Más acciones básicas con git

git diff: Viendo qué cambió

1) Seguimos en nuestro repositorio creado en la clase anterior, con nuestro historial de dos commits:

```
usuario@PC ~/probandoGIT $ git log --oneline
d785346 (HEAD -> master) Segundo commit
fb84183 Nuestro primer commit
```

2) Comprobaremos que no ha cambiado nada desde el último commit:

```
usuario@PC ~/probandoGIT $ git status
En la rama master
nada para hacer commit, el árbol de trabajo esta limpio
```

3) Ahora, con nuestro editor, abriremos el archivo saludo.txt, y comprobaremos que tiene el siguiente contenido:
HOLA
CHAU

Realizaremos un cambio, para que quede del siguiente modo:
HOLA
HASTA LUEGO
NOS VEMOS

Guardamos y salimos del editor.

4) Volvemos a comprobar el estado de nuestro repositorio:

```
usuario@PC ~/probandoGIT $ git status
En la rama master
Cambios no rastreados para el commit:
  (usa "git add <archivo>..." para actualizar lo que será confirmado)
  (usa "git checkout -- <archivo>..." para descartar los cambios en el directorio de trabajo)
    modificado:     saludo.txt
sin cambios agregados al commit (usa "git add" y/o "git commit -a")
```

Vemos que, como esperábamos, ha cambiado el archivo `saludo.txt`, por lo que se encuentra en estado "modificado" (modified). Pero, ¿qué es lo que cambió en ese archivo?

5) Para ello, podemos utilizar el comando git diff:

```
usuario@PC ~/probandoGIT $ git diff
diff --git a/saludo.txt b/saludo.txt
index c97e08c..43d7e2e 100644
--- a/saludo.txt
+++ b/saludo.txt
@@ -1,2 +1,3 @@
 HOLA
-CHAU
+HASTA LUEGO
+NOS VEMOS
```

Vemos que se está mostrando la diferencia entre:

- la versión de `saludo.txt` del área de staging (`a/saludo.txt`) y
- la versión de `saludo.txt` de nuesro directorio de trabajo (`b/saludo.txt`)

Explicaremos en detalle las últimas 4 líneas

`HOLA`	Esta línea no ha cambiado
`-CHAU`	Esta línea se ha eliminado, ya no forma parte del archivo.
`+HASTA LUEGO`	Esta línea se ha agregado.
`+NOS VEMOS`	Esta línea también se ha agregado.

6) Agreguemos ahora la nueva versión del archivo `saludo.txt` a la staging area:

```
usuario@PC ~/probandoGIT $ git add saludo.txt
```

7) Comprobemos el nuevo estado del repositorio:

```
usuario@PC ~/probandoGIT $ git status
En la rama master
Cambios a ser confirmados:
  (usa "git reset HEAD <archivo>..." para sacar del área de stage)
    modificado:     saludo.txt
```

8) Veamos ahora que el comando `git diff`, proporciona una salida vacía, puesto que el directorio de trabajo y la staging area son idénticos:

```
usuario@PC ~/probandoGIT $ git diff
```

Pero podemos necesitar comparar:

- el estado del repositorio en el último commit con
- el estado de la staging area

9) Para ello, utilizaremos el modificador --staged:

```
usuario@PC ~/probandoGIT $ git diff --staged
diff --git a/saludo.txt b/saludo.txt
index c97e08c..43d7e2e 100644
--- a/saludo.txt
+++ b/saludo.txt
@@ -1,2 +1,3 @@
 HOLA
-CHAU
+HASTA LUEGO
+NOS VEMOS
```

Lo que estamos viendo ahora, es la diferencia entre el último commit y la staging area.

10) Hagamos commit de nuestros últimos cambios:

```
usuario@PC ~/probandoGIT $ git commit -m "Cambio en la despedida del saludo"
[master 2724ff8] Cambio en la despedida del saludo
 1 file changed, 2 insertions(+), 1 deletion(-)
```

11) Si volvemos a probar `git diff --staged`, veremos que presenta una salida vacía, puesto que ya no hay diferencias respecto al último commit:

```
usuario@PC ~/probandoGIT $ git diff --staged
```

Evitando la staging area

En ciertas ocasiones puede ser útil evitar la staging area, haciendo `git add` y `git commit` en un solo comando. Vamos a probarlo.

