git log

De forma *predeterminada* , este comando muestra:

* el SHA
* el autor
* la fecha
* y el mensaje
* **el SHA** : git logmostrará el SHA completo para cada confirmación. Cada SHA es único, por lo que realmente no necesitamos ver el SHA *completo* . Podríamos arreglárnoslas perfectamente con conocer solo los primeros 6-8 caracteres. ¿No sería genial si pudiéramos ahorrar algo de espacio y mostrar solo los primeros 5 o más personajes del SHA?
* **el autor** : la git logsalida muestra el autor de la confirmación para *cada confirmación* . Podría ser diferente para otros repositorios que tienen varias personas colaborando juntas, pero para este, solo hay una persona que realiza todas las confirmaciones, por lo que el autor de la confirmación será idéntico para todas. ¿Necesitamos ver al autor de cada uno? ¿Y si quisiéramos ocultar esa información?
* **la fecha** : de forma predeterminada, git logmostrará la fecha de cada confirmación. ¿Pero realmente nos importa la fecha del compromiso? Saber la fecha puede ser importante ocasionalmente, pero normalmente conocer la fecha no es de vital importancia y puede ignorarse en muchos casos. ¿Hay alguna forma de ocultar eso para ahorrar espacio?
* **el mensaje de confirmación** : esta es una de las partes más importantes de un mensaje de confirmación ... normalmente siempre queremos ver esto

$ git log --oneline

Este comando:

* enumera una confirmación por línea
* muestra los primeros 7 caracteres del SHA de la confirmación
* muestra el mensaje de la confirmación

Eso es todo! Recuerde, la qllave sale de la git logvista. Todavía estamos usando, git logpero solo estamos pasando una bandera para cambiar cómo se muestra la información. Entonces, la **q**clave aún funciona y devuelve el terminal al símbolo del sistema.

**git log --stat Intro**

El git logcomando tiene una bandera que se puede usar para mostrar los archivos que se han cambiado en la confirmación, así como el número de líneas que se han agregado o eliminado. La bandera es --stat("estadística" es la abreviatura de "estadísticas"):

En resumen, la --statbandera se usa para alterar cómo git logmuestra la información:

$ git log --stat

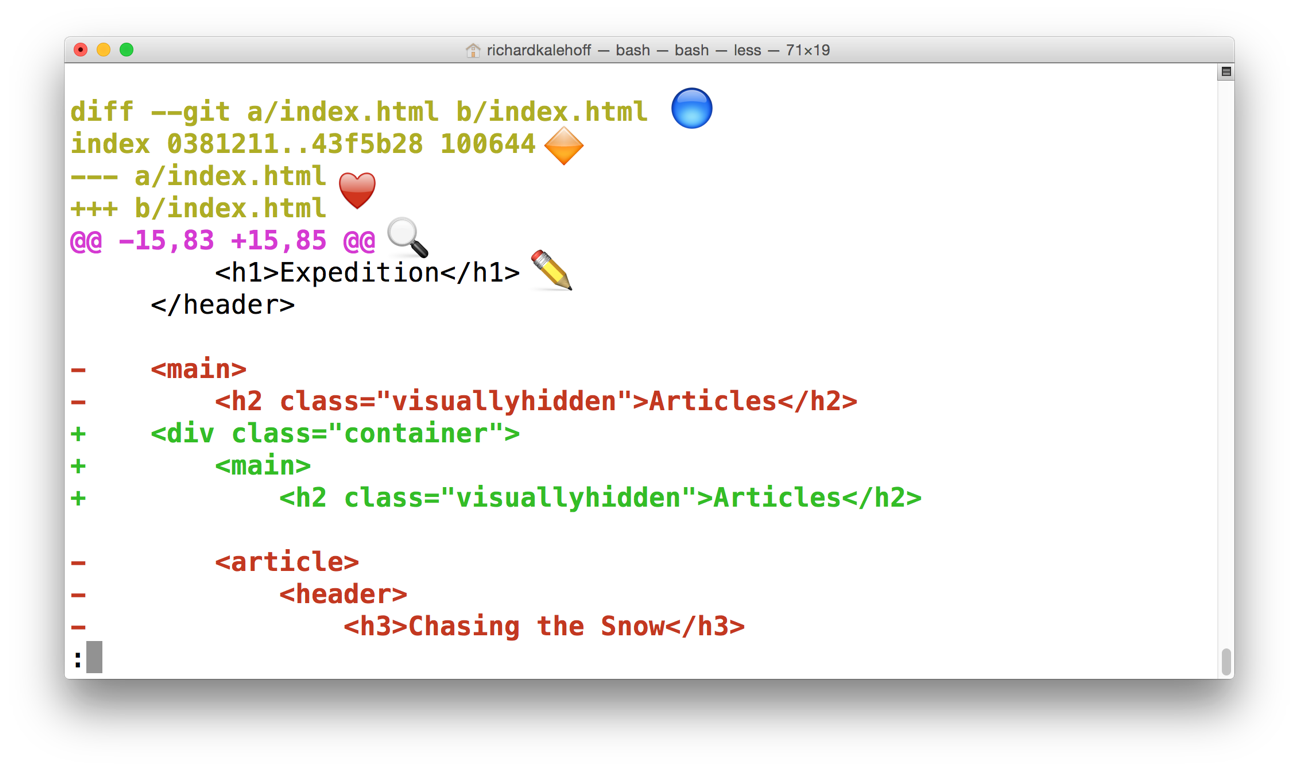
Este comando:

