

Aprendiendo a Utilizar Git

1 Inicio de un proyecto

- Para crear un archivo .js en una carpeta utilizamos 'Git Bash'. Primero debemos posicionarnos en la carpeta. Esto lo podemos hacer utilizando el comando:

```
cd <ruta de la carpeta>
```

(Es util utilizar el comando ls para listar los elementos de uan carpeta)

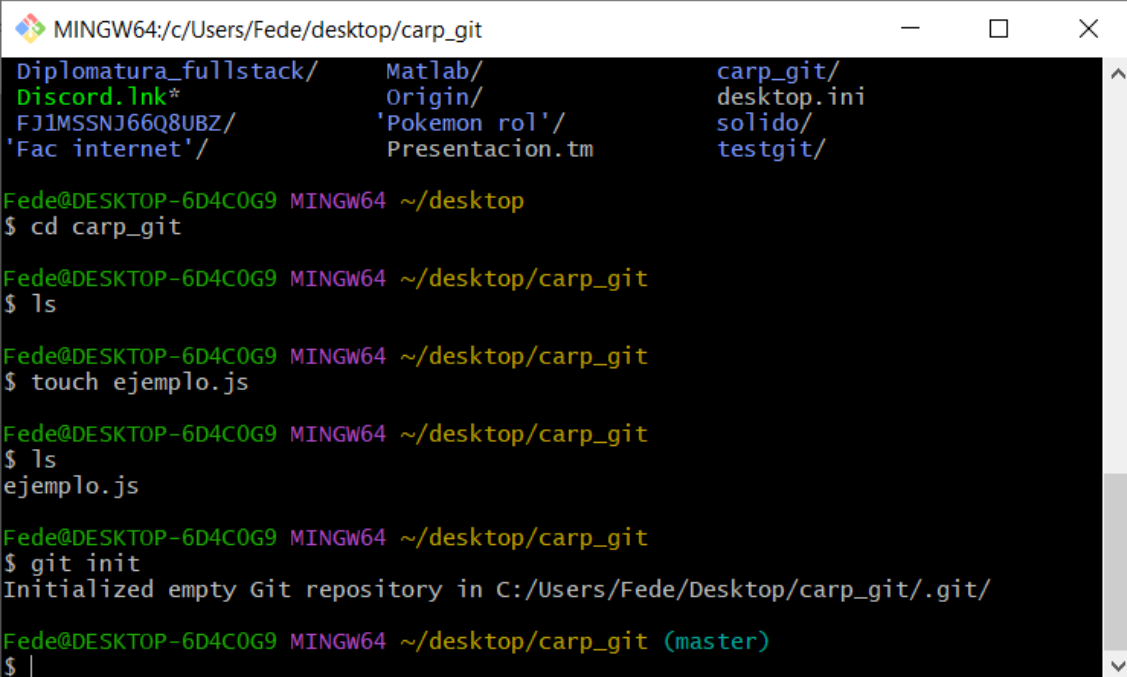
- Ahora si creamos el archivo .js utilizando el comando:

```
touch <nombre_archivo>.js
```

- Para inicializar el proyecto git es necesario crear la carpeta .git Esto puede hacerse utilizando 'Git Bash' con el comando:

```
git init
```

Esto crea la carpeta .git como **elemento oculto**



```
MINGW64:/c/Users/Fede/desktop/carp_git
Diplomatura_fullstack/  Matlab/  carp_git/
Discord.lnk*           Origin/  desktop.ini
FJ1MSSNJ66Q8UBZ/      'Pokemon rol'/  solido/
'Fac internet'/        Presentacion.tm  testgit/

Fede@DESKTOP-6D4C0G9 MINGW64 ~/desktop
$ cd carp_git

Fede@DESKTOP-6D4C0G9 MINGW64 ~/desktop/carp_git
$ ls

Fede@DESKTOP-6D4C0G9 MINGW64 ~/desktop/carp_git
$ touch ejemplo.js

Fede@DESKTOP-6D4C0G9 MINGW64 ~/desktop/carp_git
$ ls
ejemplo.js

Fede@DESKTOP-6D4C0G9 MINGW64 ~/desktop/carp_git
$ git init
Initialized empty Git repository in C:/Users/Fede/Desktop/carp_git/.git/

Fede@DESKTOP-6D4C0G9 MINGW64 ~/desktop/carp_git (master)
$
```

Figura 1. Tras ejecutar el comando Git init obtenemos como respuesta que se ha inicializado un repositorio git vacio.

- Utilizando el comando:

```
code .
```

Abrimos una instancia de virtual studio. Esta instancia nos muestra la carpeta junto con nuestro archivo .js

- Abrimos una nueva terminal desde la pestaña 'Terminal' en la instancia de Code. Despues de abrirla podemos agregar una terminal de git.