1) Cambiemos con nuestro editor el archivo saludo.txt, de tal manera que ahora tenga el siguiente contenido.

```
HOLA
QUE TAL
HASTA LUEGO
NOS VEMOS
```

Guardamos y salimos.

2) Verifiquemos el estado de nuestro repositorio

```
usuario@PC ~/probandoGIT $ git status
En la rama master
Cambios no rastreados para el commit:
  (usa "git add <archivo>..." para actualizar lo que será confirmado)
   (usa "git checkout -- <archivo>..." para descartar los cambios en el directorio de trabajo)
    modificado:     saludo.txt
sin cambios agregados al commit (usa "git add" y/o "git commit -a")
```

No vamos a hacer `git add saludo.txt` y luego `git commit -m "…"`, sino que haremos ambos pasos en uno.

3) Ejecutaremos el siguiente comando:

```
usuario@PC ~/probandoGIT $ git commit -a -m "Agregado 'que tal'"
[master 2d82d33] Agregado 'que tal'
 1 file changed, 1 insertion(+)
```

Vemos que se ha hecho el commit, aún cuando el archivo `saludo.txt` no estaba en la staging area. En realidad, el modificador **`-a`** hace lo siguiente:

- primero se hace `git add` sobre todos los archivos que estén en estado "`modificado`" (modified)
- luego, se hace el commit

4) Verifiquemos que realmente se realizó el commit al repositorio:

```
usuario@PC ~/probandoGIT $ git log --oneline
2d82d33 (HEAD -> master) Agregado 'que tal'
2724ff8 Cambio en la despedida del saludo
d785346 Segundo commit
fb84183 Nuestro primer commit
```

Vemos que ahora HEAD apunta al último commit (el que realizamos en el paso 3), sin necesidad de haber ejecutado `git add`.

Tengamos en cuenta que el modificador `-a` **no** funciona con archivos nuevos, sino solamente con archivos que ya formaban parte del repositorio y han sido modificados. Esto es porque los archivos nuevos se encuentran con el estado "sin seguimiento" (untracked) y no en estado "modificado" (modified). Para agregar un archivo nuevo a un commit, hay que ejecutar ambos comandos (`git add` y `git commit`) por separado.

Eliminando archivos del repositorio

1) Creemos un nuevo archivo con nuestro editor, escribamos cualquier cosa en él, y guardémoslo en nuestro directorio de trabajo con el nombre de archivoNuevo.txt

2) Comprobemos que nuestro archivo nuevo se encuentra con el estado "sin seguimiento" (untracked).

```
usuario@PC ~/probandoGIT $ git status
En la rama master
Archivos sin seguimiento:
  (usa "git add <archivo>..." para incluirlo a lo que se será confirmado)
    archivoNuevo.txt
no hay nada agregado al commit pero hay archivos sin seguimiento presentes (usa "git add"
para hacerles seguimiento)
```

3) Agreguemos ahora el archivoNuevo.txt a la staging area de nuestro repositorio:

```
usuario@PC ~/probandoGIT $ git add archivoNuevo.txt
```

(recordemos que este comando no presenta ninguna salida).

4) Revisemos que el comando anterior haya surtido efecto:

```
usuario@PC ~/probandoGIT $ git status
En la rama master
Cambios a ser confirmados:
  (usa "git reset HEAD <archivo>..." para sacar del área de stage)
    nuevo archivo:  archivoNuevo.txt
```

5) Hagamos commit con el nuevo archivo:

```
usuario@PC ~/probandoGIT $ git commit -m "Agregado archivoNuevo.txt"
[master 37c1e8e] Agregado archivoNuevo.txt
 1 file changed, 1 insertion(+)
 create mode 100644 archivoNuevo.txt
```

6) Comprobemos ahora qué hay en nuestro directorio de trabajo, con el comando **`ls`**

```
usuario@PC ~/probandoGIT $ ls
archivoNuevo.txt  saludo.txt
```