* muestra los archivos que se han modificado
* muestra el número de líneas que se han agregado / eliminado
* muestra una línea de resumen con el número total de archivos y líneas modificados que se han agregado / eliminado

## git log -p

El git logcomando tiene una bandera que se puede utilizar para mostrar los cambios reales realizados en un archivo. La bandera es la --patchque se puede acortar a solo -p:

$ git log -p



Usando la imagen de arriba, hagamos un resumen rápido de la git log -psalida:

* 🔵 - el archivo que se muestra
* 🔶 - el hash de la primera versión del archivo y el hash de la segunda versión del archivo
  + no suele ser importante, por lo que es seguro ignorar
* ❤️: la versión anterior y la versión actual del archivo
* 🔍 - las líneas donde se agrega el archivo y cuántas líneas hay
  + -15,83indica que la versión anterior (representada por -) comenzaba en la línea 15 y que el archivo tenía 83 líneas
  + +15,85indica que la versión actual (representada por +) comienza en la línea 15 y que ahora hay 85 líneas ... estas 85 líneas se muestran en el parche a continuación
* ✏️ - los cambios reales realizados en la confirmación
  + Las líneas que son rojas y comienzan con un signo menos ( -) estaban en la versión original del archivo pero han sido eliminadas por la confirmación
  + Las líneas que son verdes y comienzan con un signo más ( +) son líneas nuevas que se han agregado en la confirmación

Así es; ¡puedes combinar banderas! git log -p --statmostrará la información de las estadísticas sobre la información del parche. En realidad, el orden no importa; git log --stat -ptambién mostrará la información de las estadísticas sobre la información del parche.

¡Así es! git log -p -wmostrará la información del parche, pero no resaltará las líneas donde *solo* se han producido cambios en los espacios en blanco.

## git log -p Resumen

En resumen, la -pbandera (que es la misma que la --patchbandera) se usa para alterar cómo git logmuestra la información:

$ git log -p

Este comando agrega lo siguiente a la salida predeterminada:

* muestra los archivos que se han modificado
* muestra la ubicación de las líneas que se han agregado / eliminado
* muestra los cambios reales que se han realizado

Ya sabe cómo "registrar" información con:

* git log
* git log --oneline
* git log --stat
* git log -p

Pero, ¿sabía que puede proporcionar el SHA de una confirmación como argumento final para todos estos comandos? Por ejemplo:

$ git log -p fdf5493

Al proporcionar un SHA, el git log -pcomando comenzará en ese compromiso . ¡No es necesario desplazarse por todo! Tenga en cuenta que también mostrará todas las confirmaciones que se realizaron antes del SHA proporcionado.

