Apuntes de Git

migueluisV

Realizadas: Junio 2022

Índice

Índice de Figuras

1 Conceptos

1.1 ¿Qué es Git?

Es un software de control de versiones creado por el desarrollador de Linux: Linus Torvalds, es un software libre.

Funciona para tener un control, dar mantenimiento y desarrollar aplicaciones de forma individual, como en equipo.

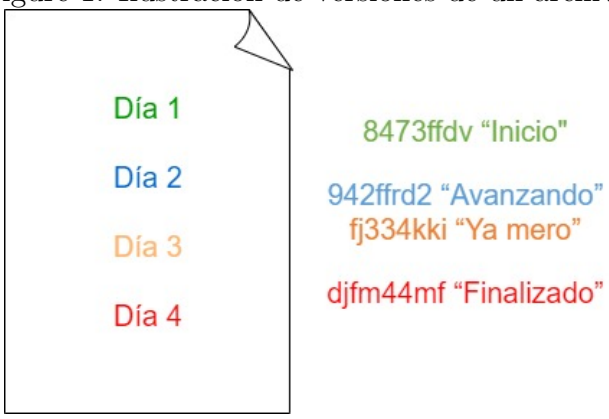
Logra estas características de desarrollo, mantenimiento y control guardando, en un registro, todos los cambios que sufre un archivo o conjunto de archivos.

1.2 Funcionamiento de Git

Supongamos que tenemos un archivo *index.html*, al principio tiene 0 líneas de código, después de dos horas, ya tiene más de 100 líneas y decidimos guardamos el progreso, entonces, al principio del trabajo, nuestro archivo era una versión (sin nada), después de las dos horas, ya es otra versión completamente distinta, Git se encarga de registrar las versiones que se van creando cuando uno decide que ha concluido con una versión, con una jornada laboral, o con un requisito de una lista de trabajo, estos registros tiene forma de "instantánea" (como una foto), la cual está compuesta por un código alfanumérico, por ejemplo: 8745dsd; y un mensaje entre comillas ("") que la describe, como lo podría ser: "Comienzo del proyecto".

Conforme vayamos trabajando el archivo y guardando los cambios, se irán creando más instantáneas de nuestro trabajo, como vemos en la *Figura 1*:

Figure 1: Ilustración de versiones de un archivo



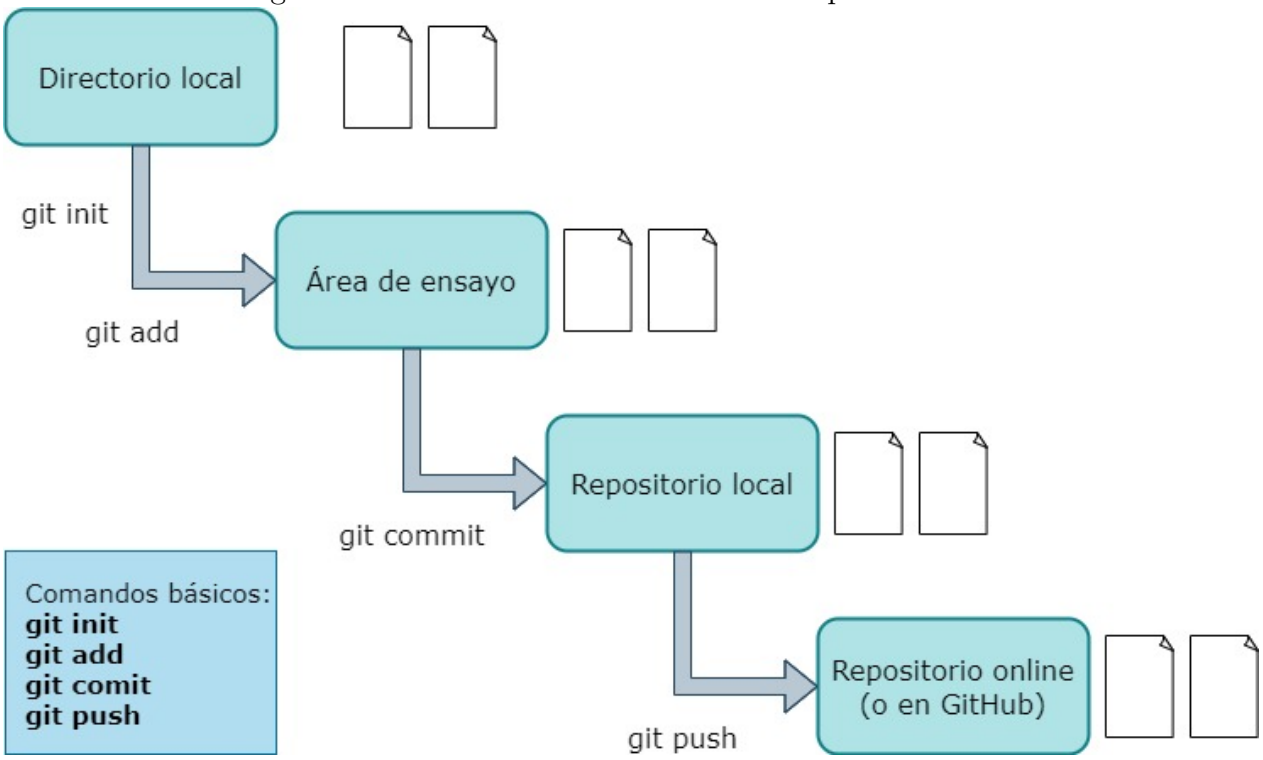
Git guarda las instantáneas que tiene de nuestro trabajo, así que, si durante el día 10 de trabajo, algo falla, podemos regresar a la instantánea de nuestro archivo en el día 7, para revisar que salió mal. Esto es muy útil para proyectos con muchos archivos.

1.3 Estructura de un repositorio local

Cuando decidimos crear un directorio local con un proyecto personal o grupal, al utilizar el comando **git init**, en nuestro directorio, se crea una carpeta invisible llamada **.git**, la cual

está constituida por dos partes: el **área de ensayo** y el **repositorio local**, el primero nos muestra todos aquellos archivos o ficheros que tienen seguimiento, es un área temporal; la segunda área almacena todos los archivos que tienen seguimiento y que están listos para que se les tome una *instantánea*. La *Figura 2* puede dar a entender un poco más este concepto:

Figure 2: Paso entre la estructura de un repositorio local



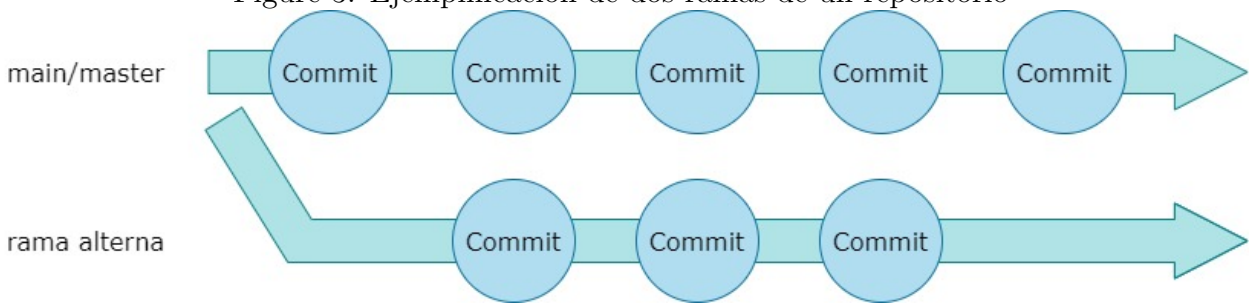
Nota:los comandos básicos serán explicados más adelante.

1.4 Ramas

Una **rama** es una copia de un archivo (que viene de un original) o archivos de nuestro repositorio local que es trabajada por un usuario o usuarios en específico, Git te permite crear más de una *rama* para que más de una persona o grupos trabaje en un archivo o archivos del proyecto, así, abarcando más tareas y trabajando más rápido, al final, cuando los usuarios terminaron su trabajo, Git puede unificar y combinar todas las ramas para que sea un solo archivo, archivos o repositorio con todo el trabajo de todos los usuarios.

En caso de que, varios usuarios hayan modificado en sus ramas el mismo archivo en la misma sección de código, a la hora de unificar todas las ramas, habrá un conflicto donde Git avisa y pregunta qué se hará en ese caso. Una rama podría tener el siguiente aspecto de la *Figura 3*, para que sean más fáciles de visualizar y entender:

Figure 3: Ejemplificación de dos ramas de un repositorio



Vemos que la rama *main/master* es la principal o la origen, a partir del primer **commit**, nace una **rama alterna** que es exactamente igual a la rama main/master, sin embargo, lo que se haga en esta última no afecta a su rama alterna, del mismo modo que lo que se haga en la rama alterna no afecta a su rama origen o principal.

Las ramas más comunes son **main** y **master**.

Trabajando en consola y sin utilizar un editor de texto, para poder crear una rama nueva en nuestro repositorio local, utilizamos el comando **git branch [nombre]**; puede usar el comando **git status -s** para ver el registro de la creación de la rama y **git branch** para ver las ramas existentes en el repositorio; puede utilizar el comando **git checkout [nombre rama]** para moverse a una rama existente para poder trabajar sobre ella.

Observación: en caso de que se esté trabajando en una rama ajena a main o master (por ejemplo, *rama v2.1*) y se desee volver a main, usando el comando **git checkout [nombre rama]**, en algunos editores de texto, como Dreamweaver, se nos notificará que se han hecho cambios desde fuera del editor de texto, por lo que podemos cargarlos en nuestro proyecto, sabemos que en main no hemos realizado cambios, pero si en v2.1, entonces, al pasa a una rama sin dichos cambios, se nos notifica en algunos editores de texto. Esto es una consideración a tomar en cuenta al moverse entre ramas.

Para poder mezclar, combinar o unir el trabajo de varias ramas en una sola, es requerido primero estar en la rama principal (main o master), se utiliza el comando **git merge [nombre rama]**, en caso de que se hayan modificado el mismo archivo en líneas de código similares o iguales, lanzará un error diciendo donde está ubicado el error, para que nosotros determinemos que solución dar.

1.5 Tags

Las **tags** (etiquetas) son identificadores que determinan hasta un cierto punto nuestro repositorio online, es decir, si tenemos 20 archivos que conforman nuestro repositorio, a estos 20 archivos los podemos denominar como "versión 1.0", entonces, utilizamos estas etiquetas para definir la versión del repositorio y poder hacer que la gente vea en qué versión se encuentra nuestro trabajo, y poder descargarla. Estas etiquetas funcionan también para indicar que nuestro repositorio está *en proceso* o *terminado*.

Para poder crear una etiqueta en consola escribimos el comando **git tag [nombre] "descripción"** (por ejemplo: *git tag 5-5-20v1 "versión 1.0"*), en este momento, solo hemos creado la etiqueta, podemos apreciarlo usando el comando **git status -s**, todavía falta subirla, para ello, usando el comando **git push tags**.

2 Comandos de Git

Enlistamos todos los comandos vistos en el curso de YouTube que aprendimos.

- Comandos para comenzar a trabajar con un repositorio local:
  - **git init:** comienza el seguimiento de versiones del proyecto o archivos, en el directorio donde está nuestro trabajo.
  - **git add *[archivo]*:** es usado para darle seguimiento a un o unos archivos, los archivos que sean agregados pasan al área de ensayo. P. e: *git add index.php*.
  - **git add .:** les da seguimiento a todos los archivos de una carpeta.
  - **git commit -m "*mensaje*":** pasa todos los archivos del área de ensayo al repositorio local, una vez estando ahí, toma la instantánea de los archivos. P. e: *git commit -m "Primer avance"*.
  - **git commit -am "*mensaje*":** agrega todos los archivos que han sido modificados recientemente del área de ensayo al repositorio local, una vez estando ahí, toma la instantánea de los archivos.
  - **git status -s:** enlista todos los archivos que tienen seguimiento o si están en el área de ensayo.
  - **git log –oneline:** enlista todas las instantáneas creadas de nuestro proyecto.
  - **git reset –hard *[código instantánea]*:** regresa a una instantánea creada previamente. P. e: *git reset –hard 90ffrtyu*.
  - **git remote add origin *[dirección repositorio online]*.git:** a esta dirección se va a subir nuestro repositorio local. Debe haber primero ingresar las credenciales de su cuenta donde está el repositorio utilizando los siguientes comandos:
    - **git config –global user.name "*nombre usuario GitHub*"**
    - **git config –global user.email "*correo electrónico GitHub*"**
- Comandos para ramas:
  - **git branch:** enlista todas las ramas creadas o existentes hasta el momento.
  - **git branch *[nombre]*:** crea una nueva rama en el repositorio local.
  - **git checkout *[nombre rama]*:** pasamos a trabajar a la rama seleccionada.
  - **git brand -d *[nombre rama]*:** elimina la rama seleccionada.
  - **git merge *[nombre rama]*:** comienza el proceso de unificar varias ramas en una sola. En caso de que se hayan hecho cambios en un mismo archivo desde varias ramas (problema de concurrencia), Git lanzará mensaje de error y no podrá concretar la mezcla hasta que nosotros lo resolvamos.
  - **git branch -M main:** asigna a qué rama se va a subir el repositorio local.
  - **git push -u origin main:** sube todo el contenido del repositorio local al de la dirección web.