7) Supongamos ahora que deseamos eliminar el archivoNuevo.txt de nuestro repositorio. Para ello, utilizaremos el comando **`git rm`**, del siguiente modo:

```
usuario@PC ~/probandoGIT $ git rm archivoNuevo.txt
rm 'archivoNuevo.txt'
```

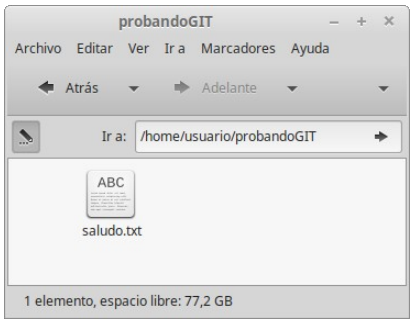
8) Comprobemos que hemos marcado para borrar el archivo de nuestro repositorio (el archivo se borrará cuando hagamos el próximo commit).

```
usuario@PC ~/probandoGIT $ git status
En la rama master
Cambios a ser confirmados:
  (usa "git reset HEAD <archivo>..." para sacar del área de stage)
    borrado:        archivoNuevo.txt
```

9) Pero, además, veamos que **el comando git rm ha eliminado el archivo de nuestro directorio de trabajo**:

```
usuario@PC ~/probandoGIT $ ls
saludo.txt
```

Como puede verse, el archivoNuevo.txt ya no está en la carpeta, situación que podemos comprobar también abriendo la ventana de la carpeta con el gestor de archivos del sistema operativo:



10) Por último, confirmemos la eliminación haciendo el commit correspondiente:

```
usuario@PC ~/probandoGIT $ git commit -m "Borado archivoNuevo.txt"
[master 74f8bb0] Borado archivoNuevo.txt
 1 file changed, 1 deletion(-)
 delete mode 100644 archivoNuevo.txt
```

Cambiar el nombre de un archivo

1) Volvamos a crear un archivo con un contenido cualquiera, llamado nombreIncorrecto.txt.

2) Comprobemos su situación:

```
usuario@PC ~/probandoGIT $ git status
En la rama master
Archivos sin seguimiento:
  (usa "git add <archivo>..." para incluirlo a lo que se será confirmado)
    nombreIncorrecto.txt
no hay nada agregado al commit pero hay archivos sin seguimiento presentes (usa "git add"
para hacerles seguimiento
```

3) Agreguémoslo a la staging area:

```
usuario@PC ~/probandoGIT $ git add nombreIncorrecto.txt
```

(recordemos que este comando no muestra una salida)

4) Volvamos a verificar el estado:

```
usuario@PC ~/probandoGIT $ git status
En la rama master
Cambios a ser confirmados:
  (usa "git reset HEAD <archivo>..." para sacar del área de stage)
    nuevo archivo:  nombreIncorrecto.txt
```

5) Ahora, hagamos commit del nuevo archivo:

```
usuario@PC ~/probandoGIT $ git commit -m "Agregado nombreIncorrecto.txt"
[master 2f8653e] Agregado nombreIncorrecto.txt
 1 file changed, 1 insertion(+)
 create mode 100644 nombreIncorrecto.txt
```

6) Supongamos ahora que deseamos cambiar el nombre de `nombreIncorrecto.txt` por `nombreCorrecto.txt`. Para eso, utilizaremos el comando **`git mv`**:

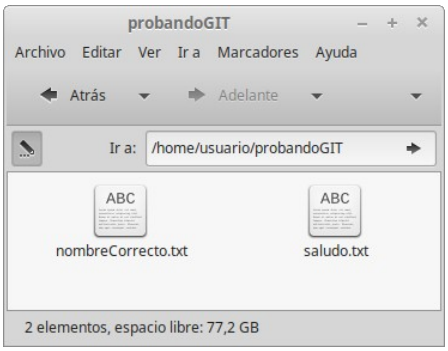
```
usuario@PC ~/probandoGIT $ git mv nombreIncorrecto.txt nombreCorrecto.txt
```

(este comando tampoco muestra ninguna salida)

7) Verifiquemos la nueva situación:

```
usuario@PC ~/probandoGIT $ git status
En la rama master
Cambios a ser confirmados:
  (usa "git reset HEAD <archivo>..." para sacar del área de stage)
    renombrado:     nombreIncorrecto.txt -> nombreCorrecto.txt
```

Cuando hagamos el próximo commit, se renombrará el archivo.



