
GIT
&
GITHUB
@blueslaboratory

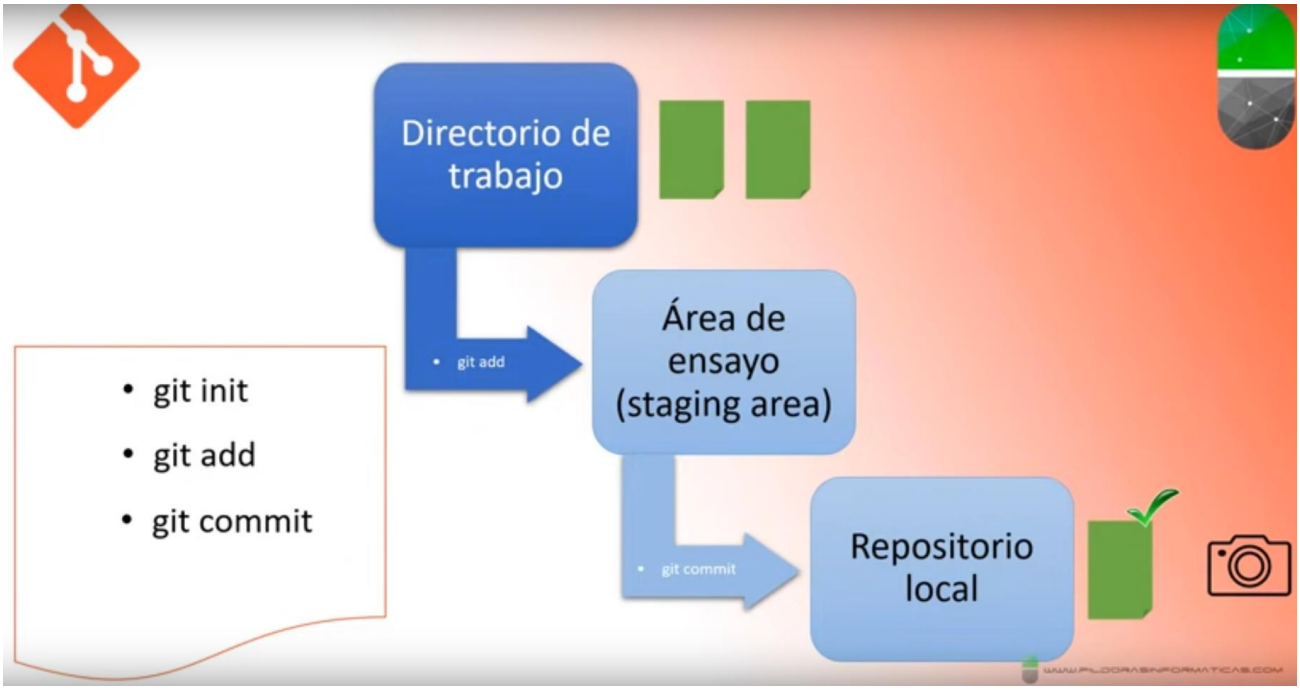
Curso GIT – Píldoras informáticas

Vídeo 1. Curso Git. Introducción

- **Qué es Git**: es un software libre de control de versiones.
- **Para qué sirve**:
  - Facilita el desarrollo y mantenimiento de aplicaciones tanto de forma individual como al trabajar en equipo ya que guarda un registro de todos los cambios que sufren los archivos.
  - Permite revertir los cambios de un archivo o varios a una instantánea de estos que haya sido guardada con anterioridad, esto es especialmente útil cuando trabajamos con frameworks que trabajan con cientos de archivos.
  - Facilita el trabajo en equipo de varios programadores en un mismo proyecto con la creación de ramas, líneas temporales/flujos de trabajo, que permiten a los programadores trabajar sobre un mismo archivo al mismo tiempo. Al terminar el trabajo estas ramas se pueden fusionar.
- **Descarga e instalación de Git**: https://git-scm.com/downloads

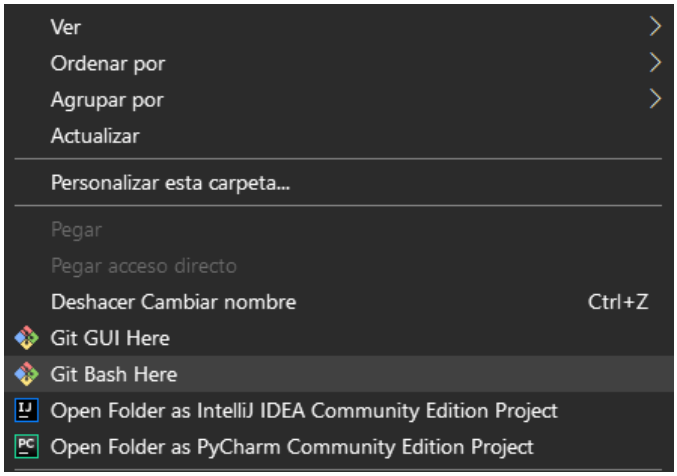
Vídeo 2. Curso Git. Creando repositorio

- **Crear repositorio:**



En la consola de Git podemos configurar el name y el email:
git config --global user.email "correocuentagithub@gmail.com"
git config --global user.name "blueslaboratory"

Para abrir la consola lo mejor es situarnos en el directorio en el que queremos comenzar a hacer el seguimiento, click con el botón derecho del ratón -> *Git Bash Here*



- **Agregar archivos y directorios al repositorio**
- **Respaldar archivos y directorios en el repositorio**

Documentación (todos los comandos): https://git-scm.com/docs/git#_git_commands

1. Cuando queremos empezar un seguimiento introducimos el comando:
   git init

2. Directorio de trabajo → Área de ensayo (staging area). Lleva el archivo al staging area y se olvida de que existen los demás archivos.
   git add "nombre_del_archivo.extension"
   Llevar al staging área todos los archivos:
   git add .

3. Descartar los cambios en el staging area después de hacer un add:
   git reset --hard

4. Área de ensayo (staging area) → Repositorio Local. Para llevar el archivo al repositorio local, donde se toma la instantánea del archivo, se utiliza el comando:
   git commit
   Se puede hacer un commit con una descripción:
   git commit -m "Comienzo del proyecto"

5. Comprobar el status del directorio de trabajo. Sólo te informa de los archivos que no están agregados al repositorio y que no tienen copia de respaldo, informándonos de si ni siquiera se le está haciendo un seguimiento o de si está en el área de ensayo. En cuanto se le hace un cambio al archivo, vuelve a aparecer:
   git status -s

6. Cómo puedo ver todos los commits, instantáneas en el repositorio local, que tengo:
   git log --oneline

7. Restaurar el archivo a la primera instantánea (borra las instantáneas anteriores) :
   git reset --hard "*código de la instantánea a la que queremos volver*"

```
Alejandro@W10 MINGW64 /e/ /Pildoras informáticas - GIT y GitHub/Testing GIT 2
(master)
$ git log --oneline
c199a42 (HEAD -> master) agregar titulo + primer parrafo
d25a3fe comienzo del proyecto testing git

Alejandro@W10 MINGW64 /e/ /Pildoras informáticas - GIT y GitHub/Testing GIT 2
(master)
$ git reset --hard d25a3fe
HEAD is now at d25a3fe comienzo del proyecto testing git
```

