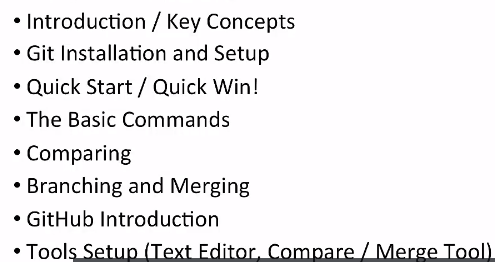
**GIT Course**

# Introduction

## Course Goals and Objectives:

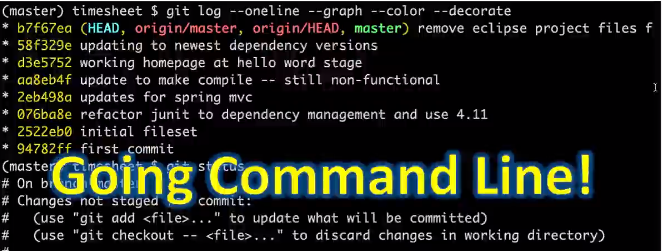


## Course Overview



## Why Command Line?

Empezando con la linea de commando es la major manera de aprender GIT, es una forma estándar de comunicar como GIT hace algo:



## Why Source Control?

El Source Control (o control de origen), en un inicio se describe como una copia de seguridad de cada versión de la fuente/fichero que has hecho. De esta forma, se crea una historia o un rastro de los cambios, lo cual permite deshacer los cambios que realizó o restauró desde un estado conocido anterior.

¡Es un control de versiones!

Tipologías de sistemas de Control de versiones:

* Centralized (centralizada) => Requieren de una conexión a un servidor centralizado que es la mejor fuente para la colección de archivos versionados.
  + Free: Subversion, CVS
  + Commercial: ClearCase, Perforce, Team Foundation Server (TFS)
  + Requieres connection to central server for most operations
* Decentralized / Distributed => No requiere de una conexión a un servido centralizado, ya que permite que la mayoría de las operaciones sean locales.
  + GIT
  + Mercurial (Hg)
  + Most operations are local
  + Central server not requiered

## Key GIT Terminology

### **The GIT Repository**

El **repositor** contiene ficheros, históricos de cambios de versiones y cualquier configuración especial que se mantienen juntas y están manejados por GIT. Por tanto, contiene todos los ficheros relacionados con un proyecto o aplicación.

### **The States of GIT**

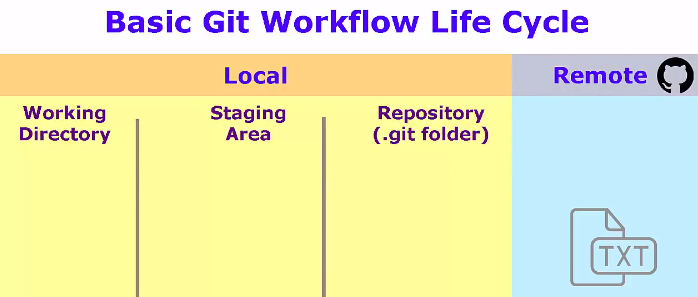
#### Local States

Estados locales relacionados con los archivos administrados por Git:

* **Working directory** => Repositorio de trabajo => El directorio de trabajo es el directorio o carpeta en su computadora que contiene todos los archivos de proyecto o aplicación. Los archivos dentro del directorio de trabajo, pueden o no ser administrados por Git. Dentro del directorio devtrabajo hay una carpeta oculta llamada ". git "que contiene el repositorio git real.
* **Staging área / pre-commit holding area** => Área de preparación => Es un área de espera para poner en cola los cambios para la próxima confirmación. Dado que los archivos en el área de preparación aún no se han confirmado, puede moverlos dentro y fuera del área de preparación sin afectar el repositorio de Git y su historial de cambios.
* **Coming – Git Repository (history)** => Repositorio de GIT o el historial de confirmación => Es el área en la que se confirman los archivos editados en el repositorio de trabajo.

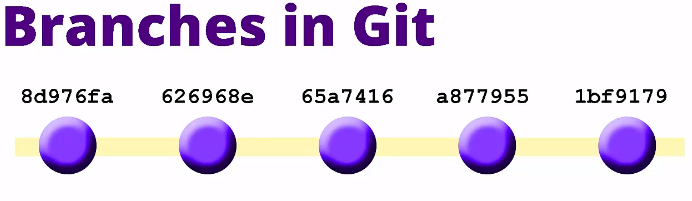
#### Remote Repository (GitHub)

El último estado del flujo de trabajo de GIT. Una vez que se finaliza el trabajo en local, se sube la información a remoto.



### **Master Branch (Marca maestra)**

Son líneas de tiempo que contienen los cambios realizados en las diversas versiones de los ficheros/proyectos:



En GIT, las ramas contienen commits. Cuando comenzamos, GIT nos proporciona una rama predeterminada denominada **Master**.

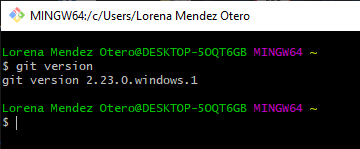
# GIT Installation

**Installation for Windows:**



A continuación, se procede con la instalación de GIT para Windows.

Una vez instalado el programa, nos mostrará la versión disponible:



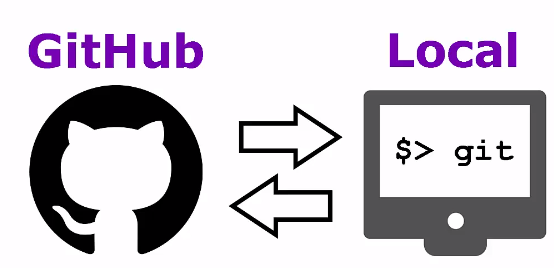
Link para releases notes de GIT:

[file:///C:/Program%20Files/Git/ReleaseNotes.html](file:///C:\Program%20Files\Git\ReleaseNotes.html)

# GIT Quick Start

## Starting with GitHub and Project Setup

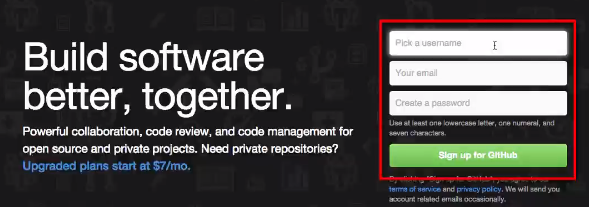
Workflow de GitHub:



Enlace a la página principal de GitHub:

<https://github.com/>

Para crear una cuenta en GIT, hay que crear el siguiente formulario:

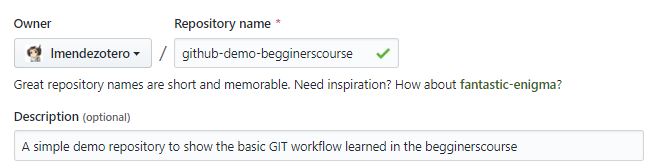


Una vez que la cuenta esta creado, vamos a crear un nuevo repositorio:

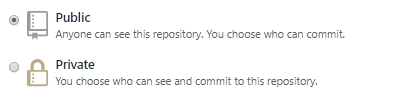
<https://github.com/new>

Ejemplo de la creación de un repositorio:

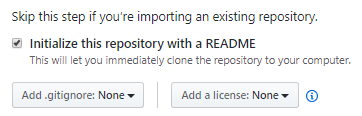
1. Añadimos el owner, el nombre del repositorio y la una descripción breve de lo que guardaremos en el respositorio.



1. Indicaremos que el repositorio sea público:



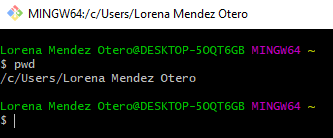
1. Indicar que el repositorio se inicie con un README (comenzará nuestro repositorio con un simple archivo README)



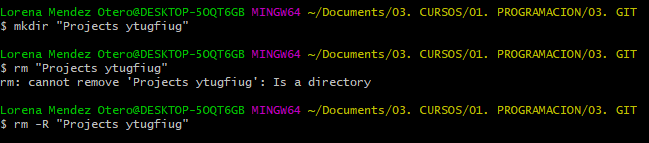
Una vez creado el repositorio, necesitamos preparar nuestro sistema local para poder continuar:



Para arrancar Git Bash, introduzca **pwd**:



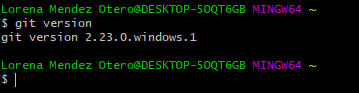
Crear una carpeta denominada “projects” debajo del directorio de inicio del usuario para administrar todos los proyectos juntos en un mismo directorio. Para ello, introduzca el COMANDO “***MKDIR***”:



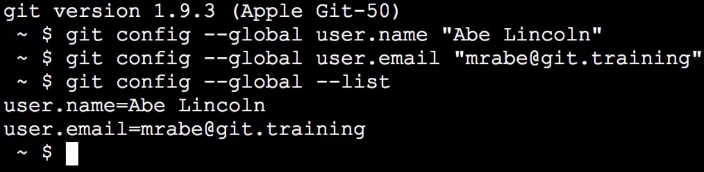
## Configuration, Clone and Git Basic Workflow