8) Pero veamos que **en nuestro directorio, el nombre del archivo ya cambió:**

```
usuario@PC ~/probandoGIT $ ls
nombreCorrecto.txt  saludo.txt
```

Podemos verificarlo también viendo la carpeta gráficamente:

9) Hagamos finalmente commit de nuestro cambio, para dejarlo registrado en el repositorio.

```
usuario@PC ~/probandoGIT $ git commit -m "Renombrado un archivo"
[master 665d0c2] Renombrado un archivo
 1 file changed, 0 insertions(+), 0 deletions(-)
 rename nombreIncorrecto.txt => nombreCorrecto.txt (100%)
```

10) Veamos el historial de los cuatro últimos commits, que reflejan los cambios de esta sección y de la anterior:

```
usuario@PC ~/probandoGIT $ git log -4 --oneline
665d0c2 (HEAD -> master) Renombrado un archivo
2f8653e Agregado nombreIncorrecto.txt
74f8bb0 Borado archivoNuevo.txt
37c1e8e Agregado archivoNuevo.txt
```

Vemos que el último comando **`git log`** utiliza el modificador **`-4`**, para indicar que queremos ver los últimos cuatro commits. Obviamente, puede reemplazarse en número 4 por cualquier otro.

Deshaciendo cambios en git

Todos nos equivocamos. En este apartado veremos cómo deshacer cambios.
En primer lugar, veremos cómo deshacer cambios en nuestro directorio de trabajo.
1) Abramos con el editor el archivo saludo.txt, borremos todo su contenido (dejémoslo vacío) y guardemos.
2) Comprobemos el estado del repositorio.

```
usuario@PC ~/probandoGIT $ git status
En la rama master
Cambios no rastreados para el commit:
 (usa "git add <archivo>..." para actualizar lo que será confirmado)
 (usa "git checkout -- <archivo>..." para descartar los cambios en el directorio de trabajo)
    modificado:     saludo.txt
sin cambios agregados al commit (usa "git add" y/o "git commit -a")
```

3) Agreguemos ahora el archivo modificado a la staging area:

```
usuario@PC ~/probandoGIT $ git add saludo.txt
```

(recordemos que este comando no muestra ninguna salida)
4) Volvamos a revisar el estado del repositorio.

```
usuario@PC ~/probandoGIT $ git status
En la rama master
Cambios a ser confirmados:
  (usa "git reset HEAD <archivo>..." para sacar del área de stage)
    modificado:     saludo.txt
```

5) Veamos qué cambios hay entre lo que está en la staging area y lo que había en el último commit:

```
usuario@PC ~/probandoGIT $ git diff --staged
diff --git a/saludo.txt b/saludo.txt
index 4df762a..e69de29 100644
--- a/saludo.txt
+++ b/saludo.txt
@@ -1,4 +0,0 @@
-HOLA
-QUE TAL
-HASTA LUEGO
-NOS VEMOS
```

Vemos que se han eliminado las cuatro líneas que tenía el archivo.

¡Un momento! ¡Esto no es lo que queríamos! Nos damos cuenta ahora de que no queremos que el archivo saludo.txt quede vacío. ¿Cómo deshacer los cambios?

6) Lo primero que debemos hacer es quitar el archivo saludo.txt de la staging area. Para eso, podemos usar el siguiente comando:

```
usuario@PC ~/probandoGIT $ git reset HEAD saludo.txt
Cambios fuera del área de stage tras el reset:
M   saludo.txt
```

7) Comprobemos ahora que el archivo, que estaba en estado "preparado" (staged), ahora está en estado "modificado" (modified), es decir que salió de la staging area.

```
usuario@PC ~/probandoGIT $ git status
En la rama master
Cambios no rastreados para el commit:
 (usa "git add <archivo>..." para actualizar lo que será confirmado)
  (usa "git checkout -- <archivo>..." para descartar los cambios en el directorio de trabajo)
    modificado:     saludo.txt
```