Al hacer **git log --oneline** podemos comprobar que se pierden las instantáneas posteriores a la versión restaurada:

```
Alejandro@W10 MINGW64 /e/ /Pildoras informáticas - GIT y GitHub/Testing GIT 2 (master)
$ git log --oneline
d25a3fe (HEAD -> master) comienzo del proyecto testing git
```

Vídeo 3. Curso Git. Subiendo a GitHub

- **Algunos comandos más:**

1. Para agregar al área de ensayo (staging area) todos los cambios:
   git add .

2. Hacer un add y un commit a la vez:
   git commit --am "mi súper comentario"

```
Alejandro@W10 MINGW64 /e/ /Pildoras informáticas - GIT y GitHub/Testing GIT 2 (master)
$ git status -s
AM css/testing-git.css
A  js/testing-git.js
 M testing-git.html

Alejandro@W10 MINGW64 /e/ /Pildoras informáticas - GIT y GitHub/Testing GIT 2 (master)
$ git commit -am "parrafo y tam fuente"
[master d876e95] parrafo y tam fuente
 3 files changed, 9 insertions(+), 1 deletion(-)
 create mode 100644 css/testing-git.css
 create mode 100644 js/testing-git.js

Alejandro@W10 MINGW64 /e/ /Pildoras informáticas - GIT y GitHub/Testing GIT 2 (master)
$ git log --oneline
d876e95 (HEAD -> master) parrafo y tam fuente
d25a3fe comienzo del proyecto testing git
```

```
Alejandro@W10 MINGW64 /e/ /Pildoras informáticas - GIT y GitHub/Testing GIT 2 (master)
$ git status -s
```

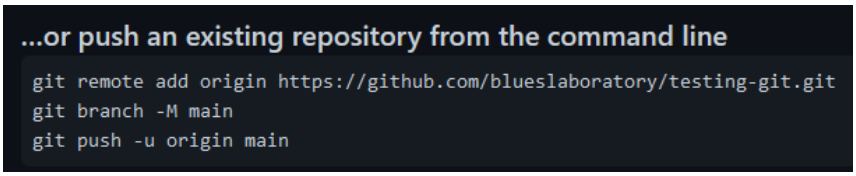
(no aparece nada en el status porque ya está todo agregado en el repositorio)

Si quiero hacer un commit con todos los archivos del staging área (es obligatorio poner un comentario):
git commit .

3. Abrir el editor VIM por si nos equivocamos al poner un mensaje en un commit:
   git commit --amend

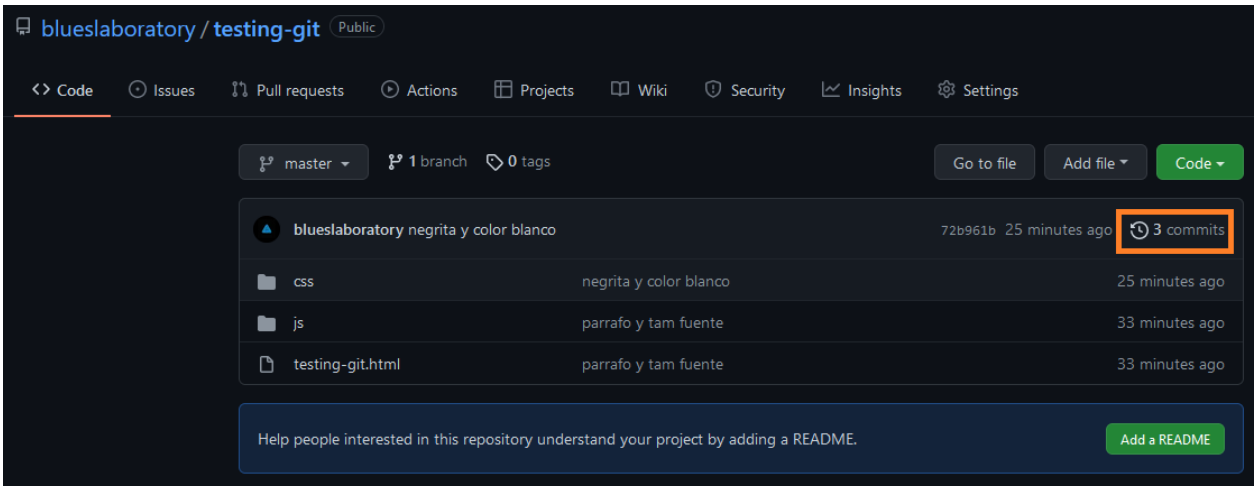
- **Subir un repositorio a GitHub:**

1. Creamos el repositorio en la página de GitHub: https://github.com/new
2. Una vez creado:



Nos interesan la primera y la tercera instrucción:
git remote add origin https://github.com/blueslaboratory/testing-git.git
git push -u origin main
git push origin master → (si no funciona: **git push -u origin main**)

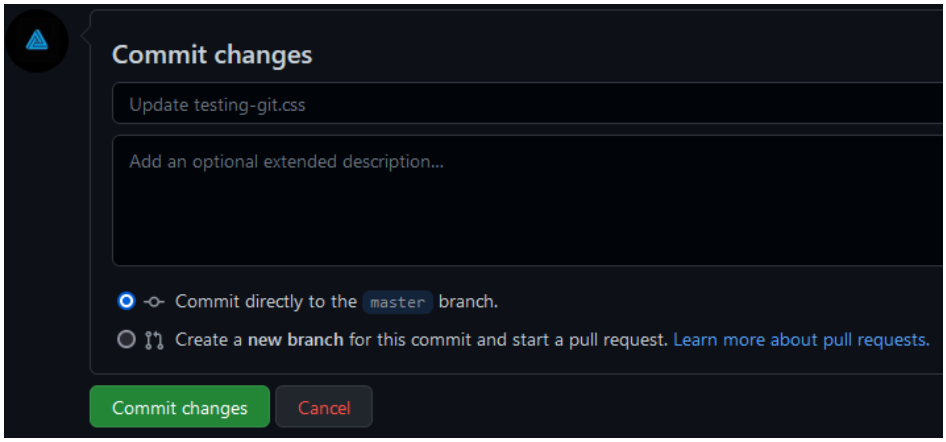
Estas instrucciones suben a GitHub todo el repositorio, donde se pueden ver los commits:



Vídeo 4. Curso Git. Clonación y tags

- **Editar desde GitHub:**

Y seleccionaríamos en este caso:



Al hacer cambios en remoto no los tendremos guardados en local. Podemos traernos nuestros cambios en remoto a local con el comando:
git pull https://github.com/blueslaboratory/testing-git.git