- Comandos para subir y clonar repositorios:
  - **git pull:** jala todo el contenido de un repositorio online a nuestro repositorio local. En caso de que muestre algún error al escribir este comando, al final del mismo, agregue la dirección .git del repositorio online.
  - **git clone *[dirección .git]*:** clona o copia los archivos de un repositorio online a un directorio local de nuestra computadora.
  - **git push tags:** sube todas las etiquetas creadas del repositorio local al online.

3 Trabajando con Git en consola

Los siguientes puntos describen el proceso de trabajo con Git utilizando solamente el software oficial de Git en nuestro sistema operativo, usando además un editor de texto como auxiliar, más adelante se describirá el uso de Git con un editor de texto mejorado y botones que fácilitan el uso y comprensión de Git.

i. Creamos un directorio local en nuestro sistema operativo para nuestro proyecto.

ii. Debemos instalar Git en nuestra máquina (https://git-scm.com/downloads).

iii. Una vez instalado, debemos tener a la mano un editor de texto para poder crear los archivos de nuestro proyecto en nuestro directorio local (Notepad++, Bloc de Notas, Visual Studio Code, Dreamweaver, etc.).

iv. Creamos un archivo en nuestro directorio local, ya sea por medio del Explorador de archivo de nuestro sistema operativo o por el editor de texto, crearemos un archivo llamado *index.html*.

v. Abriremos la consola Git Bash en la ubicación donde está nuestro directorio local (en Windows, clic derecho sobre cualquier parte vacía del directorio, y damos clic en la opción **Git Bash Here**), con esto podremos trabajar Git en el directorio local.

vi. Para no entrar en detalle sobre cuántos y qué es lo que contienen los archivos de este proyecto ejemplo, el archivo *index.html* tiene como contenido la estructura básica de un archivo html, este será la primera versión o instancia de nuestro proyecto en Git.

vii. En la consola Git Bash, escribimos el comando **git init**, recordemos que la consola está ubicada en el directorio local de nuestro proyecto, por lo que, Git creará una carpeta escondida **.git**, donde estará la información necesaria para crear el repositorio local.

viii. Si escribimos el comando **git status -s** veremos la lista de los archivos en el directorio local de nuestro proyecto, pero como ninguno ha sido agregado al área de ensayo ni tienen seguimiento, para lograr esto, debemos escribir el comando **git add index.html**, repetimos el comando **git status -s** y ahora si nuestro único archivo tiene seguimiento.

ix. Escribimos el comando **git commit -m "Primer avance"** para crear la primer instantánea o versión de nuestro proyecto, este comando manda nuestro archivo *index.html* al repositorio de Git. Si escribimos el comando **git log –oneline** veremos la instantánea que acabamos de crear, con su código y mensaje.

x. Si volvemos a escribir el comando **git status -s**, veremos que *index.html* ya no está presente, esto es debido a que ahora está ubicado en el repositorio, no en el área de ensayo.

xi. Ahora bien, si realizamos cualquier cambio en *index.html* (por ejemplo, agregar un Header 1 con un mensaje de bienvenida), y escribimos el comando **git status -s**, veremos que este volverá a ser listado como un archivo en el directorio local, pero debe estar en el área de ensayo para que podamos realizar un commit nuevamente, repetimos el comando **git add index.html** para que este tenga seguimiento nuevamente.

xii. Repetimos el comando **git commit -m "Segundo avance"** para crear otra instantánea con nuestras modificaciones.

xiii. En github.com, debemos tener un repositorio de prueba creado y su enlace a la mano, para poder subir desde la consola todos los archivos del repositorio local al que está ubicado en GitHub. Primero debemos configurar a qué cuenta y repositorio se va a subir el repositorio local, para ello, escribimos los comandos **git config –global user.name** y **git config –global user.email**, después de las últimas palabras de cada comando, debemos escribir el nombre de usuario y correo electrónico de la cuenta de GitHub a donde se subirá el proyecto, después de configurarlos, escribimos el comando **git remote add origin**, al final de este comando, va la dirección **.git** de nuestro repositorio (se encuentra en la página principal del repositorio online de GitHub), después escribimos el comando **git branch -M main**, este comando indica a qué rama se subirá el repositorio local, preferiblemente lo haremos a la rama **main**, ya que si no se establece este comando, se subirá en automático a la rama **master**, finalmente, escribimos el comando **git push -u origin main**, este comando se encarga de subir el repositorio local a GitHub. Una vez hayamos hecho todo esto, podemos refrescar nuestro repositorio en GitHub para ver nuestro proyecto en línea.

xiv. En caso de que se haga alguna modificación desde GitHub en el repositorio online, podemos obtener los archivos o modificaciones online en nuestro repositorio local, para ello, utilizamos el comando **git pull** o **git pull *[dirección .git]***, para obtener los archivos modificados, siempre y cuando en la consola ya estemos trabajando en el directorio local.

xv. En caso de que hayamos perdido nuestro proyecto de directorio local, podemos utilizar el comando **git clone** para poder copiar todo lo que esté en un directorio online a nuestra computadora, y así seguir trabajando en la última versión que realizamos. Refrescamos la pestaña de GitHub en nuestro navegador, y veremos que ya existe una versión 1.0 de nuestro proyecto.

4 Trabajando con Git en Visual Studio Code y consola

Visual Studio Code (VSC a partir de ahora) cuenta, de manera nativa, con herramientas Git para poder crear repositorios locales y poderlos subir a GitHub de una manera muy visual y fácil de comprender.

Primero que nada, debemos tener nuestro directorio local con nuestro proyecto en el sistema operativo, en VSC, es recomendable crear **áreas de trabajo** o **Workspaces** para nuestro directorio local, en caso de que se cierre accidentalmente VSC, cargamos nuestro área de trabajo y todo estará justo como lo dejamos.

Las siguientes Figuras muestran como crear un *Workspace*:

Figure 4: Agregando directorio local a VSC

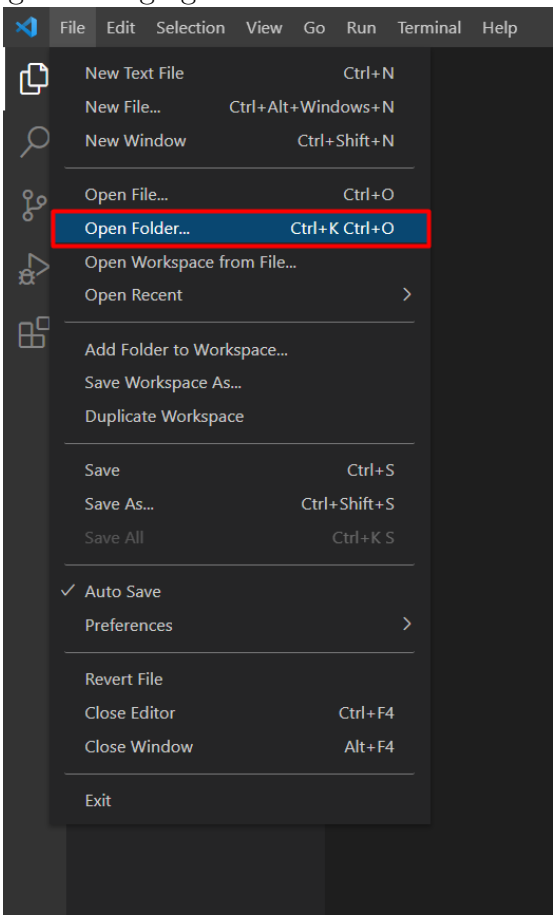


Figure 5: Seleccionando directorio local

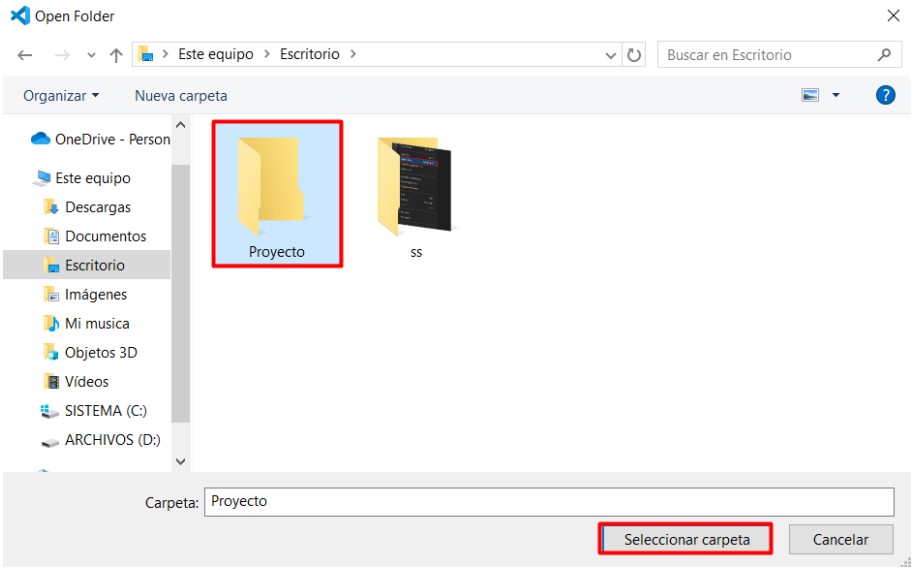


Figure 6: Autorizando directorio local en VSC

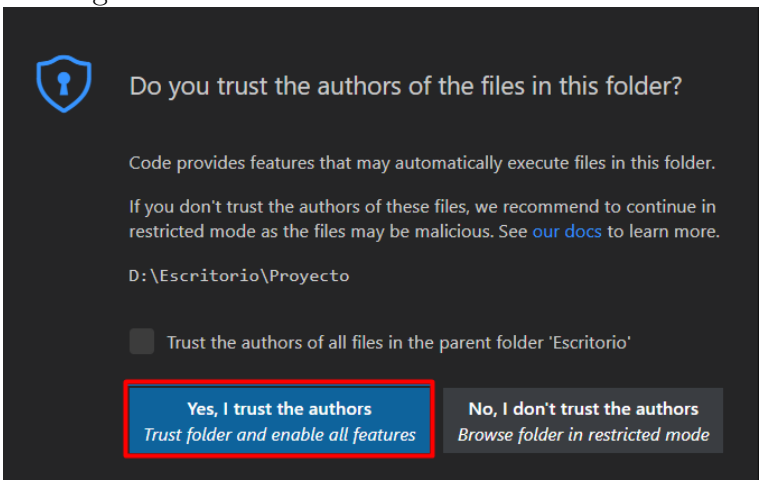


Figure 7: Agregando Workspace a VSC

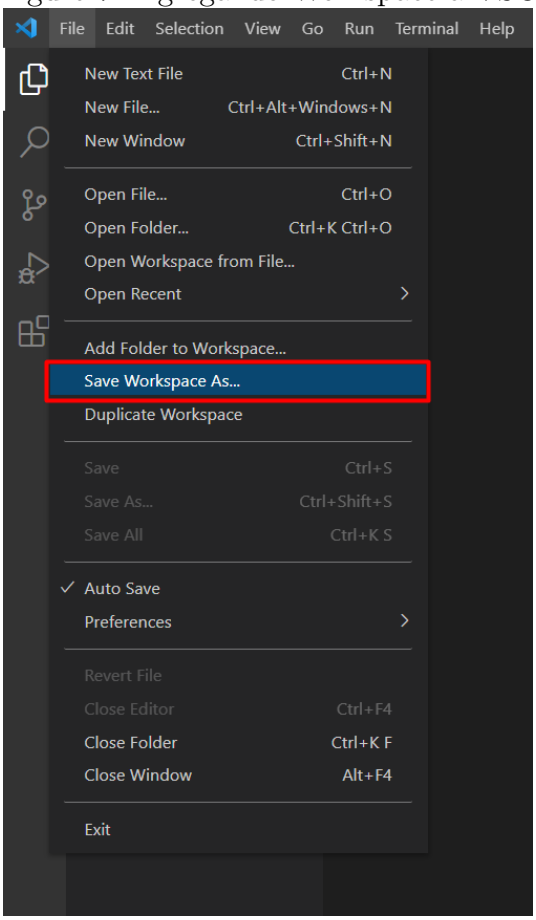
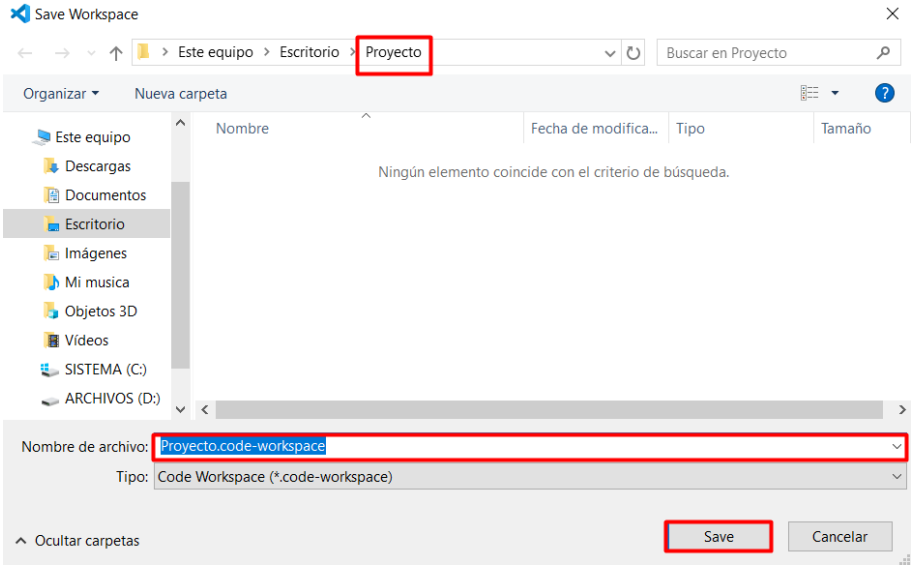