### ¿Qué hace git show?

El git showcomando mostrará solo una confirmación . Así que no se alarme cuando no pueda encontrar otras confirmaciones, solo muestra una. La salida del git showcomando es exactamente la misma que la del git log -pcomando. Entonces, por defecto, git showmuestra:

* el compromiso
* el autor
* la fecha
* el mensaje de confirmación
* la información del parche

Sin embargo, git showse puede combinar con la mayoría de las otras banderas que hemos visto:

* --stat - para mostrar cuántos archivos se cambiaron y el número de líneas que se agregaron / eliminaron
* -po --patch- este es el predeterminado, pero si --statse usa, el parche no se mostrará, así que pase -ppara agregarlo nuevamente
* -w - para ignorar los cambios en los espacios en blanco
* Ejecutarlo como en el ejemplo anterior solo mostrará la confirmación más reciente. Normalmente, se proporciona un SHA como argumento final:
* $ git show fdf5493

 git log --onelineencontrar el SHA de la confirmación, luego lo usé git log --statcon el SHA para encontrar la información correcta.

Así que modificamos nuestro archivo. Git ve que se ha modificado. Así que lo estamos haciendo bien hasta ahora. Ahora recuerde, para realizar una confirmación, el archivo o archivos que queremos confirmar deben estar en el índice de ensayo. ¿Qué comando usamos para mover archivos del directorio de trabajo al índice de ensayo? ¡Lo tienes - git add!

Aunque solíamos git addagregar *archivos recién creados* al índice de etapa, usamos el mismo comando para mover *archivos modificados* al índice de etapa.

Use el git addcomando para mover el archivo al índice de etapas, ahora. Verifica que esté ahí con git status.

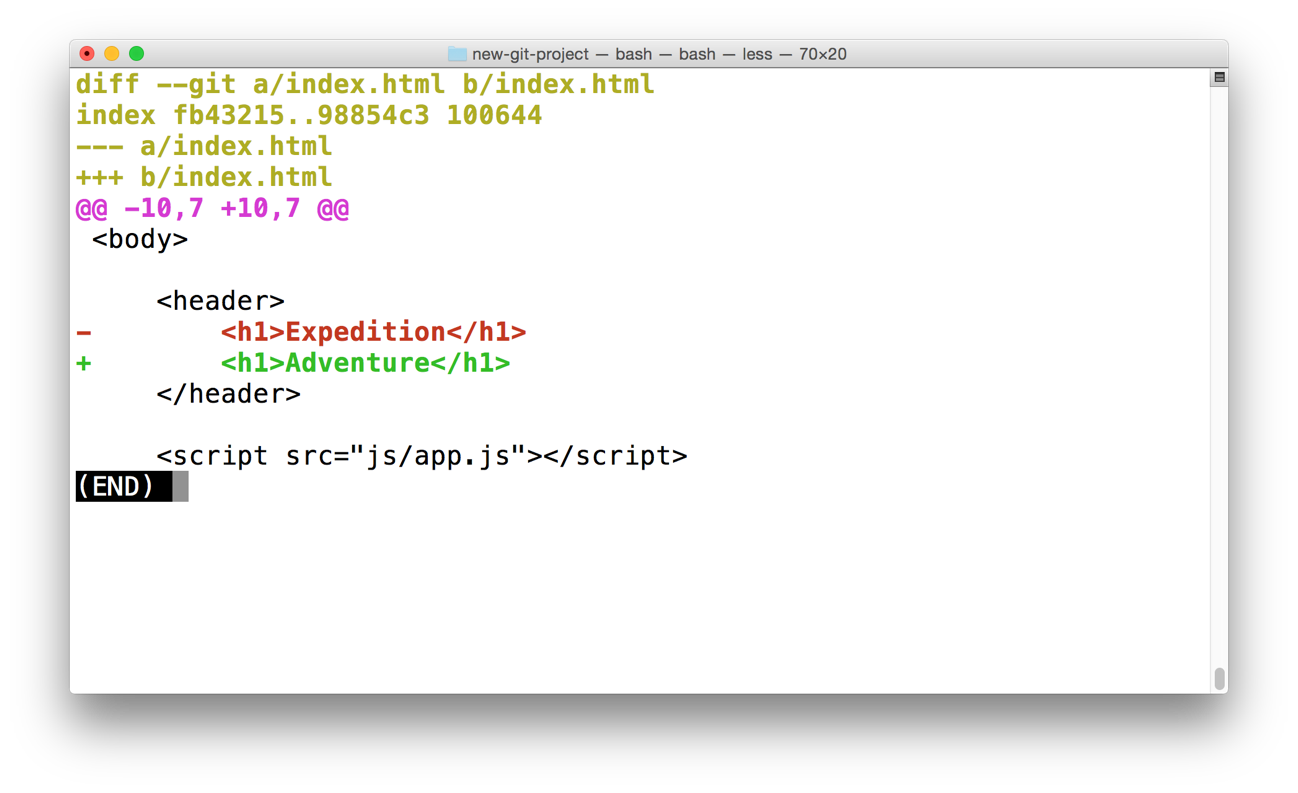
¡El git diffcomando se usa para encontrar esta información!

