**\_notas de Git y Gitub (20230403)**

Cualquier error que detecte por favor avise al alumno Hector Duran en EGG o juliuxar en github.com

**Poner en vista Web (Vista > Diseño Web)**

Basadas en la serie de 5 videos hechos por egg (serie 1):

El ultimo video es un repaso.

<https://www.youtube.com/playlist?list=PLgwlfcqa5h3zLFaZgnPzmkDzLUYuJyzY3>

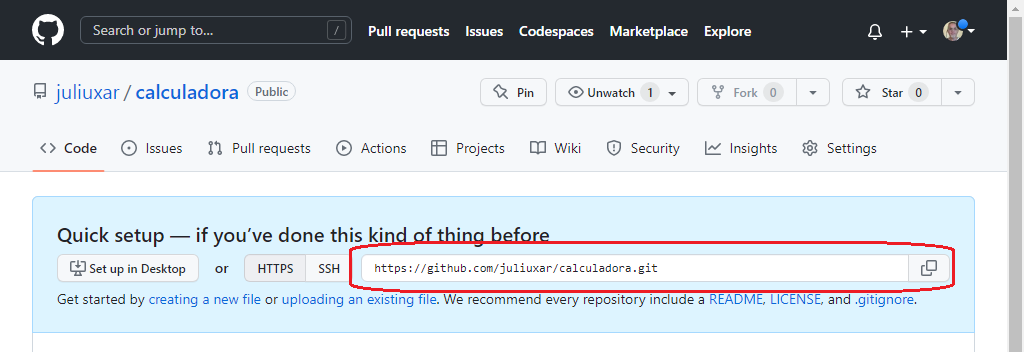
Luego seguir con esta serie de videos, “Moverse entre diferentes commits” (serie 2)

<https://www.youtube.com/playlist?list=PLgwlfcqa5h3xypUOclgVoHegHt9xYL81U>

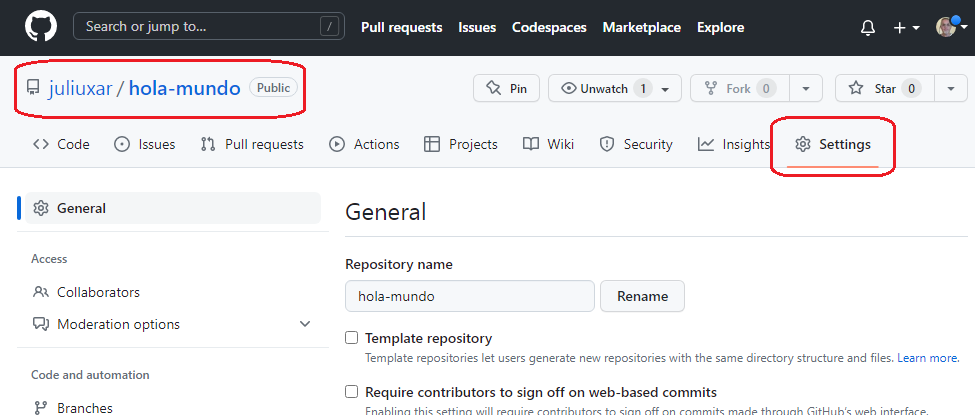
VIDEOS 1, 2 Y 3 de la serie 1

A. Previamente en www.github.com

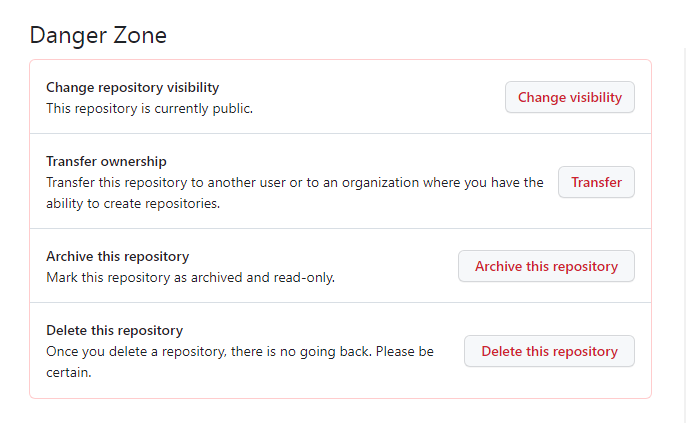
1. Hacerse miembro de github, y crear el primer repositorio con el botón **New**. Supongamos que creamos el repositorio Calculadora porque ya tenemos un proyecto Calculadora.
2. La pantalla mostrara algo asi: tomar nota del URL rodeado de rojo que luego usaremos en la pc.