Figure 8: Seleccionando dirección donde guardar Workspace y nombrándolo



Vemos que en la *Figura 4* y *5* se abre y ubica el directorio local de nuestro proyecto

en VSC, la *Figura 6* muestra un mensaje (puede aparecer o no) donde confirmamos que confiamos en este directorio, una vez cargado el directorio en VSC, debemos guardarlo como un *área de trabajo*, la *Figura 7* lo guarda como tal y la *Figura 8* le da una ubicación y nombre al área de trabajo, puede estar ubicado en el mismo directorio local de nuestro proyecto y tener el mismo nombre, o no.

Para **abrir** un Workspace en VSC se siguen las siguientes Figuras:

Figure 9: Opción para abrir un Workspace

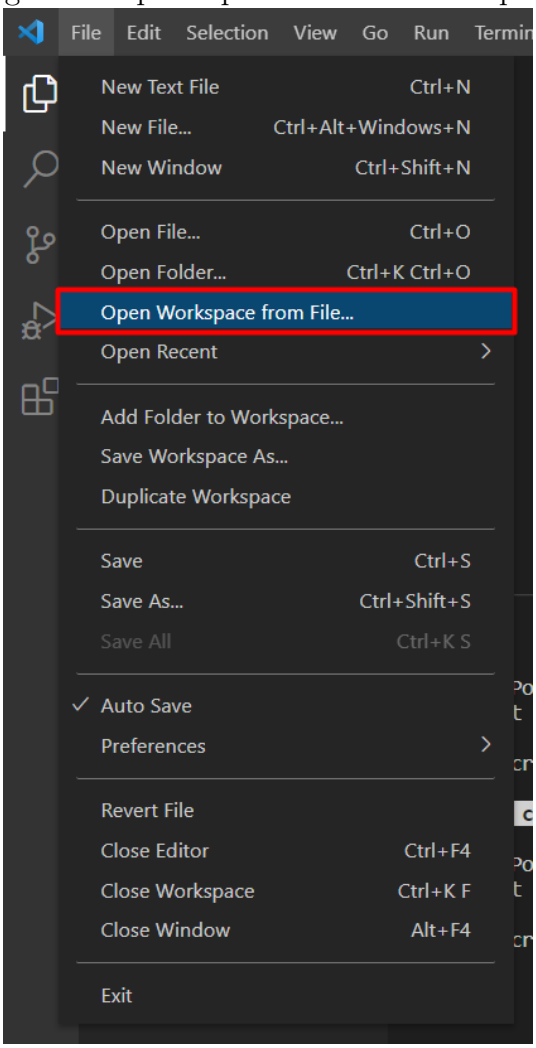
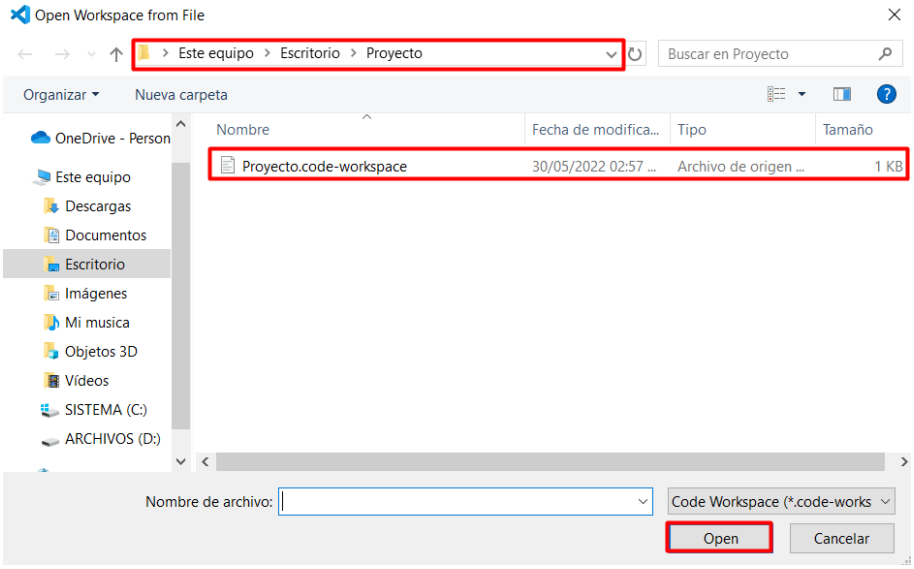


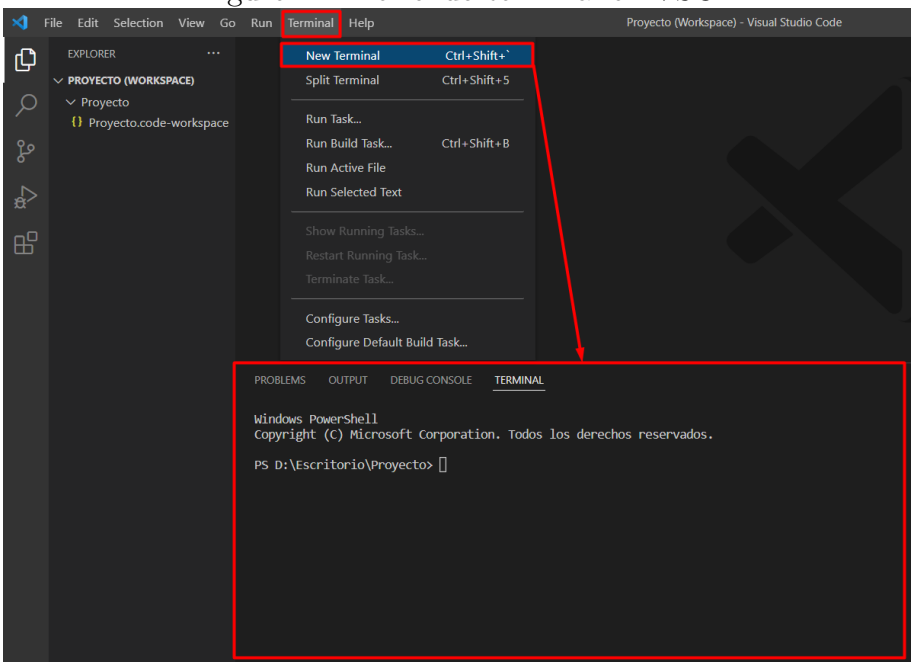
Figure 10: Buscando el Workspace en su ubicación



En la misma pestaña **File** se encuentra la opción para abrir un Workspace desde un archivo, como se ve en la *Figura 9*, tendremos que buscar nuestro archivo de .code-workspace donde lo almacenamos, como podemos apreciar en la *Figura 10*.

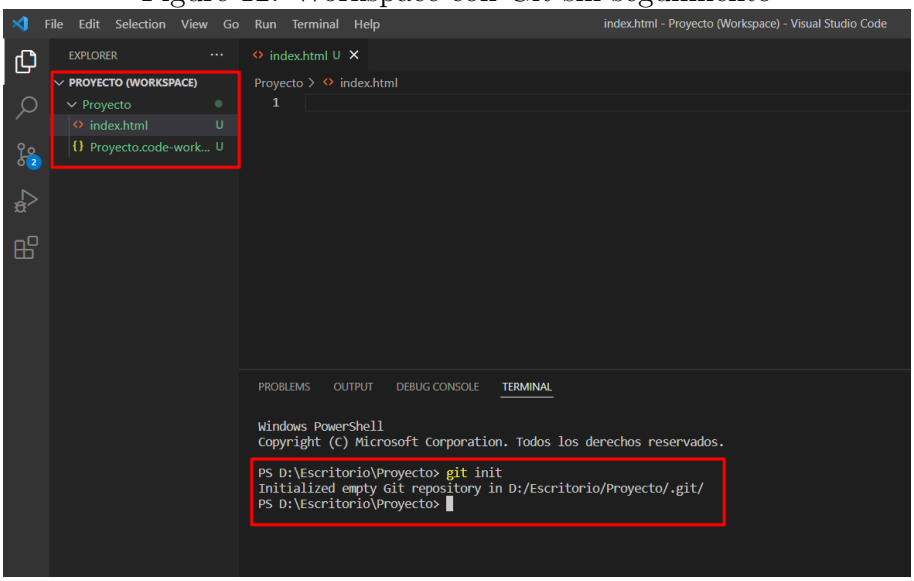
Todo el proceso anterior fue para poder cargar nuestro directorio local a VSC, ahora, es momento de que Git le comience a dar seguimiento. Agregamos al Workspace un archivo *intdex.html*. Si recordamos como fue realizado ese proceso en consola, son prácticamente los mismos comandos, pero VSC te lo muestra de una manera más visual: la *Figura 11* muestra que podemos abrir terminales (consolas) en nuestro editor de texto.

Figure 11: Abriendo terminal en VSC



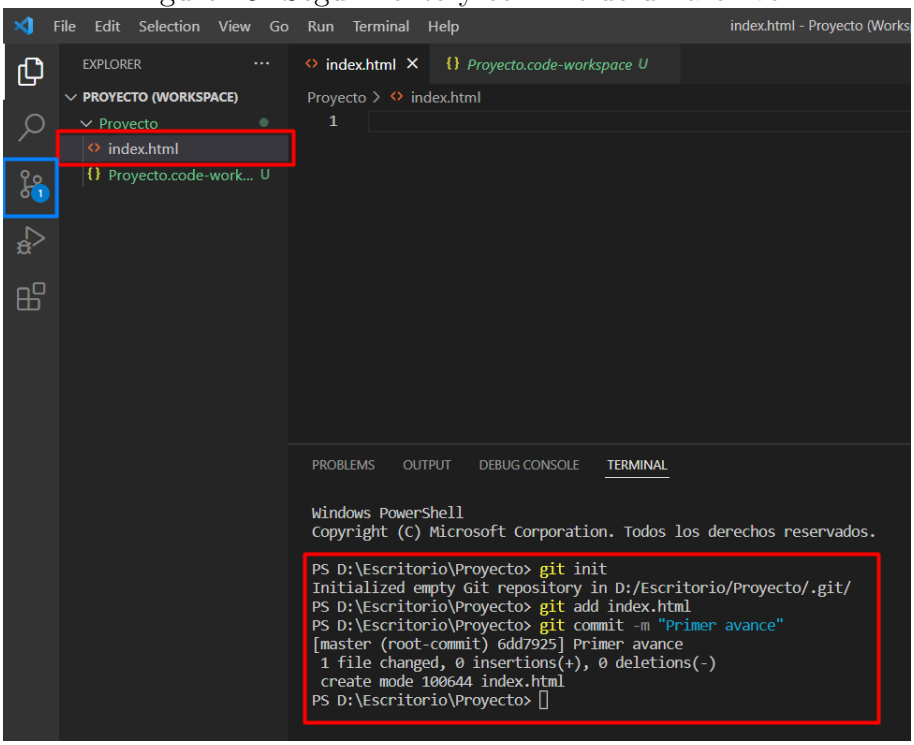
En esta terminal, escribiremos los comandos git vistos anteriormente, comenzaremos con **git init**. Vemos en la *Figura 12* que el explorador de nuestro Workspace se ha coloreado en verde, y han aparecido algunas letras al lado de los archivos, estas serán explicadas más adelante.

