**Fundamentos en Git**

**3. Agregamos archivos a Work Tree y confirmamos cambios.**

Estamos usando visual Studio para ver con más presentación los procesos de Git. En este caso se ha modificado la estructura de un HTML y en Visual Studio podemos ver las modificaciones y las abre como un work trhee.

Git status no devuelve información acerca de la rama en la que nos encontramos, en el caso del ejemplo estamos en la rama master, donde estaremos trabajando los código. Podemos tener múltiples rámas de desarrollo.

* Los commits son confirmaciones de cambio.
* Los archivos untracked son aquellos que no tienen un seguimiento o un control de versiones.
* Git add –all fuerza a git a meter todos los archivos.

4. Ahora revisemos la sección de código fácilito:

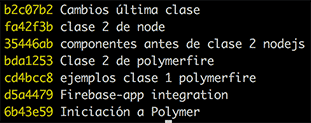
Con el comando check out lo que podemos hacer es volver entre los commits del pasado. Sin afectar a los últimos realizados, se quedan guardados estos y siempre es posible volver a interactuar con ellos.

**Cambios en un archivo**: el código git checkout – en el caso de la clase es index.html nos permite revertir los cambios que le hemos hecho a un archivo.

**Cambios en un stage**: Para esto cuando ya el archivo está registrado en un stage lo que podemos hacer primero es aplicar reset HEAD que lo que hace es que le estamos pidiendo que regrese el archivo index al último commit. Luego de eso aplicamos git checkout -- para que revierta los cambios en el último commit. Cuando ya hemos guardado un archivo en el stage lo primero que tenemos que hacer es sacarlo del stage y eso se logra con el reset HEAD.

Cambios en un commit: En el caso de que a última hora hayas guardado algo que no querías guardar o borraste algo. Lo primero que se hace es buscar el último commit o el commi en el que se hayan hecho los cambios que en este caso supongamos que de verdad es el primero: git log oneline: Lo que hace es que nos muestra de una manera más sencilla la cantidad de commits realizados.

**Nota:** Esta opción en realidad es un atajo de escribir estas dos opciones: "--pretty=oneline --abbrev-commit". La primera indica que justamente que pongas un commit por línea y la segunda te indica que muestres la cadena resumida con el identificador del commit, en vez de la cadena de 40 bytes hexadecimal normal. El identificador resumido es suficiente para referirse de manera inequívoca a un commit, ya que el resto del identificador contiene información sobre el usuario que hizo el commit y otros datos de control.



Lo que podemos hacer a continuación de lo que buscamos es sacar el último commit para así poder hacer un cambio. Así pasamos a git reset del commit que queremos cambiar, paso siguiente será revisar el estatus del repositorio para saber si se ha concretado el cambio, de ser así solamente se aplica git restore para reestablecer los cambios hechos.

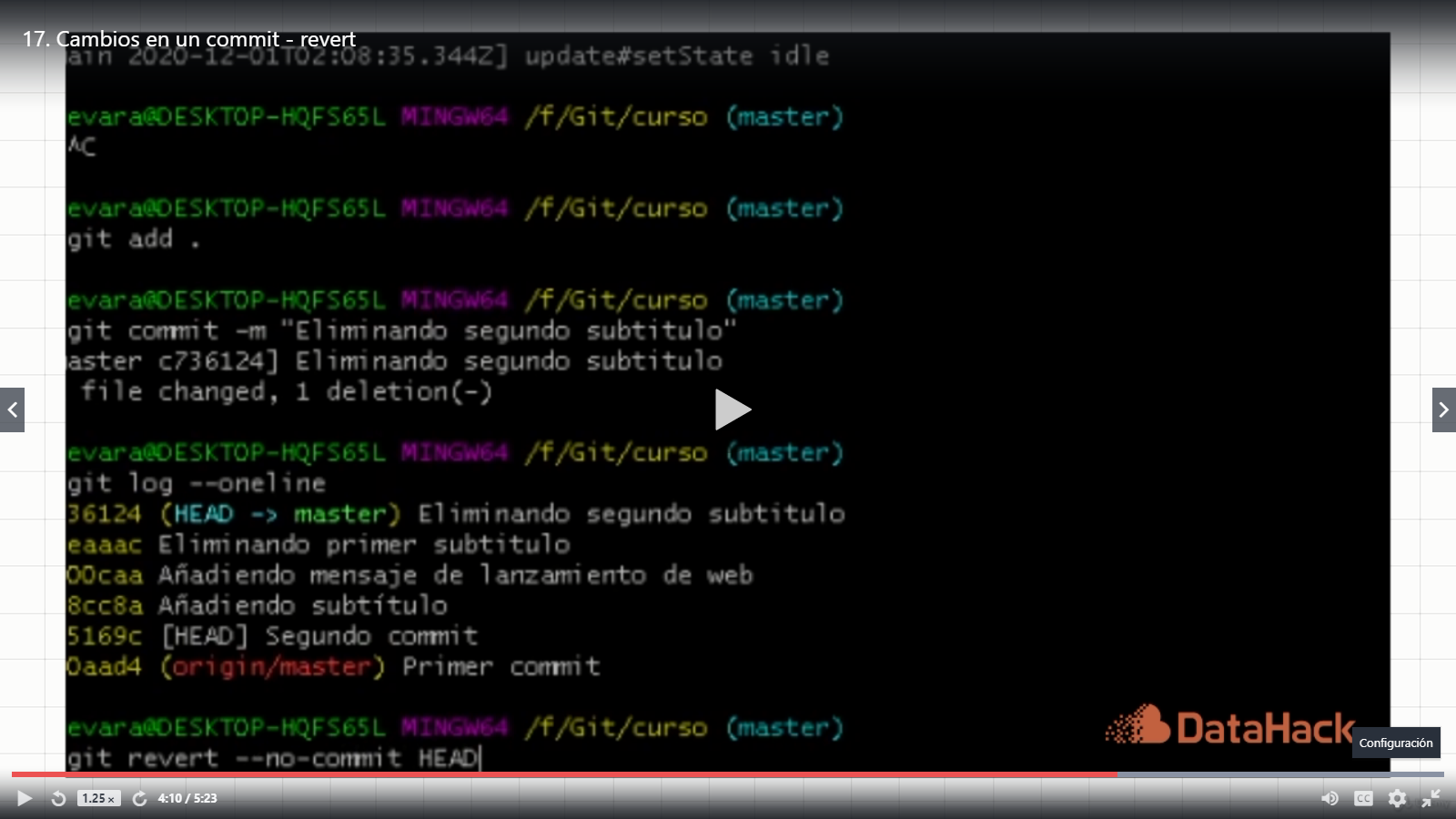
Para borrra un commit de manera fuerte y a fondo lo que tenemos que hacer el mítico git reset hard que este lo que hace es borrra el comit del respositorio y del disco duro.