1. En las practicas es posible que creemos repositorios fallidos y quiéranos borrarlos. Para borrar un repositorio sobre github:
2. Entrar al repositorio, sección Settings. Se vera una pantalla que comienza asi :



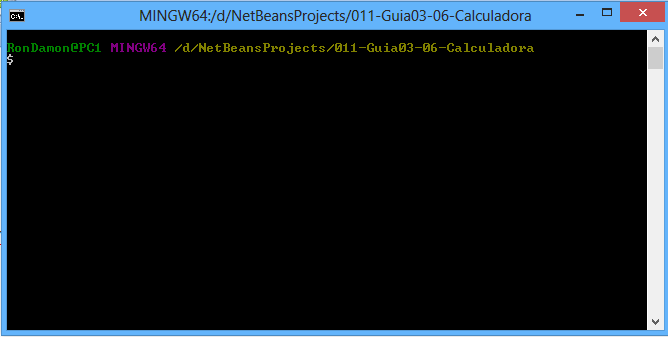
1. Ir hacia abajo, lado derecho hasta hallar “Danger Zone”. Dentro de esa sección encontraremos la subseccion Delete this repository. Pedira que se escriba el <nombre del usuario>/< nombre del repositorio> y apretar el botón que dice “Yo entiendo las consecuencias de esta acción” o algo por el estilo, en ingles. Luego pedirá la clave de acceso a GitHub. Y será borrado.



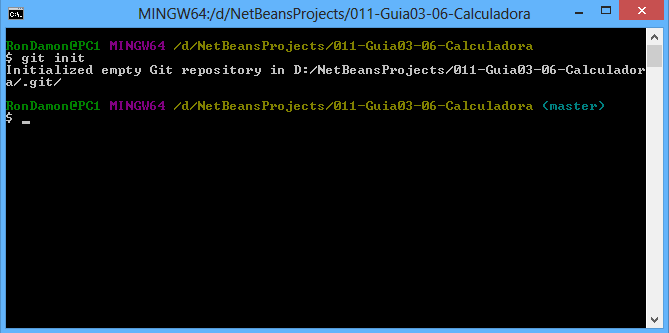
B. En la PC: creacion del primer repositorio local

Advertencia: tipear siempre las ordenes, no copiarlas y pegarlas en bash. Si se copia una orden de un ejemplo de Word, pdf o la web, pueden haber caracteres de formato que molestaran a bash y dara error. Si se insiste en copiar, se puede pegar la frase en notepad, marcarla y copiarla allí y luego pegarla en bash (porque notepad solo maneja texto plano). Pero lo mas seguro es escribirla uno mismo.

1. Abrir el explorador de archivos de Windows e ir hasta la carpeta deseada. Con el mouse marcar la carpeta de un proyecto en java.
2. **git bash here**: Hacer raton derecho dentro de la carpeta y en el menú clickear “Git bash here”. Esto no crea nada, simplemente se entra a la carpeta visualizándosela al estilo consola. Por ejemplo, se abre git bash en la carpeta del proyecto java “Calculadora”. El prompt termina con el nombre del proyecto (Calculadora), en amarillo. No hay ninguna marca que indique que la carpeta este siendo administrada por git, porque no lo esta.



1. **git init**: para crear un nuevo repositorio ejecutar Git Init. Se crea una subcarpeta oculta de nombre .git y varias subcarpetas. Si no se ve, configurar el explorador de Windows para que se vea. El prompt ahora termina en **(master)**. Ahora la carpeta esta en la lista de carpetas de Git, como “recién inicializada para administrar”. Pero nada mas.



Si damos la orden ls o dir, veremos las carpetas y archivos estándar del proyecto java ordenadas alfabeticamente:

build/ build.xml manifest.mf nbproject/ src/ test/

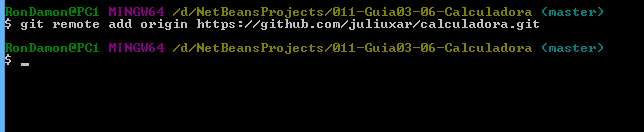
Obviamente las que terminan en “/” son nombres de carpeta. Pero no se vera la carpeta .git que esta oculta.

1. **git remote add origin**: para vincular el recién creado repositorio en la maquina local con el repositorio en github.com ejecutar

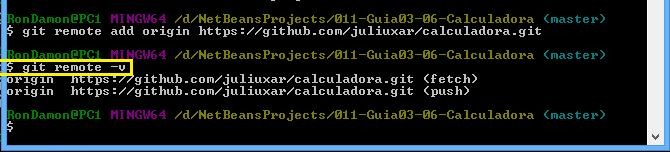
git remote add origin <URL del repositorio>

usando el URL obtenido de github.com en el paso 2.

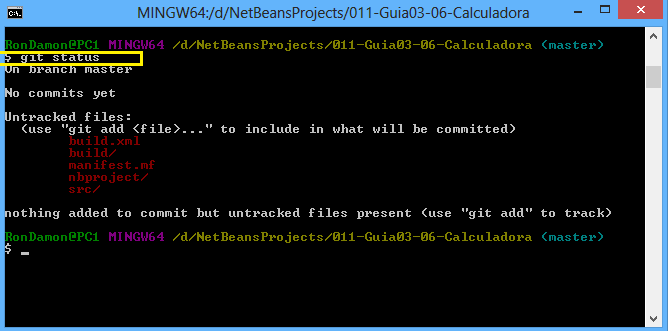
* + - La repuesta fue, ninguna, como se ve debajo.
    - Ver nota de “repositorio recién creado en github”, de allí se saca el 4º parámetro, el URL
    - “origin” se refiere a nuestro repositorio especificado en github
    - Git remote –v: para verificar si se hizo bien ¿?
    - Al final del propt de bash aparecerá entre paréntesis en letra celeste la palabra “master”, indicando que la carpeta local esta vinculada con la carpeta remota dentro de la cuenta de uno en github.