Figure 12: Workspace con Git sin seguimiento



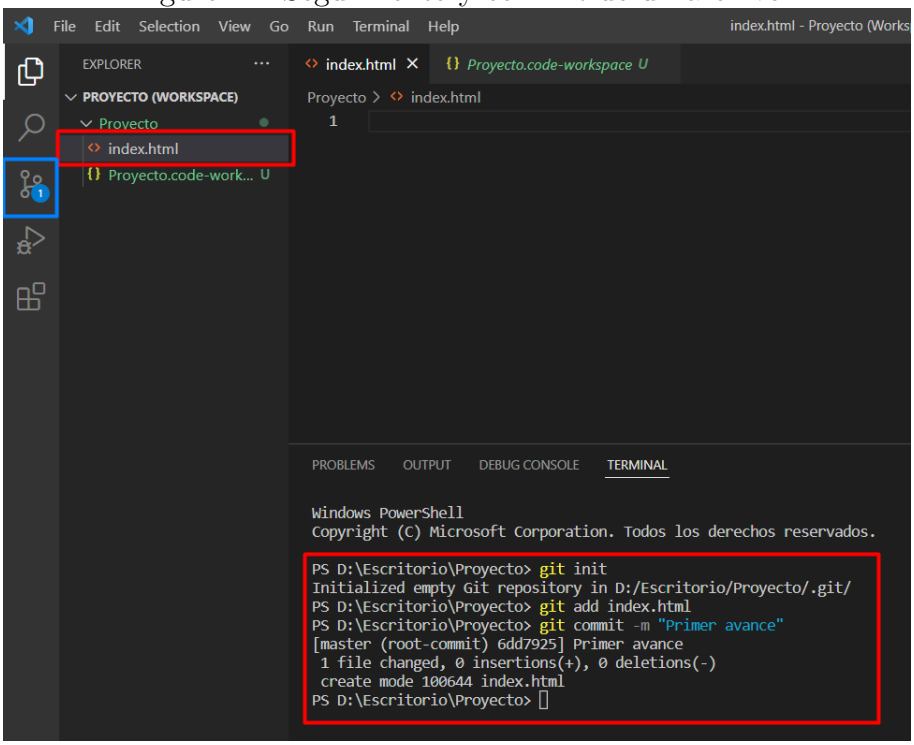
Entonces, debemos agregar *index.html* al área de ensayo, dándole seguimiento, con el comando **git add index.html**, una vez hecho esto, podemos realizar un primer commit (aunque el archivo esté vacío) usando el comando **git commit -m "Primer avance"**, como se ve en la *Figura 13*.

Figure 13: Seguimiento y commit de un archivo



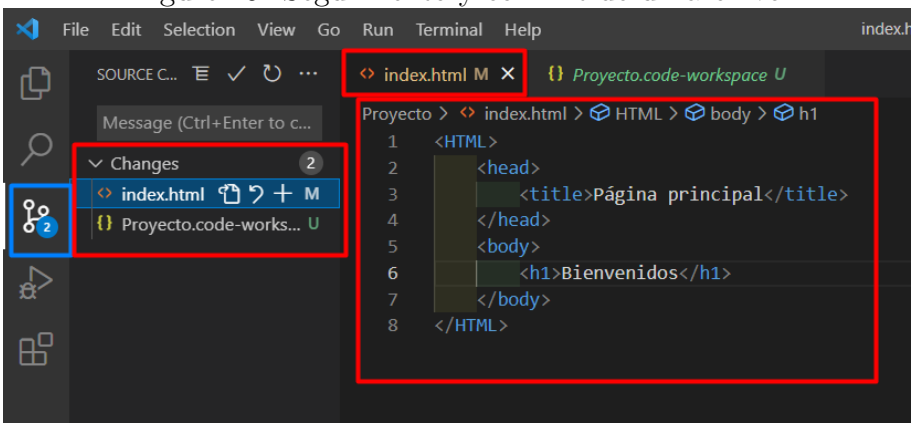
Si nos fijamos, encerramos una opción en el menú lateral izquierdo con un cuadrado azul, esta opción nos muestra todos aquellos archivos que no tienen seguimiento o que han sido modificados recientemente, por lo que necesitan que se les realice alguna acción o tarea, esta opción es como una lista de archivos a los cuales darles seguimiento en VSC, cuando usamos el comando **git init**, esta opción tenía 2 tareas o 2 archivos con pendiente para realizar seguimiento: *index.html* y *Proyecto.code-workspace*, al darle seguimiento a index.html nada más, esta lista de tareas bajó a 1, al segundo archivo mencionado no se le hará seguimiento. Vemos entonces, en la *Figura 14* que en nuestro explorador de Workspace, el archivo index.html ya no tiene la letra U a su costado, ni está coloreado de verde, esto es porque ya tiene seguimiento, al realizar alguna modificación, este archivo volverá a estar coloreado.

Figure 14: Seguimiento y commit de un archivo



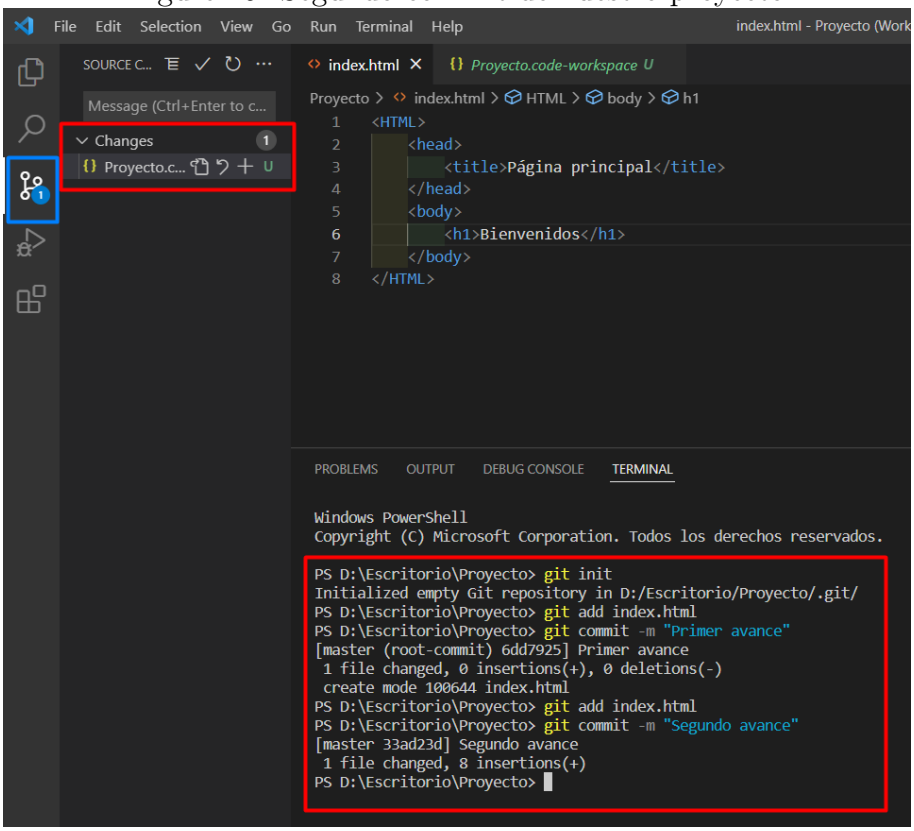
Ahora, realizaremos una modificación para ver el aspecto de VSC (*Figura 15*):

Figure 15: Seguimiento y commit de un archivo



Otra vez, nuestra lista de tareas aumenta a 2 (recuadro azul), añadimos una estructura básica a *index.html* y este tiene la letra M a su costado, podemos realizar un segundo commit. No olvide guardar todos los cambios en VSC y su Workspace antes de hacer un **git add** y **git commit**. Veamos la *Figura 16* para ver el segundo commit.

Figure 16: Segundo commit de nuestro proyecto



Las letras que aparecen significan:

- **U**: unfollow, sin seguir, significa que los archivos no tienen seguimiento.
- **M**: modified, modificado, significa que a un archivo se le han hecho cambios, por lo que se le debe dar seguimiento nuevamente.

5 VSC y sus herramientas visuales para trabajar con Git

Este editor cuenta con más herramientas que nos harán prescindir de la terminal.

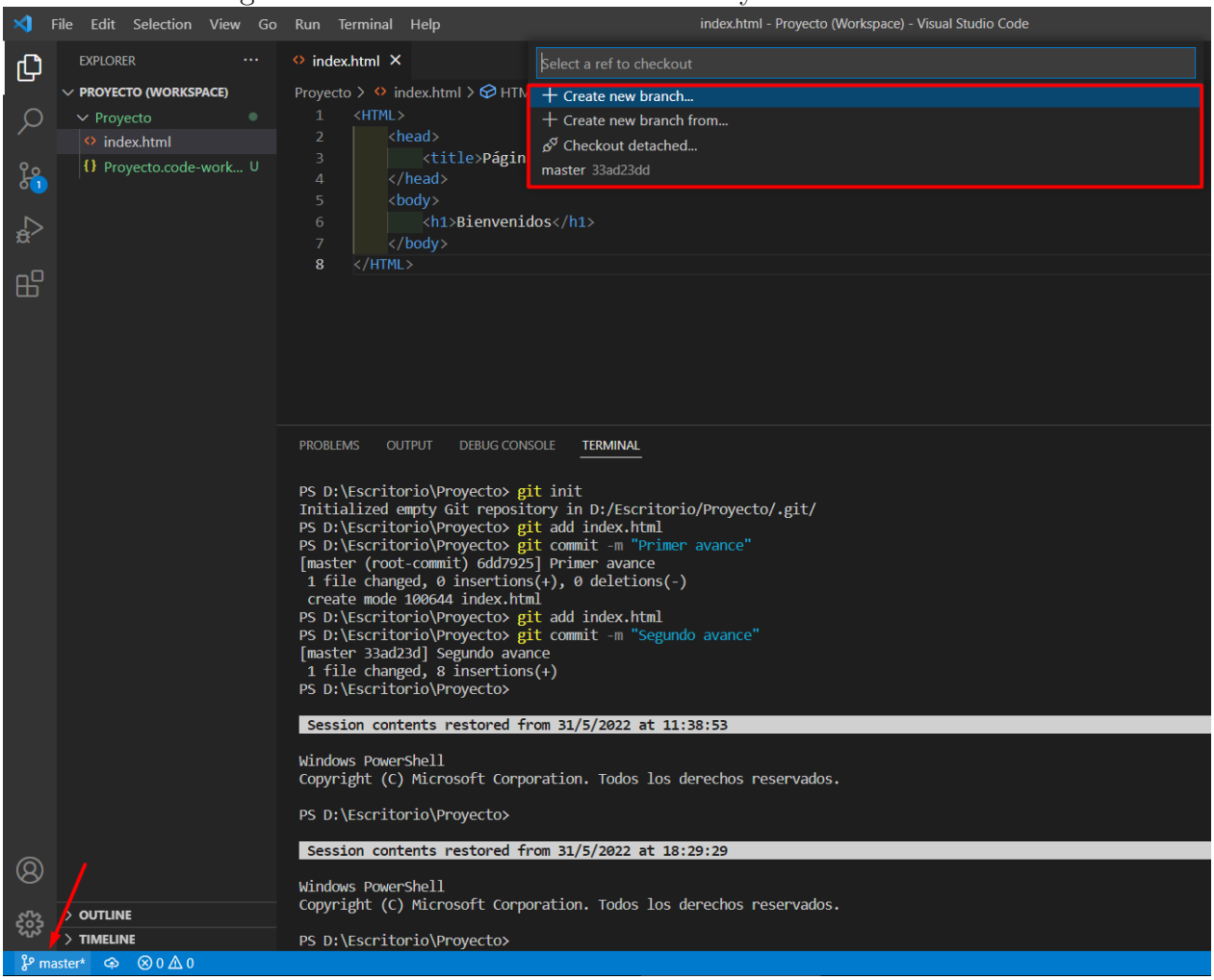
5.1 Creando ramas

Por ejemplo, para **crear una rama** de Git en VSC podemos utilizar dos formas:

i. **Con la barra de tareas inferior azul**: VSC posee una barra de tareas color azul en la parte inferior de su interfaz, donde suele mostrar los errores y alertas de un código, en que línea y columna estamos ubicados, entre otra información, en la zona izquierda de esta barra, encontramos un texto que nombra a la rama en donde estamos trabajando actualmente con nuestro proyecto en Git. Si damos clic sobre este texto, nos abre una

