educación IT
CODE YOUR FUTURE

Git:
Desarrollo colaborativo

Módulo 5

Reset

El comando Reset

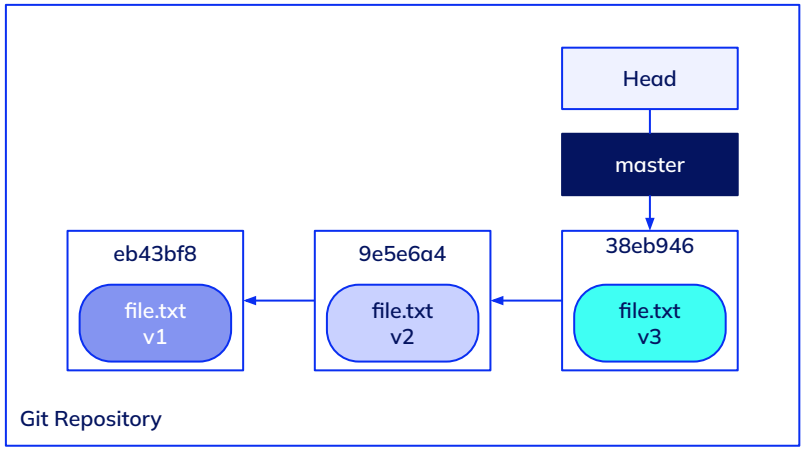
Si el trabajo que estuvimos haciendo no fue aún sincronizado con el repositorio remoto, todavía tenemos la posibilidad de **ordenar nuestro historial de cambios y borrar aquellos `commits` que estuvieron de más,** con el siguiente comando:

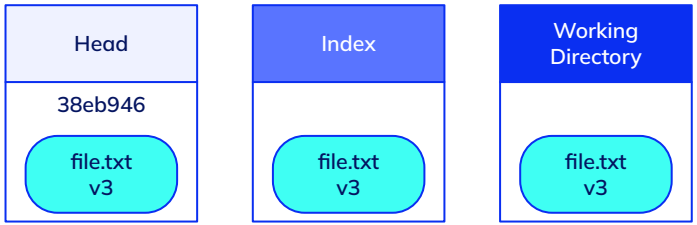
```
> git reset [tipo] <commit>
```

Para entender mejor este comando, en el siguiente slide seguiremos un ejemplo más claro.

Supongamos que estamos trabajando en un repositorio donde tenemos un archivo llamado *file.txt* al que ya le hicimos varios cambios.

Como muestra el gráfico, el `HEAD` está actualmente ubicado en el `commit 38eb946` viendo la versión 3 de nuestro archivo en el *branch master*. Por detrás nuestro hay dos `commits` más, es decir, los dos de las versiones anteriores del mismo archivo.





Supongamos entonces que lo que necesitamos es deshacer el último **commit**, donde nos encontramos ahora mismo, para volver a una versión anterior del archivo. Podríamos ejecutar el siguiente comando:

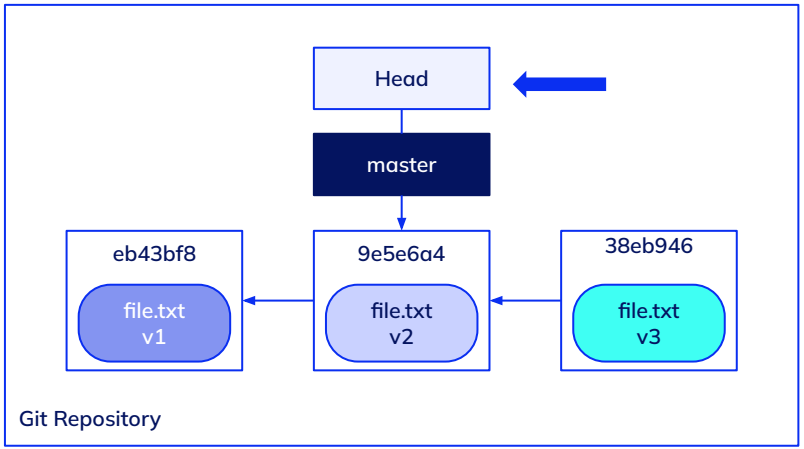
```
> git reset --soft 9e5e6a4
```

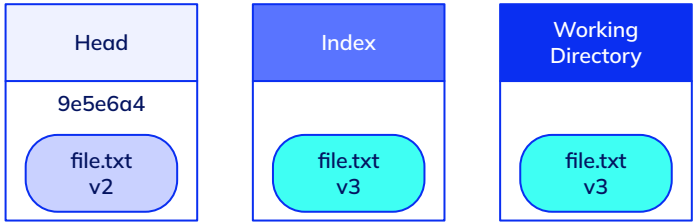
Lo primero que hará el comando **reset** es intentar **mover el puntero HEAD hasta el commit** que se especificó. ¿En qué se diferencia esto de hacer un `checkout` y movernos libremente también a ese `commit`?

