muestra de una forma fácil como hacerlo:



Para ver el historial de los *merged* y el arbol usamos:

***Git log –graph –oneline –all –decorate*** (Todos con --).

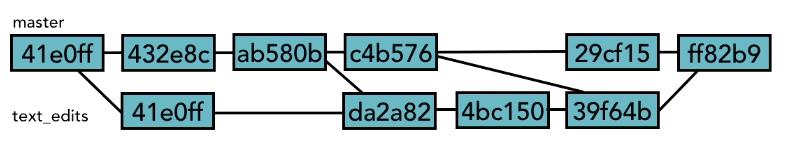


* Usar una herramienta de *merging*.



Estrategias para reducir conflictos

* Mantener las lineas cortas
* Mantener commits *pequeños* y *enfocados*.
* Tener cuidado con los espacios, tabs, y enters.
* Lo mejor es hacer *merging* frecuentemente.
* Rastrear cambios de *master.* No esperar muchos cambios en master, pero más bien estar mirando si hay cambios en *master* para incorporarlos oportunamente en la *rama* si es necesario.



# Capítulo 12. Escondiendo (stashing) cambios

**Stash:** es el lugar donde se puede guardar cambios temporalmente sin tener que hacerles *commit* al repositorio. No hace parte del directorio de trabajo, staging index ni en el repositorio. No tienen un SHA asociado

Es obligatorio hacer *commit o stash* los cambios si se quiere cambiar entre ramas

Yojhan@YOJHAN-PC MINGW64 /d/YR/Tutoriales/2018/Linkedin/1. Become a Web Developer/6 Git Essential Training/Ex\_Files\_GIT\_EssT/Chapter 12/12\_01/explore\_california (shorten\_title)

$ git stash save “Change mission page title”

Saved working directory and index state On shorten\_title: Change mission page title

* Para ver los elementos en el *stash* se usa el siguiente comando:

**$ git stash list**

stash@{0}: On shorten\_title: changed mission page title

Los stash están disponibles aun cuando cambios de rama. Es decir, que si trabajos en una rama, hicimos unos cambios y nos dimos cuenta que no era la rama que queriamos usar, podemos hacer stash y despues cambiar de rama y pasar esos cambios a la otra.

* Para ver los cambios que hubo en un *stash* usamos el siguiente comando, referenciando el *stash* con el código que nos arrojo previamente:

$ git stash show -p stash@{0}

diff—git a/mission.html b/mission.html

index b79f44d..6c493c0 100644

--- a/mission.html

+++ b/mission.html

@@ -3,7 +3,7 @@

<html lang=“en“>

<head>

<meta http-equiv=”Content-Type” content=”text/html; charset=utf-8”>

<title>Explore California: Mission</title>

<title>Explore California: Our Mission</title>

<link href=”assets/stylesheets/main.css” rel=”stylesheet” type=”text/css” media=”all”>

<script src=”assets/javascripts/jquery-1.5.1.min.js”></script>

<script src=”assets/javascripts/jquery-ui-1.8.10.custom.min.js”></script>

Recuperar cambios guardados en el Stash

Sirve con un *merging* ya que se pueden presentar conflictos. Hay dos comandos a tener en cuenta. Ambos toman lo que está en el *stash* y lo pone en el directorio de trabajo; sin embargo, hay diferencias:

* Git stash pop: pone lo del stash en el directorio de trabajo y lo quita del stash
* Git stash apply: pone lo del stash en el directorio de trabajo, pero deja una copia en el stash

Yojhan@YOJHAN-PC MINGW64 /d/YR/Tutoriales/2018/Linkedin/1. Become a Web Developer/6 Git Essential Training/ExChapter 12/12\_03/explore\_california (master)

$ git stash pop stash@{0}

Auto-merging mission.html

On branch master

Changes not staged for commit:

(use “git add <file>...” to update what will be committed)

(use “git checkout -- <file>...” to discard changes in working directory)

modified: mission.html

no changes added to commit (use “git add” and/or “git commit -a”)

Dropped stash@{0} (f5a1464fc7aad3c76c77f2e4fd397ac4a825ca59)

Cuando se recupera el *stash* pasa al directorio de trabajo sin quedar en el *staging index*.