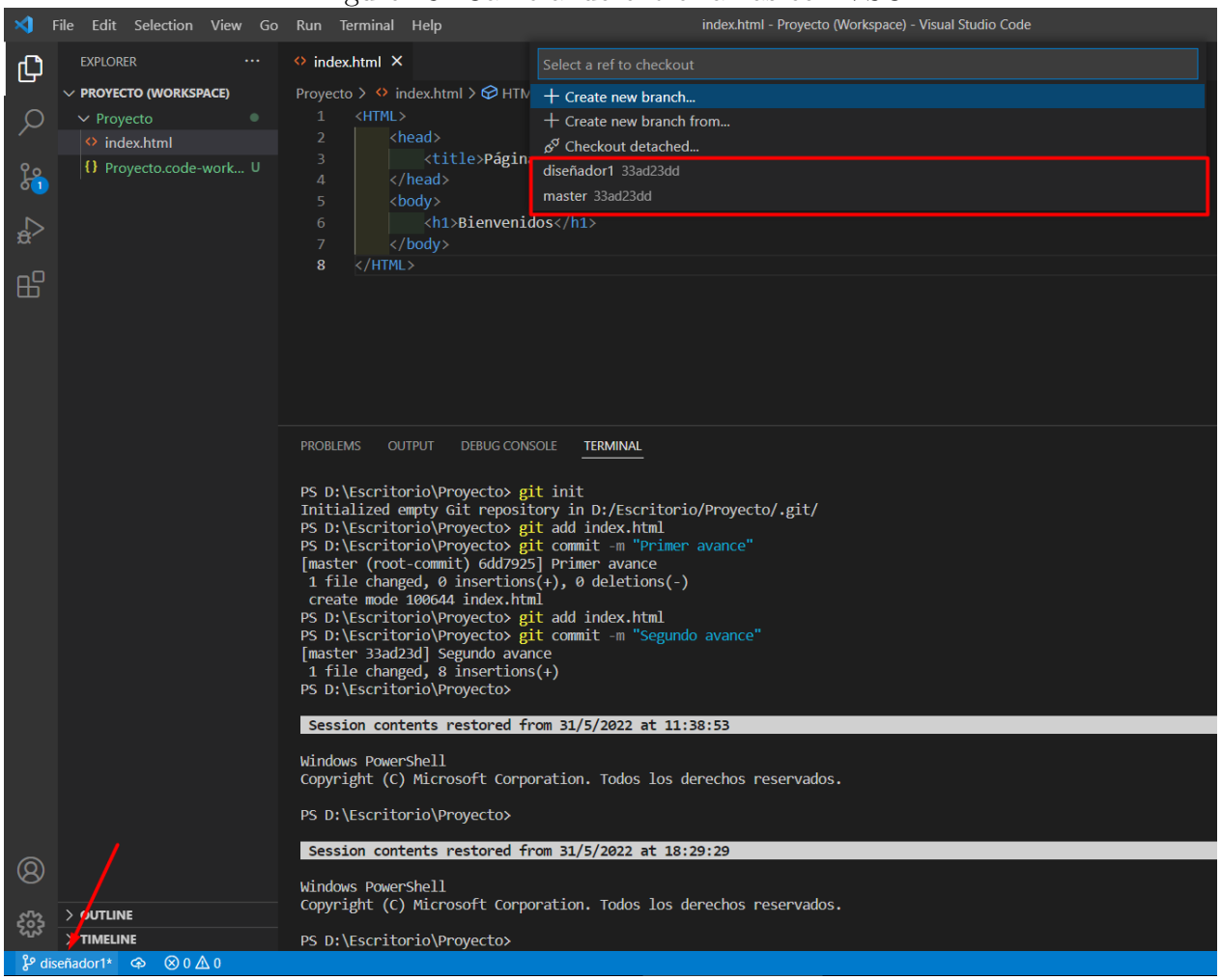
barra de acciones en la parte superior central de VSC dándonos algunas opciones, como la de **Crear rama** y ver las ramas existentes, como se ve en la *Figura 17*:

Figure 17: Creando una rama con VSC y su barra de tareas



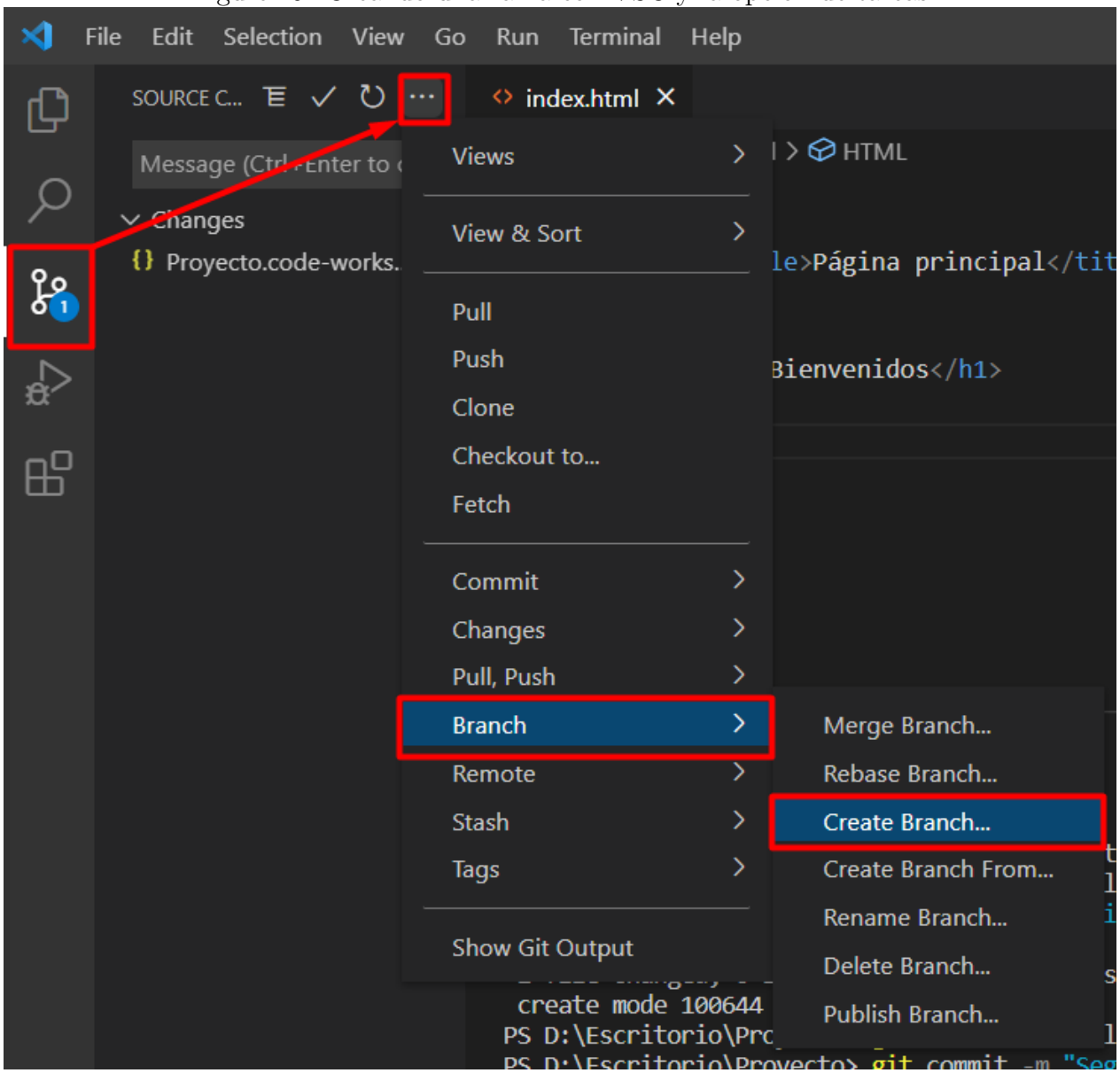
Al dar clic en **Create new branch...** nos pedirá el nombre de esta, una vez escrito, damos ENTER para continuar y la rama se ha creado, VSC nos situa en la rama recién creada para comenzar a trabajar en ella directamente. Si volvemos a dar clic sobre el nombre de la rama en la barra de tareas, él menú desplegado, además de darnos la posibilidad de crear otra rama, nos muestra las dos existentes hasta el momento, como se ve en la *Figura 18*:

Figure 18: Cambiando entre ramas con VSC



ii. **Con el menú de tareas lateral y sus tres puntos**: esta opción es un poco más completa, y nos sitúa en un espacio donde aparecen muchas más opciones para trabajar con Git. Nos vamos a la opción de tareas de Git, en este punto, tenemos una tarea pendiente, que es darle seguimiento al archivo de Workspace de nuestro proyecto, pero lo dejaremos como está, vemos que encima de **Changes** hay una caja de texto para escribir algo, encima de esta, hay un botón con la forma de tres puntos, si damos clic a dicho botón, se nos despliegan múltiples características Git que podemos usar, entre ellas, está la opción **Branch**, si la seleccionamos, podemos encontrar la opción de **Create branch...**, además de las opciones para unir ramas, renombrar alguna, borrar, entre más, como vemos en la *Figura 19*:

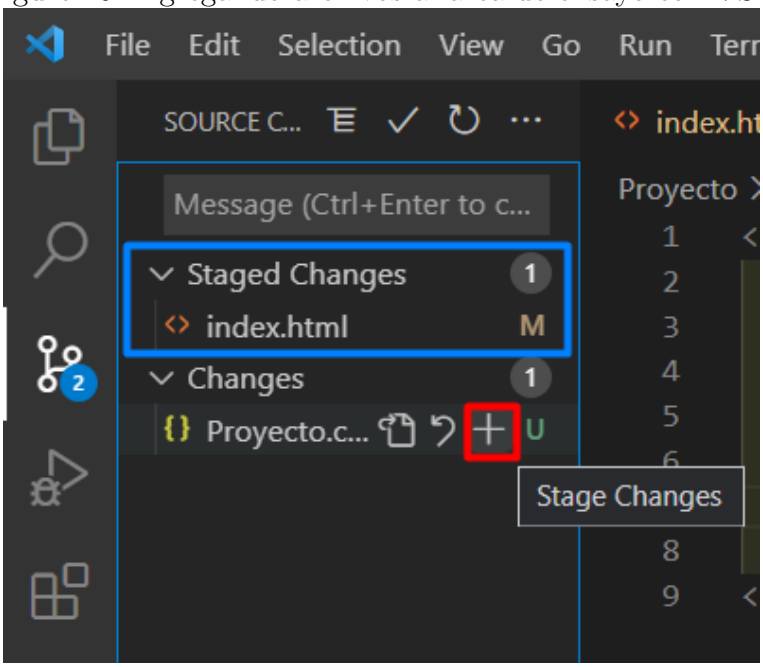
Figure 19: Creando una rama con VSC y la opción de tareas



5.2 Agregando archivos al área de ensayo

Nos quedaremos en la rama diseñador1. Este punto es bastante simple, si realizo alguna modificación en *index.html*, sabemos que este archivo aparecerá en el menú de tareas de Git, si situamos el puntero del mouse encima del archivo con letra M, veremos que aparecen varias opciones, entre ellas, un icono de Más (+), el cual realiza la tarea de agregar el archivo al área de ensayo, lo único que debemos hacer es darle clic, y nuestro archivo tendrá seguimiento (*Figura 20*).

Figure 20: Agregando archivos al área de ensayo con VSC

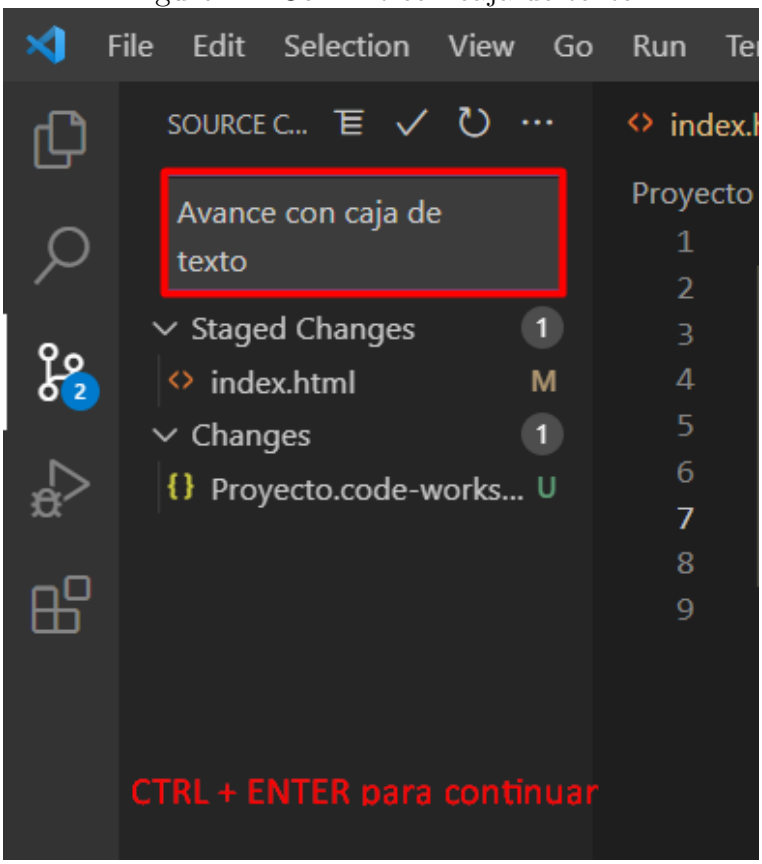


El archivo *index.html* ya tiene seguimiento, lo vemos encerrado en el recuadro azul, mientras que el archivo de Workspace no lo tiene, por eso aparece el botón + entre sus opciones.