- **Crear tags (etiquetas)**

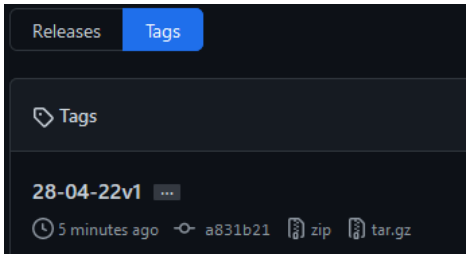
Se utiliza el siguiente comando:
git tag 28-09-22v1 -m "v1 del proyecto"

```
Alejandro@W10 MINGW64 /e/ /Pildoras informáticas - GIT y GitHub/Testing GIT 2
(master)
$ git tag 28-09-22v1 -m "v1 del proyecto"
```

Para editar un tag y eliminar el anterior tag:
git tag newTag oldTag
git tag -d oldTag

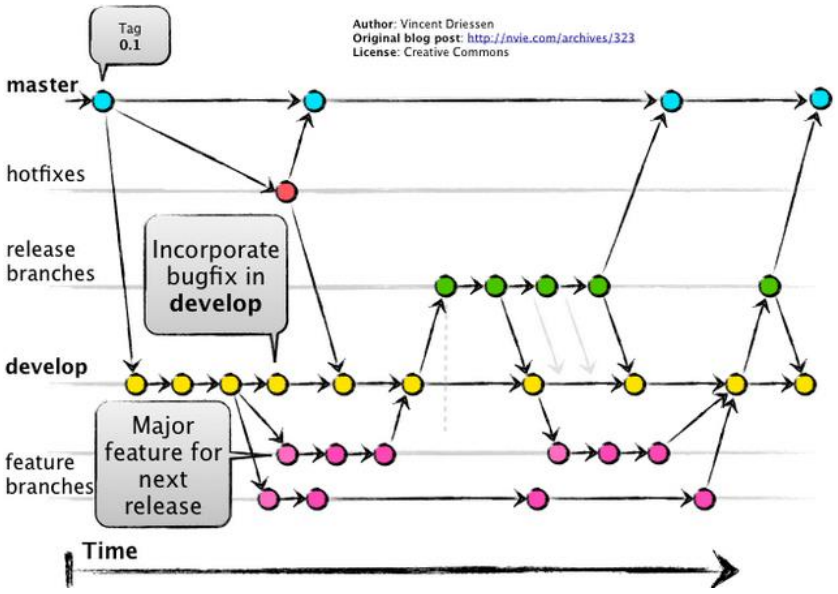
```
Alejandro@W10 MINGW64 /e/ /Pildoras informáticas - GIT y GitHub/Testing GIT 2
(master)
$ git tag 28-09-22v1 -m "v1 del proyecto"

Alejandro@W10 MINGW64 /e/ /Pildoras informáticas - GIT y GitHub/Testing GIT 2
(master)
$ git log --oneline
a831b21 (HEAD -> master, tag: 28-09-22v1, origin/master) error ortografico
04b2461 Update testing-git.css
0989550 Update testing-git.css
5cc42dd Update testing-git.html
72b961b negrita y color blanco
d876e95 parrafo y tam fuente
d25a3fe comienzo del proyecto testing git

Alejandro@W10 MINGW64 /e/ /Pildoras informáticas - GIT y GitHub/Testing GIT 2
(master)
$ git tag 28-04-22v1 28-09-22v1

Alejandro@W10 MINGW64 /e/ /Pildoras informáticas - GIT y GitHub/Testing GIT 2
(master)
$ git tag -d 28-09-22v1
Deleted tag '28-09-22v1' (was 9cc565e)

Alejandro@W10 MINGW64 /e/ /Pildoras informáticas - GIT y GitHub/Testing GIT 2
(master)
$ git log --oneline
a831b21 (HEAD -> master, tag: 28-04-22v1, origin/master) error ortografico
04b2461 Update testing-git.css
0989550 Update testing-git.css
5cc42dd Update testing-git.html
72b961b negrita y color blanco
d876e95 parrafo y tam fuente
d25a3fe comienzo del proyecto testing git
```

Para subir los tags de local a remoto:
git push --tags



- **<u>Clonación de repositorio en local</u>** (GitHub → Local)
  Vamos a simular un desastre, imaginemos que perdemos el proyecto, no pasa nada:
  git clone https://github.com/blueslaboratory/testing-git.git

Vídeo 5. Curso Git. Ramas o Branches I



Una rama o branch es una línea de trabajo, flujo de trabajo en el tiempo. Se trabaja con la rama master/main, pero se pueden crear ramas paralelas a esta para hacer cambios en el funcionamiento del proyecto y pruebas que después se pueden fusionar con la rama master/main.

Crear una rama es como si el proyecto se dividiese en 2 copias y puedes continuar trabajando en cualquiera de esas 2 copias. Puedes borrar la rama paralela si algo va mal y el código de la rama master/main no se verá afectado.

Para crear una rama utilizamos el comando branch:
git branch nombreRama

Para ver las ramas del proyecto y en que rama me encuentro:
git branch

Para moverme a la nueva rama de trabajo:
git checkout nombreRama

```
Alejandro@W10 MINGW64 /e/ /Pildoras informáticas - GIT y GitHub/Testing GIT 2 (master)
$ git checkout javascript
Switched to branch 'javascript'

Alejandro@W10 MINGW64 /e/ /Pildoras informáticas - GIT y GitHub/Testing GIT 2 (javascri
pt)
$ git branch
* javascript
  master

Alejandro@W10 MINGW64 /e/ /Pildoras informáticas - GIT y GitHub/Testing GIT 2 (javascri
pt)
$ git add .

Alejandro@W10 MINGW64 /e/ /Pildoras informáticas - GIT y GitHub/Testing GIT 2 (javascri
pt)
$ git commit -m "saludo javascript agregado"
[javascript 7d2ffcb] saludo javascript agregado
 3 files changed, 8 insertions(+), 2 deletions(-)

Alejandro@W10 MINGW64 /e/ /Pildoras informáticas - GIT y GitHub/Testing GIT 2 (javascri
pt)
$ git log --oneline
7d2ffcb (HEAD -> javascript) saludo javascript agregado
a831b21 (tag: 28-04-22v1, origin/master, origin/HEAD, master) error ortografico
04b2461 Update testing-git.css
0989550 Update testing-git.css
5cc42dd Update testing-git.html
72b961b negrita y color blanco
d876e95 parrafo y tam fuente
d25a3fe comienzo del proyecto testing git
```

Si hacemos ahora un checkout a la rama master:
git checkout master

El editor de código que tengamos abierto, (Visual Studio Code yo), es capaz de detectar que los cambios que hemos realizado en nuestra nueva rama, (javascript), NO están en la rama master, es por ello que recarga los archivos y únicamente se verá aquella versión de estos que se encontraban en la rama master, es decir, no aparecen los cambios de la rama de la que nos hemos movido (javascript).