1. Hacer **git remote –v** para verificar que la vinculación fue bien hecha.



1. Se puede usar **git status**: para mostrar alguna información del estado actual del repositorio.



En rojo aparecen carpetas y archivos del proyecto. De las carpetas falta una (test) y los dos archivos son externos a las carpetas.

build.xml

build/

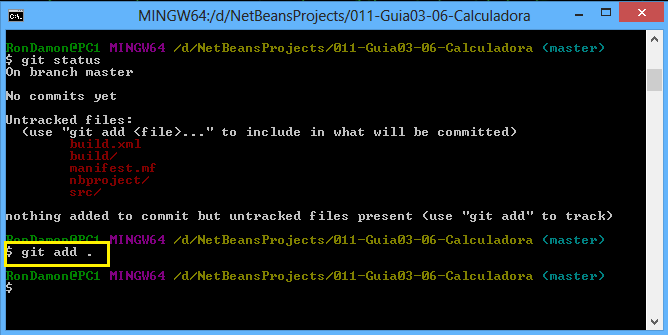
manifest.mf

nbproject/

src/

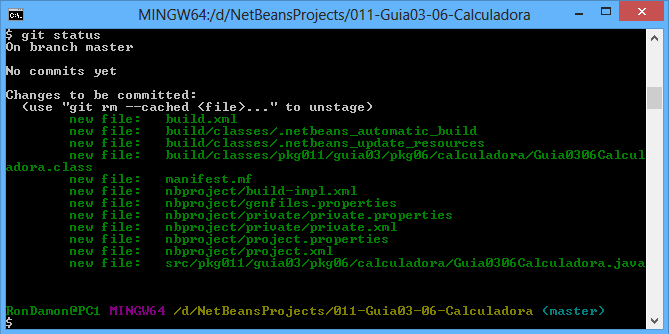
Los archivos marcados en rojo son archivos que no están siendo seguidos por git (todo el proyecto de Java). Para que git se encargue de ellos debe señalárselos con la siguiente orden.

1. **git add .** No confundir con git remote add. El punto solitario del final significa “todos los archivos”. En mi caso la ejecución no entrego ningún mensaje a diferencia del video.



Pueden hacerse especificaciones de exactamente **que** archivos incluir en el seguimiento usando la línea de comandos. Tambien puede escribirse una lista dentro de un archivo de texto de extensión .md, pero escrito en un lenguaje de marcado ligero parecido al HTML llamado MarkDown, de allí la extensión “md”.

Se puede usar **git status** que mostrara algo similar a lo siguiente: todos los archivos agregados listos para hacer el commit (el punto de restauración del proyecto java para poder retroceder si fuera necesario).



1. **git commit**. Sirve para hacer un “punto de restauración”. El formato debe seguir el modelo de abajo.

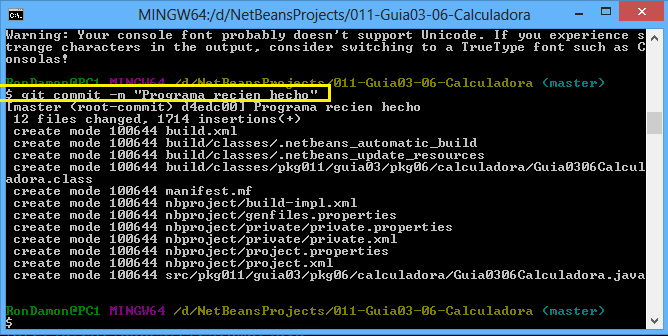
git commit -m "<mensaje de confirmación>"

El mensaje de confirmación es una descripción que caracteriza el punto de restauración.

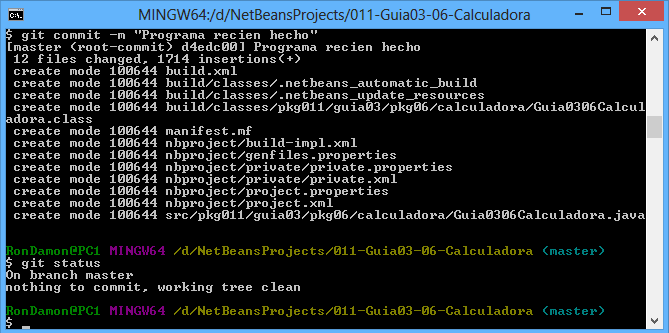
|  |
| --- |
| **Nota**: si se pretende no ingresar el comentario, git insiste y habilita, en el angulo superior izquierdo, un renglón para agregar la descripción omitida, con letra amarilla (en la imagen se muestra la zona de la pantalla recuadrada en amarillo)    Aca se va entrando la descripción faltante…    Una vez entrada la descripción no se da enter, sino que se apreta tecla escape y a continuación se tipea :wq (que aparecerá abajo, a la izquierda) y se le da enter.  :wq significa “write and quit”, es decir, “escriba y quite” (la pantalla de objeccion).    y se vuelve a la pantalla normal de git. |