5.3 Realizando commits

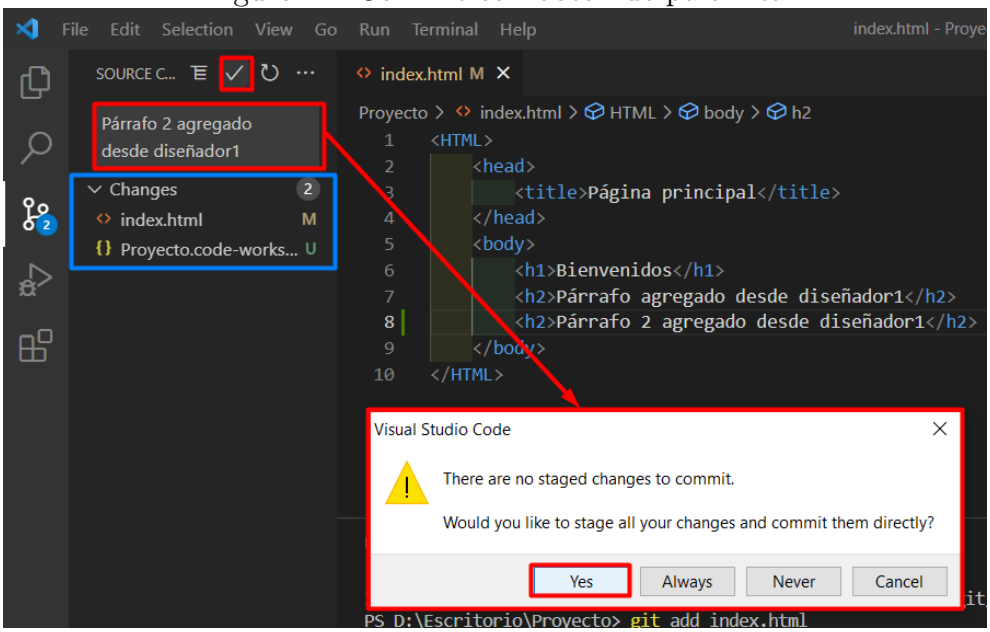
Aquí se nos presenta una opción bastante útil en caso de tener varios archivos. Como dijimos anteriormente, hay una caja de texto en el menú de tareas en la cual podemos ingresar un texto, este texto es el mensaje que tendrá un commit, al escribirlo y dando CTRL + ENTER, se llevará a cabo un commit, pero el alcance que este tendrá solo aplicará a los archivos que tienen seguimiento (*Figura 21*).

Figure 21: Commit con caja de texto



En cambio, si repetimos la acción de escribir el mensaje, teniendo archivos con y sin seguimiento (letra M y U) y presionamos el botón con una palomita, VSC nos lanzará una alerta que dice que si queremos realizar el commit usando todos los archivos (con y sin seguimiento) siempre, nunca o solo en esta ocasión, nosotros diremos que solo en esta ocasión (*Figura 22*).

Figure 22: Commit con botón de palomita



Y si recordamos cómo crear ramas con VSC, sabremos que si damos clic al botón de los tres puntos, ahí también aparece la opción de commits.

5.4 Unir ramas y conflictos de concurrencia

Una vez terminado el trabajo en dos ramas distintas, damos clic al botón de los tres puntos en el menú de tareas, dentro de la opción **Branch**, encontramos la opción **Merge branch...**, VSC toma la rama donde estamos trabajando como rama inicial, y nos despliega una lista de las demás ramas para unirlas, como vemos en la *Figura 23* y *Figura 24*:

Figure 23: Uniendo varias ramas

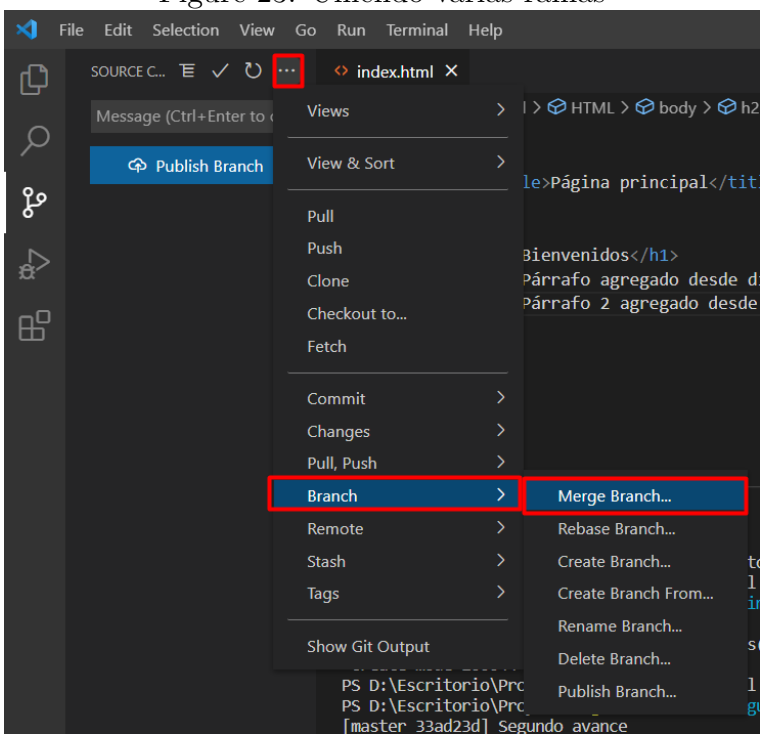
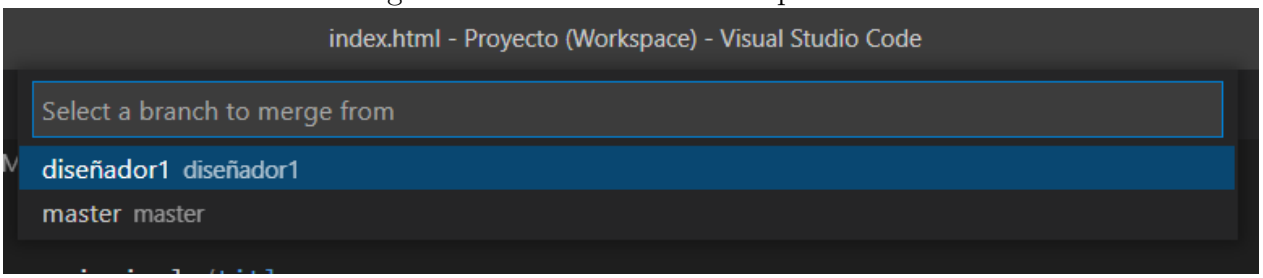
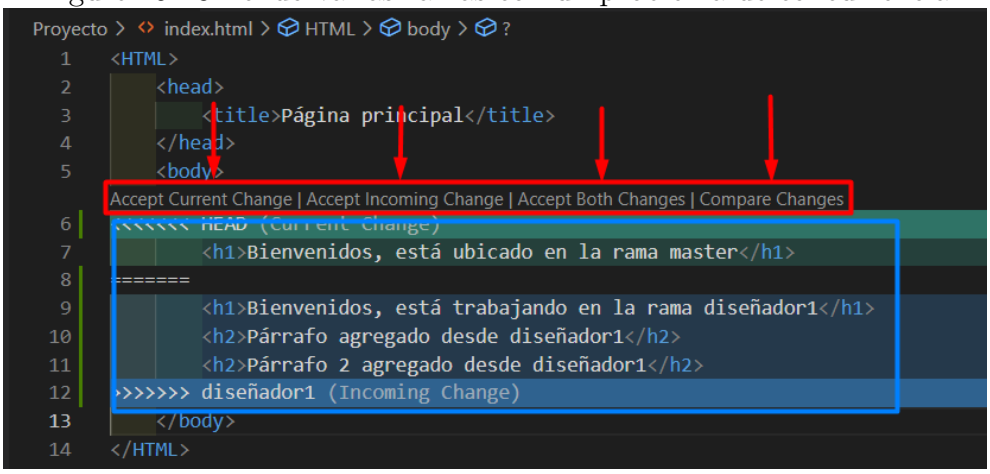


Figure 24: Selección de ramas para unir



Esto en caso de que no hubiera ningún conflicto de concurrencia, pongamos la siguiente situación: en **master**, agregamos más texto al **Header 1** y hacemos un commit, en **diseñador1** volvemos a hacer cambios en el *Header 1* y volvemos a realizar un commit, finalmente, realizamos el proceso de unión de ramas, sin embargo, hicimos un problema de concurrencia a propósito para ver como lo trata VSC en la *Figura 25*:

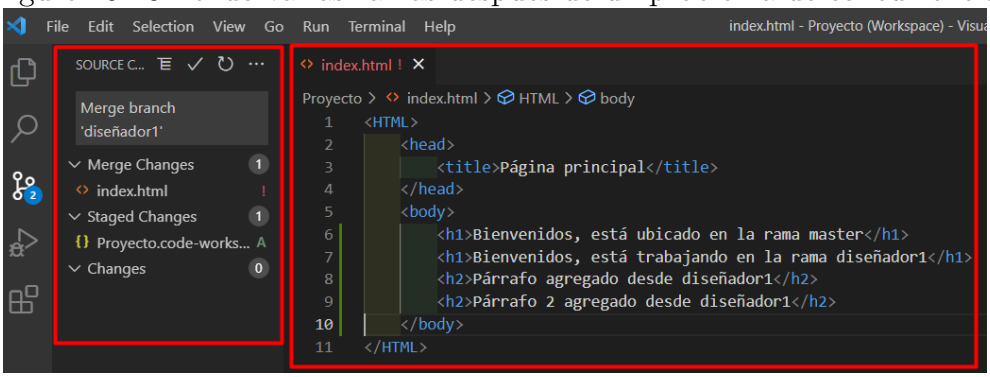
Figure 25: Uniendo varias ramas con un problema de concurrencia



Vemos que VSC nos lanza una advertencia de que hay un problema de concurrencia, en **master** y en **diseñador1** se han realizado cambios en la misma línea de código, por lo que se nos ofrecen las siguientes opciones: aceptar el cambio de la rama actual, aceptar el cambio de la rama con la que se desea mezclar, aceptar ambos cambios y comparar ambos cambios, dependiendo de nuestras necesidades, aceptamos una u otra, nosotros aceptaremos los dos cambios.