Modificamos el html agregando un 3er párrafo en la rama master y lo pasamos del directorio de trabajo al repositorio local:
git add .
git commit -m "agregado 3er párrafo"

Al hacer **git log --oneline** veremos únicamente los commits de la rama master en este caso. Es como si se tratase de 2 proyectos diferentes.

```
Alejandro@W10 MINGW64 /e/ /Pildoras informáticas - GIT y GitHub/Testing GIT 2 (javascript)
$ git checkout master
Switched to branch 'master'
Your branch is up to date with 'origin/master'.

Alejandro@W10 MINGW64 /e/ /Pildoras informáticas - GIT y GitHub/Testing GIT 2 (master)
$ git add .

Alejandro@W10 MINGW64 /e/ /Pildoras informáticas - GIT y GitHub/Testing GIT 2 (master)
$ git commit -m "agregado 3er parrafo"
[master 608c996] agregado 3er parrafo
 1 file changed, 4 insertions(+)

Alejandro@W10 MINGW64 /e/ /Pildoras informáticas - GIT y GitHub/Testing GIT 2 (master)
$ git log --oneline
608c996 (HEAD -> master) agregado 3er parrafo
a831b21 (tag: 28-04-22v1, origin/master, origin/HEAD) error ortografico
04b2461 Update testing-git.css
0989550 Update testing-git.css
5cc42dd Update testing-git.html
72b961b negrita y color blanco
d876e95 parrafo y tam fuente
d25a3fe comienzo del proyecto testing git
```

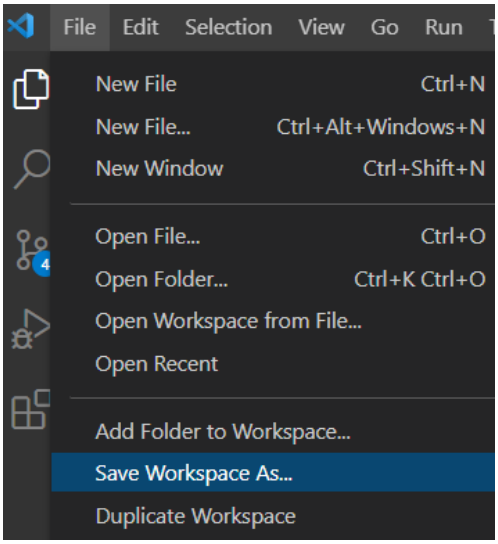
Vamos a hacer un merge, fusionar lo que hice en la rama nueva (javascript) con lo de la rama master:

1. El merge se debe hacer desde la rama master:
   git checkout master

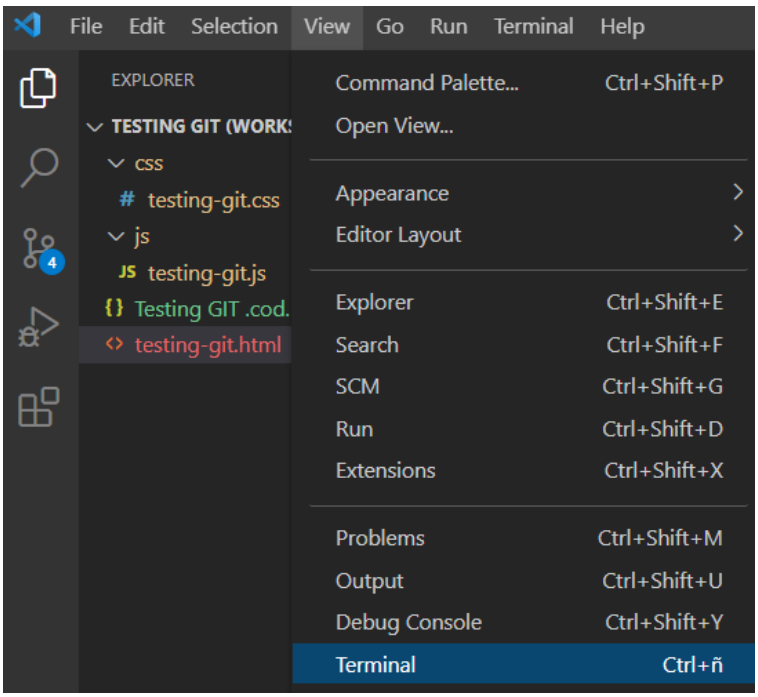
2. Ahora hacemos el merge, que da problemas y veremos cómo resolverlos en el siguiente video:
   git merge nombreRama

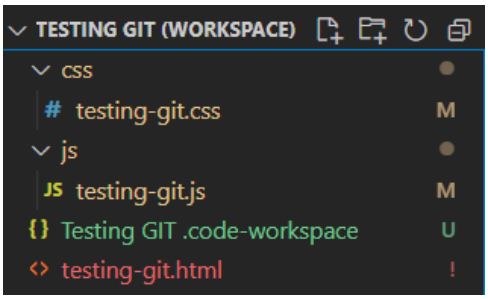
Vídeo 6. Curso Git. Visual Studio Code con Git. Ramas o Branches II

Lo primero que haremos será guardar un Workspace para que si queremos restaurar el proyecto en VSC con los archivos abiertos tal y como los teníamos en una sesión anterior esto sea posible:



VSC cuenta con una terminal incorporada:





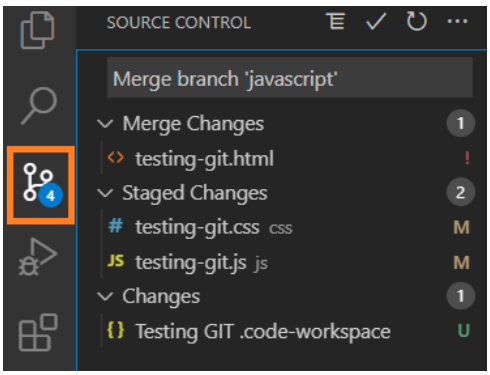
La **U** quiere decir **Unfollowed**, significa que no se le está haciendo un seguimiento al archivo.

La **M** es de **Modified**.

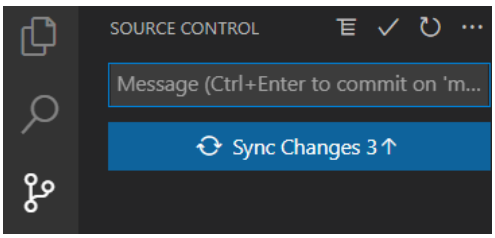
La **A** es de **Agreggated**, pasa después de hacer un add, cuando los cambios están en el staging area.