## git diff

El git diffcomando se puede usar para ver los cambios que se han realizado pero que aún no se han confirmado.

$ git diff

¡Para verlo git diffen acción, necesitamos algunos cambios no comprometidos! En index.html, reformulemos el título. Cambie el título de "Expedición" a "Aventura". Guarde el archivo y ejecútelo git diffen la Terminal.



En resumen, el git diffcomando se usa para ver los cambios que se han realizado pero que aún no se han confirmado:

$ git diff

Este comando muestra:

* los archivos que se han modificado
* la ubicación de las líneas que se han agregado / eliminado
* los cambios reales que se han realizado

## Curso intensivo globbing

Digamos que agrega 50 imágenes a su proyecto, pero quiere que Git las ignore todas. ¿Significa esto que debe enumerar todos y cada uno de los nombres de archivo en el .gitignorearchivo? ¡Oh Dios, no, eso sería una locura! En su lugar, puede utilizar un concepto llamado [globbing](https://en.wikipedia.org/wiki/Glob_(programming" \t "_blank) .

Globbing le permite usar caracteres especiales para hacer coincidir patrones / caracteres. En el .gitignorearchivo, puede utilizar lo siguiente:

* Se pueden usar líneas en blanco para espaciar
* # - marca la línea como comentario
* \* - coincide con 0 o más caracteres
* ? - coincide con 1 carácter
* [abc] - coincide con a, b, \_o\_ c
* \*\*- coincide con directorios anidados - a/\*\*/zcoincide
  + Arizona
  + a / b / z
  + a / b / c / z

*CUIDADO: En el comando anterior ( git tag -a v1.0) -ase usa la bandera. Esta bandera le dice a Git que cree una bandera*anotada*. Si no proporciona la bandera (es decir git tag v1.0), se creará lo que se llama una etiqueta*ligera*.*

*Se recomiendan las etiquetas anotadas porque incluyen mucha información adicional, como:*

* *la persona que hizo la etiqueta*
* *la fecha en que se hizo la etiqueta*
* *un mensaje para la etiqueta*

*Debido a esto, siempre debe usar etiquetas anotadas.*

## Bandera de Git Log's --decorate

Como ha aprendido, git loges una herramienta bastante poderosa para permitirnos verificar las confirmaciones de un repositorio. Ya hemos visto un par de sus banderas, pero es hora de agregar una nueva a nuestro cinturón de herramientas. La --decoratebandera nos mostrará algunos detalles que están ocultos en la vista predeterminada.

## *CABEZA -> maestro?*

*¿Notó que, además de la información de la etiqueta que se muestra en el registro, --decoratetambién se revela HEAD -> master? ¡Esa es información sobre una*sucursal*! A continuación, veremos las sucursales en Git.*

## Eliminar una etiqueta

¿Qué pasa si accidentalmente escribiste mal algo en el mensaje de la etiqueta, o escribiste mal el nombre real de la etiqueta (en v0.1lugar de v1.0)? ¿Cómo puedes arreglar esto? La forma más sencilla es eliminar la etiqueta y crear una nueva.