Una vez escogida la opción, VSC da seguimiento al archivo donde ocurrió el problema, escribe un texto en la caja de mensaje para realizar un commit, se puede dejar o se puede cambiar, y podemos realizar el commit con el archivo ya mezclado sin problemas (*Figura 26*)

Figure 26: Uniendo varias ramas después de un problema de concurrencia



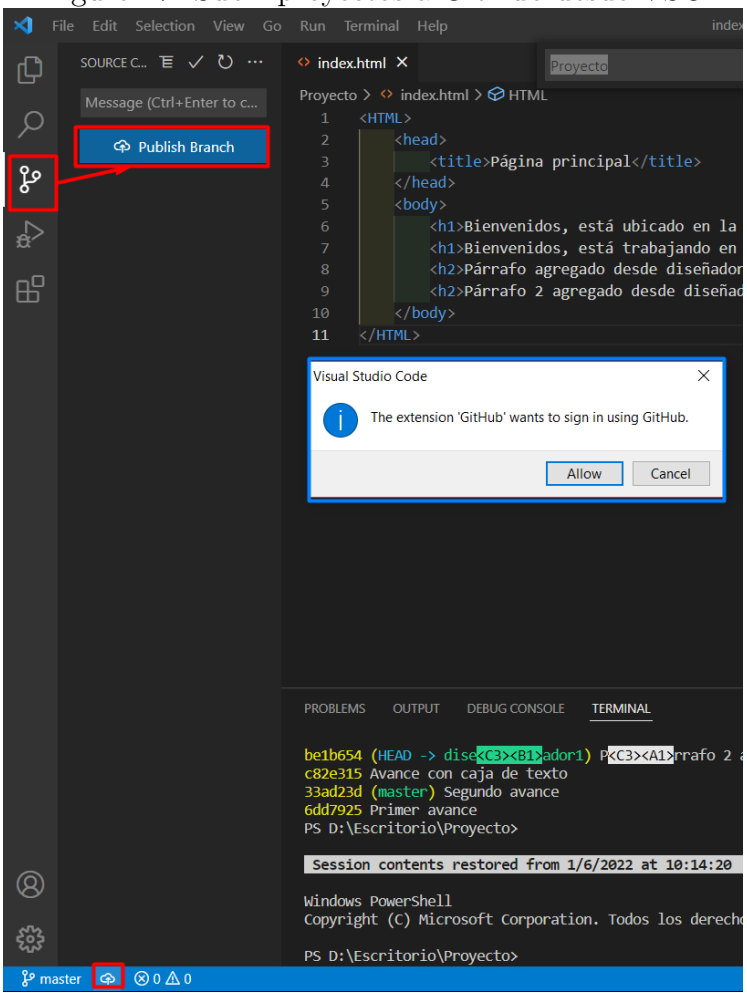
Nota: en caso de que se presente algún error para realizar el commit en este punto, puede poner el puntero del mouse encima del archivo que tuvo los problemas de concurrencia y dar clic al botón de +, para darle seguimiento.

5.5 Sincronizando y subiendo proyecto de VSC a GitHub

Podemos subir nuestro proyecto a GitHub de dos formas, desde el **panel de tareas de Git** y desde la **barra de tareas de VSC**, en nuestra situación, no hemos ingresado ninguna

cuenta de GitHub ni enlace de repositorio, por lo que, para ambas opciones, VSC nos pedirá que ingresemos la información mencionada. Los recuadros azules en la *Figura 27* muestran la ubicación de las dos opciones para subir proyectos desde VSC:

Figure 27: Subir proyectos a GitHub desde VSC

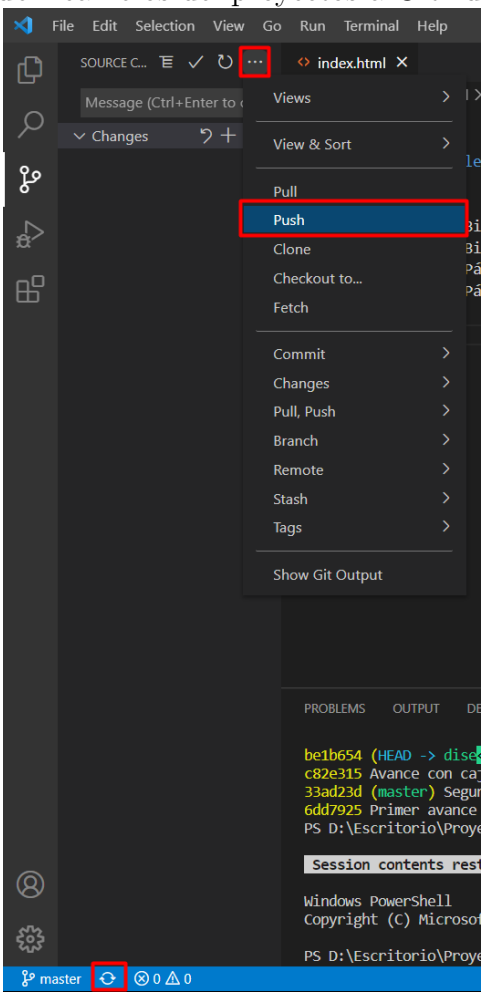


En caso de que, en tu navegador estés loggeado en GitHub, VSC solamente te pedirá que des clic a un botón para dar autorización a GitHub de poder trabajar en VSC y que ingreses tu contraseña, en caso contrario, pedirá que ingreses tu correo, contraseña, y des autorización.

Una vez iniciada sesión en GitHub con VSC, este último te preguntará a qué tipo de repositorio subir tu proyecto local: si a un repositorio público o privado, escoge alguna opción, posterior a ello, comenzará el proceso para subir el proyecto local a GitHub, debes tener en cuenta de que, cuando se suba, VSC creará un repositorio en GitHub con el mismo nombre del proyecto local y ahí lo publicará.

Si realizamos alguna modificación en el archivo *index.html* y realizamos un commit nuevo, para subir este cambio a GitHub, presionamos alguna de las dos opciones mostradas en la *Figura 28* para subir nuestros cambios, a esta acción se le conoce como **push**:

Figure 28: Subir cambios del proyectos a GitHub desde VSC



Si previamente ya teníamos un repositorio creada en GitHub donde deseamos que se suba nuestro proyecto local, podemos acudir a la **Paleta de comandos** para poder obtener acceso a los comandos que ya conocemos de Git de una forma más visual y fácil de comprender. Abrimos la *pPaleta de comandos* yéndonos al opción **View** en la barra superior de VSC, la primer opción desplegada es la que requerimos. Una vez aparece la barra, escribimos la palabra "git" y, en ese momento, se nos abren todos los comandos de Git permitidos en VSC, estando en ese menú, podemos buscar el comando **Git: Add Remote...**, ahí escribiremos la dirección del repositorio que ya tenemos creado, como vemos en la *Figura 29* y *30*.

Figure 29: Subir proyecto a GitHub con la Paleta de comandos

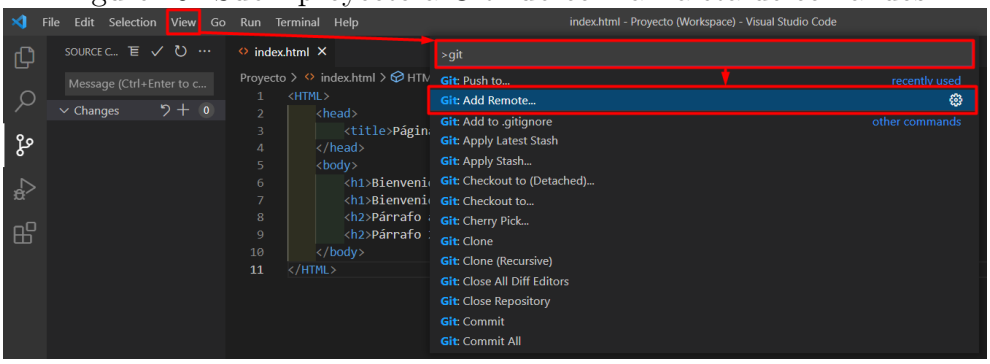
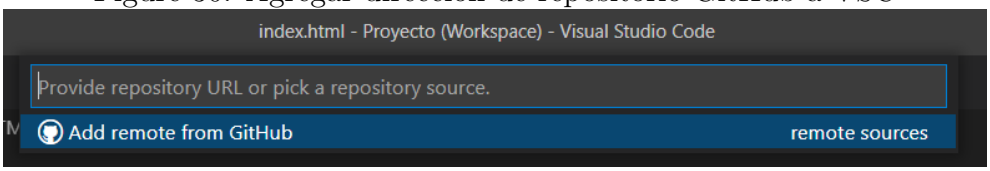
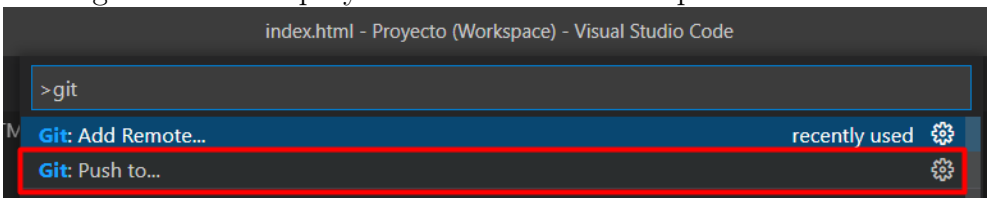


Figure 30: Agregar dirección de repositorio GitHub a VSC



Una vez agregado el repositorio a VSC, volvemos a abrir la Paleta de comandos, escribimos "git" y buscamos el comando **Git: Push To...** (*Figura 31*) y lo seleccionamos; en este punto, tenemos la dirección del repositorio que se creó automáticamente cuando agregamos nuestra cuenta de GitHub a VSC y la dirección del repositorio que acabamos de añadir, seleccionamos la dirección más reciente y el proyecto se habrá subido.

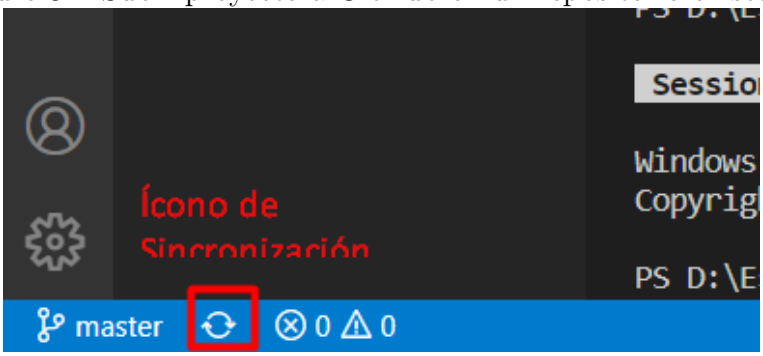
Figure 31: Subir proyecto a GitHub en un repositorio existente



Como vemos, para subir un proyecto local a GitHub primero debemos estar loggeados en VSC con nuestra cuenta de GitHub, con esa información, VSC detecta crea automáticamente un repositorio online con el mismo nombre que el proyecto local, subiendo ahí nuestro trabajo, sin embargo, si buscamos que se suba a otro repositorio, debemos agregar la dirección de este a VSC con el comando **Add Remote...** y, posterior a ello, utilizar el comando **Push To...**.

Sabemos que se pueden editar archivos en GitHub, si en nuestro archivo *index.html* le agregamos un párrafo desde GitHub y queremos que aparezca en VSC, simplemente debemos darle clic al icono de **sincronización** en la **barra de tareas** (*Figura 32*). Para que aparezca el cambio de VSC a GitHub, realizamos el commit con nuestros cambios y realizamos un push a el repositorio que deseemos.

Figure 32: Subir proyecto a GitHub en un repositorio existente

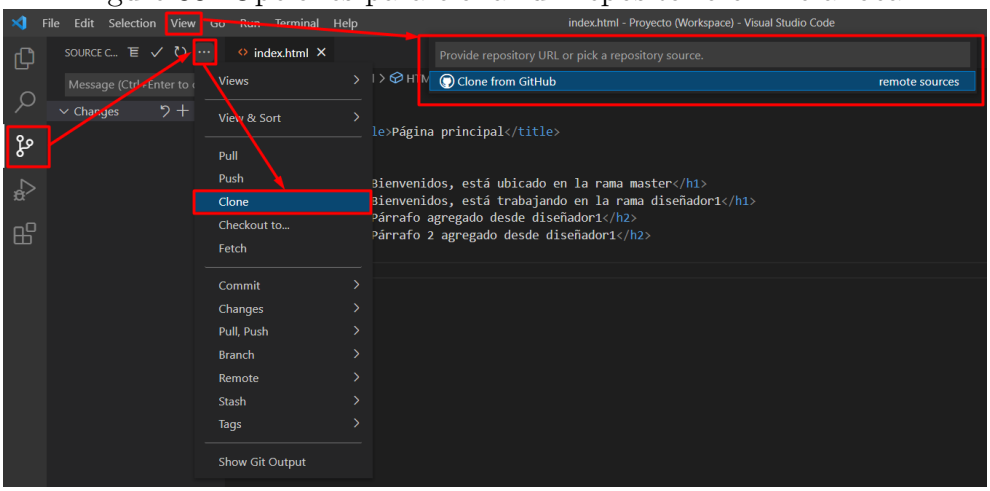


Si queremos ahorrarnos el paso de la creación de un repositorio automático para, posterior, agregar la dirección del repositorio que realmente queremos trabajar, siga los pasos de la siguiente sección.

5.6 Clonación de proyectos

Con la dirección del repositorio que queremos pasar de GitHub a VSC, nos vamos al editor de texto, en el **Panel de tareas de Git** o **Paleta de comandos** buscamos la operación **Clone**, como vemos en la *Figura 33*.