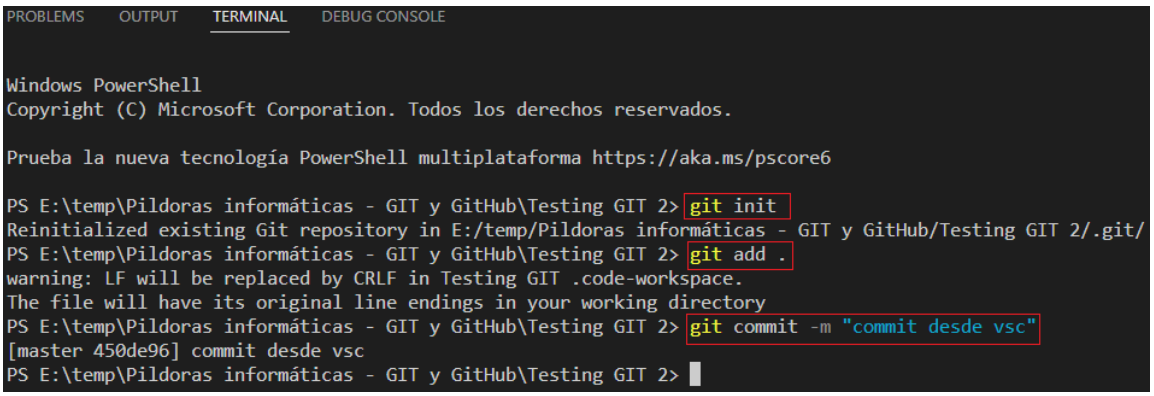
La **!** es porque hay un **Conflicto**.

Tareas pendientes:



Tras hacer un **git commit -m "commit desde vsc"** desaparecen todas las tareas pendientes.





Vídeo 7. Curso Git. Visual Studio Code con Git. Ramas o Branches III

Aquí vamos a ver qué herramientas nos ofrece VSC para trabajar con ramas/branches.

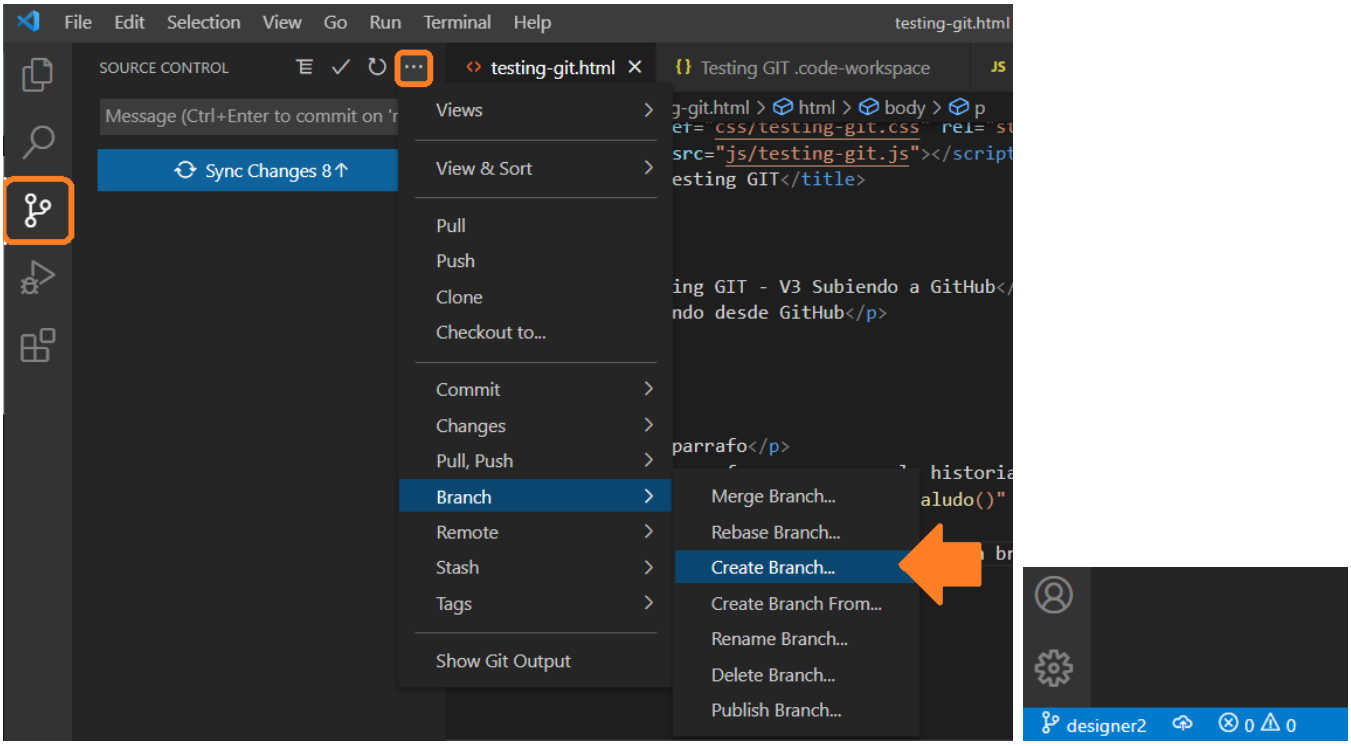
Vamos a modificar los archivos en las 2 ramas, master y javascript, y tras hacer los commits vamos a hacer un merge en VSC. Si al cambiar de rama en el VSC sale el error:
error: The following untracked working tree files would be overwritten by checkout:
Please move or remove them before you switch branches.

Utilizamos el siguiente comando para limpiar los archivos o bien hacemos un commit.
Opción 1: **git clean -d -f**
Opción 2: **git commit --am "add y commit a la vez"**

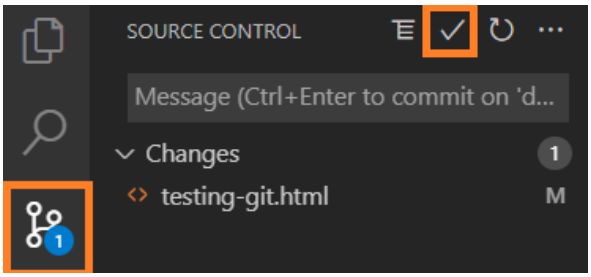
Para hacer el merge, recordamos que nos tenemos que colocar en la rama master:
git checkout master

Para borrar la rama después de hacer el merge (no aparecerá en el oneline):
git branch -d nombreRama
git log --oneline

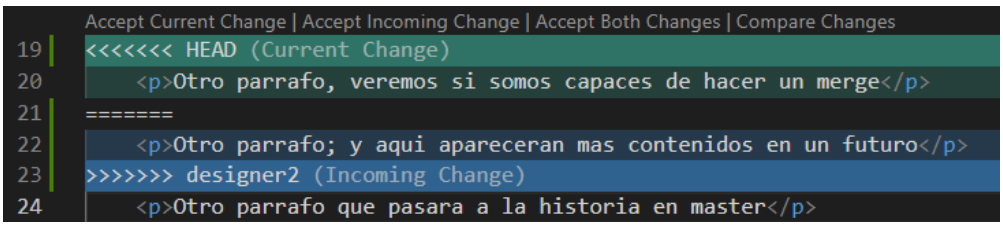
Desde VSC no tenemos por qué utilizar la consola:



En la esquina inferior izquierda se puede ver la rama en la que nos encontramos, tras crear la rama *designer2*, el checkout a esta rama ha sido automático. Para hacer un commit sólo hace falta darle al check o también podemos escribir un mensaje en el cuadro de texto y pulsar CTRL + Enter.