El **checkout** solo mueve el **HEAD** y permite ver nuestro **Working Directory** en ese momento mientras que el **reset** no solo se encarga de esto sino también de editar el puntero del **branch** donde estemos trabajando y modificarlo para que tome el mismo cambio.

En forma gráfica:

```
> git reset --soft HEAD~
```





Lo más probable es que si se ejecuta el comando **`git status`** en ese momento, se pueda observar el cambio introducido en el `commit` previo como parte del *Stage Area* del `commit` actual, como si los cambios se hubiesen hecho recién.

También se observa:

1. El puntero del *branch* donde estábamos trabajando también se movió con nosotros.
2. Utilizamos el flag **`--soft`** para ejecutar el comando. Explicaremos esto a continuación.

Tipos de Reset

Existen tres tipos de **`Reset`**.

- `--soft`
- `--mixed`
- `--hard`

`reset soft`

Es aquel en donde **los cambios que hemos intentado resetear aún se mantienen visibles, pero ya no forman parte de ningún `commit` en nuestro *branch***.

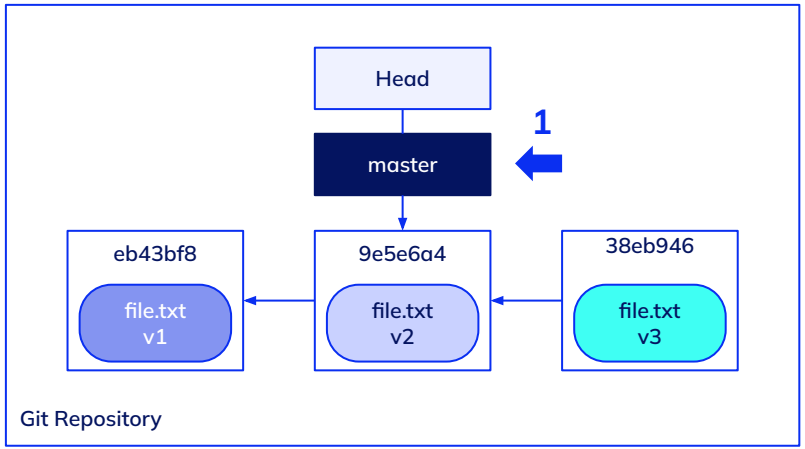
Por otro lado, **estos cambios se van a ver en los archivos** y, sobre todo, **reflejados en el *Stage Area* cuando ejecutemos un `git status`** y veamos que esos cambios que intentamos deshacer forman parte del próximo `commit`, en el caso en que hagamos uno en este mismo momento. Podemos hacer referencia del último gráfico para entender mejor el mecanismo.

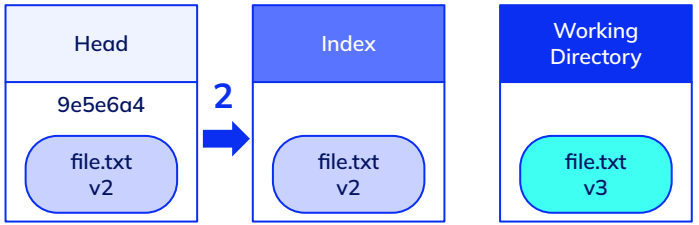
`reset mixed`

Es aquel en el que **los cambios también se siguen manteniendo visibles en nuestros archivos,** es decir, no fueron totalmente deshechos pero si consultamos el `status` de nuestro repositorio podríamos observar que **no forman parte del *Stage Area***.

Es decir, que se van a ver como si se hubieran hecho recién pero como si todavía no se hubiera corrido el comando **`git add`**.

```
> git reset [--mixed] HEAD~
```