Cambios en un commit (Revert): Primero cuando ya tenemos un cambio y después ya no lo deseamos en el código, lo que haremos es hacer un git revert a la etiqueta HEAD que es el último commit hecho. Nos lanzará a lo que es la pantalla de Revert está “eliminando subtítulos”.

Cuando ocurre esto lo que se hace es que se revienten los cambios hechos, sin eliminar el commit donde fueron hechos los cambios.

Hizo algo acá medio loco que fue resetear el commmit de introducción a clase. Lo que ha ocurrido con eso es que ya no existe ni el revert ni las modificaciones posteriores al formato original.

Supongamos que se harán dos cambios, seriados de manera que tengamos uno y posteriormente el otro cambio:



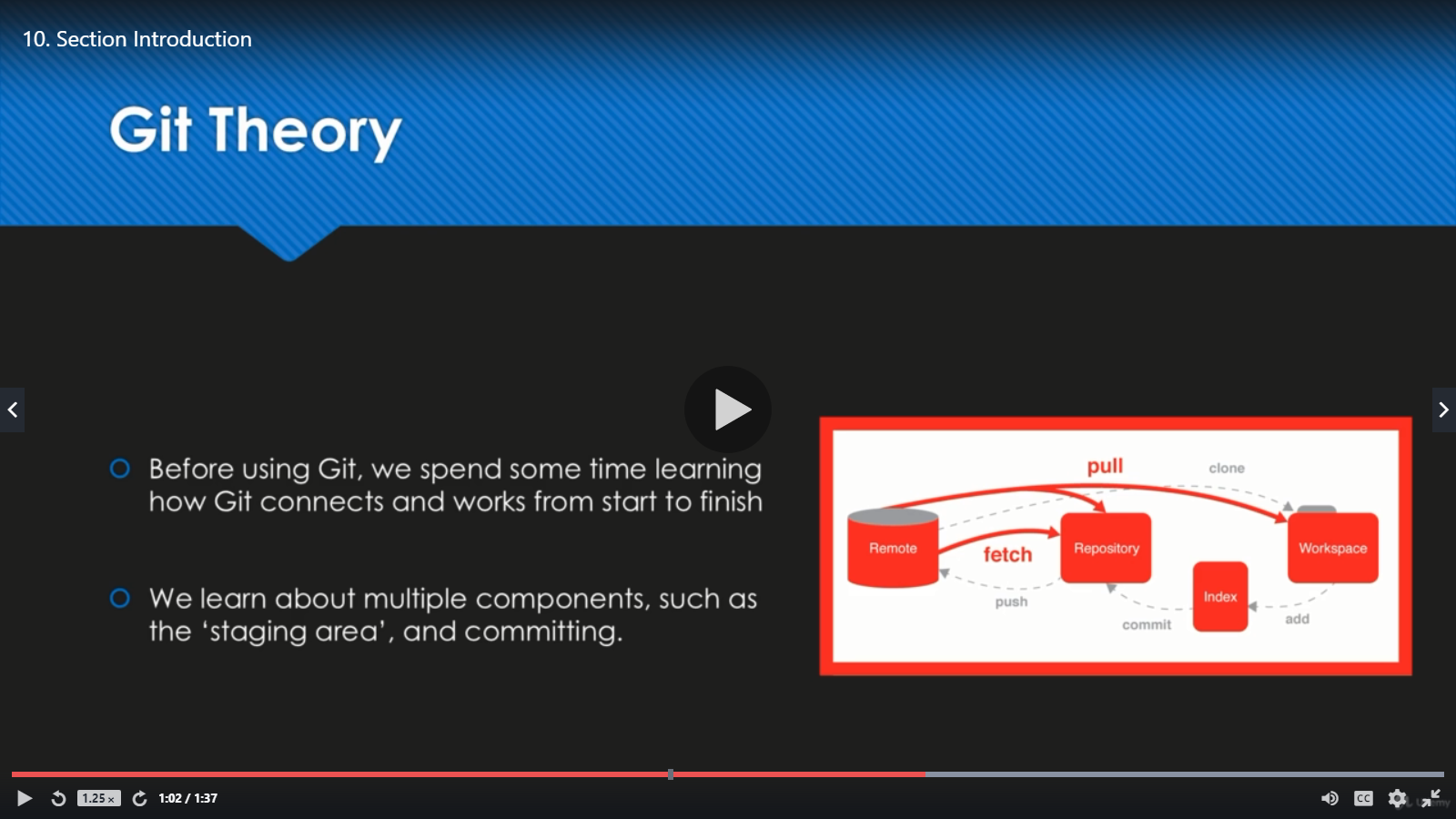
Lo que se hace es colocar que se quiere revertir una confirmación que no es la etiqueta HEAD (git revert –no-commit HEAD)

Es: (git revert –no-commit HEAD-1)

Luego: (git revert –continue)

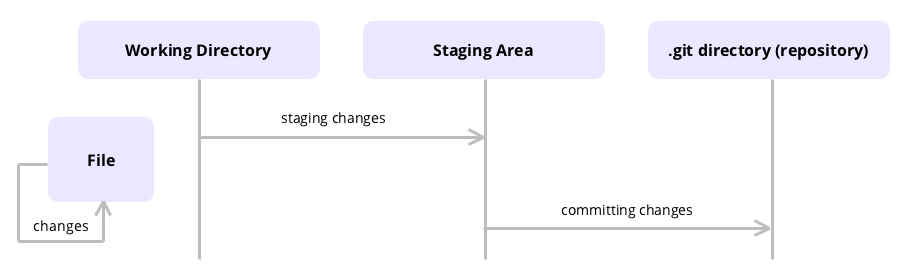
Nos aparecerá una notificación del commit que estamos revirtiendo.

Se han aarreglado dos commits que se han borrado y así se evitará que se eliminen commits que interfieran con nuestro trabajo.



¿Cuándo agregamos una confirmación por primera vez se crea el famoso stage? No lo sé veamos la respuesta mi hermano.

El comando de confirmación no guarda cambios en servidores remotos, solo en los repositorios locales de Git.



Rm –cached file.txt en el caso del ejemplo. Lo que hará este comando es que toma el archivo en cuestión y lo remueve del stage.

*Definición de ramas:* nnasm

Git almacena una rama como referencia a un commit o confirmación.

Representa el extremo de una serie de commits

No es un contenedor de commits.

Una rama representa una línea independiente de desarrollo, es una abstracción en la cuestión de procesos preparación y confirmación.

El comando ( Git Branch ) nos permite:

1. Crear
2. Enumerar
3. Cambiar

Lo que no te permite es:

1. Cambiar entre ramas
2. Volver a unir un historial bifurcado

*Creación de ramas:* La

Para la creación de una rama y transportarte como ese commit mágico lo que se hace es git checkout -b production2

Hemos aprendido a crear una nueva rama sin el git Branch, al menos la parte de cambiarle el nombre me resulta buenísimo, creo que es con -m. Cómo borrar una rama, incluso como copiar una rama en otra.

Importante por que se me ha olvidado hace un momento. Para hacer un cambio de rama se utiliza el checkout (nombre de la rama en cuestión).

También importante para cambiar de rama lo que se hace es checkout (nombre de la rama).

Acabamos de hacer algo épico. Con el comando Git pull origin (esta es el repositorio de donde se hicieron los cambios)master(que es mi respositorio local).

Siempre o por lo menos cuando se entregen proyectos siempre deben de ir acompañados por un readme.md

Este puede contener muchos elementos dentro de si. Es importante poner todos des detalles actuales y del porvenir, si es que hay bugs en la plataforma o lo que se esté desarrollando