Por ejemplo, lanzar

git commit –m "Programa recien hecho”



Los archivos mostrados son los incluidos en el commit. Si damos nuevamente la orden **git status** se vera algo asi:



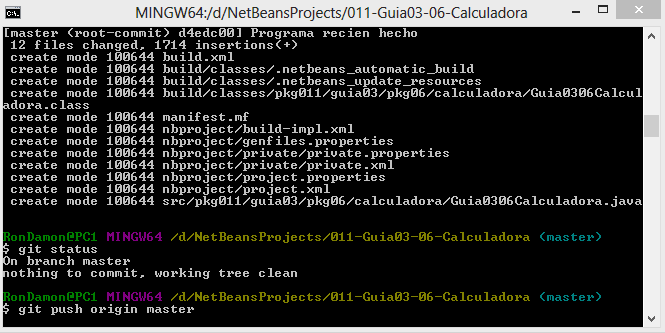
El mensaje “On branch master, nothing to commit, working tree clean” significa algo asi como

“Situados en la rama “master”. Nada para someter a commit, trabajando árbol limpio (sin ramificaciones)”

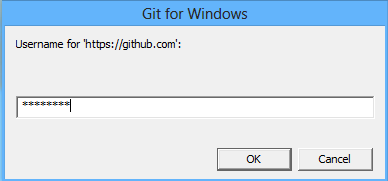
Quiere decir no pudo hacerse commit porque no hubo cambios, recién es la primer versión del proyecto.

Pero eso no impedira que lo mandemos a la nube, al repositorio en github.

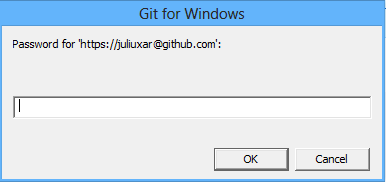
1. **git push origin master** envía el repositorio a github.



En este caso git pregunta el nombre de usuario presentando un pequeño dialogo para escribirlo:

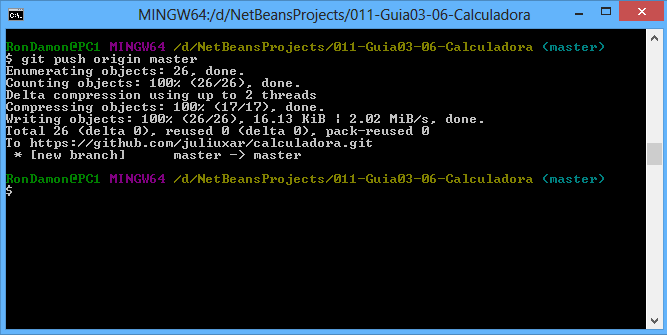


A continuación pide la clave. Pero desde Agosto de 2021 no se ingresa mas la clave, sino el token que ya debimos haber conseguido en su momento.



Seguramente hay maneras mas practicas de entrar usuario y token. Tambien hay una orden para que se guarde usuario y token en la cache hasta el final de la sesión, de modo que se pongan solos.

Una vez entrado el usuario y el token, se hace un procesamiento y muestra algo asi:

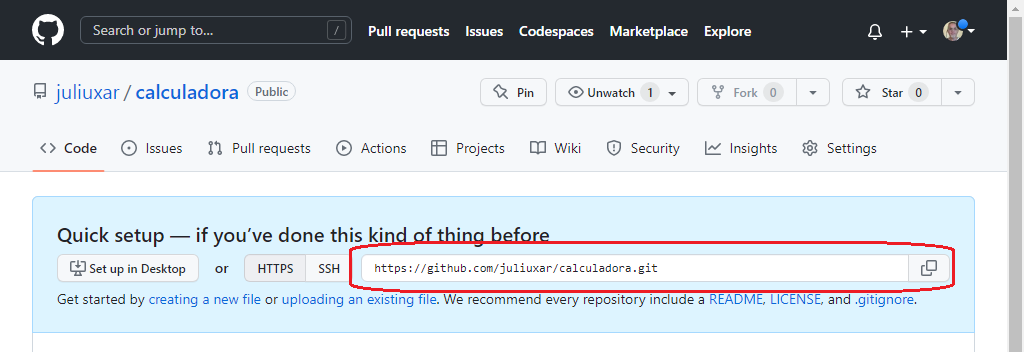


Aparentemente se subieron 26 archivos.