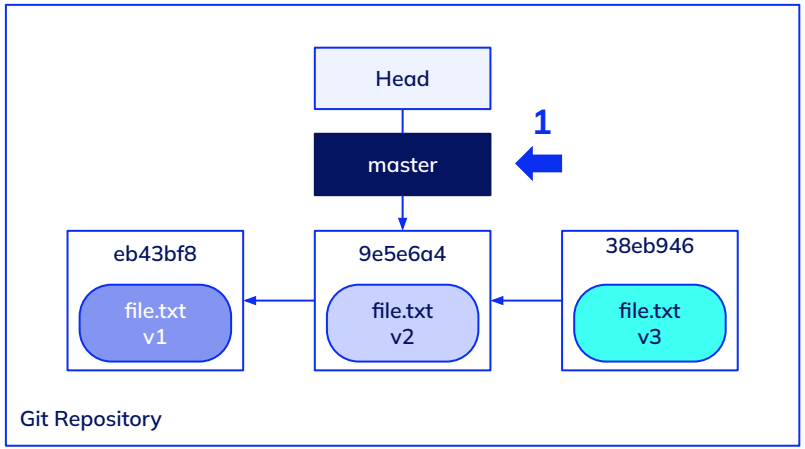


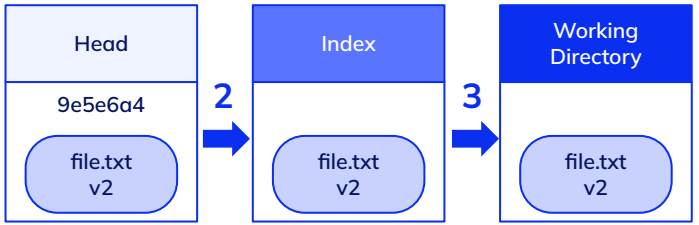
`reset hard`

Es el más “peligroso”. El **`reset`** va a comenzar por lo mismo que los demás, corriendo el `HEAD` al `commit` establecido pero **todos los cambios que se hayan hecho durante ese tiempo no solo no van a formar parte del *Stage Area* sino que no vamos a verlos tampoco en los archivos actuales**, es decir nuestro ***Working Area***.

Podríamos decir que, en este caso, “perdimos” un `commit` porque ya no vamos a tener ese libre acceso a los cambios en caso de arrepentirnos a tiempo como con los otros dos tipos.

```
> git reset --hard HEAD~
```





Git Blame

Supongamos que hemos sido capaces de identificar a través de nuestro código **un error *(bug)*** y lo que necesitamos es **deducir en qué momento se introdujo ese cambio.**

La anotación de archivos o *file annotation* va a ser nuestra mejor amiga. Se puede acceder a esta herramienta extra con:

```
> git blame
```

Permite saber el último `commit` en el que fue introducida una línea específica de un archivo.

Pongamos como ejemplo las primeras líneas de un archivo llamado *server.js:*

```
const express = require('express');
const app = express();
const fetch = require('node-fetch');
const LocalStorage = require('node-localstorage').LocalStorage;
const localStorage = new LocalStorage('./.data');
const fs = require('fs');
const OctokitApp = require('@octokit/app');
const request = require('@octokit/request');

class Server {

  constructor() {
    this.basicStr = 'basic';
    this.oAuthStr = 'OAuth';
    this.serverStr = 'Server-to-Server';
    this.searchStr = 'Search';
    this.userStr = 'User';
    this.state = {
      authType: '',                // || 'OAuth' || 'Server-to-Server'
      authTarget: this.searchStr,  // || 'User'
      clientId: process.env.GH_CLIENT_ID,
      oAuthToken: localStorage.getItem('oauth'),
      oAuthState: String(Math.random() * 1000000),
      rateLimitRemaining: '',
      rateLimitTotal: '',
      rateResetDate: '',
      serverToken: ''
    };

    this.startup();
    this.api();
  }
```

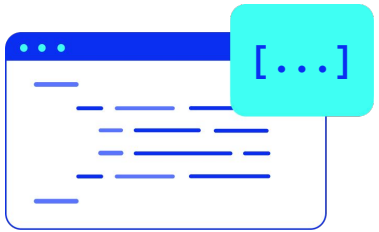
Si contamos desde arriba hacia abajo, vamos a prestar **especial atención a la línea 10 en adelante.** De esta manera podemos realizar:

```
> git blame -L 10,20 server.js
8b638d1e (John Bohannon 2019-06-04 09:12:11 -0400 10) class Server {
8b638d1e (John Bohannon 2019-06-04 09:12:11 -0400 11)
8b638d1e (John Bohannon 2019-06-04 09:12:11 -0400 12)   constructor() {
8b638d1e (John Bohannon 2019-06-04 09:12:11 -0400 13)     this.basicStr = 'basic';
8b638d1e (John Bohannon 2019-06-04 09:12:11 -0400 14)     this.oAuthStr = 'OAuth';
8b638d1e (John Bohannon 2019-06-04 09:12:11 -0400 15)     this.serverStr = 'Server-to-Server';
8b638d1e (John Bohannon 2019-06-04 09:12:11 -0400 16)     this.searchStr = 'Search';
8b638d1e (John Bohannon 2019-06-04 09:12:11 -0400 17)     this.userStr = 'User';
8b638d1e (John Bohannon 2019-06-04 09:12:11 -0400 18)     this.state = {
8b638d1e (John Bohannon 2019-06-04 09:12:11 -0400 19)       authType: '', // || 'OAuth' || 'Server-to-Server'
8b638d1e (John Bohannon 2019-06-04 09:12:11 -0400 20)       authTarget: this.searchStr,    // || 'User'
```

Lo que estamos haciendo es ejecutar el comando **`git blame`** con varios parámetros adicionales, en particular **`-L 10,20`** (indica que lo único que necesitamos revisar del archivo son las líneas de código que van desde la línea 10 a la 20).