Eliminar cambios en el Stash

Yojhan@YOJHAN-PC MINGW64 /d/YR/Tutoriales/2018/Linkedin/1. Become a Web Developer/6 Git Essential Training/Ex\_Files\_GIT\_EssT/Chapter 12/12\_04/explore\_california (shorten\_title)

**$ git stash drop stash@{0}**

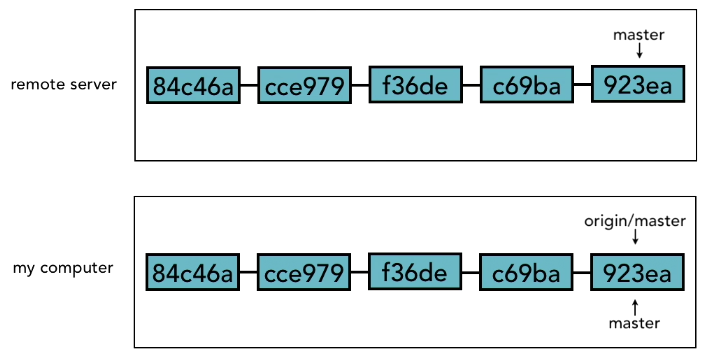
Dropped stash@{0} (bf2b3b6ac42eb43d3f1e571ca19625432bd1b862)

También se puede hacer ***git stash clear*** para borrar todos los elementos en el stash.

# Capítulo 13. Repositorios remotos

**(13-03 CARPETA EN LA QUE SE REALIZO TODO)**

Esquema general del trabajo en un repositorio remoto:



El proceso general es que cuando se trabaja de esta forma, uno hace los *commits* localmente en la rama *master*, después se hace un *fetch* para obtener la última versión del repositorio que se alojará en lo que se marca como *origin/master* (este siempre estará sincronizado con el servidor remoto) y después se debe realizar el *merging* de nuestro trabajo con el que acabamos de obtener haciendo *fetch*. Por último, se hace un *push (up)* al servidor y guardar los cambios.

Añadir un repositorio remoto

***Git remote (-v):*** Arroja los repositorios que conoce. Se puede tener más de un repositorio remoto conectado por proyecto. Con el -v nos da más información:

Yojhan@YOJHAN-PC MINGW64 /d/YR/Tutoriales/2018/Linkedin/1. Become a Web Developer/6 Git Essential Training/Ex\_Files\_GIT\_EssT/Chapter 13/13\_03/explore\_california (master)

$ git remote -v

origin <https://github.com/YojhanLR/Explore_California.git> (fetch)

origin <https://github.com/YojhanLR/Explore_California.git> (push)

***Git remote add nombre(origin) url:*** Añade el repositorio. Por convención, origin es el nombre más adecuado.

Yojhan@YOJHAN-PC MINGW64 /d/YR/Tutoriales/2018/Linkedin/1. Become a Web Developer/6 Git Essential Training/Ex\_Files\_GIT\_EssT/Chapter 13/13\_03/explore\_california (master)

$ git remote add origin <https://github.com/YojhanLR/Explore_California.git>

***Git remote rm nombre(origin): Eliminar el repositorio***

Hacer PUSH

Lo que se quiere hacer es p*ush* al código a la rama remota y cuando se hace un *push* en realidad lo que hace es *push* una rama. Así que estamos en una rama y le decimos a Github: *push* esta rama a la rama correspondiente en el servidor remoto

Así que decimos *push* al repositorio (alias referenciado por origin) esta rama llamada *master.*

Yojhan@YOJHAN-PC MINGW64 /d/YR/Tutoriales/2018/Linkedin/1. Become a Web Developer/6 Git Essential Training/Ex\_Files\_GIT\_EssT/Chapter 13/13\_03/explore\_california (master)

**$ git push -u origin master**

fatal: HttpRequestException encountered.

Error al enviar la solicitud.

Username for ‘https://github.com’: YojhanLR

Counting objects: 141, done.