### **Configurar Git Bash**

Para ver la versión de Git que tenemos instalada, introduce “git version”:

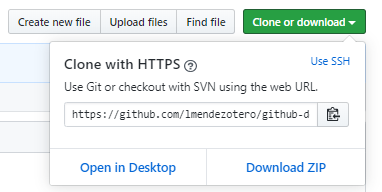


Configuración del nombre y correo electrónico del usuario (en este caso Lorena):



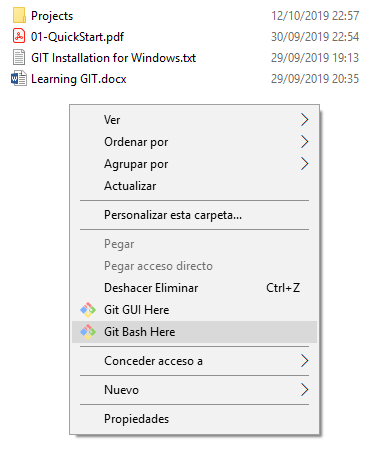
### **Clonar un repositorio**

Para clonar un repositorio, tenemos que volver a Git Hub y seleccionar la opción “Clone with HTTPS”:

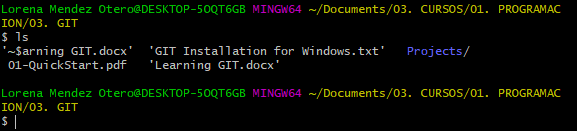


Luego copiamos la URL y la pegamos en Git Bash.

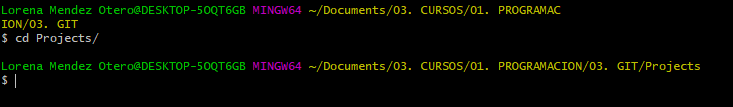
Para realizar correctamente el clon, nos vamos a la carpeta de Git en local y :

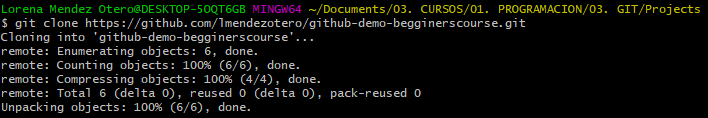


Una vez abierto Git Bash, chequeamos los directorios disponibles para guardar el clon introduciendo el COMANDO “***LS***”:

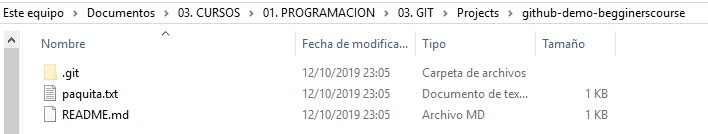


El directorio disponible es Projects, por lo que vamos a indicar que vamos a crear aquí nuestro clon. Cambiamos el nombre del directorio introduciendo el COMANDO “***CD***”:



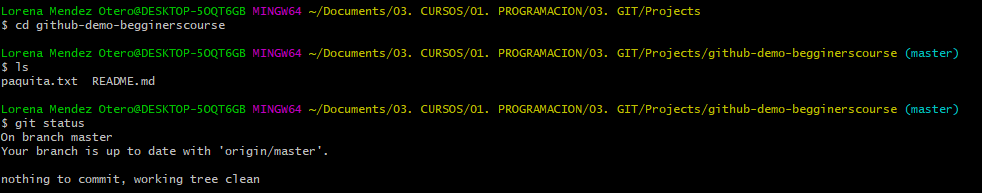
Ahora que ya estoy en Projects, inserto el comando “**GIT CLONE**” + URL copiada para crear el clon: 

Resultado de la visualización del clone en Local:



El comando GIT CLONE establece una relación entre el repositorio en GitHub y la referencia origin.

Para ver el status del repositorio clonado, introduzca el nombre del repositorio clonado y luego el COMANDO “***GIT STATUS***”:

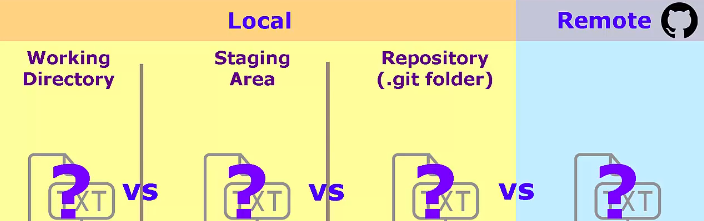


On branch master es la rama predeterminada para un repositorio Git.

“GIT STATUS” también nos dice que master está actualizado con origin/master, lo que se refiere a la rama maestra en la versión de GitHub del repositorio.

Finalmente, GIT STATUS nos dice si el directorio de trabajo está limpio (clean).

Por tanto, utilizamos GIT STATUS para ver si se ha producido algun cambio entre el directorio de trabajo, el área de preparación, nuestro repositorio local y remoto:



### **Crear un fichero en el repositorio**

Para crear un fichero nuevo en el repositorio, introducimos el COMANDO “ECHO” seguido del nombre del fichero entre “”. Luego, añadimos >> para canalizar el contenido de ECHO en un fichero txt start.txt:



Nombre del fichero: start.txt

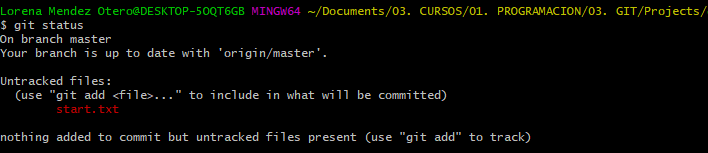
Contenido del fichero: “Test Git Quick Start: Paquita es guapa”

Utilizamos el COMANDO “***CAT***” para mostrar el contenido del fichero (en Python equivale a PRINT):

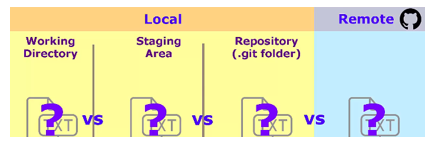


### **Git Basic Workflow**

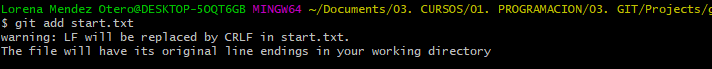
¿Qué significa tener un archivo sin seguimiento? => Un archivo en nuestro directorio de trabajo que aun no se ha añadido a Git.



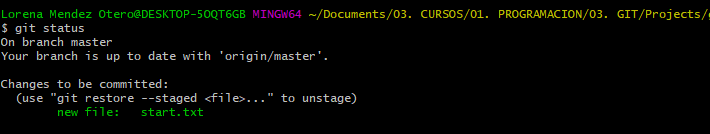
Por tanto, tenemos que añadir el fichero a la Stagin Area.



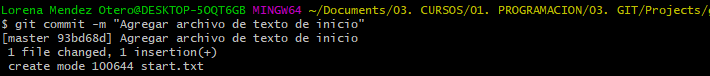
Para ello, añadimos el COMANDO “GIT ADD” y luego el nombre del fichero:



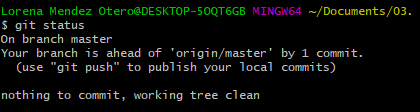
Chequeo del resultado final con GIT STATUS:



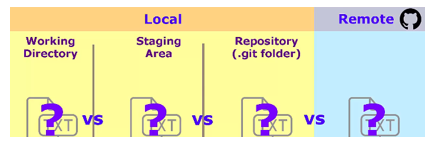
Ahora el fichero está en la Stagin Area, pero pendiente de introducir una confirmación en el commit (parte remarcada en rojo). Para confirmar el nuevo archivo introducimos el COMANDO “***GIT COMMIT -M***” y entre “” el mensaje de confirmación “Agregar archivo de texto de inicio”:



Si introducimos GIT STATUS, nos encontramos en que estamos de vuelta en un estado de directorio de trabajo limpio y que nuestra rama maestra está por delante de “origin/master” por una confirmación:



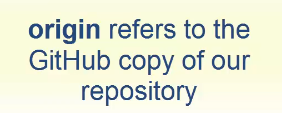
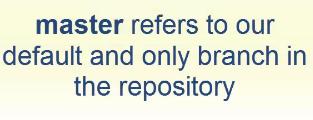
Por tanto, el archivo se ha movido hasta el área de Repositorio Local:



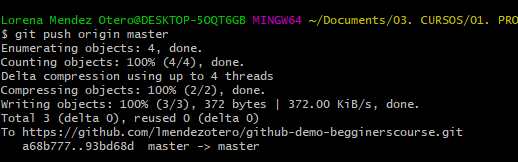
Como no hay confirmaciones pendientes, Git marca el fichero como “limpio”.

