Comandos utilizados y vistos en clases:

- git init:

Esto crea un subdirectorio nuevo llamado .git, el cual contiene todos los archivos necesarios del repositorio – un esqueleto de un repositorio de Git. Todavía no hay nada en tu proyecto que esté bajo seguimiento.

- git fetch:

Descarga los cambios realizados en el repositorio remoto.

- git merge <*nombre_rama*>:

Impacta en la rama en la que te encuentras parado, los cambios realizados en la rama "nombre_rama".

- git pull:

Unifica los comandos *fetch* y *merge* en un único comando.

- git commit -m "<mensaje>":

Confirma los cambios realizados. El "mensaje" generalmente se usa para asociar al *commit* una breve descripción de los cambios realizados.

- git push origin <*nombre_rama*>:

Sube la rama "nombre_rama" al servidor remoto.

- git status:

Muestra el estado actual de la rama, como los cambios que hay sin commitear.

- git add <*nombre_archivo*>:

Comienza a trackear el archivo "nombre_archivo".

- git checkout -b <*nombre_rama_nueva*>:

Crea una rama a partir de la que te encuentres parado con el nombre "nombre_rama_nueva", y luego salta sobre la rama nueva, por lo que quedas parado en esta última.

- git checkout -t origin/<*nombre_rama*>:

Si existe una rama remota de nombre "nombre_rama", al ejecutar este comando se crea una rama local con el nombre "nombre_rama" para hacer un seguimiento de la rama remota con el mismo nombre.

- git branch:

Lista todas las ramas locales.

- git branch -a:

Lista todas las ramas locales y remotas.

- git branch -d <*nombre_rama*>:

Elimina la rama local con el nombre “nombre_rama”.

- git push origin *<nombre_rama>*:

Commitea los cambios desde el branch local origin al branch “nombre_rama”.

- git clone <https://link-con-nombre-del-repositorio>:

Sirve para descargar el código fuente existente desde un repositorio remoto (como Github, por ejemplo).

- Git revert:

A veces, necesitaremos deshacer los cambios que hemos hecho. Hay varias maneras para deshacer nuestros cambios en local y/o en remoto (dependiendo de lo que necesitemos), pero necesitaremos utilizar cuidadosamente estos comandos para evitar borrados no deseados.

Una manera segura para deshacer nuestras commits es utilizar git revert. Para ver nuestro historial de commits, primero necesitamos utilizar el git log --oneline:

```
Cem-MacBook-Pro:my-new-app cem$ git log --oneline
3321844 (HEAD -> master) test
e64e7bb Initial commit from Create React App
```

histórico de git en mi rama master.

Entonces, solo necesitamos especificar el código de comprobación que encontrarás junto al commit que queremos deshacer:

```
git revert 3321844
```

Después de esto, verás una pantalla como la de abajo -tan solo presiona shift + q para salir:

```
Revert "test"

This reverts commit 332184490ef2b5db289d85ed3f1a13ff2d5f94b9.

# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
#
# Committer: Cem <cem@Cem-MacBook-Pro.local>
#
# On branch master
# Changes to be committed:
#       modified:   src/App.js
#       deleted:    src/components/myFirstComponent.js
#
~
~
~
~
~
~
~
~
~
"~/Desktop/my-new-app/.git/COMMIT_EDITMSG" 14L, 384C
```

El comando git revert deshará el commit que le hemos indicado, pero creará un nuevo commit deshaciendo la anterior:

```
Cem-MacBook-Pro:my-new-app cem$ git log --oneline
cd7fe6f (HEAD -> master) Revert "test"
3321844 test
e64e7bb Initial commit from Create React App
```

commit generado con el git revert.

La ventaja de utilizar git revert es que no afecta al commit histórico. Esto significa que puedes seguir viendo todos los commits en tu histórico, incluso los revertidos.

Otra medida de seguridad es que todo sucede en local a no ser que los enviemos al repositorio remoto. Por esto es que git revert es más seguro de usar y es la manera preferida para deshacer los commits.