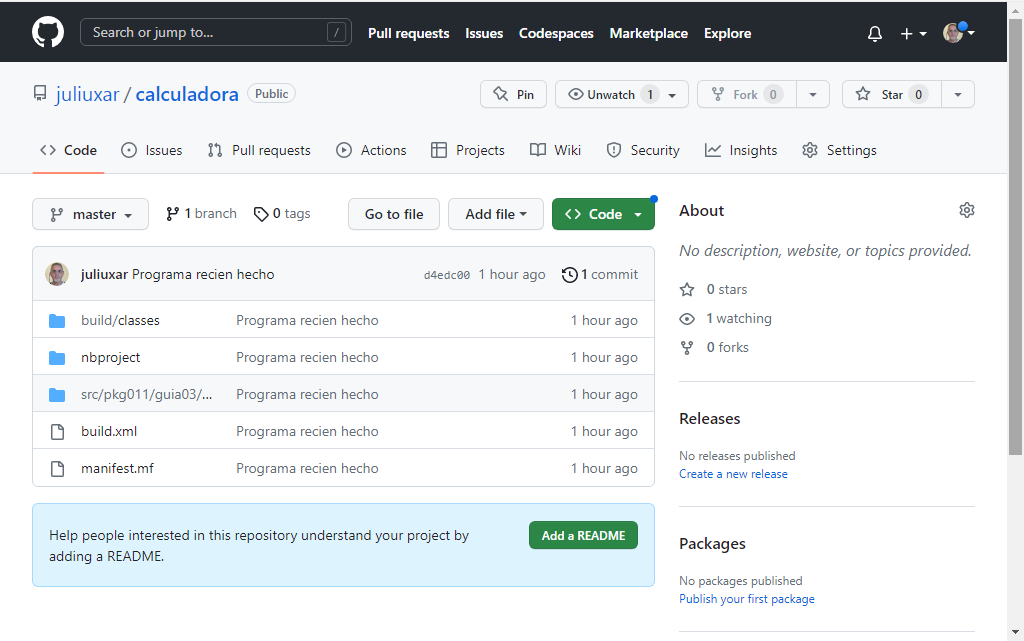
De la rama master en local se hizo una rama master en remoto (master -> master)

Si vamos a github no aparece nada, pero debemos refrescar con ctrl+Shift+R. F5 solo recarga la pagina, ctrl + Shift + R limpia la cache del navegador y recarga la pagina. Es mas completo.

Lo que se veía como (abajo seguían una serie de ordenes de git, como ejemplo para el operador novicio)

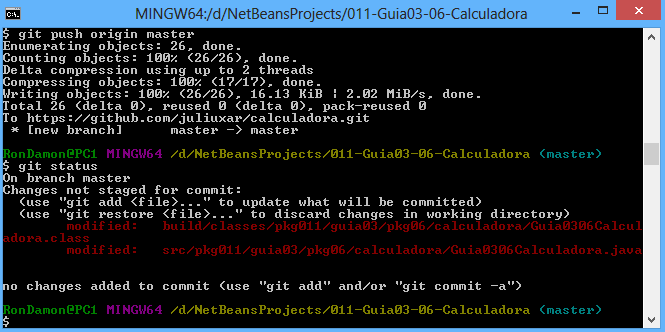


Ahora se ve asi, con carpetas y archivos:



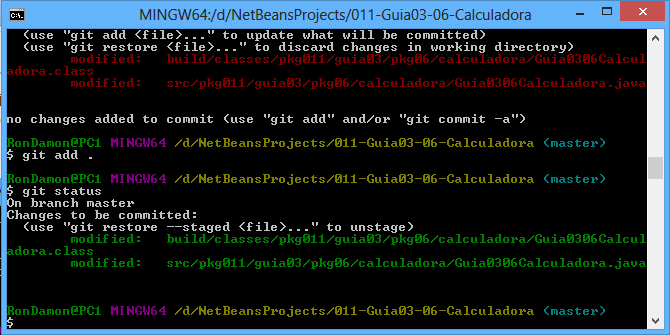
1. Con fines de aprendizaje, entraremos a nuestro proyecto y haremos algún pequeño cambio para ver como reacciona git cuando encuentre diferencias entre el estado inicial del proyecto y el actual.

9.1 - Hacemos un **git status**:



En letras rojas, arriba, se marca lo modificado. Un archivo de clase de extensión .class y otro de extensión .java

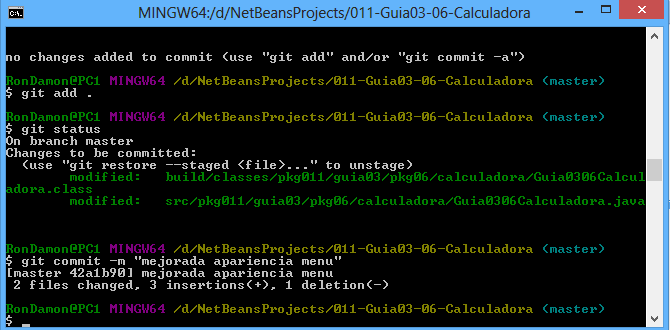
9.2 - Para que git revise lo que se haya modificado damos la orden **git add .** y a continuación (para fines de aprendizaje) un **git status**.