Delta compression using up to 4 threads.

Compressing objects: 100% (135/135), done.

Writing objects: 100% (141/141), 1.02 MiB | 341.00 KiB/s, done.

Total 141 (delta 47), reused 0 (delta 0)

remote: Resolving deltas: 100% (47/47), done.

remote:

remote: Create a pull request for ‘master’ on GitHub by visiting:

remote: <https://github.com/YojhanLR/Explore_California/pull/new/master>

remote:

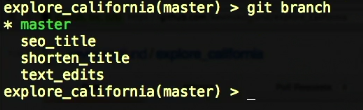
To <https://github.com/YojhanLR/Explore_California.git>

\* [new branch] master -> master

Branch master set up to track remote branch master from origin.

**¡Si no se hace con el -u (la primera vez), no hace que sea una rama rastreada (tracking branch)!**

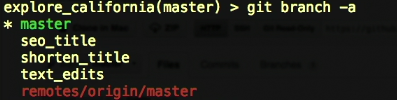
* Ver las ramas:



* Ver las ramas remotas:



* Ver ambas. Ramas locales y remotas:



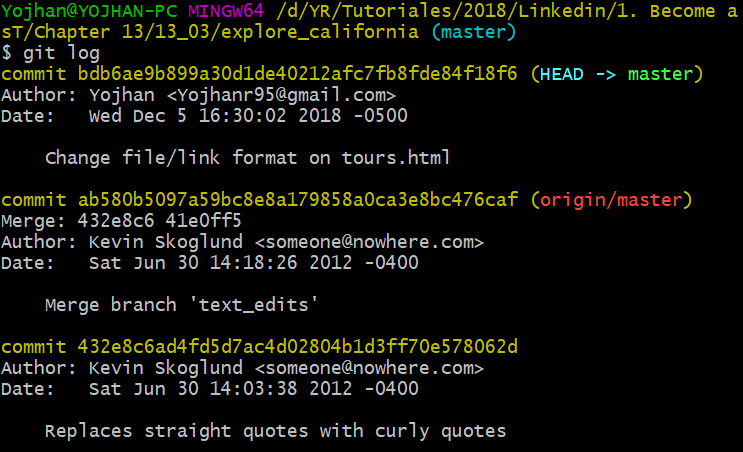
Clonar un repositorio

Yojhan@YOJHAN-PC MINGW64 /d/YR/Tutoriales/2018/Linkedin/1. Become a Web Develop\_Files\_GIT\_EssT/Chapter 13/13\_05

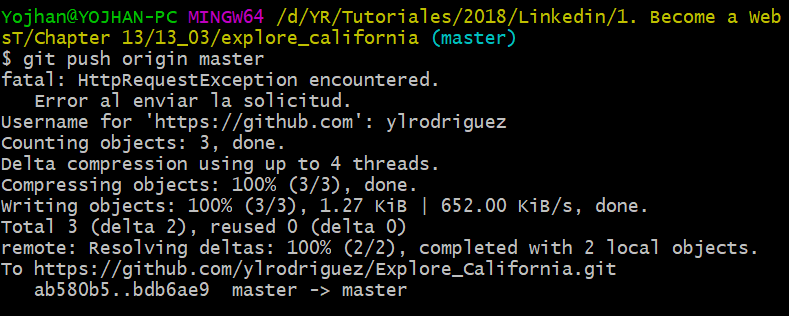
***$ git clone*** [***https://github.com/ylrodriguez/Explore\_California.git***](https://github.com/ylrodriguez/Explore_California.git)***.***

Pushing cambios

Al hacer los cambios y hacer *commit* los cambios están en nuestro repositorio local. Solo existen estos cambios aquí.