Para lanzar el fichero al Repositorio en remoto (GitHub), tenemos que hacer un paso final denominado **PUSH**  => Impuslo. Para ello, introduzca el COMANDO “***GIT PUSH***” seguido de “origin master”:

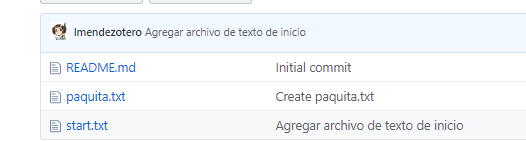


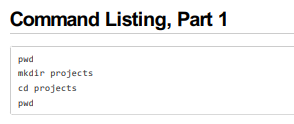
Después de presionar ENTER, nos vamos al browser en donde tenemos GitHub y refrescamos la página para ver los cambios. Nos pedirá introducir nombre de usuario y contraseña de GitHub. Una vez introducidas las credenciales, vemos los resultados:

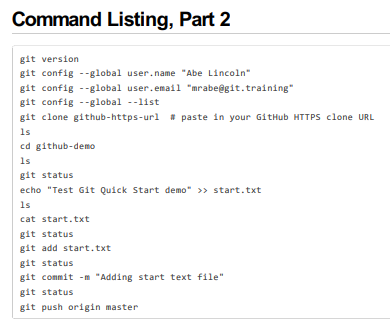


Nos vamos al browser de Internet en GitHub y refrescamos la página de nuevo:



Comandos de Git analizados en esta sección:

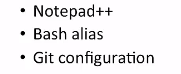




# Text Editor Installation

## Text Editor Installation Overview

Instalaremos el editor de texto **Notepadd++** y veremos cómo configurarlo para que funcione correctamente con Git.



### **Instalación de Notepadd++**

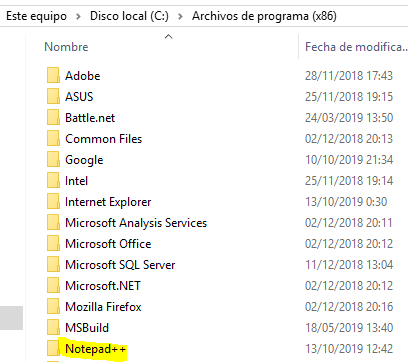
<https://notepad-plus-plus.org/downloads/>

Icono del programa **Notepad++** en el escritorio:



### **Configuración de Notepad++ en Git**

Primero, tenemos que encontrar en dónde se ha instalado Notepad++. Para ello, nos vamos al disco Local C y abro la carpeta “Archivos de programa (x86)”:

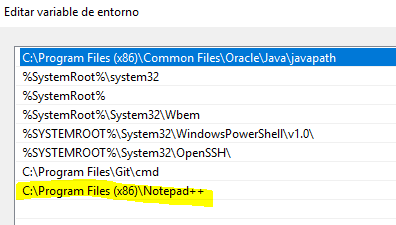




Copiamos la dirección de la ruta en al que está instalado Notepadd++:

C:\Program Files (x86)\Notepad++

Tenemos que crear un nuevo path en Variables de Entorno, que se encuentra en las Opciones Avanzadas de las Propiedades de Este equipo:



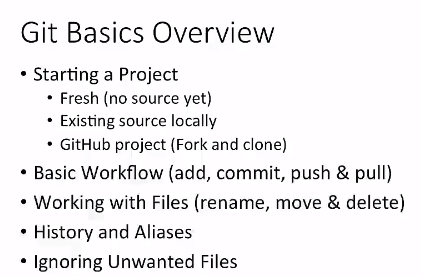
De esta forma, al introducer “Notepad++” en Git Bash se abrirá automáticamente el editor de texto.

## Text Editor for Windows

¿???????

# Basic GIT Commands

Hay 3 maneras de empezar un proyecto en GIT:



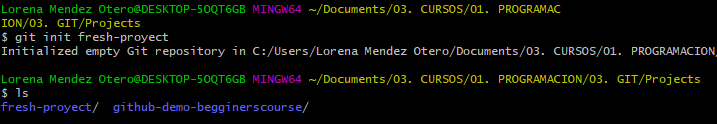
## Starting with a Fresh Project

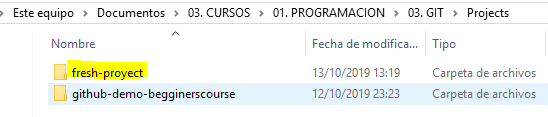
Vamos a iniciar un Proyecto sin ningún código fuente existente. Para ello, utilizaremos información de la siguiente página:

<https://hipsum.co/>

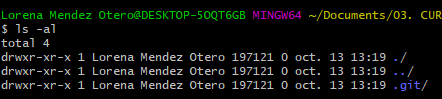
Voy a crear un nuevo repositorio en mi carpeta de Projects, por lo que la abro y arranco Git Bash desde allí.

Introduzco el COMANDO “***GIT INIT***” para crear un nuevo proyecto seguido del nombre del proyecto (fresh-project):



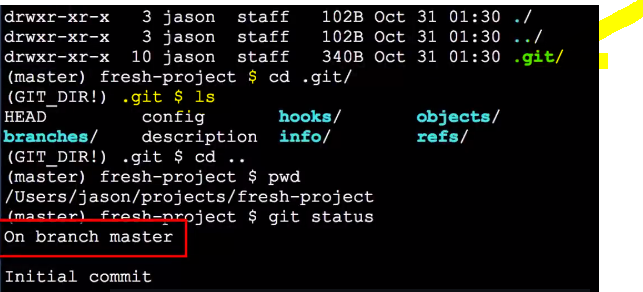


Si añadimos el comando LS, veremos que la carpeta fresh-proyect no tiene ningun archivo. Sin embargo, si añadimos **LS –AL**, nos encontramos con que el programa le dice al comando list que enumere todos los archivos y carpetas, incluidos los archivos y carpetas de puntos; mientras que "-l" le dice que haga una lista en el formato de listado. Ahora puedes ver que tenemos un ". git ".



Ahí es donde vive el repositorio de Git; la carpeta en la que estoy ahora es en realidad la carpeta de trabajo. Entonces si entro en el ". git ", puedes hacer un" ls ", y puedes ver todas las carpetas y archivos que componen el repositorio de Git internamente:

**NO ENTIENDO CÓMO LOGRAR ESTO:**

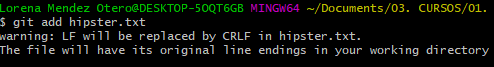


Continuamos con el proyecto y nos vamos a la página de Hipsum para copiar texto de relleno. Copiamos texto, lo guardamos en las notas y creamos un fichero nuevo insertando el COMANDO “MATE” seguido del nombre del archivo hipster.txt:

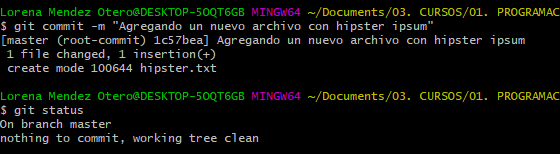


**NO LO HE LOGRADO CON MATE, PERO SI CON ECHO!!!!**

Actualmente estamos en el directorio de trabajo, por lo que Git aún no está rastreando esto. Para que Git rastree este archivo, simplemente necesitamos agregar/mover ese archivo al índice de Git o al Stagin Area de Git. Para hacerlo, simplemente escribimos "git add", espacio y luego el archivo que deseamos agregar.



Ahora, si hacemos nuestro "git status", Git nos dice que tenemos "Cambios para confirmar". Eso significa este nuevo archivo, "hipster. txt ", ahora está en el área de preparación de Git. Tenemos un archivo que está esperando ser confirmado, pero aún no está confirmado. Para confirmar los cambios, introducimos “**GIT COMMIT**” y el mensaje de confirmación “Agregando un nuevo archivo con hipster ipsum”:



¿Y si quiero retroceder hacía atrás?

Inserto el COMANDO “***RM-RF***” espeficiando la eliminación de la carpeta que quiero hacer:

Por tanto, ya no hay más repositorios de Git.