Una etiqueta de Git se puede eliminar con la -dbandera (¡para eliminar !) Y el nombre de la etiqueta:

$ git tag -d v1.0

## El git branchcomando

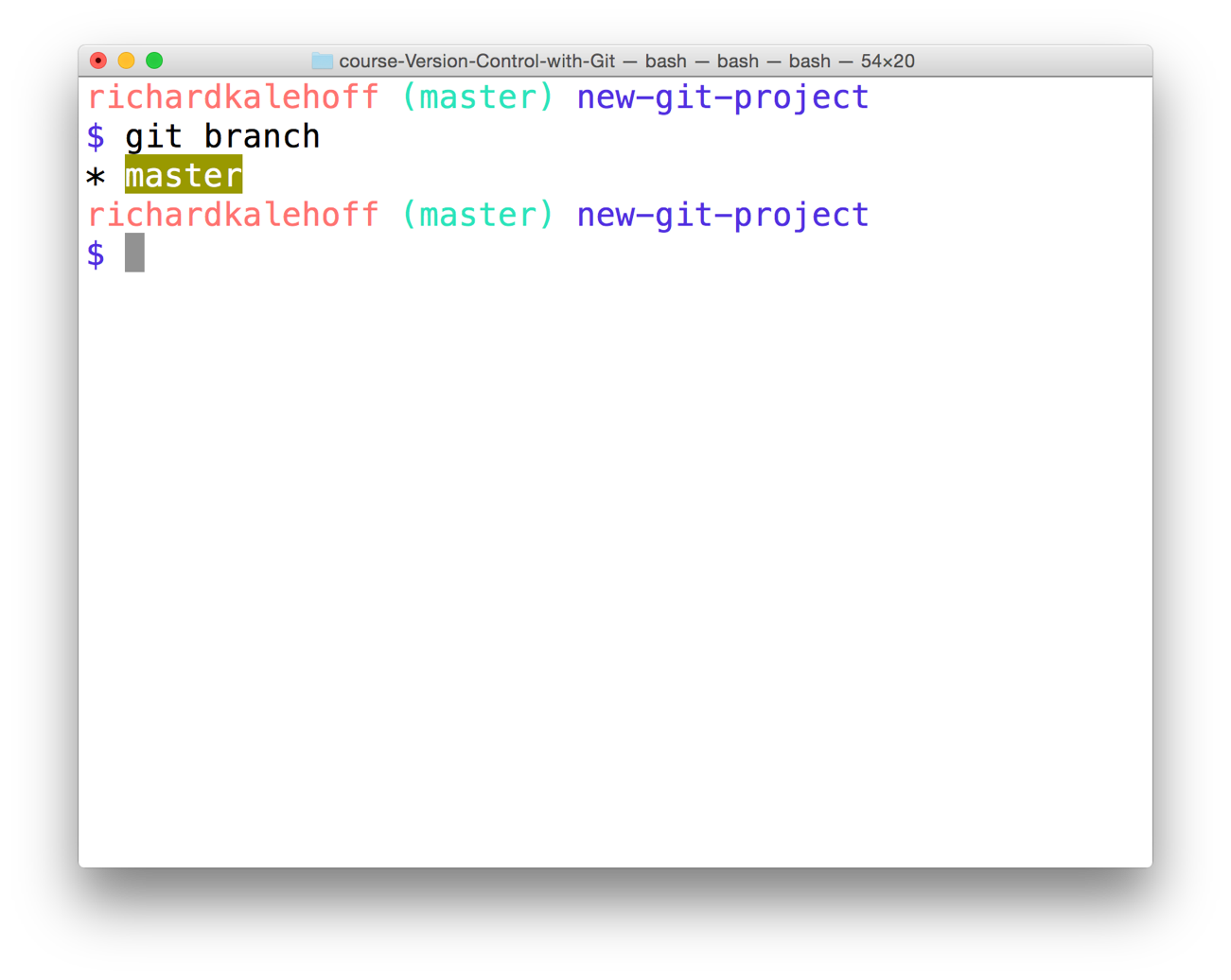
El git branchcomando se usa para interactuar con las ramas de Git:

$ git branch

Puede utilizarse para:

* enumerar todos los nombres de sucursales en el repositorio
* crear nuevas ramas
* eliminar ramas

Si escribimos solo git branch, enumerará las ramas en un repositorio:



## Crear una sucursal

Para crear una rama, todo lo que tiene que hacer es usar git branchy proporcionarle el nombre de la rama que desea que cree. Entonces, si desea una rama llamada "barra lateral", ejecute este comando:

$ git branch sidebar

## El git checkoutcomando

Recuerde que cuando se realiza una confirmación, se agregará a la rama actual. Entonces, aunque creamos el nuevo sidebar, no se le agregarán nuevas confirmaciones ya que aún no lo hemos cambiado . Si hiciéramos una confirmación en este momento, esa confirmación se agregaría a la masterrama, no a la sidebarrama. Ya hemos visto esto en la demostración, pero para cambiar entre ramas, necesitamos usar el checkoutcomando de Git .