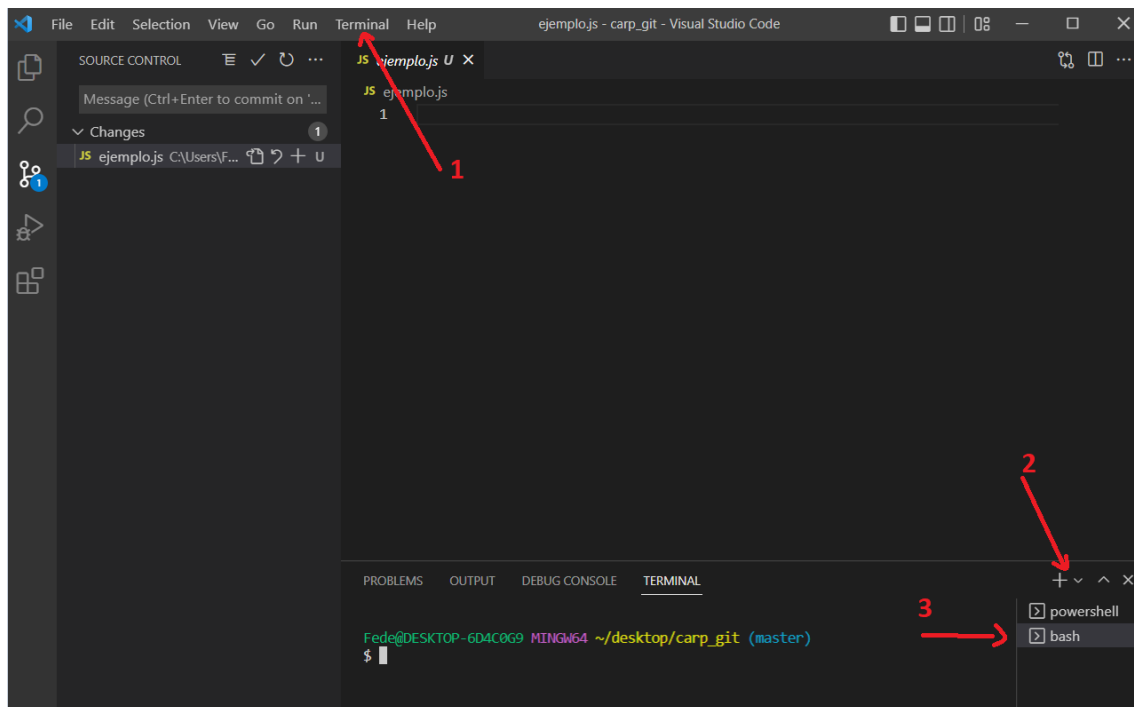


Figura 2. Apertura de una terminal de Git. Esta instancia de terminal es la misma que la que habíamos abierto anteriormente con git bash

- El comando:
git status
nos da información sobre el trabajo que estamos realizando en git. Nos indica en qué rama estamos trabajando y si hay 'commits'.

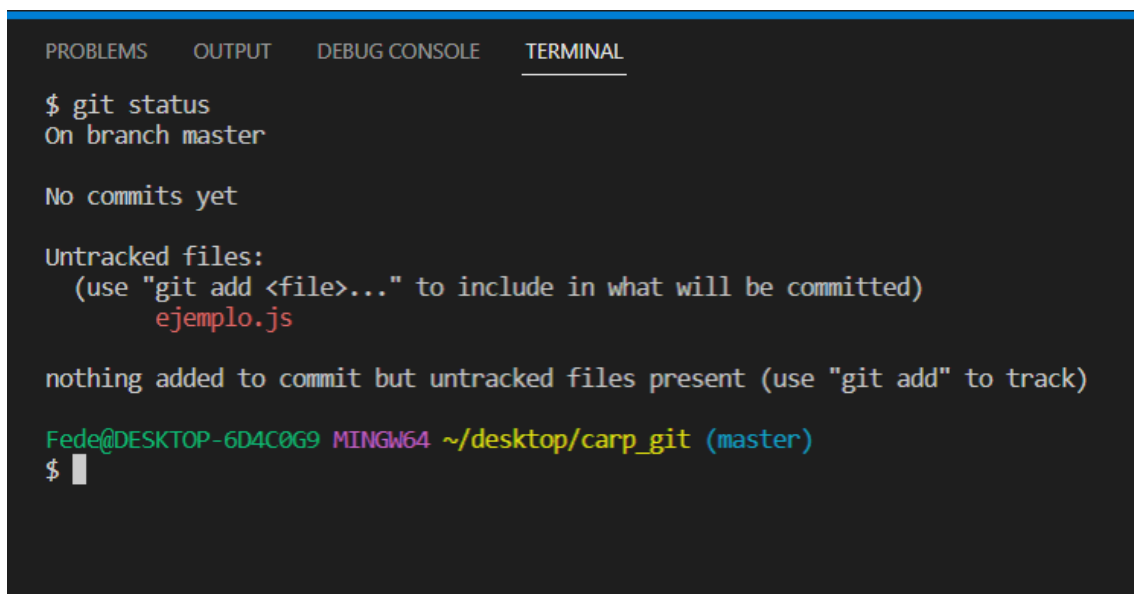


Figura 3. Al utilizarlo por primera vez observamos que se nos indica que no hay 'commits' todavía. Como sugerencia se nos da que incluyamos un archivo para ser committed utilizando el comando: git add <archivo>

- En las extensiones se nos aconseja instalar: git graph

Para ello nos vamos a la barra izquierda de code, a la ultima pestaña inferior denominada 'Extensions' y en el buscador escribimos git graph.

- La herramienta de git graph nos permite visualizar de forma grafica los sucesivos pasos que vamos realizando en git. Para inicializarla debemos ir a la barra inferior (color azul) y hacer click donde dice git graph

- Otra herramienta aconsejable es: git lens

Las extensiones aparecen en la barra lateral izquierda.

- Inicializamos el 'tracking' de un archivo utilizando la sugerencia que nos da la terminal de git (Figura 3).

```
git add <archivo>.js
```

- Si ahora utilizamos el comando:

```
git status
```

Veremos que se esta haciendo un seguimiento del archivo

- Podemos agregar mas archivos .js utilizando la pestaña para agregar archivos que aparece en la carpeta de git en visual studio. Creamos otro archivo .js

Si comparamos los dos archivos existentes, veremos que asociado a un archivo hay una letra A mientras que al otro, en este caso el nuevo archivo creado, se asocia una letra U. La letra U nos indica que no se esta haciendo seguimiento de este archivo, por lo cual, para seguirlo debemos ejecutar el mismo codigo que nos permitio hacer seguimiento del otro archivo:

```
git add <archivo2>.js
```

- Un archivo categorizado como A , se dice que se encuentra en la 'Staging area'

- Podemos dejar de hacer seguimiento de un archivo utilizando el comando:

```
git rm --cached <file>
```

- Para 'commitear' un archivo desde la ventana de comandos podemos escribir:

```
git commit -m '<algo>'
```

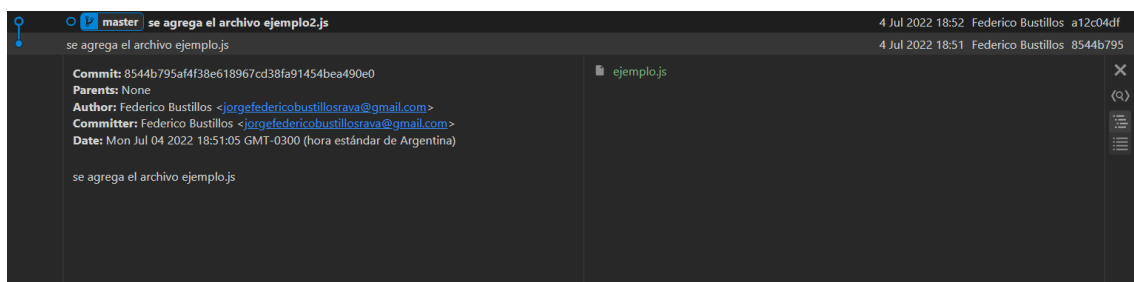


Figura 4. Ejemplo de muestra de los archivos commiteados utilizando git graph en visual estudio. El primer cambio se registra en el punto inferior (punto azul bien marcado.) Al hacer click podemos ver el nombre, que es el caracter alfanumerico largo que se ve. El autor y otros detalles.