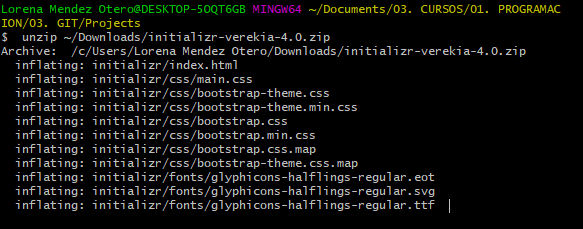
**NO LO HE LOGRADO!**

## Adding Git to an exist project (git init)

Para añadir Git a un Proyecto ya inexistente, vamos a utilizar la web:

<http://www.initializr.com/>

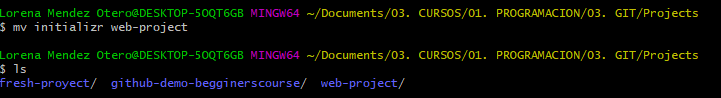
Descargamos unos archivos de dicha página web. A continuación, arrancamos Git Hash y descomprimimos los ficheros descargados de la web de initializr añadiendo el COMANDO “***UNZIP***”:



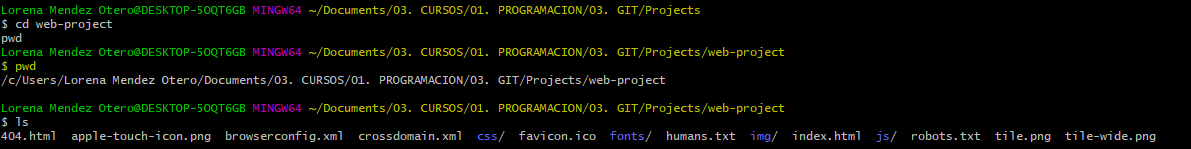
Chequeo de que existe la carpeta Initialir que contiene todos los documentos descomprimidos en la carpeta de Projects:



Vamos a cambiarle el nombre a la carpeta de Initializr por “web-project” utilizando el COMANDO “***MV***”:



Arrancar la carpeta web-project y ver todos los archivos que hay dentro de ella:



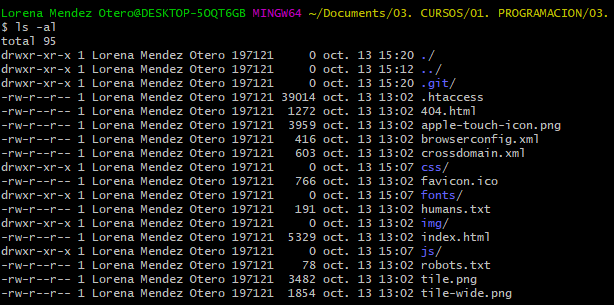
### **Agregar control de fuente Git al proyecto**

Para agregar control de fuente Git al proyecto, utilizamos el COMANDO “**GIT INIT**”:



Por tanto, con “INIT” Git inicializará un nuevo repositorio, usando la carpeta actual como el directorio de trabajo actual.

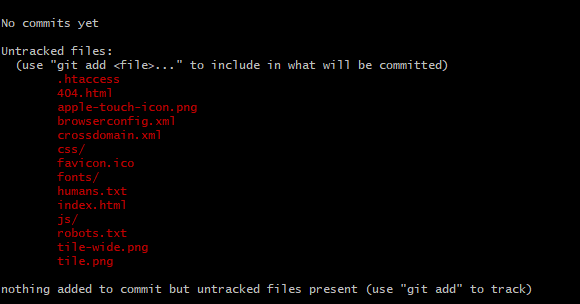
Ahora si hago un "ls -al", puedo ver que tengo mi ". git ", que es donde realmente vive el repositorio de Git, junto con los otros archivos que forman parte de este proyecto:



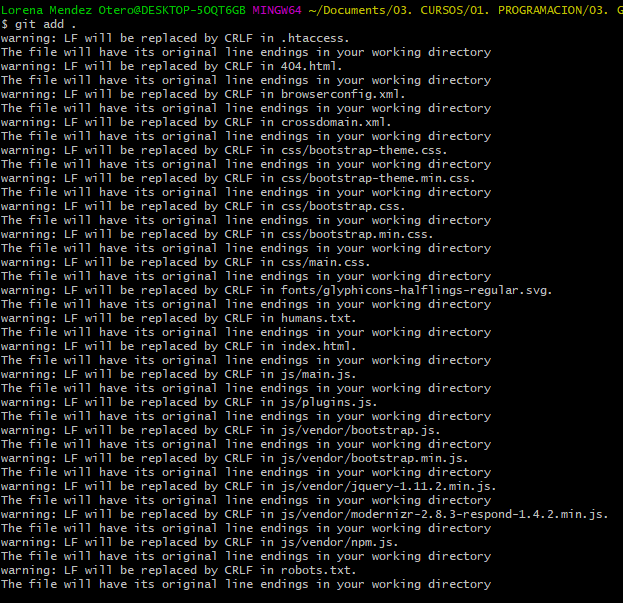
Ahora, si hago un "estado de git", Git me recordará que estoy en la rama maestra, que es la rama predeterminada que obtienes cuando creas un nuevo repositorio de Git.



Dado que aún no hemos realizado ninguna confirmación, estamos acumulando nuestra confirmación inicial, y luego Git nota un montón de archivos que aún no está rastreando:

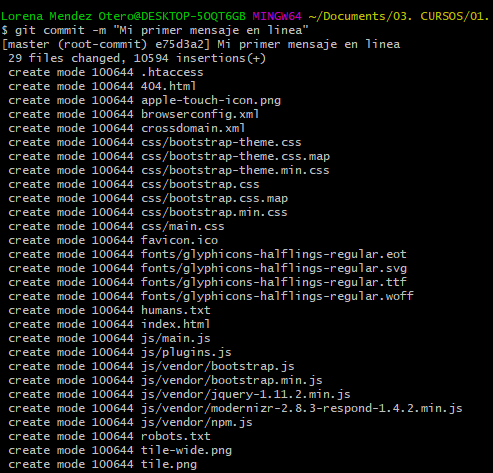


Entonces, para agregar todos estos archivos a la vez al área de preparación de Git y al índice de Git, vamos a escribir "git add" espacio y un punto:

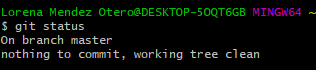


Ahora hay muchos más archivos de los que teníamos antes, y eso se debe a que el período agregará recursivamente todos los archivos que forman parte de este proyecto.

Añadiremos el mensaje de comprobación usando el comando “**GIT COMMIT -M**” seguido del mensaje entre comillas “Mi primera confirmación en línea”:

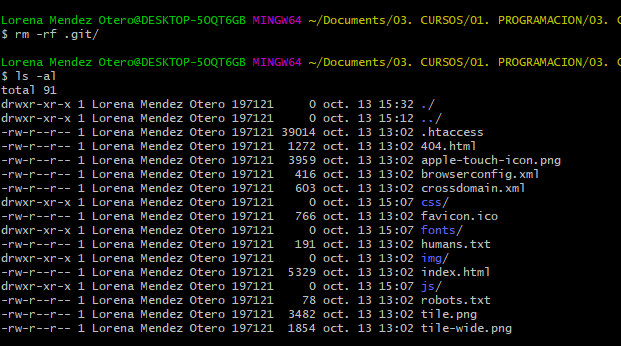


Utilizamos “Git Status” para chequear que el commit se ha hecho correctamente y el directorio de trabajo está limpio:



Para eliminar una carpeta/archivo, añadimos el comando “**RM -RF**” seguido del nombre de la carpeta/archivo que quiero eliminar.

Luego, haremos una comprobación con el comando “LS -AL” y vemos que ya no existe la carpeta .git/:

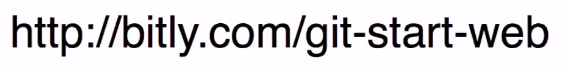


Eliminación de la carpeta de proyecto web-project utilizando el comando “**RM -RF**”:



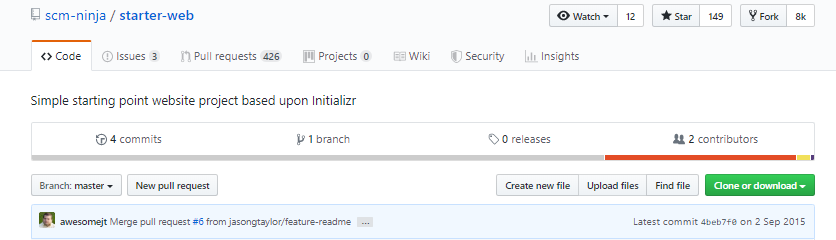
## Starting on GitHub by joining an existing project (git clone)

Vamos a demostrar cómo unirnos a un Proyecto de GitHub ya existente.



### **Copiar el proyecto en GitHub**

Primero, tenemos que asegurarnos de que nos hemos unido a nuestra cuenta de GitHub y luego entrar en el repositorio de **scm-ninja / starter-web**:



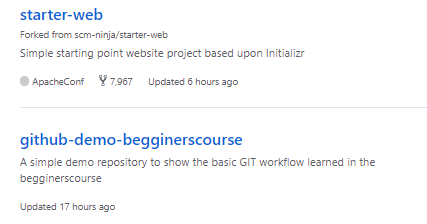
A continuación, clicamos en el botón de Fork (también conocido como "Bifurcación") para crear una copia del repositorio en nuestro espacio personal:



Ahora tenemos nuestra propia copia personal de ese repositorio para trabajar:



Resultado en GitHub:

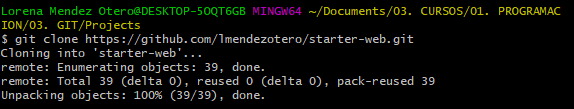


### **Clonar el repositorio de GitHub en Local**

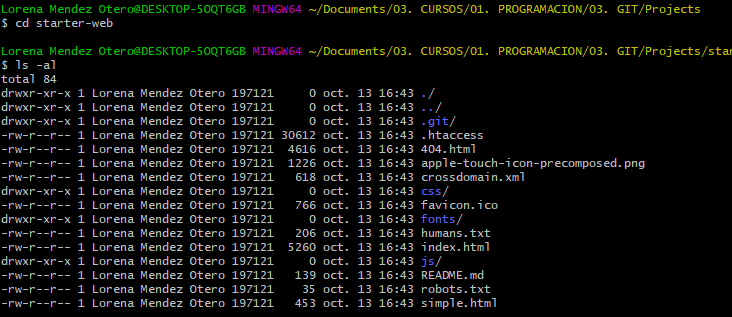
Luego, necesitamos clonar el repositorio a nuestro sistema local.