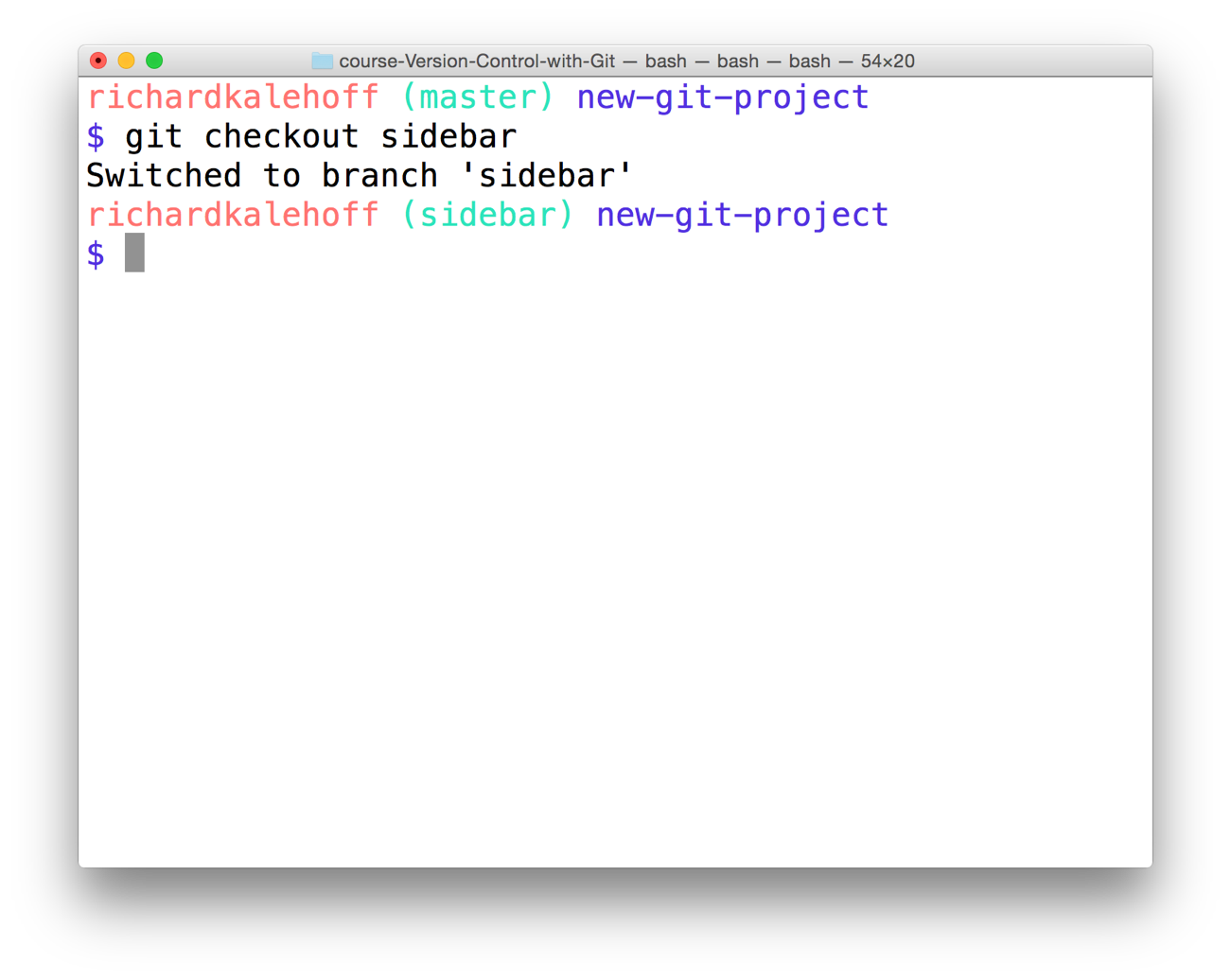
$ git checkout sidebar

Es importante comprender cómo funciona este comando. Al ejecutar este comando:

* eliminar todos los archivos y directorios del directorio de trabajo que Git está rastreando
  + (los archivos que Git rastrea se almacenan en el repositorio, por lo que no se pierde nada)
* vaya al repositorio y extraiga todos los archivos y directorios de la confirmación a los que apunta la rama

Así que esto eliminará todos los archivos a los que hacen referencia las confirmaciones en la rama maestra. Los reemplazará con los archivos a los que hacen referencia las confirmaciones en la rama de la barra lateral. Esto es muy importante de entender, así que regrese y lea estas dos últimas oraciones.