Figure 33: Opciones para clonar un repositorio online a local



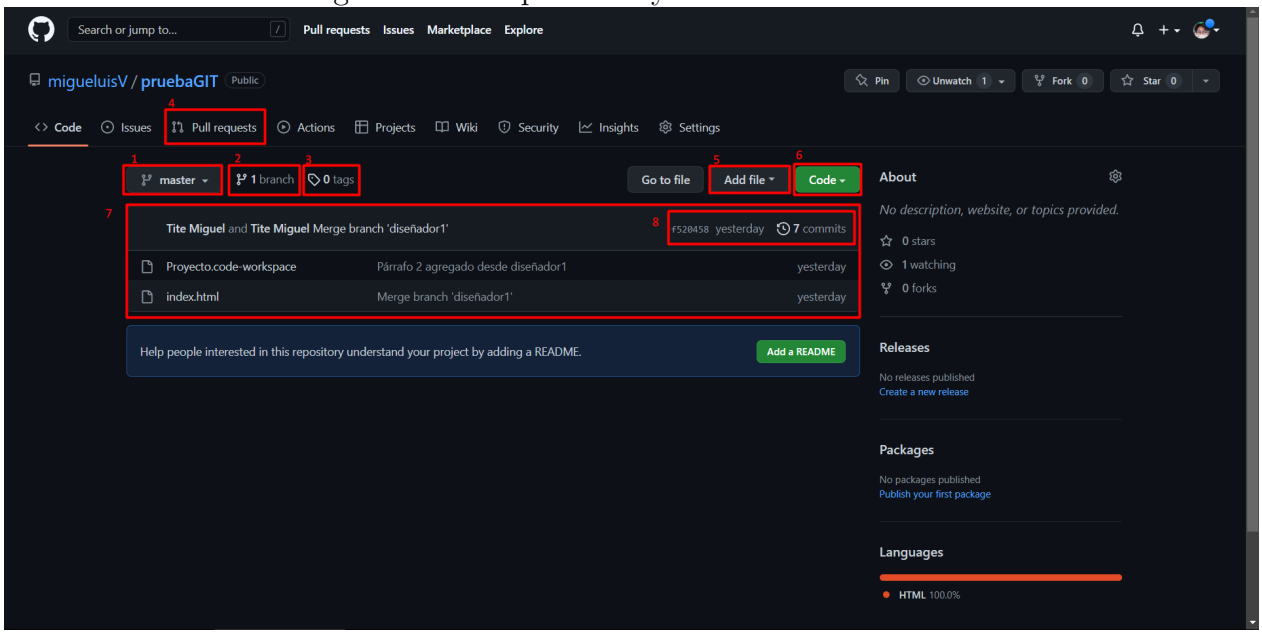
En ambas situaciones, debemos escribir en la caja de texto la dirección **.git** del repositorio online que buscamos clonar, posterior a ello, VSC nos mostrará un Explorador de archivos para seleccionar una carpeta donde guardar todos los archivos del repositorio online y, finalmente, comenzará el proceso de clonación, después de esperar un poco, tendremos el repositorio online en nuestra máquina.

6 Trabajando en GitHub

Nos alejaremos un poco de VSC, en esta sección, simplemente vamos a destacar algunas

herramientas o funciones que nos ofrece GitHub en su plataforma en línea. Ingresamos a Github e iniciamos sesión en nuestra cuenta, accedemos a cualquier repositorio en línea con el que contemos, en esta ocasión, mostraremos como prueba el repositorio que hemos utilizado previamente en estos apuntes (pruebaGIT), la *Figura 34* muestra algunos de sus funciones y componentes:

Figure 34: Componentes y funciones de GitHub



Donde:

i. muestra la rama donde actualmente estamos trabajando, muestra todos los archivos de la rama.

ii. muestra la cantidad de **ramas** existentes en el repositorio.

iii. muestra la cantidad de **etiquetas** existentes en el repositorio.

iv. muestra las **solicitudes de mezcla de ramas** pendientes por revisar y aprobar.

v. opción para **agregar un archivo** de nuestra computadora o crearlo en GitHub.

vi. muestra la dirección **.git** de nuestro repositorio.

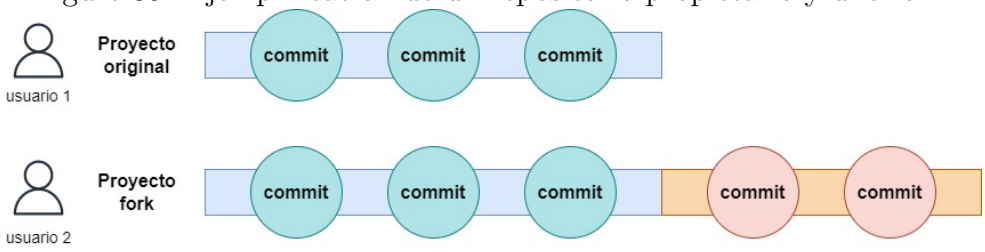
vii. muestra todos los archivos de una rama del repositorio.

viii. muestra la **cantidad de commits** que posee la rama.

7 Fork

Si recordamos el concepto de *ramas*, sabemos que es la copia de los archivos de nuestro proyecto para que más de una persona pueda trabajar en nuestro repositorio, al final, se mezclan ambas ramas para que se unifique el proyecto, entonces, el concepto de **fork** va más allá de una rama, sigue la misma lógica que las ramas, pero a escala de repositorios. Un *fork* es la copia del repositorio de un usuario en nuestra cuenta o cuenta de algún usuario que se haya visto interesado en el repositorio que busca copiar, el usuario original puede realizar commits y modificaciones en su trabajo original, mientras que el usuario que tiene una copia del repositorio puede clonarlo y hacer lo mismo, el usuario con el repositorio fork puede realizar una solicitud de merge al usuario propietario y este puede aceptarla, comentarla, revisarla, compararla o declinarla. La *Figura 35* muestra como se ven un repositorio propietario y uno fork:

Figure 35: Ejemplificación de un repositorio propietario y uno fork



Si vamos a un repositorio nuestro en GitHub en la esquina superior derecha, aparece un botón con el texto **Fork** (*Figura 36*) y *37*:

Figure 36: Ubicación del botón Fork en GitHub

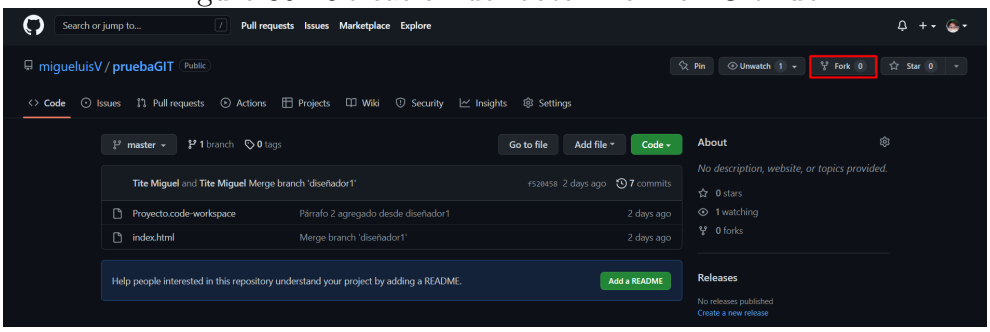
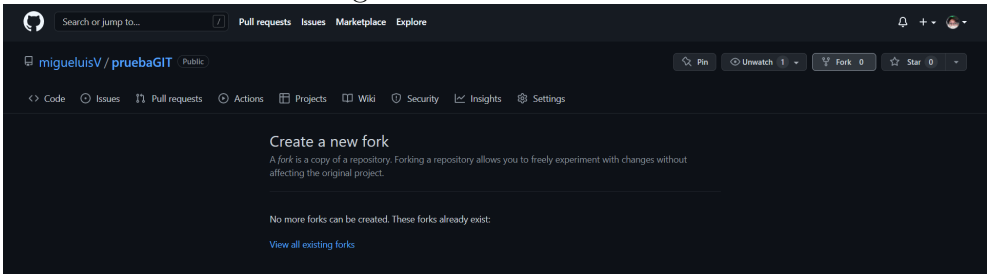


Figure 37: Menú de Fork



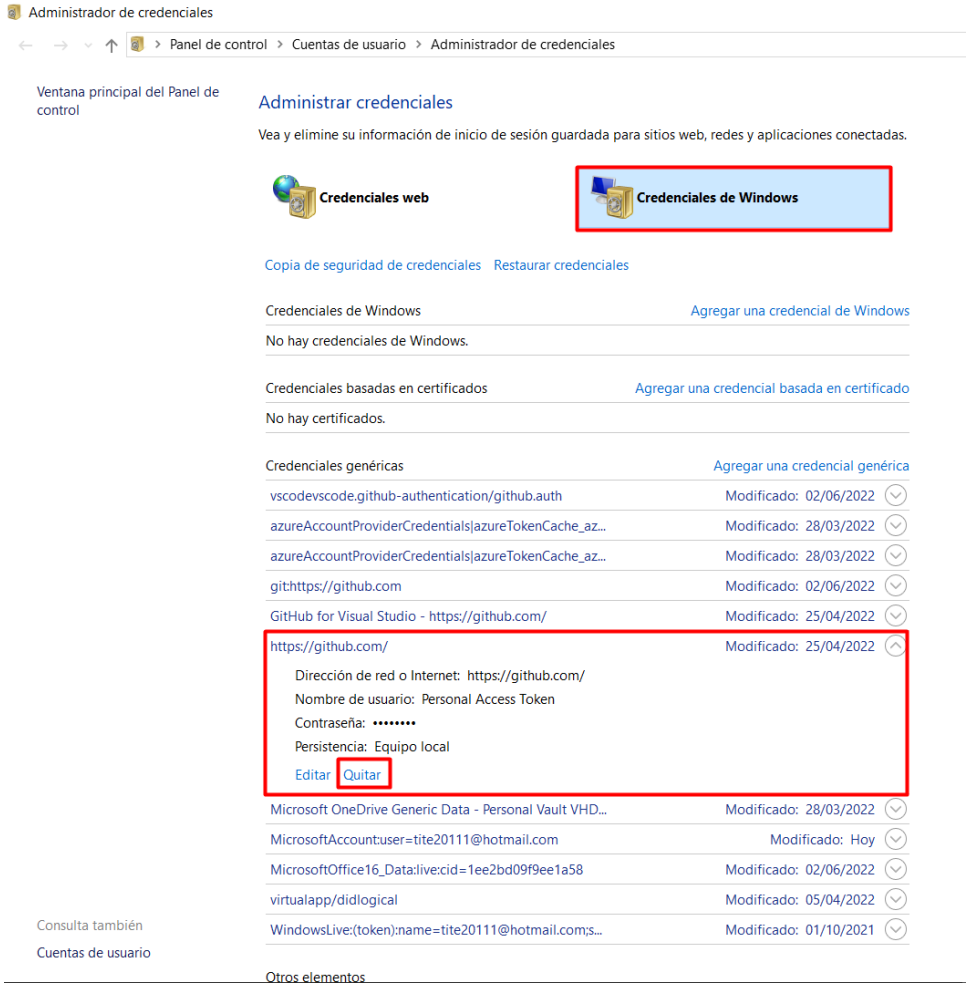
Como es nuestro propio proyecto, no podemos crear un fork, sin embargo, si estuviésemos en una cuenta distinta, buscamos nuestro usuario, y buscamos el repositorio en que estamos interesados, el mismo botón **Fork** está ubicado donde mismo, le damos clic y debe aparecer un texto para realizar un *fork* a los repositorios de la cuenta externa.

Algo que se debe tomar en consideración es que previamente, a lo largo de este documento, ingresamos las credenciales de GitHub de una cuenta (cuenta A), ya sea en la consola o en VSC para poder hacer **push** y **commits** de nuestros proyectos a GitHub, entonces, la información de la cuenta A fue almacenada en el sistema donde estemos trabajando, en nuestro caso, Windows, así que, si realizamos un fork de un repositorio A de la cuenta A a una cuenta B (repositorio B) en GitHub, debemos de salir de la cuenta A para ingresar como la cuenta B en nuestra computadora para trabajar el repositorio B, para hacer esto, debemos **eliminar las credenciales** del sistema, revise el siguiente subtítulo.

7.1 Eliminando credenciales de GitHub de Windows

Damos clic en el botón de **Windows**, abrimos la **Configuración**, escribimos en la barra de búsqueda "credenciales" y abrimos el programa **Administrador de credenciales**, abierto el programa, damos clic a la opción **Credenciales de Windows** y buscamos aquella que se llame **git:https://github.com**, simplemente damos clic sobre esta credencial, se abre un pequeño menú y damos clic a la opción **Quitar**, como se ve en la *Figura 38*.

Figure 38: Eliminando credencial de GitHub en Windows



Con esto, podemos volver a iniciar sesión en Git o VSC con una cuenta B o diferente a la cuenta A y, como resultado, poder trabajar los repositorios de la cuenta B o repositorios *fork*.