Así que vaya a nuestras opciones de clonación y asegúrese de que esté seleccionado "HTTPS" y, si tiene este botón aquí, "Copiar al portapapeles", puede usarlo, o simplemente puede copiar la url que está en este texto campo.

Luego, voy a "proyectos de CD" porque es la ubicación en la que clonaré mi repositorio Git. Para hacerlo, uso el comando “GIT CLONE”, espacio, y luego pegue la URL que acabamos de copiar de la página web de GitHub:



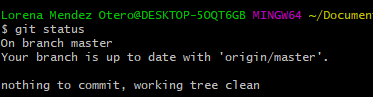
Veamos todos los ficheros dentro del repositorio clonado:



En la parte superior, notarás un ". carpeta git "; ese es el repositorio real de Git.

Hacemos un “GIT STATUS” y nos encontramos con que estamos en la rama maestra; master es la rama predeterminada que obtienes con un repositorio de Git. Y, dado que clonamos nuestro repositorio Git desde una ubicación remota, la siguiente línea nos dice que nuestra rama está "actualizada con 'origin / master'. ".

El nombre "origen" es una referencia a nuestro repositorio GitHub, es nuestra referencia remota, y "/ master", es la rama maestra en ese repositorio remoto.



Finalmente, la última línea nos dice que no hay nada que comprometer, porque todavía no hemos hecho nada y que nuestro directorio de trabajo está limpio, lo cual, nuevamente, tiene sentido.

## Basic Git Workflow (add, commit, pull & push)

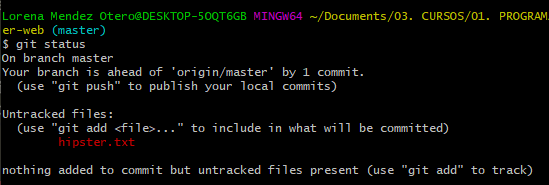
Vamos a demostrar cómo agregar nuevos archivos a un repositorio de Git. Utilizaremos las siguientes URLs:



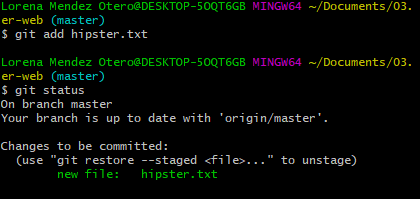
Las dos últimas URLs son para conseguir texto de relleno.

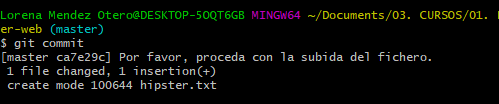
Vamos a agregar un fichero nuevo utilizando el COMANDO “ECO” llamada hipster.txt.

Al iniciar el comando GIT STATUS, Git nos dice que hay un fichero (hipster.txt) pendiente de seguimiento y dice como incorporarlo para Git pueda rastrearlo:

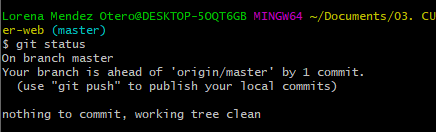


Por lo tanto, utilizaremos el COMANDO “***GIT ADD***” para lanzar el fichero al Stagin Area y luego añadimos el commit con el COMANDO “***GIT COMMIT***”:

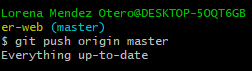




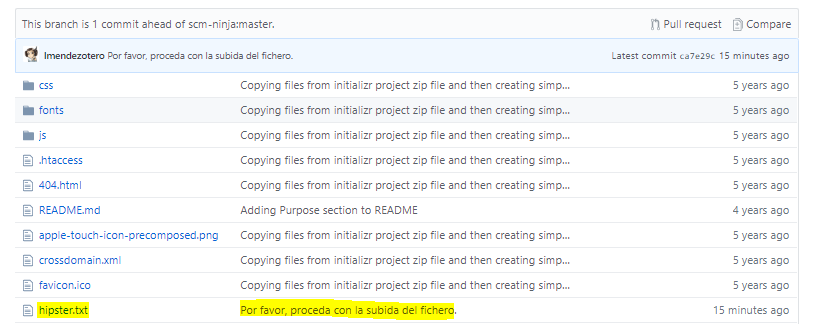
Si hacemos otro GIT STATUS, Git nos dirá que estamos por delante de "origen / maestro" por una confirmación. También nos dice que no tenemos nada que comprometer y que nuestro directorio de trabajo está limpio.



El comando commit es una operación local, ya que no ha impacto en el repositorio en remoto (GitHub). Entonces, para lanzar mi fichero de local a remoto tengo que utilizar el comando GIT PUSH. Por tanto, insertando "git push origin master" logro empujar cualquier confirmación desde el repositorio local al repositorio remoto por nombre de referencia, así como la rama que desea empujar.



Ver resultado en remoto:



Ahora, debería explicar qué significan "origen" y "maestro". Por una parte, "origen" es el nombre del repositorio remoto del que clonamos: de ahí que se llame origen. Por otra parte, “maestro” es nuestra rama maestra, es la única rama que tenemos en este momento.

*La operación inversa a GIT PUSH es GIT PULL*. Utilizando el comando "***git pull origin master***", esto actualizará nuestro repositorio con cualquier cambio que pueda haber ocurrido en el repositorio remoto hasta nuestro repositorio local, solo para asegurarnos de que estamos actualizados antes de hacer cualquier envío copia de seguridad en nuestro repositorio remoto.

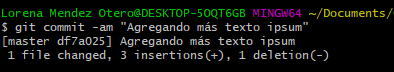
## Tracked Files

He añadido un cambio en el fichero de hipster.txt añadiendo texto adicional de la web:

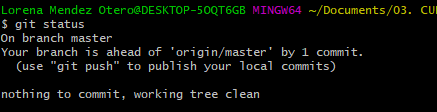
<https://hipsum.co/>



Por ahora, digamos que estoy contento con estos cambios y que solo quiero confirmarlos directamente. Así que realmente puedo atajar el proceso de agregar y confirmar en un solo paso escribiendo "**git commit -am**" (**-a** para agregar el escenario, y luego **"m"** para mi mensaje de confirmación. Por tanto, entonces estoy haciendo un mensaje de confirmación en línea, "Agregando más texto ipsum".

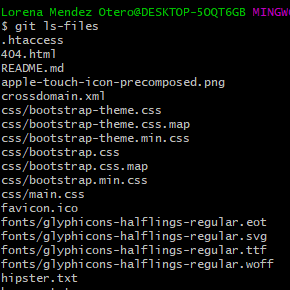


Añadimos GIT STATUS para ver el estado del fichero modificado:

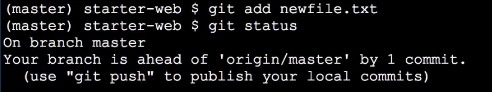


*“Git commit -am” solo se puede hacer con archivos que se están rastreando*. =>> **¿Qué es un archivo rastreado?** Un archivo rastreado es cualquier archivo que Git conoce y rastrea activamente. Ese sería cualquier archivo que ya se haya confirmado en el repositorio de Git, o cualquier archivo que se haya agregado al índice de Git o al área de preparación de Git.

Una forma de averiguar si Git está rastreando su archivo, es usar el comando "***git ls-files***", que me da una lista de todos los archivos que Git está rastreando en el repositorio actual.

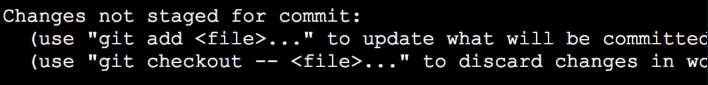


*En caso de que un archivo nuevo “txt " no estuviese en la lista*, podemos añadirlo al listado de ficheros rastreados utilizando el comando "***git add newfile***". txt " y a continuación "git status” nos encontramos con el nuevo fichero en la lista.



## Editing Files

Cuando estamos haciendo una edición muy simple y directa a un archivo existente, uno de los puntos principales del área de preparación es que desarrollamos nuestros cambios para el compromiso; es decir, solo queremos usar el parámetro "-a" en el comando "commit" cuando no queremos agregar más cambios a ese commit.



Cuando tenemos un cambio que puede abarcar varios archivos, definitivamente queremos usar el área de preparación. Entonces, si hago un "estado de git", nuestro cambio se ha movido al área de preparación, y modifiquemos el archivo una vez más.