Sin embargo, lo curioso es que ambos sidebary masterapuntan a la misma confirmación , por lo que parecerá que nada cambia cuando cambie entre ellos. Pero el símbolo del sistema mostrará "barra lateral", ahora:



## Ramas en el registro

La información de la rama en el símbolo del sistema es útil, pero la forma más clara de verla es mirando la salida de git log. Pero al igual que tuvimos que usar la --decoratebandera para mostrar etiquetas de Git, la necesitamos para mostrar ramas.

$ git log --oneline --decorate

## Eliminar una rama

Una rama se utiliza para desarrollar o corregir el proyecto que no afectará al proyecto (ya que los cambios se realizan en una rama). Una vez que realice el cambio en la rama, puede combinar esa rama en la masterrama (esta "combinación de ramas" se llama "fusión" y la veremos en breve).

Ahora, después de fusionar los cambios de una rama, probablemente ya no la necesitará. Si desea eliminar la rama, usaría la -dbandera. El siguiente comando incluye la -dbandera que le dice a Git que elimine la rama provista (en este caso, la rama "barra lateral").

$ git branch -d sidebar

## Resumen de la rama de Git

En resumen, el git branchcomando se usa para administrar ramas en Git:

*# to list all branches*

$ git branch

*# to create a new "footer-fix" branch*

$ git branch footer-fix

*# to delete the "footer-fix" branch*

$ git branch -d footer-fix

Este comando se usa para:

* enumerar las sucursales locales
* crear nuevas ramas
* quitar ramas

## El comando Fusionar

El git mergecomando se usa para combinar ramas de Git:

$ git merge <name-of-branch-to-merge-in>

Cuando ocurre una fusión, Git:

* mira las ramas que se van a fusionar
* mira hacia atrás a lo largo del historial de la rama para encontrar una única confirmación que ambas ramas tengan en su historial de confirmaciones
* combinar las líneas de código que se cambiaron en las ramas separadas juntas
* se compromete a registrar la fusión
* **git init**creará un nuevo repositorio local GIT. El siguiente comando de Git creará un repositorio en el directorio actual:

git init

* Como alternativa, puedes crear un repositorio dentro de un nuevo directorio especificando el nombre del proyecto:

git init [nombre del proyecto]

* **git clone** se usa para copiar un repositorio. Si el repositorio está en un servidor remoto, usa:

git clone nombredeusuario@host:/path/to/repository

* A la inversa, ejecuta el siguiente comando básico para copiar un repositorio local:

git clone /path/to/repository

* **git add**se usa para agregar archivos al área de preparación. Por ejemplo, el siguiente comando de Git básico indexará el archivo temp.txt:

git add <temp.txt>

* **git commit**creará una instantánea de los cambios y la guardará en el directorio git.

git commit –m “El mensaje que acompaña al commit va aquí”

* Ten en cuenta que los cambios confirmados no llegarán al repositorio remoto.
* **git config** puede ser usado para establecer una configuración específica de usuario, como el email, nombre de usuario y tipo de formato, etc. Por ejemplo, el siguiente comando se usa para establecer un email:

git config --global user.email tuemail@ejemplo.com

* La opción -global le dice a GIT que vas a usar ese correo electrónico para todos los repositorios locales. Si quieres utilizar diferentes correos electrónicos para diferentes repositorios, usa el siguiente comando:

git config --local user.email tuemail@ejemplo.com

* **git status**muestra la lista de los archivos que se han cambiado junto con los archivos que están por ser preparados o confirmados.

git status

* **git push**se usa para enviar confirmaciones locales a la rama maestra del repositorio remoto. Aquí está la estructura básica del código:

git push  origin <master>

* Reemplaza <master> con la rama en la que quieres enviar los cambios cuando no quieras enviarlos a la rama maestra.
* **git checkout**crea ramas y te ayuda a navegar entre ellas. Por ejemplo, el siguiente comando crea una nueva y automáticamente se cambia a ella:

command git checkout -b <branch-name>

* Para cambiar de una rama a otra, sólo usa:

git checkout <branch-name>

* **git remote**te permite ver todos los repositorios remotos. El siguiente comando listará todas las conexiones junto con sus URLs:

git remote -v

* Para conectar el repositorio local a un servidor remoto, usa este comando:

git remote add origin <host-or-remoteURL>

* Por otro lado, el siguiente comando borrará una conexión a un repositorio remoto especificado:

git remote <nombre-del-repositorio>

* **git branch** se usa para listar, crear o borrar ramas. Por ejemplo, si quieres listar todas las ramas presentes en el repositorio, el comando debería verse así:

git branch

* Si quieres borrar una rama, usa:

git branch -d <branch-name>

* **git pull**fusiona todos los cambios que se han hecho en el repositorio local con el directorio de trabajo local.

git pull

* **git merge** se usa para fusionar una rama con otra rama activa:

git merge <branch-name>

* **git diff**se usa para hacer una lista de conflictos. Para poder ver conflictos con respecto al archivo base, usa:

git diff --base <file-name>

* El siguiente comando se usa para ver los conflictos que hay entre ramas antes de fusionarlas:

git diff <source-branch> <target-branch>

* Para ver una lista de todos los conflictos presentes usa:

git diff

* **git tag**marca commits específicos. Los desarrolladores lo usan para marcar puntos de lanzamiento como v1.0 y v2.0.

git tag 1.1.0 <instert-commitID-here>

* **git log**se usa para ver el historial del repositorio listando ciertos detalles de la confirmación. Al ejecutar el comando se obtiene una salida como ésta:

commit 15f4b6c44b3c8344caasdac9e4be13246e21sadw

Author: Alex Hunter <alexh@gmail.com>

Date:   Mon Oct 1 12:56:29 2016 -0600

* **git reset**sirve para resetear el index y el directorio de trabajo al último estado de confirmación.

git reset - -hard HEAD

* **git rm**se puede usar para remover archivos del index y del directorio de trabajo.

git rm filename.txt

* **git stash** guardará momentáneamente los cambios que no están listos para ser confirmados. De esta manera, pudes volver al proyecto más tarde.

git stash

* **git show**se usa para mostrar información sobre cualquier objeto git.

git show

* **git fetch**le permite al usuario buscar todos los objetos de un repositorio remoto que actualmente no se encuentran en el directorio de trabajo local.

git fetch origin

* **git ls-tree**te permite ver un objeto de árbol junto con el nombre y modo de cada ítem, y el valor blob de SHA-1. Si quieres ver el HEAD, usa:

git ls-tree HEAD

* **git cat-file**se usa para ver la información de tipo y tamaño de un objeto del repositorio. Usa la opción -p junto con el valor SHA-1 del objeto para ver la información de un objeto específico, por ejemplo:

git cat-file –p d670460b4b4aece5915caf5c68d12f560a9fe3e4

* **git grep** le permite al usuario buscar frases y palabras específicas en los árboles de confirmación, el directorio de trabajo y en el área de preparación. Para buscar por www.hostinger.com en todos los archivos, usa:

git grep “www.hostinger.com”

* **gitk** muestra la interfaz gráfica para un repositorio local. Simplemente ejecuta:

gitk

* **git instaweb**te permite explorar tu repositorio local en la interfaz GitWeb. Por ejemplo:

git instaweb –http=webrick

* **git gc** limpiará archivos innecesarios y optimizará el repositorio local.

git gc

* **git archive**le permite al usuario crear archivos zip o tar que contengan los constituyentes de un solo árbol de repositorio. Por ejemplo:

git archive - -format=tar master

* **git prune**elimina los objetos que no tengan ningún apuntador entrante.

git prune

* **git fsck** realiza una comprobación de integridad del sistema de archivos git e identifica cualquier objeto corrupto

git fsck

* **git rebase** se usa para aplicar ciertos cambios de una rama en otra. Por ejemplo:

git rebase master