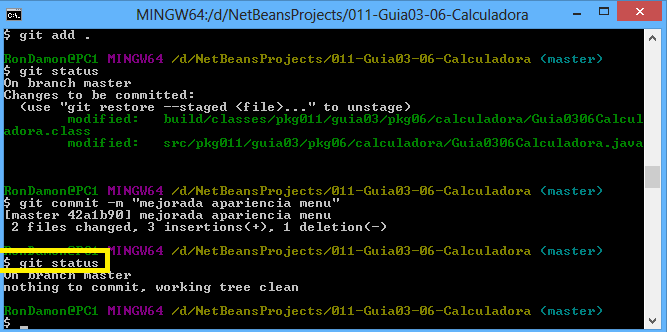
Veremos los dos archivos antes marcados en rojo, ahora están en verde: el **git add .** los hallo modificados y los agrego a la lista de archivos para hacer la próxima “imagen” del proyecto.

9.3 - Y ahora hacemos, de nuevo, un commit, que será

**git commit** –m “mejorada apariencia menu”. Y esta vez no apareció el mensaje “On branch master, nothing to commit, working tree clean” que aparecio en el punto 7 luego de lanzar un git status.



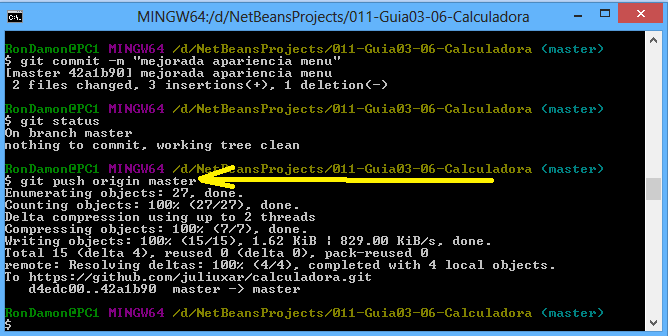
9.4 - Ahora lanzamos otro **git status**.



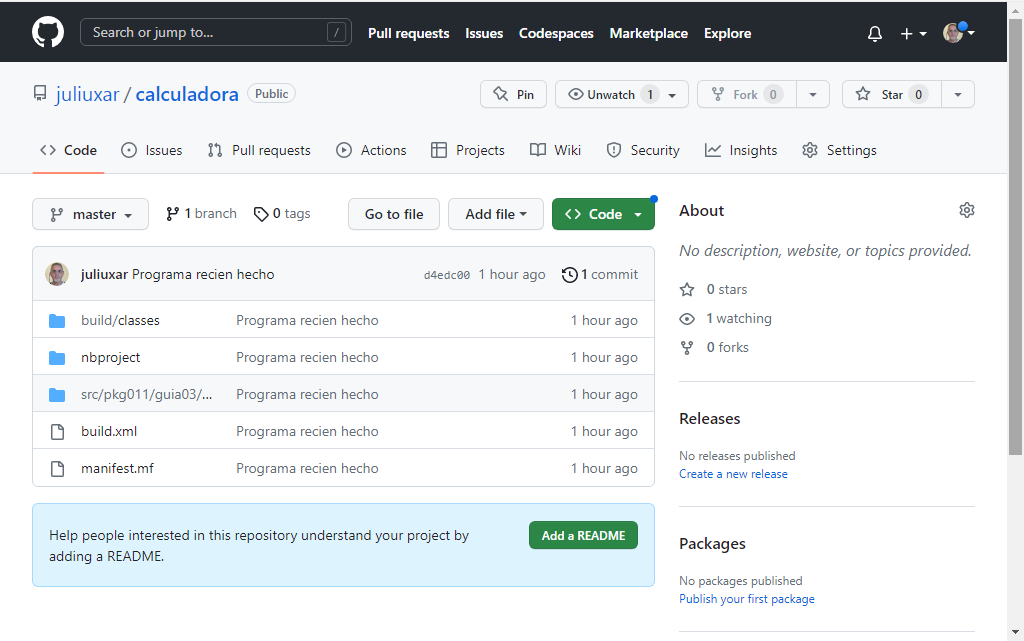
Y nos vuelve a repetir, como en el git status del punto 7, que no hay mas cambios que procesar (On branch master. Nothing to commit, working tree clean), los cambios que hubo fueron procesados por el ultimo **git commit**.

9.5 -Mientras tanto en [www.github.com](http://www.github.com) no llego la noticia del segundo commit (o segundo punto de restauración). Para actualizarlo, lanzamos otro

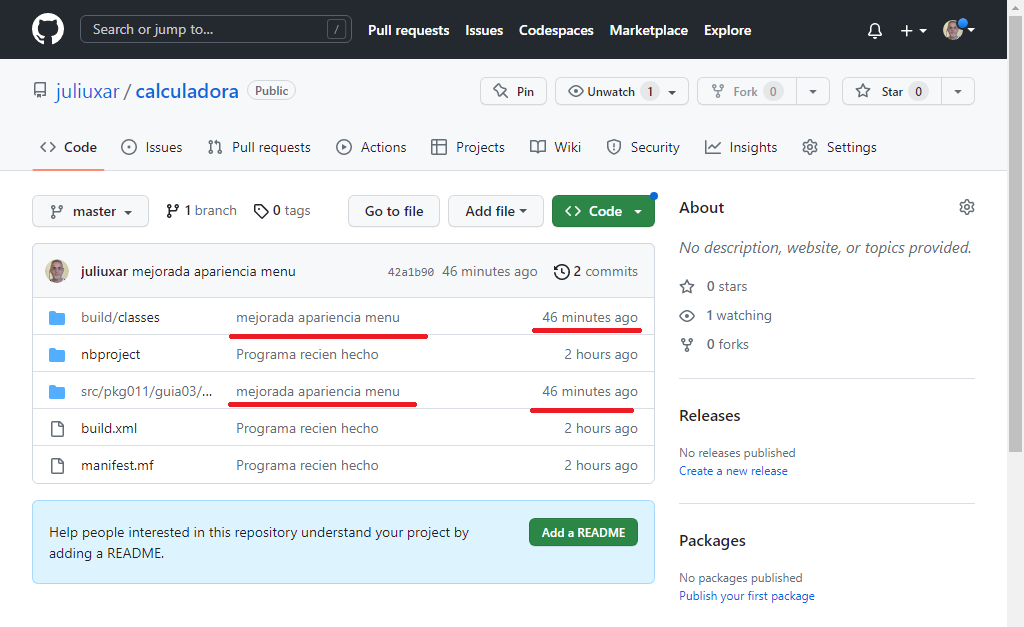
**git push origin master**



9.6 – Vamos a github para ver los cambios. Esta es la vista anterior.

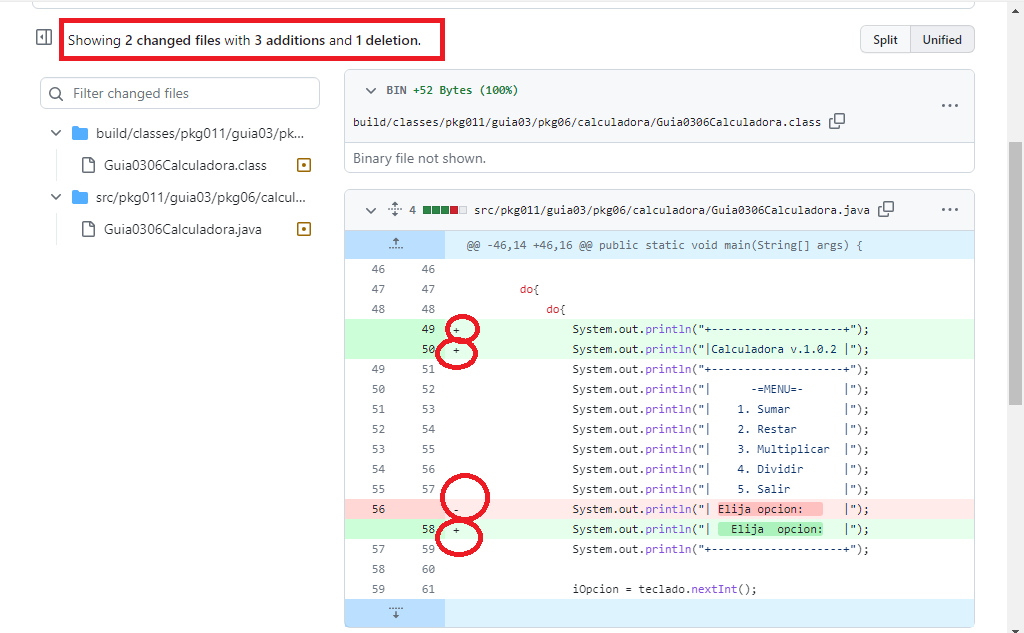


Actualizamos con ctrl + shift + R y esta es la nueva vista



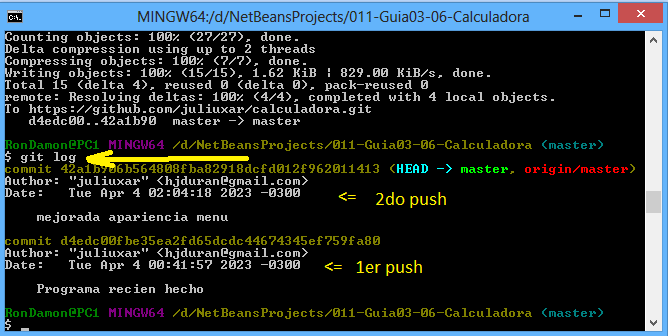
Si se clickea el comentario subrayado con rojo nos conducirá a otra pantalla donde veremos los cambios en el código fuente.

Ver las zonas marcadas. A la derecha hay un renglón rosado que fue quitado (ver el signo (-) dentro de un circulo rojo). Debajo se ve un nuevo renglón de color verde, con el signo mas. Mas arriba, en renglones 49 y 50 se ven dos agregados.



1. Tambien podemos ver los cambios si en bash entramos la orden

**git log**



Configuraciones globales básicas de git

Creacion de nombre y e-mail para la auditoria de git, que permite saber quien hizo cada cosa.