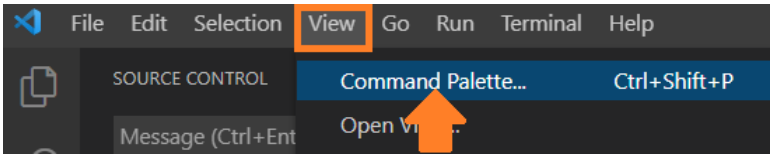
Ahora buscamos el conflicto a la hora de hacer un merge entre *designer2* y *master* editando el mismo parrafo en las 2 branches y haciendo los commits:

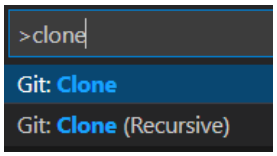


Current change es el cambio que tenemos en la rama master.
Incoming change es el cambio realizado desde la rama designer2.
La opción **Accept Both Changes** nos deja ambos párrafos.

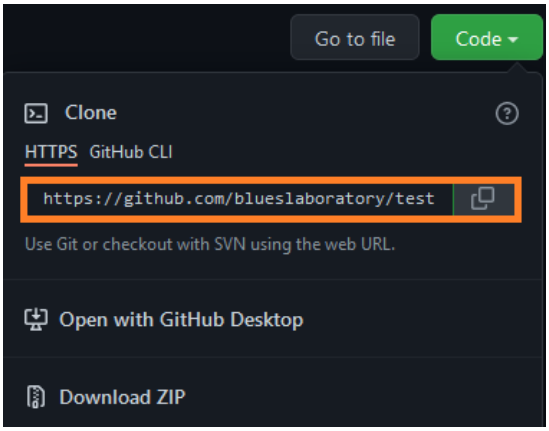
Vídeo 8. Curso Git. Visual Studio Code con GitHub. Subiendo y sincronizando el proyecto

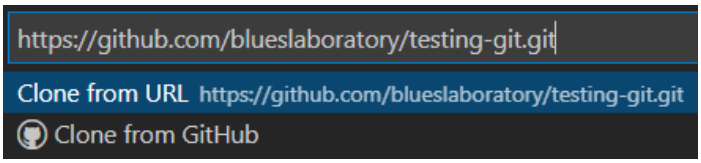
Aquí veremos cómo subir y sincronizar un proyecto en GitHub desde VSC:





Y ponemos la dirección de nuestro repositorio:

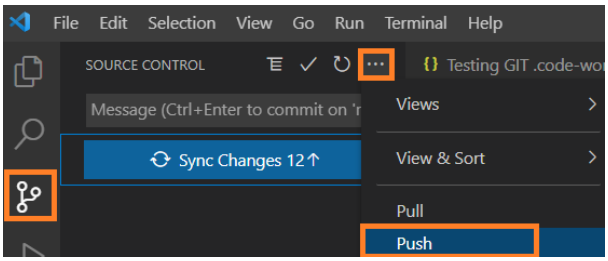




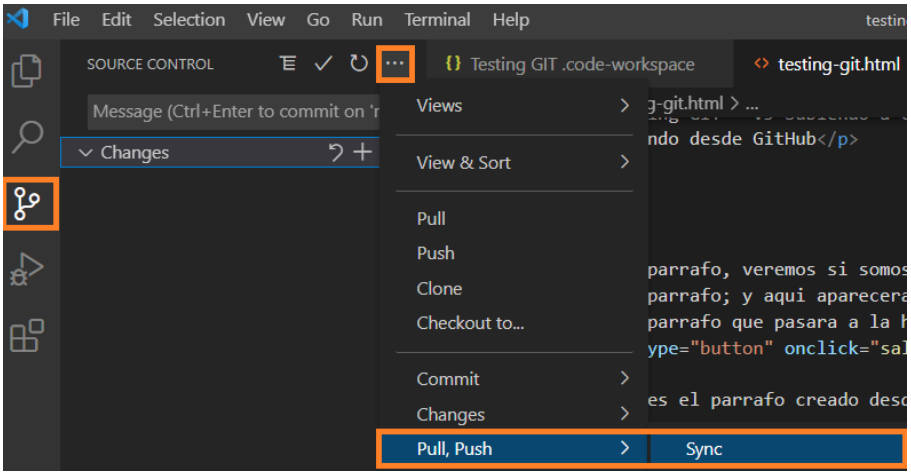
Después tenemos que seleccionar en el explorador de archivos el lugar donde queremos clonar nuestro proyecto.
Para rellenar un HTML con su esqueleto en VSC, escribir una exclamación ! y con esto te autorrellena.

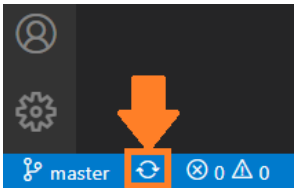
Se recomienda instalar el plugin: beautify

Para subir el repositorio a remoto hay que hacer un push, sería el equivalente a hacer:
git push origin master



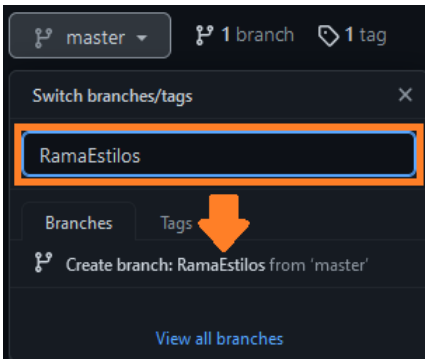
Si hacemos un cambio en los archivos desde GitHub, podemos sincronizar desde el menu o la esquina inferior izqda:

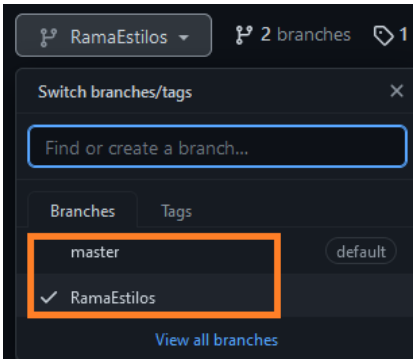




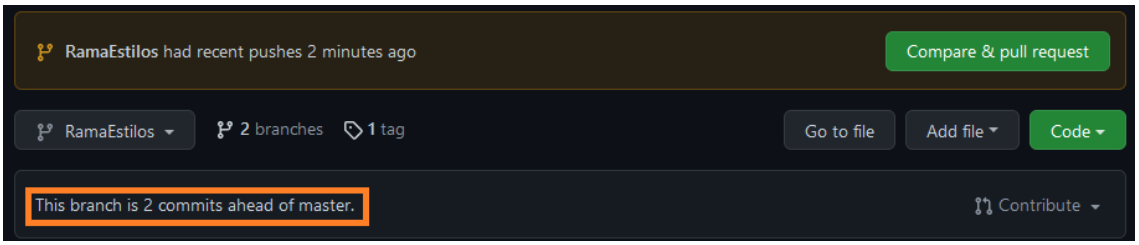
Vídeo 9. Curso Git. Visual Studio Code con GitHub II. Ramas

Para crear una rama desde GitHub nos vamos a nuestro repositorio y nos situamos donde la rama master/main:





Vamos a crear desde esta nueva rama, RamaEstilos, en GitHub una nueva hoja de estilos y a linkearla desde el html (todo editando desde GitHub). La RamaEstilos tendrá 2 commits de ventaja con respecto a la rama master:



Un pull request es una petición que el propietario de un fork, (ramificación de todo el proyecto), de un repositorio hace al propietario del repositorio original para que este último incorpore los commits que están en el fork.