Y al hacer push:

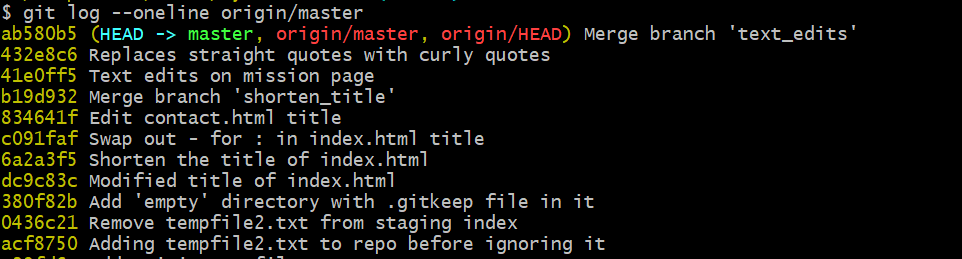


No hay necesidad de señalar el ***-u.*** porque ya la habiamos hecho una rama rastreada (tracking branch). Además, al ser una rama rastreada solo bastaba con ***git push*** porque GIT asume que es queremos ir a esa rama.

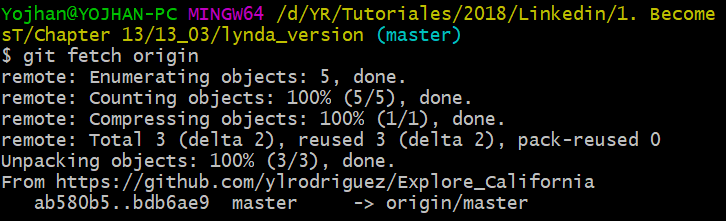
Fetching cambios de un repositorio remoto

Fetch es lo que sincroniza ***origin/master*** con lo que sea que este en el repositorio remoto.

Cuando miramos el *log* del ***origin/master*** en realidad no lo verifica con GitHub, es solo una copia local que debemos actualizar:



Asi que para actualizar el ***origin/master*** solo basta hacer ***fetch*** con:



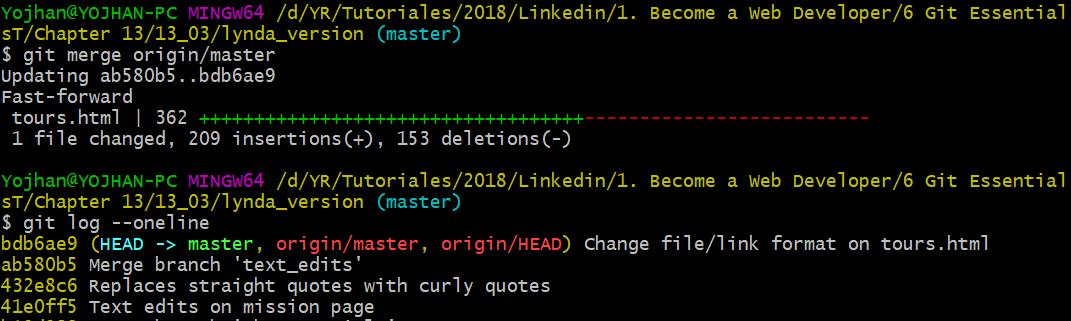
Claro que estos cambios solo están guardados en la rama de ***origin/master*** y no en nuestra copia local ***master*.**

* Siempre hacer FETCH antes de trabajar.
* Hacer FETCH antes de hacer PUSH.
* Hacer FETCH frecuentemente.
* Hacer FETCH y despues MERGE

Hacer MERGE en los cambios FETCHED (origin/master 🡪 master)

Los cambios obtenidos desde un ***fetched*** solo están guardados en la rama de ***origin/master*** y no en nuestra copia local ***master*.**

Para hacer el *merging* es igual que cualquier otra rama, siendo posible que sea Fast Forward o que haya conflictos que deban ser resueltos.



**También existe un comando bastante útil como lo es** que hace todo en un solo paso:

***Git pull = git fetch + git merge.***

* Es conveniente
* Para los principiantes olvida que en realidad son dos pasos

Haciendo CHECKOUT en ramas remotas

Si uno quiere trabajar con una rama remota se debe crear una copia local que la referencia. (No se pueden hacer ***checkout***)

Yojhan@YOJHAN-PC MINGW64 /d/YR/Tutoriales/2018/Linkedin/1. Become a Web Developer/6 Git Essential Training/Ex\_Files\_GIT\_EssT/Chapter 13/13\_03/lynda\_version (master)

***$ git branch non\_tracking origin/non\_tracking***.

Branch non\_tracking set up to track remote branch non\_tracking from origin.

También se puede hacer:

***$ git checkout -b non\_tracking origin/non\_tracking***.

Eliminar una rama remota

Simplemente los dos puntos (:) en esta forma indican eliminar la rama remota, pero mantiene una copia local

Yojhan@YOJHAN-PC MINGW64 /d/YR/Tutoriales/2018/Linkedin/1. Become a Web Developer/6 Git Essential Training/Ex\_Files\_GIT\_EssT/Chapter 13/13\_03/lynda\_version (master)

***$ git push origin: non\_tracking***.

fatal: HttpRequestException encountered.

Error al enviar la solicitud.

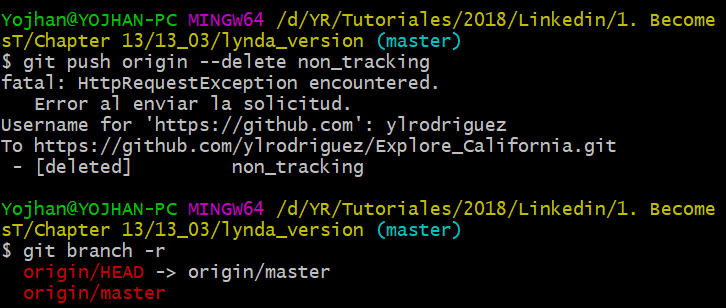
Username for 'https://github.com': ylrodriguez

To https://github.com/ylrodriguez/Explore\_California.git

- [deleted] non\_tracking

Hay una forma más fácil de hacerlo usando:

***$ git push origin --delete non\_tracking***.



Forking y Pull Request

*Forking* es copiar en el repositorio propio un proyecto cualquiera de código abierto para mejorarlo, editar alguna caracteristica, solucionar un problema, etc., y si se quiere, se puede hacer un *pull request* al original que es como alzar la mano y decir tengo algo aquí que quiero mostrar, dejando un mensaje con el problema resuelto o la caracteristica añadido. Si aceptan el *pull request* pueden añadirlo al proyecto principal.

Cambiar de usuario de Github en Git Bash

Abrir *Credenciales de Windows* y eliminar la credencial de github. Al hacer un nuevo push, se pedirá digitar la credencial de la nueva cuenta.

# Capítulo 14. Herramientas

Hacer aliases para comandos comunes

Sirven para ahorrar tiempo usando keywords definidas por nosotros:

***Git config --global alias.st “status”.***

En este caso, la palabra *st* representará *status.* Algunos de los comandos más usados que son un “estándar” son:

* Git config --global alias.co checkout
* Git config --global alias.ci commit
* Git config --global alias.br branch
* Git config --global.logg “log --graph --decorate --oneline --abbrev-commit -all”

Interfaces de usuarios graficos para GIT

* GitWeb
* Github – GitX

Git Hosting companies:

* GitHub
* Bitbucket
* Gitorious

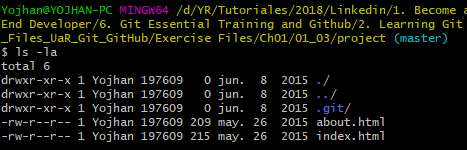
Git Self Hosting

* Gitosis = Software para instalar en un servidor
* Gitolite
* Git-core

# …………………………………Learning Git and Github

# Capítulo 1. Empezando

El comando ***ls -la*** ayuda a ver todos los archivos incluso ocultos:



***Git reset HEAD <archivo>*** permite hacer un *staging* a un archivo.



***Git rm <archivo>:*** Elimina un archivo y lo pone de una vez en el *staging.*

***Git branch -m app01 app1:***  Cambiar el nombre de una rama

# Capítulo 2. Trabajando con proyectos de GitHub

***README.md*** Esta en Markdown, en la web se puede encontrar recursos para hacer una edición correcta.

***.gitignore:*** Archivos que no se suben al repositorio y no se rastrean.

gu