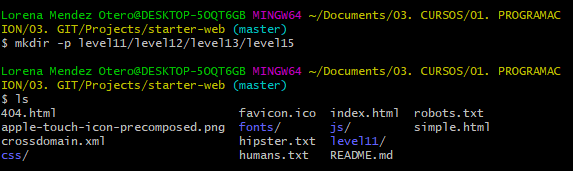
Git puede rastrear la diferencia entre los cambios que se deben confirmar, es decir, los que están en escena, y los cambios que aún no se han montado. Así que ahora necesitamos agregar esas modificaciones: "git add".

Entonces, puede que se pregunte por qué estamos usando el comando "git add" para un archivo existente. Me gusta pensar en ello como "En realidad estoy agregando los cambios que han ocurrido" de nuevo en el índice de Git, o de nuevo en el área de preparación. Si hago eso, "estado de git", ahora mis dos cambios están ahora en el área de preparación. Entonces "git commit -m" Más ipsum para hipsters "", presiona enter. Ahora hagamos un "estado de git"; Bien, volvemos a un directorio de trabajo limpio.

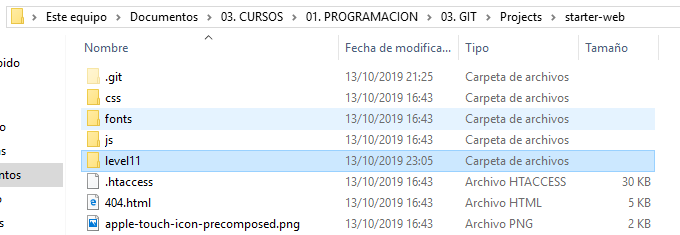
## Recursive Add

En este video, le mostraré cómo agregar archivos de manera recursiva (crear carpetas dentro de otra carpeta) a su repositorio Git.

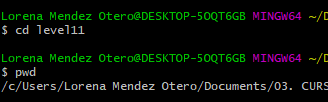
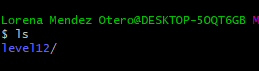
Quiero crear una estructura de carpetas de anidamiento profundo dentro de mi repositorio Git. Así que ahora voy a usar el COMANDO ***"MKDIR" con el parámetro "-p"*** para crear esta estructura de carpetas. Si hago un "ls", puede ver que tengo mi directorio de primer nivel, que es "level1".



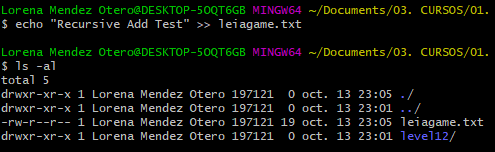
Visualización del resultado en Local:



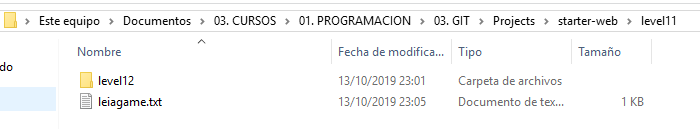
Voy a entrar el level11.

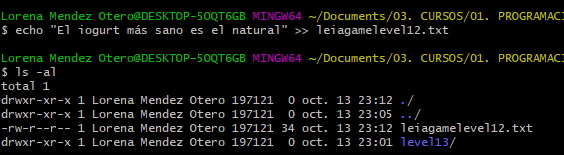
Creemos un archivo aquí denominado "leiagame. TXT". Haga un "ls", puede ver ese archivo aquí y el directorio "level12", voy a subir al directorio "level2" y voy a crear el siguiente archivo de nivel.



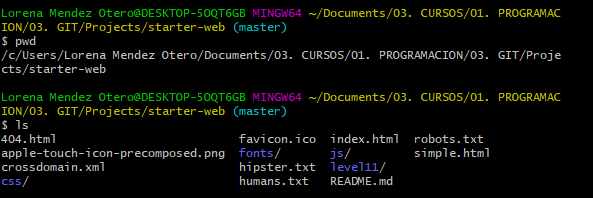
Visualización del resultado en Local:



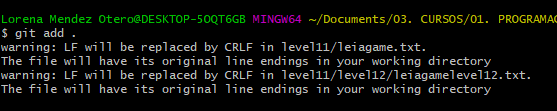
Creamos otro archivo en el level12 denominado “leigagamelevel12.txt”:



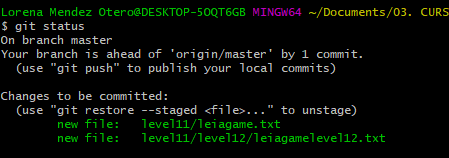
Retrocedemos a nuestra raíz "web de inicio" del repositorio de Git. Entonces hagamos un "ls"; y n "estado de git". Entonces, el comando "git status" solo le mostrará el directorio de primer nivel; No le mostrará todos los archivos recursivamente hacia abajo.



Sin embargo, definitivamente puedo agregar todos los archivos de forma recursiva, simplemente usando un parámetro adicional en el "git add". Entonces escriba "git add", espacio, y si uso un punto, agregará todos los archivos de forma recursiva a partir de ahora.



Ahora si hago un "estado de git", puedes ver ese "nivel1 / nivel1-archivo". txt "," nivel1 / nivel2 / nivel2-archivo. txt ", y así sucesivamente se han agregado al índice Git:

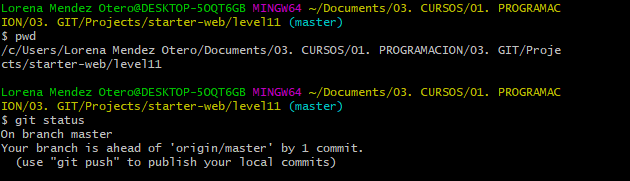


Ahora, si quisiera, simplemente puedo cometer esos cambios. Y solo voy a usar mi editor de texto para hacer el mensaje de confirmación "Agregar varios archivos de forma recursiva". Voy a guardar y cerrar, y el comando "commit" regresa, mostrando que cada uno de esos archivos se ha agregado recursivamente al repositorio de Git.

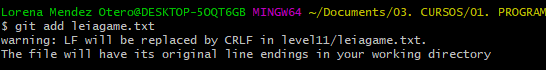
## Backing Out Changes

Mostraremos cómo anular los cambios en su repositorio de Git. Dentro del level11 de mi repositorio starter-web tengo un archivo que lo voy a modificar añadiendo texto.

Guardo y luego cierro. Voy de vuelta en mi terminal hago un "estado de git", tenemos nuestro archivo modificado.

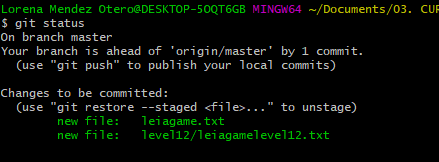


Avancemos y agreguemos esta versión del archivo al área de preparación de Git: entonces "git add level1-file". txt ", presione enter.

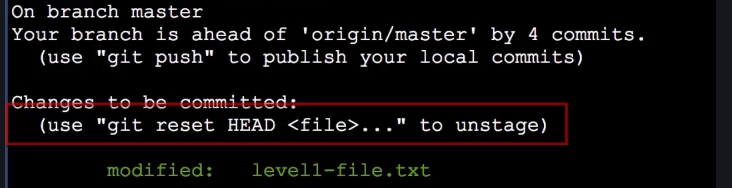


Entonces, una cosa a tener en cuenta es que puede emitir la mayoría de los comandos en su repositorio Git desde cualquier lugar dentro de su repositorio Git.

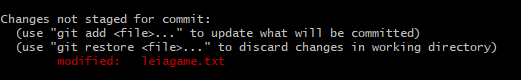
Bien, ahora si hacemos un "estado de git", tenemos nuestro archivo modificado. Nuevamente, si abro ese archivo nuevamente. De vuelta en la terminal, digamos que decido que realmente no quiero esos cambios. Lo que puedo hacer es seguir las instrucciones que se encuentran en el comando "git status".



Entonces, lo que quiero hacer es retroceder los cambios desde el área de preparación, o, sin escena. Para hacer eso, voy a usar el comando reset de Git: "git reset HEAD", y luego el nombre del archivo "leiagame". TXT".



También podemos utilizar “git restore <file>” para descartar los cambios en el área de trabajo:

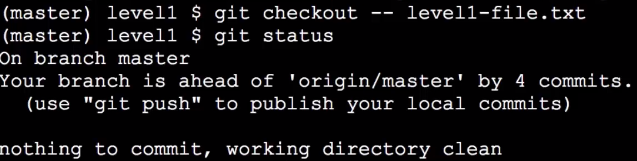


**NO LOGRO CONSEGUIR LO MISMO QUE EN EL VIDEO!**

Entonces, ese comando deshizo el archivo; entonces, si hago un "estado de git", ese archivo ha regresado a los cambios para que no se confirmen, que es solo el estado del directorio de trabajo. Si abro ese archivo, puede ver que este archivo no ha cambiado; todavía está en mi directorio de trabajo como los tres párrafos.