Vamos agregando mas cambios a los archivos para ver las funcionalidades.

- En el archivo 'ejemplo.js' agregamos la siguiente linea:

```
let a = 1;
```

En cuanto agregamos esta linea nos aparece un comentario al lado de ella , acerca de que tenemos un cambio sin comentar.

- Antes de poder 'commitear' el cambio, tenemos que 'tagearlo'. Para ello vamos a la pestania que esta a la izquierda 'Source Control' Ahi podremos ver que se encuentra nuestro archivo con el cambio y a su lado la letra M.

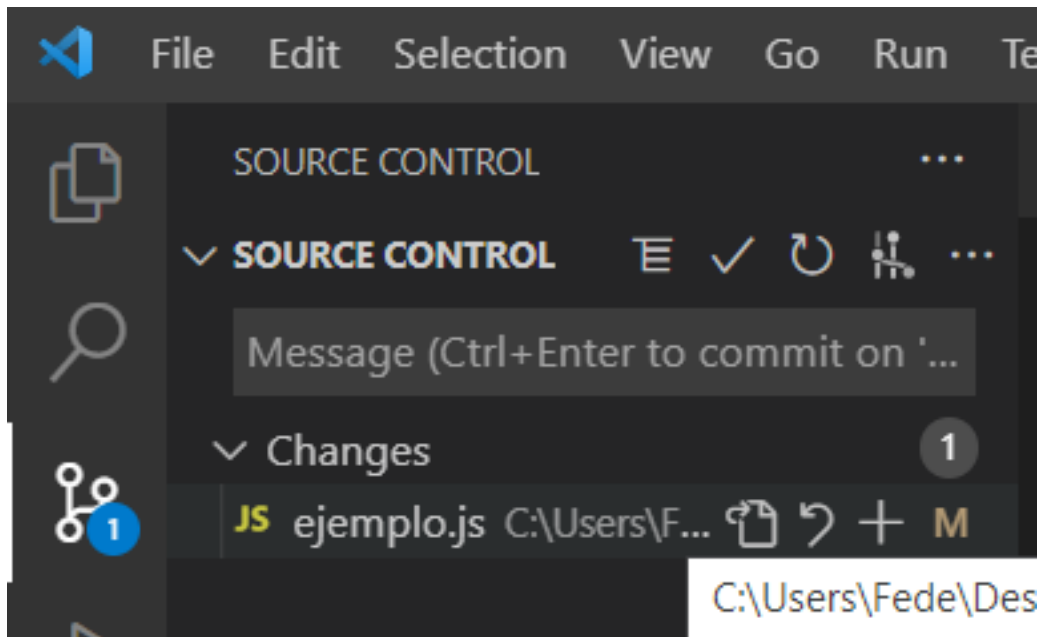


Figura 5. Si acercamos el puntero del raton a lo mostrado en 'Changes' podremos ver que nos aparecen 4 simbolos. Si hacemos click en '+' se nos dara la opcion de 'Stage'. Hacemos click en ella.

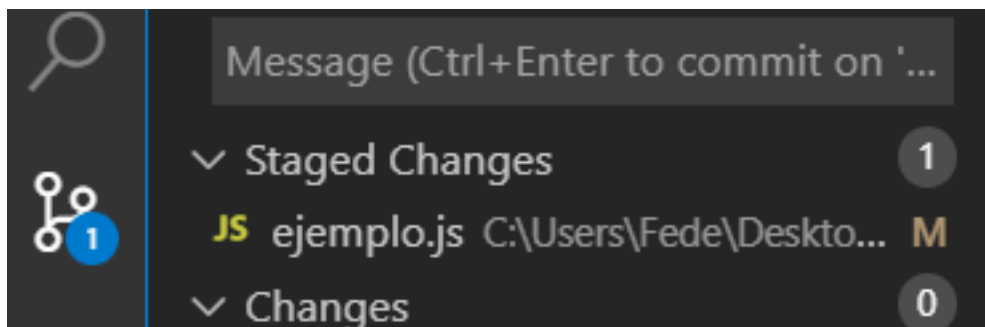


Figura 6. Ahora nos aparece una pestania que dice 'Staged Changes' Es en este momento que podemos commitear el cambio.

- Una vez que se hizo 'staged' el cambio, podemos hacer el 'commit' utilizando los comandos:
git commit -m '<texto>'

Entonces el procedimiento para agregar un commit es el siguiente:

1. Se hace un cambio, ya sea en un archivo o bien agregando un archivo
2. Tenemos que hacer 'staged' el cambio
3. Una vez 'stageado' lo comiteamos utilizando el comando:
git commit -m '<texto comentario>'

Es importante prestar atencion a la informacion otorgada en la Figura 4. Entre ellas se nos indica el archivo padre del cambio. Es decir el archivo que se modifico agregandole los cambios. Uno puede visualizar este archivo, sin los cambios. Simplemente hacemos click en el cambio de interes, a la derecha nos aparece en verde el nombre del archivo. Si acercamos el puntero y hacemos click en 'view file at this revision' podremos ver el estado del archivo antes de los cambios.

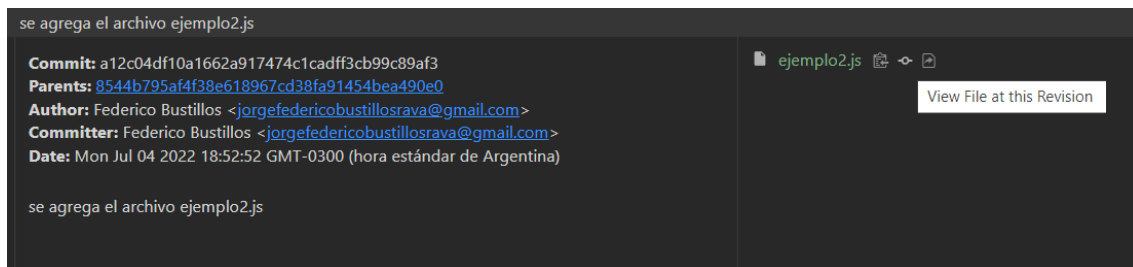


Figura 7. Como se explico para un dado cambio podemos ver la informacion relacionada con este.