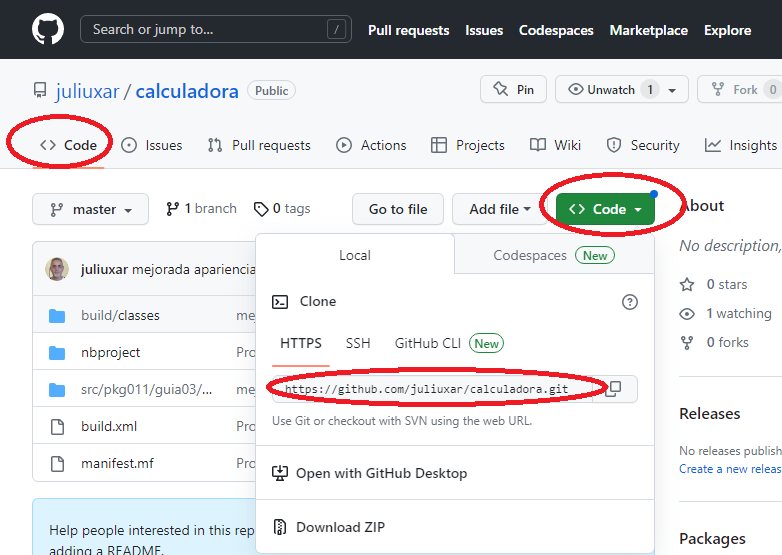
* git config –l 🡸 PARA LISTAR LA CONFIGURACION (guion medio, ele minuscula)
* git config --global user.name “<nombre del usuario de la pc>”
* git config –global user.email “<dirección de email>”
* git config –l 🡸 Para corroborar

VIDEO 4 de la serie 1: CLONAR UN REPOSITORIO (el 5to video es un repaso)

Como invitar compañeros de trabajo a nuestro proyecto para que participe y pueda colaborar bajo nuestro mismo repositorio.

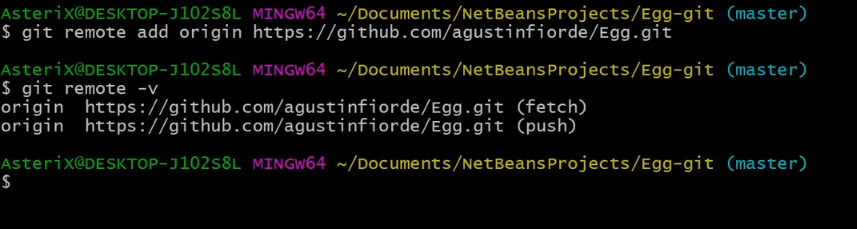
Continua la numeración de las viñetas.

1. El compañero también deberá haberse logueado en git
2. Creamos una carpeta nueva
3. Y dentro de la carpeta hacemos raton derecho y elegimos Git Bash Here
4. git init
5. Vinculamos este repositorio local, aun vacio, a un repositorio nuestro que deseemos compartir. Primero juntamos unos datos. Abrimos github.com, vamos al repositorio que queremos, sección “<> Code”, botón verde “<> Code”. Aparecera una ventana desplegable bajo el botón y de esa ventana copiamos la URL del repositorio:



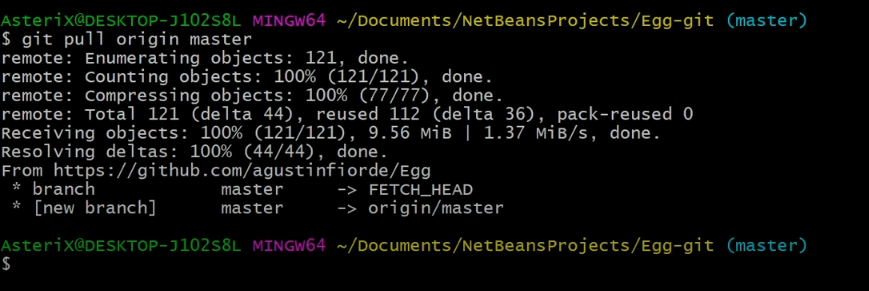
Y en git escribimos la orden git remote add origin <URL terminada en .git>

1. A continuación lo corroboramos con **git remote –v** y se vera un resultado similar a esto. Es decir, se corrobora que se ha hecho la conexión.

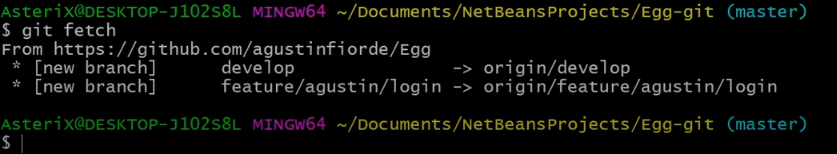


1. Hecha la conexión, traemos el proyecto para nuestra maquina usando **git pull origin master**

La siguiente es una pantalla típica luego de esa orden:



1. Si controlamos nuestra carpeta veremos que allí están todas las carpetas y archivos pulleados.
2. La orden git fech nos mostrara un resumen de la rama, que no parece ser mucha información según la imagen debajo. Según dice el youtuber, el renglón de arriba corresponde a la rama remota y el de abajo a la rama local, detalles que no son obvios por si mismos. Según se dice en el video este **pull** trae todas las ramas de trabajo.



1. Si se quiere se puede poner la orden git checkout develop

Fin del articulo