Hacemos un *Compare & pull request* y seguimos los pasos hasta finalizar el merge y borrar la nueva rama. Después nos vamos a VSC y sincronizamos para ver nuestros cambios en local.

Cuando haces el merge con 4-5 ramas diferentes es cuando descubres realmente cómo funciona el merge.

Vídeo 10. Curso Git. Fork

Un **fork** es una ramificación de todo el proyecto que se utiliza normalmente para colaborar con el **pull request**, petición al propietario del repositorio original de que acepte los commits del fork.

Lo suyo después de hacer un fork si es un proyecto complejo es clonarlo a local con git bash en la carpeta destino:
git clone **https://github.com/blueslaboratory/testing-git.git**

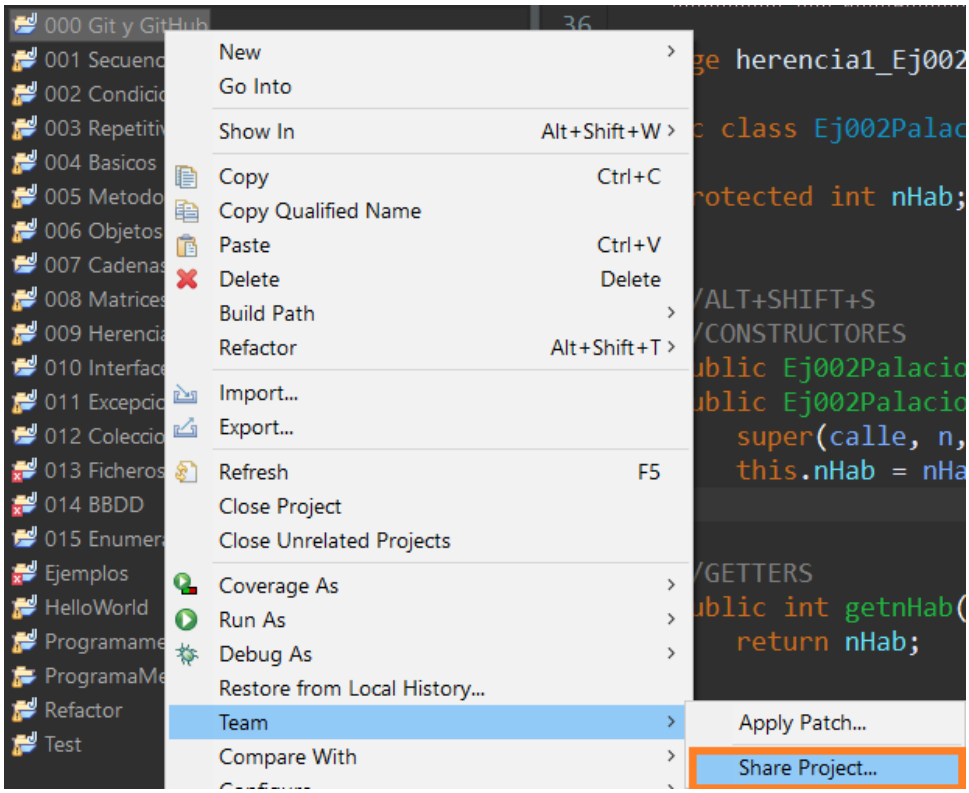
Cuando trabajamos con 2 perfiles de GitHub desde el mismo equipo será necesario borrar las credenciales para entrar desde un usuario diferente.

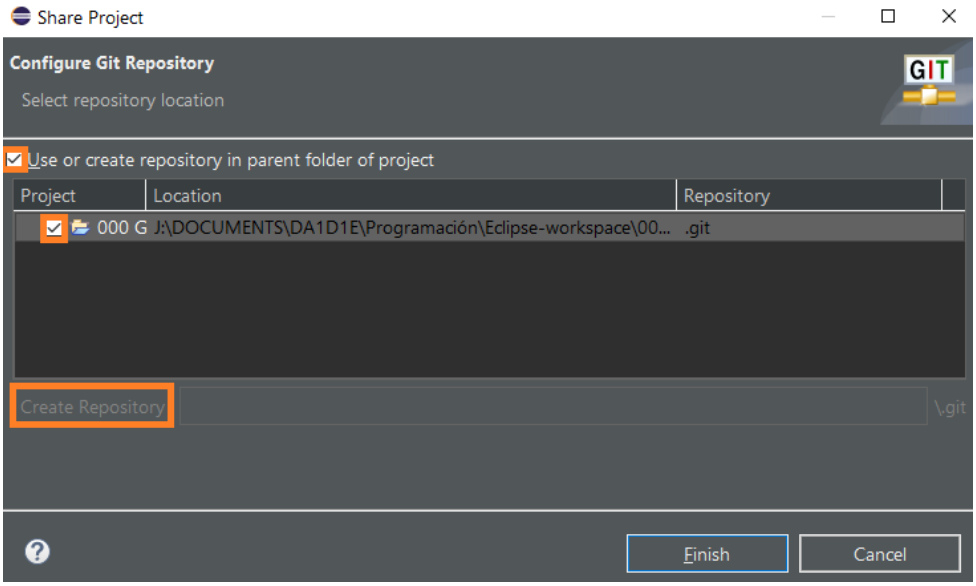
Quitar las credenciales en W10:
Panel de control\Todos los elementos de Panel de control\Administrador de credenciales

Si borramos las credenciales nos tendremos que volver a loguear al hacer un:
git push origin master

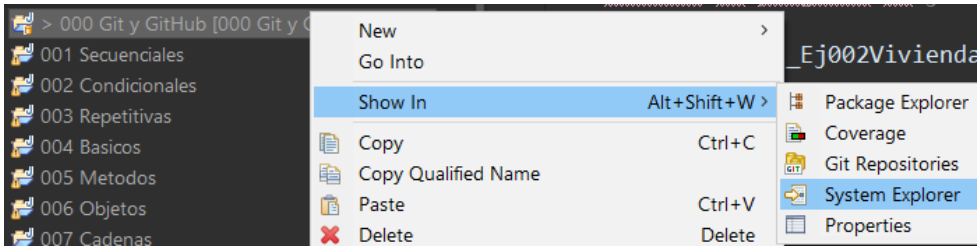
Vídeo 11. Curso Git. Final. Eclipse con Git y GitHub