Entonces, digamos que no quiero ninguno de esos cambios para este archivo. Puedo descartar los cambios en mi directorio de trabajo actual usando el comando "***git checkout***". Así que voy a seguir las instrucciones del comando "git status", así que "git checkout"; y luego dos guiones; y luego el nombre del archivo, entonces "nivel1-archivo. TXT".

Si hago eso, y ahora hago un "estado git", volveremos a un directorio de trabajo limpio. Además, si vuelvo a abrir ese archivo, puede ver que volvemos al estado del archivo que solo tiene dos párrafos; así que este es el mismo estado del archivo que se confirmó por última vez.



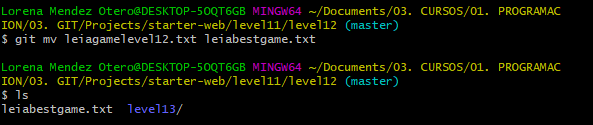
**NO LOGRO CONSEGUIRLO!**

## Renaming and Moving Files

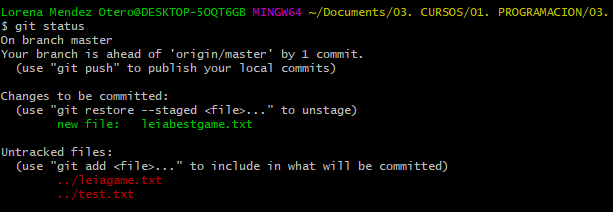
Veremos cómo mover y cambiar el nombre de los archivos en Git a través de comandos.

### **Cambio de nombre de archivos**

Para cambiar el nombre de un archivo utilizamos el comando “GIT MV” + espacio + nombre actual del fichero + nuevo nombre del fichero:



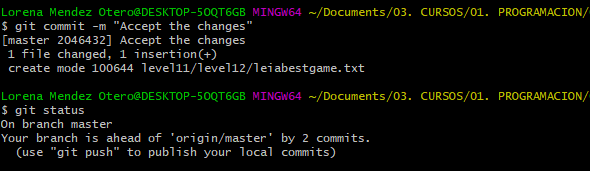
Si hago un "GIT Status", Git ha puesto en escena el hecho de que se ha producido el cambio de nombre:



Git ha puesto en escena el hecho de que se ha producido el cambio de nombre, pero no lo ha cometido. Eso significa que podría retroceder el cambio de nombre.

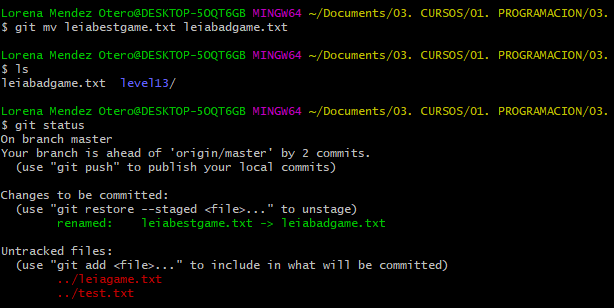
Vamos a añadir “git add” + “git commit -m” para aceptar los cambios:





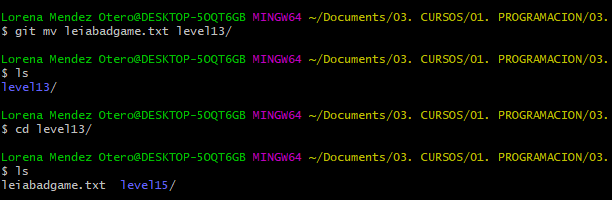
**Recomendación**: Cambiar el nombre de sus archivos antes de realizar cambios; esto facilitará a Git rastrear el hecho de que ha ocurrido el cambio de nombre.

¿Cómo retroceder al antiguo nombre de un archivo? => Utilice el comando "git mv". o bien "git mv" + actual nombre del archivo + nuevo nombre de archivo de vuelta al nombre de archivo anterior, presione Enter:



### **Movimientos de archivos**

Para mover un archivo de un directorio a otro, utilizaremos el comando “git mv” + “nombre del archivo a mover” + “carpeta de destino” para lograrlo:



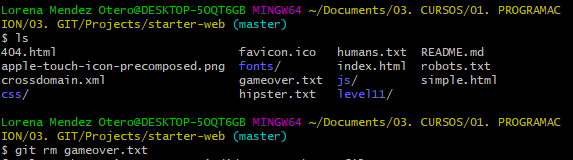
¿Cómo movemos los archivos para hacía atrás (a un nivel inferior)? => haciendo la tarea inversa con el comando “git mv”.

## Deleting Files

Demostraremos cómo eliminar archivos en Git de diversas formas.

### **Eliminar archivos NO rastreados por GiT**

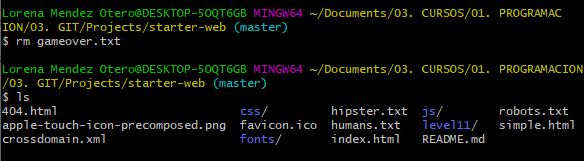
La primera será eliminar un archivo que Git aún no está rastreando, utilizando el comando para eliminar archivos “***git rm***” + nombre del archivo a eliminar:



Git se quejará de que no puede coincidir con el archivo, porque Git aún no está rastreando este archivo:

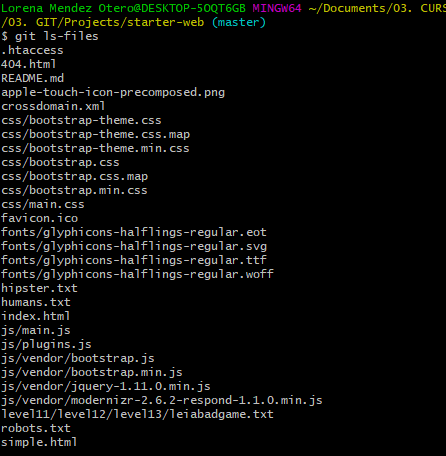


Entonces, eliminamos el archivo del sistema operativo con el comando “***rm***” + nombre del archivo a eliminar:



### **Eliminar archivos SÍ rastreados por GiT**

Para mostrar todos los archivos que Git está rastreando, introduzco el comando “***git ls-files***”:



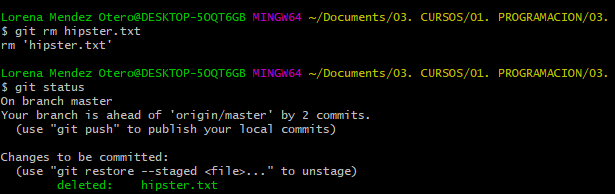
¿Cómo hacer que Git rastree un fichero? =>

Por tanto, para *eliminar un archivo que git está rastreando es incorporar “****git rm****” + nombre del archivo a eliminar*:

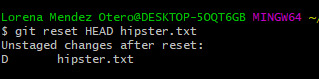
La eliminación aún no es permanente, necesitamos avanzar comprometiendo el cambio que se organiza. Para ello, introducimos un “git commit -m", para el mensaje en línea "Eliminar nuevo archivo".

### **Anular eliminaciones por etapas**

Vamos a eliminar el fichero hipster.txt => Vamos a añadir *“****git rm****” + nombre del archivo a eliminar*:



El estado de git muestra que la eliminación está actualmente organizada. Nuevamente, eso significa que el archivo no se ha eliminado permanentemente, al menos en lo que respecta a Git. Por tanto, para eliminarlo definitivamente, incorporamos un “***git reset HEAD***" + nombre del archivo:

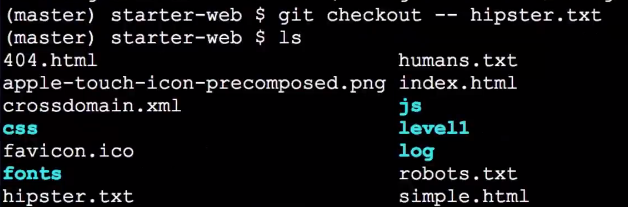


Sin embargo, si hacemos un "ls", el "hipster". txt "todavía no está en nuestro directorio de trabajo.



Entonces, si verificamos nuevamente con Git haciendo un "estado de git", entonces aquí confirmamos que Git está detectando que hay un archivo eliminado, eso es "hipster". txt ", actualmente se está rastreando pero falta en el directorio de trabajo.

Por lo tanto, el comando "reset HEAD" que especifica el archivo, simplemente eliminó la eliminación, no restauró el archivo al sistema de archivos. Por tanto, debemos descartar cualquier cambio temporal que hayamos realizado en el directorio de trabajo utilizando el comando "***git checkout***":



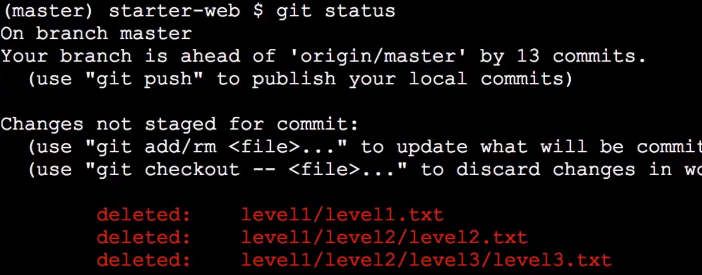
Vamos a confirmarlo => "ls", ahora podemos ver que el "hipster". txt "está de vuelta en nuestro directorio.

### **Eliminación de una estructura de carpetas**

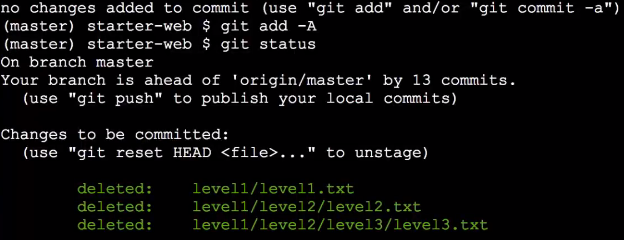
Para eliminar toda la estructura de carpetas utilizaremos el comando ***"rm -rf***", espacio, y luego el nombre de la carpeta que deseo eliminar. El "-r" recorrerá recursivamente la estructura de la carpeta mientras que el "-f" forzará la eliminación.



Al hacer un "ls" confirma que la carpeta "level1" y todos sus elementos secundarios ya no están. Si hago un "estado de git", Git observa todos los archivos que hemos eliminado que ha estado rastreando, básicamente todos los archivos de texto desde "nivel1" hacia abajo.



Entonces, para organizar la eliminación de todos esos archivos a la vez, voy a emitir "git add -A". Posteriormente, si hacemos un "estado de git", la eliminación de todos esos archivos se ha organizado.



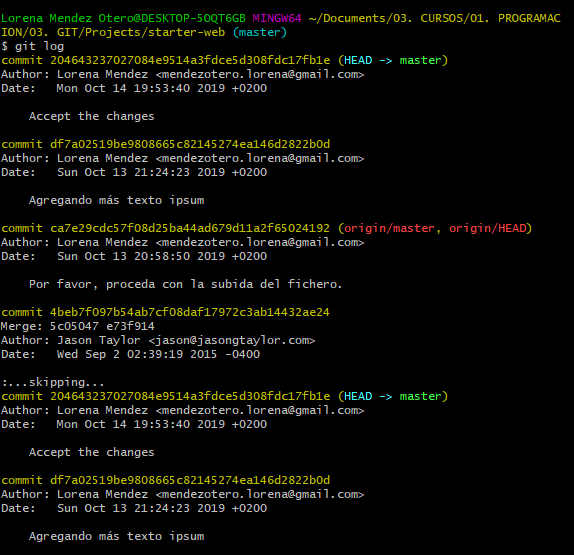
Muy bien, sigamos adelante y vamos a confirmar esto: "git commit -m", con el mensaje de confirmación: "eliminando level1 y todos los elementos secundarios".

## History

Mostraremos cómo ver la historia de Git, utilizando el comando “***git log***”.

Comando de ayuda de Git: “git help

Así que sigamos adelante y veamos qué nos da el comando predeterminado "git log": "git log". Entonces, puede ver que tenemos el historial de confirmación en orden cronológico inverso:



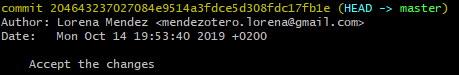
La parte superior comienza con la última confirmación, y luego trabajamos hacia atrás en

el tiempo a medida que bajamos.

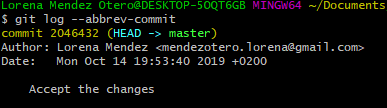
El identificador SHA-1 es como Git identifica cada uno de los commits de manera única:



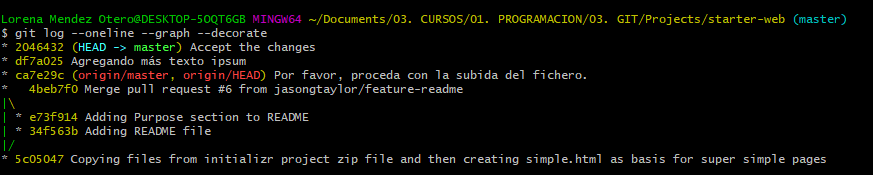
Seguido por el nombre y la dirección de correo electrónico del autor, la fecha en que ocurrió la confirmación y luego, sangrado un poco, es el mensaje de confirmación real:



Para proporcionar un historial de los commitde forma abreviada inserte el comando "***git log --abbrev-commit***" en Git Bash:

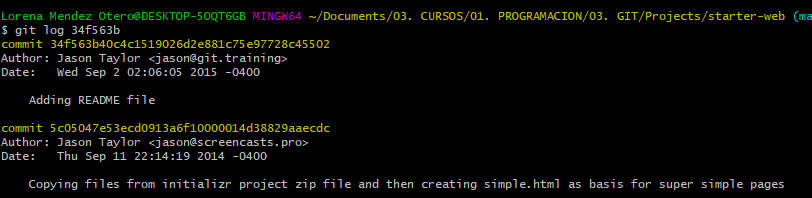


Para ver el historial de una forma más visual y resumida utilizo el comando "***git log --oneline --graph --decorate***" en Git Bash:



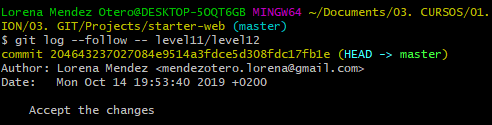
Por tanto, "--oneline" comprimirá nuestras entradas en una línea, "--graph" proporcionará un gráfico ASCII que representa el gráfico de ramificación, "--decorate" agregará cualquier etiqueta o etiqueta o cualquier cosa que anote nuestros compromisos. De esta forma tengo una vista muy diferente de mi historial de confirmaciones. En la parte superior, tenemos la rama maestra que actualmente apunta a la última confirmación.

Es posible hacer búsquedas en git por número de commit introduciendo el comando **“*git log” + numero del commit***:

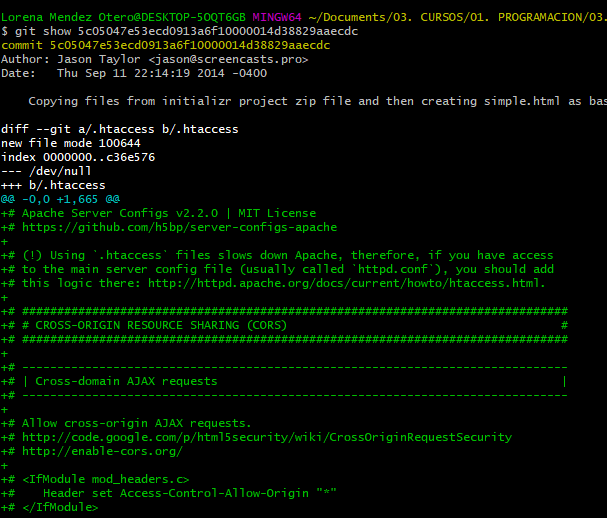


Otra opción que podría usar con el comando "git log" es la búsqueda basada en fechas utilizando el comanda "git log --since =" + entre comillas dobles "hace 3 días".

Para ver el historial de archivos renombrados utilizaremos el comando "***git log --follow***", eso seguirá los cambios de nombre; "-"; espacio; y luego la ruta al archivo:



Para mostrar información de un especifico commit, utilizamos el comando "***git show***" + referencia del commit:



## Git Alias

Vamos a demostrar cómo configurar alias dentro de Git (comandos más cortos que nos dan la misma información que un comando largo de Git).

Voy a crear mi propio comando "hist", utilizando el comando "config" de Git. Por tanto, escribo "git config --global", porque queremos que este alias esté disponible independientemente de mi repositorio, de modo que se guardará a nivel de usuario; espacio + "alias". "; y luego el nombre del comando.



Por tanto, cualquier configuración que comience con" alias. ", nos permitirá crear un alias con el nombre del comando; espacio; así, entre comillas, pondré el comando completo que quiero emitir, dejando fuera la parte" git "del comando.

## Ignoring unwanted files and folders

Demostraremos cómo excluir archivos no deseados de nuestro repositorio de Git.

La facilidad que utiliza Git es un archivo de texto llamado **". gitignore "** que Git usa para rastrear todos los archivos y carpetas que Git debería ignorar. Debemos crear un archivo nuevo denominado “. gitignore” en nuestro repositorio con el formato una carpeta "\*. TXT".

El archivo gitignore es un archivo de texto como cualquier otro archivo, y debe agregarse a Git para el control de versiones, así que hagamos eso ahora:" git add. gitignore ". Hagamos un "estado de git" y luego un "git commit”.

## Cleanup and Back to Origin (GitHub)

Cubriremos los comandos básicos de Git sincronizando nuestros cambios con GitHub. Para ello, voy a mi directorio local e introduzco el comando "git pull origin master".

GitHub => Git: GIT PULL ORIGIN MASTER

Git => GitHub: GIT PUSH ORIGIN MASTER

# Annex

## Basic Linux Commands for Beginners

<https://maker.pro/linux/tutorial/basic-linux-commands-for-beginners>