Segun se indica, se pueden agregar varios cambios y que todos ellos correspondan a un solo commit.

2 El comando Checkout

Segun pude entender el comando Checkout actualiza toda la rama de trabajo en un determinado punto del arbol. Basicamente nos cambia la cabeza del arbol. La cabeza esta representada por el punto no coloreado, o bien el punto con un orificio negro.

De acuerdo a la siguiente fuente: [https://git-scm.com/docs/git-checkout]

Updates files in the working tree to match the version in the index or the specified tree. If no pathspec was given, git checkout will also update HEAD to set the specified branch as the current branch.

[https://www.atlassian.com/es/git/tutorials/using-branches/git-checkout]

3 Subir un trabajo a GitHub

- Creamos una cuenta en GitHub
- Creamos un repositorio nuevo
- Vamos al repositorio y en la pestania 'code' hacemos click . Abajo nos da un link HTTPS a el repositorio, el cual copiamos.

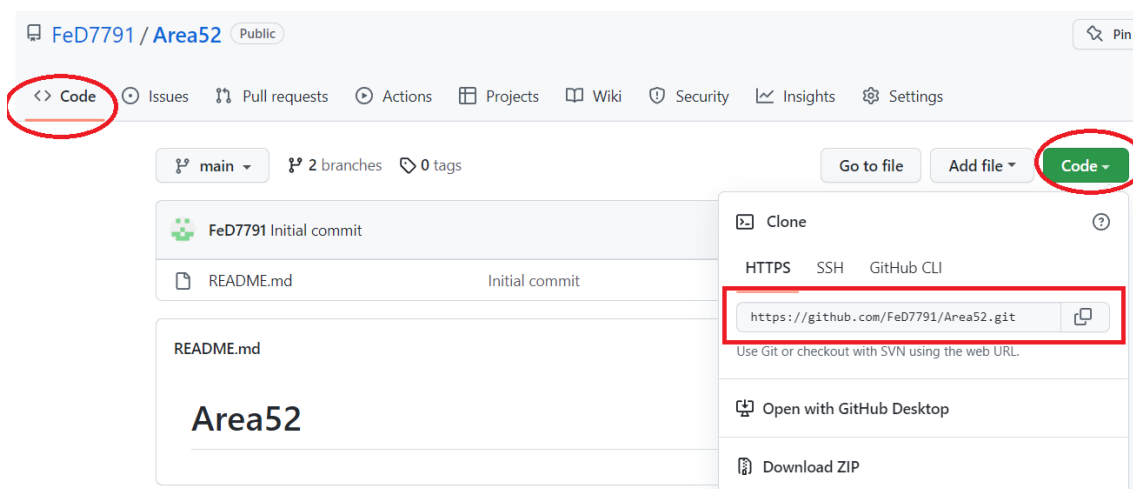


Figura 8.

- Ahora hacemos la conexion con el repositorio:

```
git remote add origin https://github.com/<usuario>/<nombre_repositorio>.git
```

- Para subir los archivos utilizamos el siguiente comando:

git push origin master

Nota, en realidad por master ponemos el nombre de la rama que estamos trabajando

Es probable que debamos acreditar credenciales de permiso

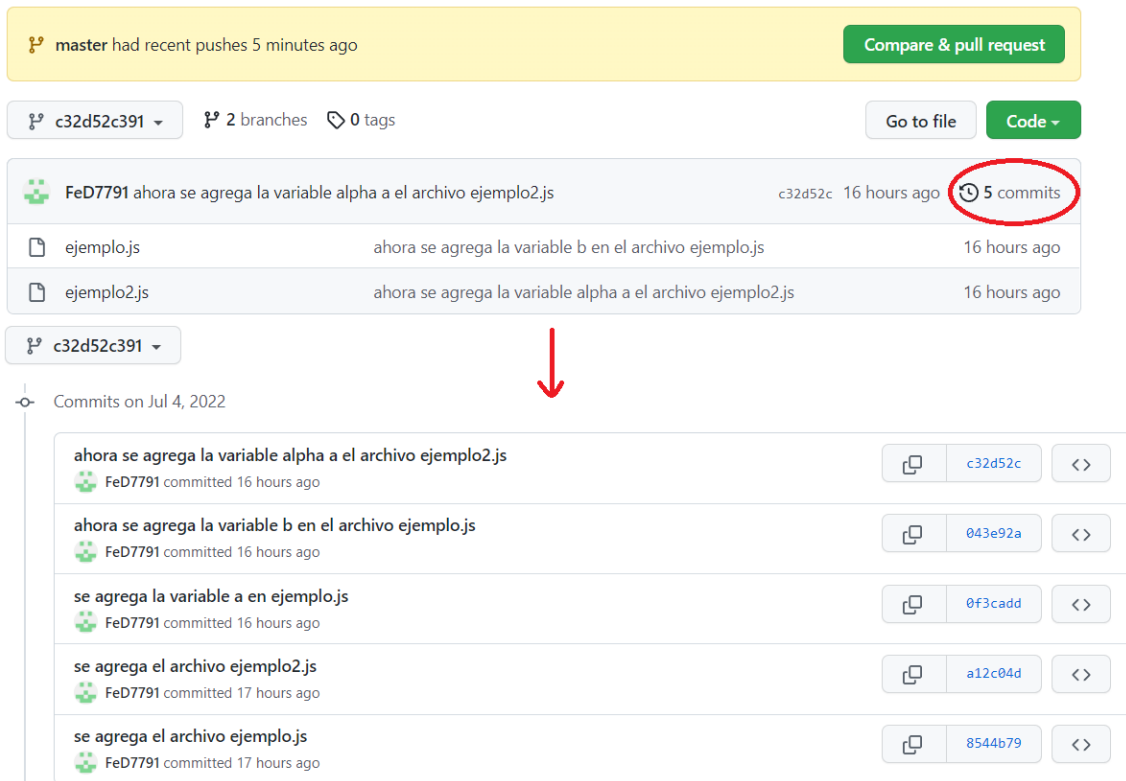


Figura 9. Podemos conseguir informacion adicional, la misma que vemos en visual studio, haciendo click en los commits. Puede verse entonces la totalidad de los cambios realizados en la rama.

3.1 git push

- Este comando se utiliza para cargar el contenido de un repositorio local a un repositorio remoto. El comando es el siguiente:

```
git push <remote> <branch>
```

Mas informacion [<https://www.atlassian.com/es/git/tutorials/syncing/git-push>]

3.2 Archivo .gitignore

Este es un archivo de texto que le dice a Git que archivos o carpetas ignorar en un proyecto. Cualquier entrada en el archivo se ignorara en los repositorios Git.

```
# Ignora archivos del sistema Mac
.DS_store

# Ignora la carpeta node_modules
node_modules

# Ignora todos los archivos de texto
*.txt

# Ignora los archivos relacionados a API keys
.env

# Ignora archivos de configuración SASS
.sass-cache
```

Figura 10. Este es un ejemplo de como puede lucir un archivo gitignore.

Mas informacion aca [<https://www.freecodecamp.org/espanol/news/gitignore-explicado-que-es-y-como-agregar-a-tu-repositorio/#:~:text=gitignore%20%2C%20es%20un%20archivo%20de,puedes%20crear%20un%20archivo%20global%20.>]

Para crear este archivo desde visual studio simplemente creamos un archivo nuevo y le damos el nombre:

.gitignore

3.3 git remote

El comando:

git remote

Nos permite crear, ver y borrar conecciones con otros repositorios. Mas informacion en el siguiente link:

[<https://git-scm.com/docs/git-remote>]

Existen opciones para mayor verbosidad: -v o bien - - verbose(Todo junto)

3.4 Editor web

En la primera imagen de la Figura 9, si hacemos click en ' . ' (el punto en el teclado), iremos a un editor de texto en git. Alli podremos seguir editando los archivos. Esta es una instancia igual a visual studio.

Aqui trataremos de agregar un cambio y de commitarlo:

- Hacemos un cambio en algun archivo
- Nos aparecera la letra M, vamos a la barra izquierda 'Source control' y en el archivo modificado le damos al '+' para stagear el cambio.
- Como observacion, se nos dira que no estamos parados en ninguna rama, esto podremos hacerlo desde la barra inferior.
- En mi caso, por desconocimiento yo termine haciendo un cambio y subiendolo en una nueva rama creada 'ramal'. Despues haciendo click en la barra inferior (el circulo para sincronizar). Subi la rama al repositorio. Esto seria hacer un push

3.5 Agregar colaborador

- Nos vamos a algun repositorio

- En la barra de menus hacemos click en 'settings'
- Al abrirse esa barra hacemos click en 'general' que despliega un menu
- Entre las opciones del menu vemos a 'Collaborators'
- Al hacer click podremos buscar a la gente a traves de nombre de usuario o email.

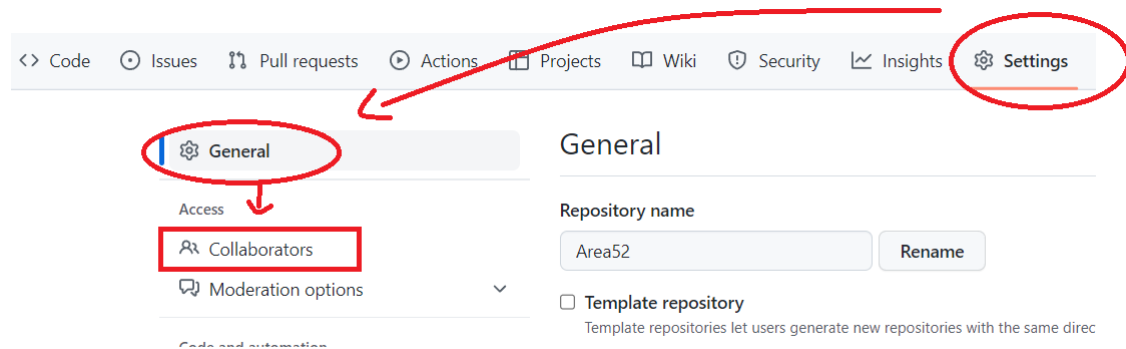


Figura 11.

4 Fetch, descargar ramas desde Github

- Estando en Visual studio, podemos ver que en la barra horizontal superior, del lado derecho hay el icono de una nube y al acercar el puntero a este, nos dice:
Fetch from remote(s)
- Al hacer click sobre este, se nos actualizan los cambios realizados en GitHub. Aqui podemos ver que en particular aparece la nueva rama creada, 'rama1'

4.1 git pull

- Este comando descarga los nuevos cambios hechos en el repositorio remoto y los integra a la rama HEAD local. (Es decir los integra a la cabeza). Estos cambios usualmente se hacen a traves de un 'Merge' pero tambien puede elegirse hacerlos a traves de un 'rebase'

```
git pull origin master --rebase
```

- Una vez que se ha establecido un seguimiento entre el repositorio local y el remoto, el comando:

```
git pull
```

Traera automaticamente los cambios

4.2 git fetch

- Si utilizamos este comando, solamente descargaremos los cambios y se dejara la HEAD y los archivos copiados sin cambios.

```
git fetch origin
```

4.3 git clone

- Para copiar un repositorio remoto utilizamos el comando git clone:

```
git clone <url_repositorio>
```

Para conseguir la url, podemos hacerlo aca:

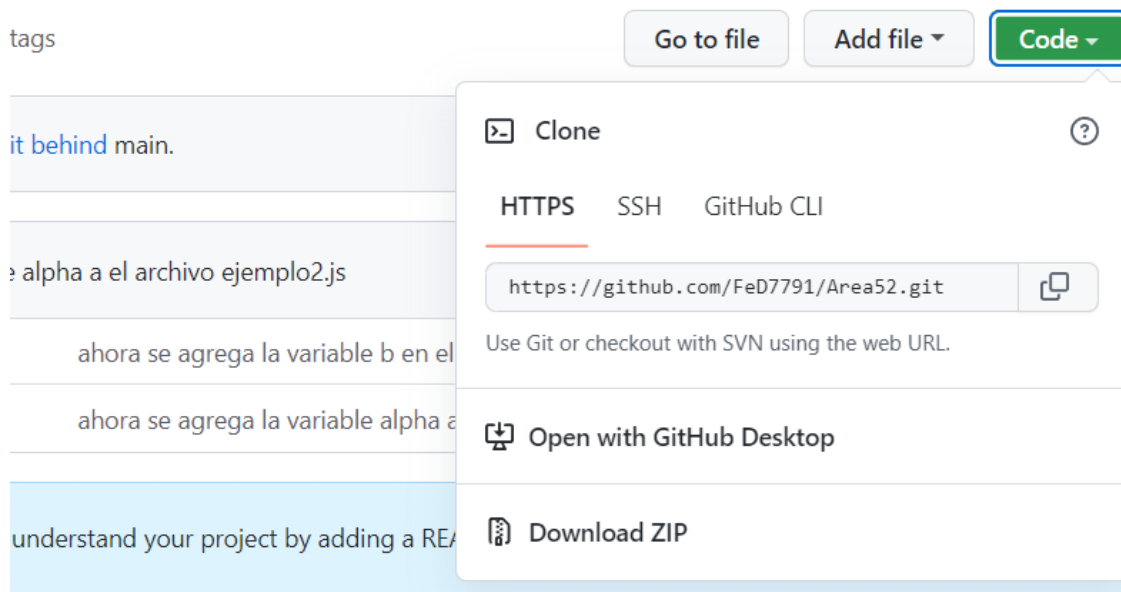


Figura 12. En gitHub tenemos en la pestania code del repositorio la url de nuestro repositorio.

Es importante que desde la terminal estemos ubicados en la carpeta donde queremos copiar el repositorio.

La conexcion entre el reopositorio clonado y el web esta establecida directamente al hacer la clonacion.

5 Ramas, creacion y gestion

5.1 Creacion de una nueva rama

Para crear una nueva rama, ejecutamos el comando:

```
git checkout -b <nombre_rama>
```

Otro comando utilizado es:

```
git branch <nombre>
```

A tenerse en cuenta, esta rama se crea en la posicion del head del codigo.

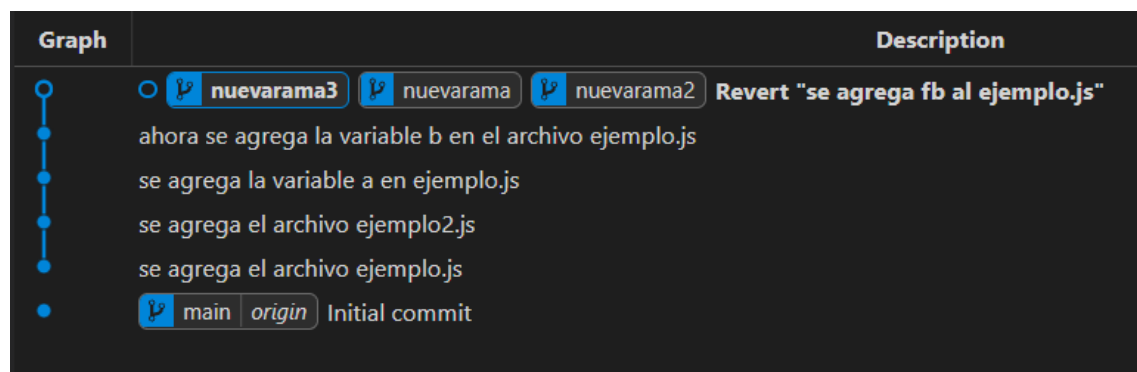


Figura 13. Puede verse en la figura, como en el 'HEAD' se han creado 3 ramas. Las distintas ramas estan una al lado de la otra, lo que significa que tienen la misma jerarquia (lo cual tiene sentido por que al crear la rama, se hace una especie de copia de la rama padre y sobre ella se haran las modificaciones que las diferenciarian)

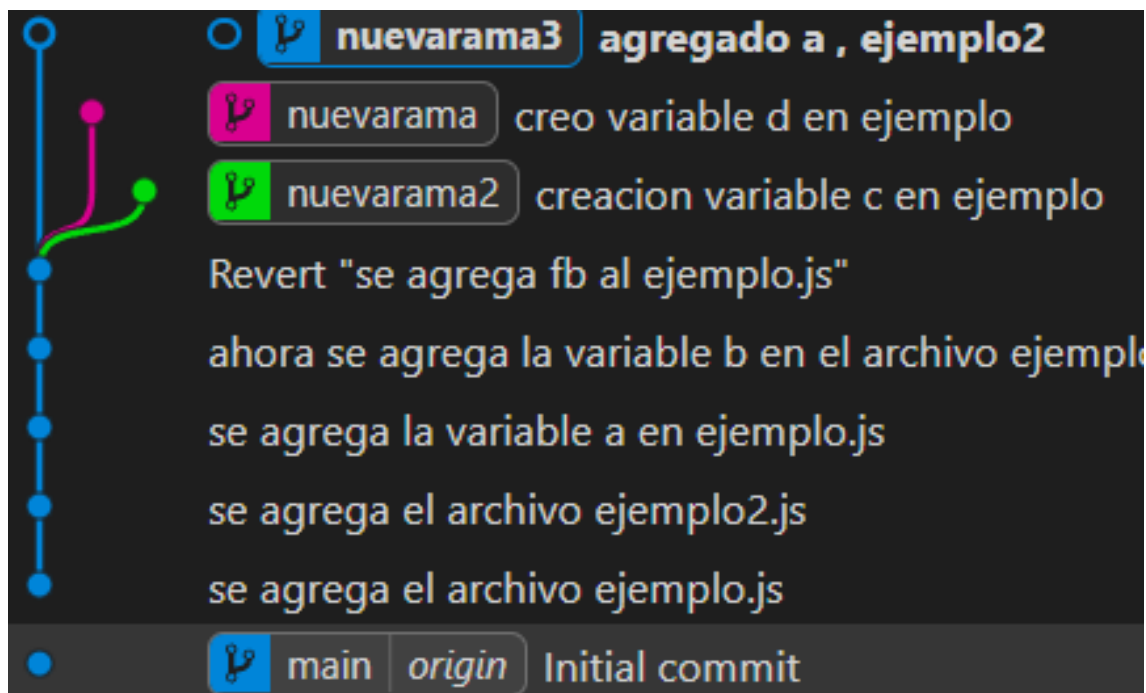


Figura 14. Se han creado 3 ramas, a través de hacer 3 modificaciones distintas. Solamente una de las ramas puede continuar como rama head.

5.2 Borrar una rama

Para borrar una rama, de manera grafica, tenemos que hacer click derecho sobre la solapa de la rama que queremos borrar y darle a eliminar.

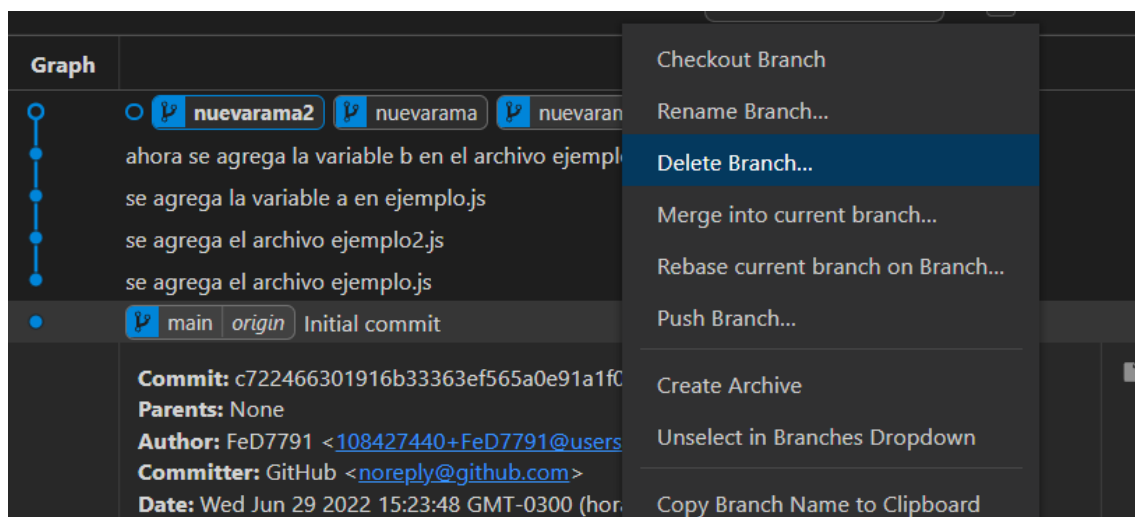


Figura 15. Lo importante a tener en cuenta antes de borrar la rama, es que este comando no se puede ejecutar sobre la 'HEAD'

5.3 Fusionar Rama

Para fusionar dos ramas podemos utilizar el comando:

```
git merge <nombre rama que quiero fusionar con la rama actual>
```

Como se indica en el comando, nosotros primero nos tenemos que parar en una rama. Sobre esta rama se traeran los cambios de otra rama.

Ejemplo:

```
git checkout rama1 #Aca me paro en la rama 1
```

```
git merge rama2
```

Trae los cambios de la rama2 a la rama1

5.4 Crear una rama en repo local y subirla a un remoto

```
git checkout -b <branch>
```

Edit files, add and commit. Then push with the -u (short for `--set-upstream`) option:

```
git push -u origin <branch>
```

6 gitignore

Para añadir archivos que no queremos seguir debemos crear primero el siguiente archivo, para ello hacemos:

```
>touch .gitignore
```

Dentro de este archivo agregamos la carpeta o archivos que no queramos seguir:

```
/
```

7 Resumen

En este resumen vamos a repasar el proceso de carga de un proyecto, que consiste en varios archivos que cargaremos a git hub, despues haremos un cambio y subiremos ese cambio a git, despues haremos un cambio en git y descargaremos ese cambio a el repositorio local. Fuente:

<https://docs.github.com/en/repositories/working-with-files/managing-files/adding-a-file-to-a-repository#adding-a-file-to-a-repository-using-the-command-line>

Nos ubicamos en la carpeta de interes y abrimos una terminal de git.

- Inicializamos un repositorio git local:

```
git init -b <nombre>
```
- Stagear los archivos en el repositorio:

```
git add .
```
- Comiteamos los cambios, en este caso, como tenemos todo un proyecto la descripcion sera para todo el conjunto.

```
git commit -m <descripcion>
```

Tener en cuenta que esto lo podemos hacer de forma grafica(esta es mi forma preferida.)

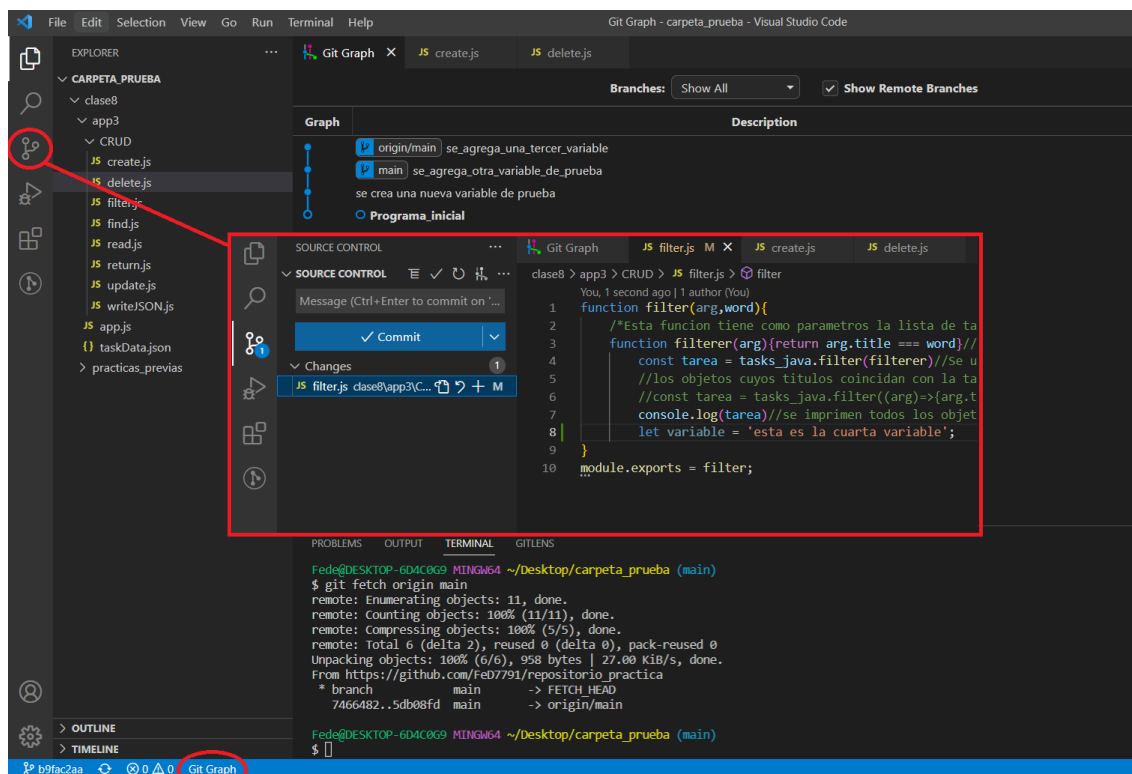


Figura 16. Esta figura explica el metodo grafico para stagear y commitear los nuevos cambios de un archivo. Ante cualquier nuevo cambio se generara una notificacion sobre el elemento indicado en rojo. Tambien se ha marcado donde se encuentra el icono de git graph para inicializarlo.

- `git remote add <algun nombre> <REMOTE_URL>` ; agrega el nuevo remoto en la direccion url de git hub
- `git remote -v` ; verifica el nuevo URL remoto
- `git push origin main` ; empuja los cambios al directorio URL remoto
- `git fetch origin main` ;descargamos los cambios [Solo los COMMIT OJO!] en el remoto al repo local
- Si quieres descargar el repositorio, lo mejor es clonarlo, para ello:
`git clone <url repositorio>`

Nota importante, cuando vos haces push o fetch, no necesariamente es `git fetch origin main`, en realidad por `main`, va el nombre de tu rama.

8 Procedimiento para descargar un repositorio desde github

Desde la terminal Gitbash

- >`git init`
- >`git clone <url>`
- >`git fetch` #(Esto descargaria todas las ramas)
- >`git branch` #Escrita asi lista la rama principal existente
- >`git branch -a` #Lista todas las ramas existentes, en rojo las que no se usan
- >`git checkout <rama_de_interes>`
- >`git checkout -b <nueva_rama>` #para crear y cambiar a una nueva rama
- >`git pull origin <rama_de_interes>` #Para asegurarnos que nuestra rama este actualizada

```
>git branch --delete <branchname> #Borrar una rama local
```

A veces sera necesario conectar con el repositorio remoto (web) sin tener que descargar su contenido. Esto puede surgir por ejemplo cuando se trabaja en un proyecto y se quiere subir el mismo mediante una rama o algo.

```
>git remote add <nombre_repo_remoto> <url_repo_remoto>
```

```
>git push -u <nombre_repo_remoto> <nombre_local_branch>#para pushear una rama del repo local al repo remoto. -u es de 'upstream'
```

Este procedimiento te genera un readme bastante piola.

Si obtenemos error:

```
error: remote <nombre_repo_remoto> exists.
```

Eso significa que ya esta seteado el remoto, simplemente usar el comando para pushear la rama al repo

8.1 Remover un repositorio remoto

Para ello utilizamos el comando:

```
git remote rm <remote-name>
```

Esto nos remueve un repositorio remoto que hayamos agregado utilizando el comando:

```
git remote add <nombre_repo_remoto> <url_repo_remoto>
```

Esto puede llegar a ser util. En concreto me sucedio que agregue un remoto, pero con el link errado, entonces como no existia habia error al pushear la rama. Al removerlo se borra el remote-name con el que esta enlazado ese url errado. Asi puedo volver a agregar el remoto nuevamente y de forma correcta.

8.2 Remover ramas locales y remotas

- Remover una rama de un repositorio remoto (web):

```
git push <remote_name> -d <remote_branch_name>
```
- Remover una rama de un repositorio local:

```
git branch - -delete <local_branch_name>
```

9 Como iniciar un nuevo proyecto

- crear el repositorio en github
- crear tu proyecto de angular por ejemplo
- ```
git init
```
- crear una rama: 

```
git checkout -b <nueva_rama>
```
- Hacer un cambio y comitear (Fundamental)
- Agregar repositorio remoto: 

```
git remote add <algun_nombre> <REMOTE_URL>
```
- Pushear la rama: 

```
git push -u <nombre_repo_remoto> <nombre_local_branch>
```

## 10 Revertir un Commit

- Primero Descargamos el repositorio a un ordenador local
- Revertimos los cambios localmente utilizando: 

```
git reset HEAD~
```
- Pusheamos devuelta los cambios con: 

```
git push origin +HEAD
```

Otras cosas utiles:

> git log : para ver el sumario de los ultimos commit, nos devuelve un codigo

> git checkout <codigo> : podemos volver al commit del codigo indicado

## 11 Caso de Emergencia, git reflog

En situaciones donde hiciste un commit, luego se te cambio de rama, y luego la perdiste puedes utilizar esto. Este comando permite recuperar el log de ramas borradas. Este problema suele surgir cuando estas parado en una rama temporal y luego te moves

- git reflog - -no-abbrev

Nos dara un listado completo de las ramas y los commits. Son codigos largos los nombres de las ramas.

- git checkout <nombre largo de la rama> : con esto volvemos a la rama anterior

## 12 Git avanzado

### 12.1 Rebase

Esto es super util.

- Estuviste trabajando en tu rama local y tenes cambios comiteados
- El remoto, por alguna razon, tiene cambios que no has incorporado en tu rama local.
- Ejecutar un git pull nos dara la clasica advertencia del merge conflict. Para salir de esta situacion podemos utilizar git rebase.

Comando : `git pull --rebase`

Que hace Git Rebase? El comando completo hace lo siguiente:

1. First, it performs a `git fetch` to get the latest changes from the remote branch you're tracking.
2. After fetching, instead of merging the remote changes with your local commits (which is the default behavior of `git pull`), it **rebases your local commits** on top of the remote branch. This means:
  - It temporarily **"undoes"** your local commits (but keeps them in the background).
  - Then, it applies the remote changes (as if they were committed first).
  - Finally, it **re-applies your local commits on top of the updated branch**, effectively placing your work on top of the changes from the remote branch.