8) En nuestro directorio de trabajo, el archivo sigue teniendo los cambios que hicimos (es decir, está vacío).

```
usuario@PC ~/probandoGIT $ git diff
diff --git a/saludo.txt b/saludo.txt
index 4df762a..e69de29 100644
--- a/saludo.txt
+++ b/saludo.txt
@@ -1,4 +0,0 @@
-HOLA
-QUE TAL
-HASTA LUEGO
-NOS VEMOS
```

9) Ahora, queremos volver el archivo saludo.txt de nuestro directorio de trabajo a como estaba en el último commit. Tengamos en cuenta que **los cambios realizados se perderán.**

```
usuario@PC ~/probandoGIT $ git checkout -- saludo.txt
```

(Este comando no muestra ninguna salida. Después de la palabra "checkout" hay dos guiones --)

10) Veamos ahora la diferencia entre el archivo saludo.txt en nuestro directorio de trabajo y en el último commit:

```
usuario@PC ~/probandoGIT $ git diff
```

Este comando devolverá una salida vacía, puesto que ya no hay diferencias en ese archico.

11) Volvamos a abrir con nuestro editor el archivo saludo.txt, y comprobemos que lo hemos vuelto a su estado original, tal como estaba antes del paso 1).

La máquina del tiempo

Con git, podemos "retroceder en el tiempo", volviendo a cualquier versión anterior de nuestro repositorio.

1) Antes de empezar, nos cercioramos de que nuestro directorio de trabajo no tiene archivos que deban ser guardados en un commit:

```
usuario@PC ~/probandoGIT $ git status
En la rama master
nada para hacer commit, el árbol de trabajo esta limpio
```

2) Veamos el contenido del archivo saludo.txt. Podemos hacerlo con el editor, o con el siguiente comando:

```
usuario@PC ~/probandoGIT $ cat saludo
HOLA
QUE TAL
HASTA LUEGO
NOS VEMOS
```

3) Revisemos ahora nuestro historial[1] de commits, con el comando:

```
usuario@PC ~/probandoGIT $ git log --oneline
665d0c2 (HEAD -> master) Renombrado un archivo
2f8653e Agregado nombreIncorrecto.txt
74f8bb0 Borado archivoNuevo.txt
37c1e8e Agregado archivoNuevo.txt
```

1 Estos pasos suponen que ya han sido realizados los ejercicios anteriores.

```
2d82d33 Agregado 'que tal'
2724ff8 Cambio en la despedida del saludo
d785346 Segundo commit
fb84183 Nuestro primer commit
```

Supongamos, que por algún motivo, queremos “volver en el tiempo” al commit que tiene el SHA ID[2] `2724ff8` y el mensaje `Cambio en la despedida del saludo`.

4) Para ello, volveremos en el tiempo, con el siguiente comando:

```
usuario@PC ~/probandoGIT $ git checkout 2724ff8
Nota: actualizando el árbol de trabajo '2724ff8'.

Te encuentras en estado 'detached HEAD'. Puedes revisar por aquí, hacer
cambios experimentales y confirmarlos, y puedes descartar cualquier
commit que hayas hecho en este estado sin impactar a tu rama realizando
otro checkout.
Si quieres crear una nueva rama para mantener los commits que has creado,
puedes hacerlo (ahora o después) usando -b con el comando checkout. Ejemplo:
  git checkout -b <nombre-de-nueva-rama>
HEAD está ahora en 2724ff8 Cambio en la despedida del saludo
```

Interpretemos esta salida[3]. Nos informa que estamos en estado ‘detached HEAD’. Lo normal es que el HEAD (el lugar que estamos “mirando”, en el que estamos “pensando” ahora, “donde tenemos la cabeza”), coincida con el último commit de la rama en la que estamos (*master*, en nuestro caso). Pero, desde el último comando, esto ya no es así.
Estamos en el pasado[4]. Los cambios (incluso los commits) que hagamos se perderán, y no afectarán el “presente”, salvo que iniciemos otra rama, tema que se verá más adelante.

5) Volvamos a ejecutar el comando del paso 1), para ver la diferencia:

```
usuario@PC ~/probandoGIT $ git log --oneline
2724ff8 (HEAD) Cambio en la despedida del saludo
d785346 Segundo commit
fb84183 Nuestro primer commit
```

Vemos que solamente figuran los tres primeros commits. Estamos en el pasado.

6) Podemos ver con el editor el contenido del archivo saludo.txt.

```
usuario@PC ~/probandoGIT $ cat saludo
HOLA
HASTA LUEGO
NOS VEMOS
```

7) El archivo tiene el contenido que tenía en el pasado, es decir, cuando hicimos el tercer commit.

Podemos cambiar lo que queramos, incluso hacer commit, que nada de eso tendrá impacto en el “presente”.

8) Volvamos nuestra máquina del tiempo al presente, con el siguiente comando:

```
usuario@PC ~/probandoGIT $ git checkout master
La posición previa de HEAD era 2724ff8 Cambio en la despedida del saludo
Cambiado a rama 'master'
```

---

2 El SHA ID será siempre distinto. Nosotros usamos el número que aparece cuando armamos este apunte, pero, cuando estés probando estos pasos, el número será distinto. Es lo esperable.

3 Transcribimos la salida en inglés, para los que tengan git instalado en su idioma original:

```
Note: checking out '2724ff8'.
You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by performing another checkout.
If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -b with the checkout command again. Example:
git checkout -b new_branch_name
HEAD is now at 2724ff8... Cambio en la despedida del saludo.
```

4 Dejamos constancia del enorme esfuerzo que estamos haciendo para no llenar el apunte de referencias a centenares de historias de ciencia-ficción que nos vienen a la mente. Con un poco de suerte, llegaremos al final del apunte sin nombrar a Marty McFly.

9) Ahora HEAD y la rama *master* coinciden nuevamente (“estamos pensando en”, “tenemos la cabeza en” la rama *master*). Comprobémoslo:

```
usuario@PC ~/probandoGIT $ git log --oneline
665d0c2 (HEAD -> master) Renombrado un archivo
2f8653e Agregado nombreIncorrecto.txt
74f8bb0 Borado archivoNuevo.txt
37c1e8e Agregado archivoNuevo.txt
2d82d33 Agregado 'que tal'
2724ff8 Cambio en la despedida del saludo
d785346 Segundo commit
fb84183 Nuestro primer commit
```

10) Supongamos ahora que queremos eliminar la versión actual del archivo saludo.txt, y reemplazarla por la versión “del pasado”. Es decir, queremos dejar el archivo saludo.txt tal como estaba en el tercer commit. Para eso, podemos usar el siguiente comando:

```
usuario@PC ~/probandoGIT $ git checkout 2724ff8 saludo.txt
```

(Dos observaciones: a) Recuerden cambiar el SHA ID 2724ff8 por el que corresponda en el repositorio de ustedes. b) Este comando no produce ninguna salida).

11) Veamos el nuevo estado del repositorio.

```
usuario@PC ~/probandoGIT $ git status
En la rama master
Cambios a ser confirmados:
  (usa "git reset HEAD <archivo>..." para sacar del área de stage)
    modificado:     saludo.txt
```

Vemos que el archivo se ha modificado, y se ha guardado en la staging area.

12) Para ver el nuevo contenido del archivo, usaremos el comando:

```
usuario@PC ~/probandoGIT $ cat saludo
HOLA
HASTA LUEGO
NOS VEMOS
```

13) ¿Qué cambió desde el último commit?

```
usuario@PC ~/probandoGIT $ git diff --staged
diff --git a/saludo.txt b/saludo.txt
index 4df762a..43d7e2e 100644
--- a/saludo.txt
+++ b/saludo.txt
@@ -1,4 +1,3 @@
 HOLA
-QUE TAL
 HASTA LUEGO
 NOS VEMOS
```

Repasemos lo que hemos hecho hasta aquí:

- “Viajamos al pasado” al tercer commit.
- Revisamos el contenido del archivo saludo.txt en aquel momento.
- “Volvimos al presente”
- Reemplazamos la versión actual del archivo por la versión del tercer commit.

15) Ahora, lo único que nos queda es hacer commit para confirmar los cambios en el repositorio:

```
usuario@PC ~/probandoGIT $ git commit -m "Revertido saludo.txt al 3er commit"
[master 75c97a0] Revertido saludo.txt al 3er commit
 1 file changed, 1 deletion(-)
```