Si analizamos el *output* que nos da la línea de comandos, podemos observar que se nos está **brindando el *hash* del `commit` que introdujo cada línea escrita en el archivo, junto con el autor, la fecha y hora.**

En este caso, todas las líneas fueron escritas por una misma persona, pero imaginemos el caso en el que se hubiera introducido un cambio en el archivo en la línea 17.



Podemos volver a realizar nuevamente la anotación de archivos para obtener el *output* que vemos debajo.

Se presenta el nombre de un autor distinto al resto en la línea 17 junto con el *hash* del `commit` en el que se realizó el cambio.

```
> git blame -L 10,20 server.js
8b638d1e (John Bohannon     2019-06-04 09:12:11 -0400 10) class Server {
8b638d1e (John Bohannon     2019-06-04 09:12:11 -0400 11)
8b638d1e (John Bohannon     2019-06-04 09:12:11 -0400 12)   constructor() {
8b638d1e (John Bohannon     2019-06-04 09:12:11 -0400 13)     this.basicStr = 'basic';
8b638d1e (John Bohannon     2019-06-04 09:12:11 -0400 14)     this.oAuthStr = 'OAuth';
8b638d1e (John Bohannon     2019-06-04 09:12:11 -0400 15)     this.serverStr = 'Server-to-Server';
8b638d1e (John Bohannon     2019-06-04 09:12:11 -0400 16)     this.searchStr = 'Search';
7737d6b0 (Horacio Gutierrez 2020-05-31 19:01:17 -0300 17)     this.test = true;
8b638d1e (John Bohannon     2019-06-04 09:12:11 -0400 18)     this.userStr = 'User';
8b638d1e (John Bohannon     2019-06-04 09:12:11 -0400 19)     this.state = {
8b638d1e (John Bohannon     2019-06-04 09:12:11 -0400 20)       authType: '', // || 'OAuth' || 'Server-to-Server'
```

Git Grep

Git permite realizar **varios tipos de búsqueda** a través de nuestros repositorios. Dos de las más populares son:

- La posibilidad de **buscar *strings* entre nuestros archivos.**
- Poder **buscar *strings* en nuestro *log*** de `commits`.

La **búsqueda de strings en nuestros archivos** se realiza con:

```
> git grep
```

Con `git grep` podemos hacer una búsqueda común como si estuviéramos trabajando en cualquier editor de código.

Continuaremos con el ejemplo del archivo *server.js.* Vamos a buscar la línea de código que sabíamos era la conflictiva en nuestro programa:

```
> git grep this.test
server.js:  this.test = true;
```

Si prestamos atención al *output* de la línea de comandos, nos está informando que el *string* **“`this.test`”** se encuentra presente en el archivo *server.js.* Si estuviera en muchas más instancias, podríamos ver un listado de varios archivos.

Es posible utilizar varios parámetros para hacer las búsquedas un poco más interesantes, como por ejemplo:

```
> git grep -n this.test
server.js:17:     this.test = true;
```

Con esta nueva búsqueda, se puede observar **no solo el nombre del archivo en el que se encuentra el *string*, también la línea en la que está presente.**

También podemos realizar algo muy interesante y es:

```
> git grep -c this.test
server.js:1
```

De esa manera, se puede saber **cuántas veces se encuentra repetida la línea conflictiva en todos los archivos** de nuestro repositorio de forma tal que podamos arreglar el error en todos lados.

Quizás no estemos buscando dónde existe un término, sino más bien **cuándo existió o fue introducido.** Para realizar esto, debemos recordar el comando **`git log`**, que se vuelve más poderoso si comenzamos a trabajar con él utilizando sus parámetros.

Se utiliza el parámetro **`-S`** para agregar el término a buscar. El comando retorna un listado de todos los **`commits`** en donde fue introducido un cambio o **`string`**.

```
> git log -S this.test
commit 7737d6b01946e9dafb689de011bfac9da694db2f
Author: Horacio Gutierrez <horacio.estevez@gmail.com>
Date:   Sun May 31 19:01:17 2020 -0300

    add new change
```

¡Sigamos trabajando!