Mouse botón derecho sobre el proyecto:

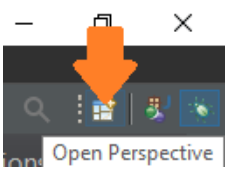


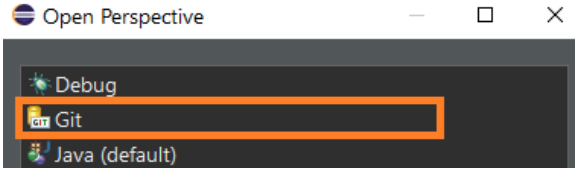


Si vemos el proyecto en el explorador de archivos veremos la carpeta .git oculta:

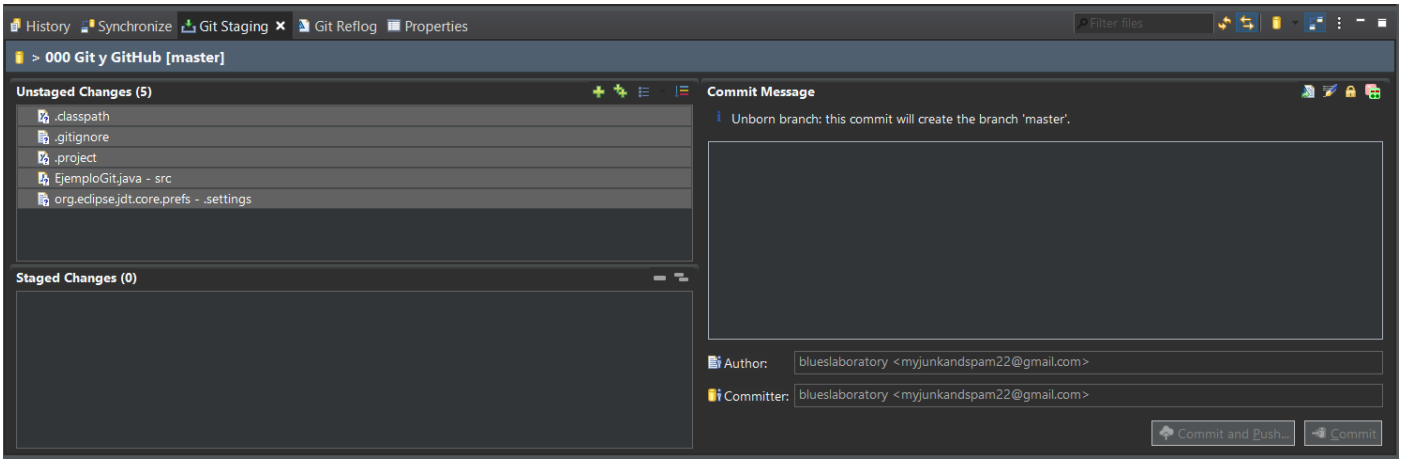


En la esquina superior derecha de Eclipse podemos ver y añadir la perspectiva desde Git:

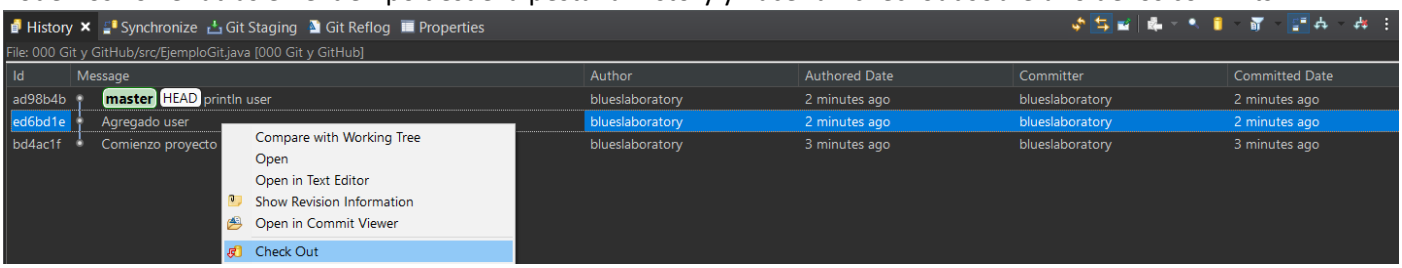




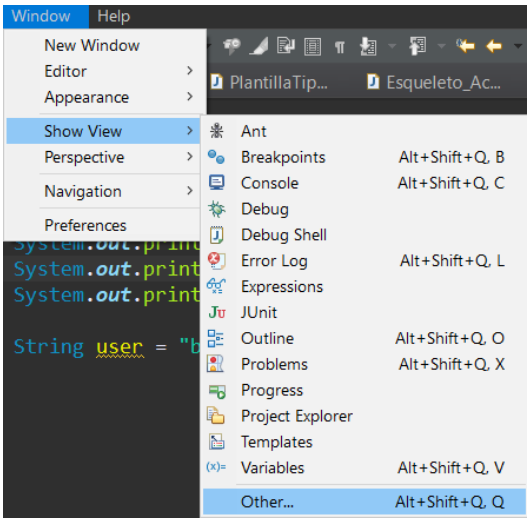
Y aquí ya desde Eclipse, pues dejarse llevar, fluir y disfrutarlo:

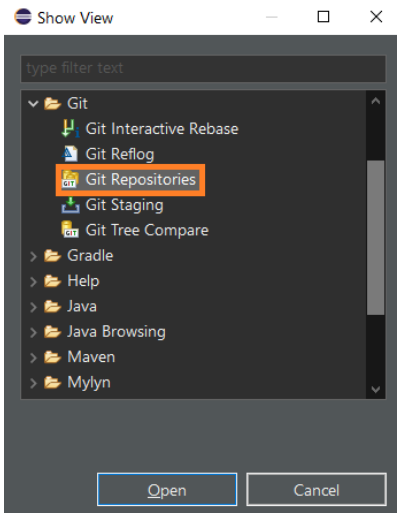


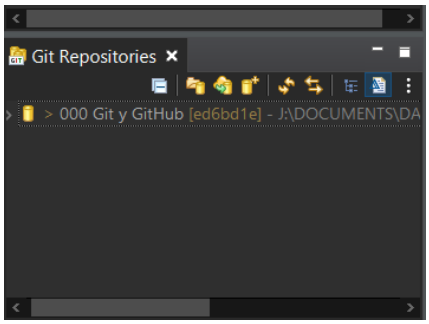
Podemos volver atrás en el tiempo desde la pestaña History y hacer un checkout sobre uno de los commits:



Ver los repositorios en Eclipse:







Vamos a hacer un Push HEAD a nuestro repositorio en GitHub: