https://www.netmentor.es/entrada/introduccion-git-github

CONTENIDO CAMBIADO

Introducción a Git y GitHub

En este tutorial veremos qué son y para que sirven tanto Git como GitHub. Personalmente pienso que tener claros estos conceptos es primordial en nuestro día ya sea laboral o cuando estamos estudiando.

Índice

* [1 - Qué es Git?](https://www.netmentor.es/entrada/introduccion-git-github#mcetoc_1ehne4tdic)
  + [1.1 - Para qué sirve Git?](https://www.netmentor.es/entrada/introduccion-git-github#mcetoc_1ehne4tdid)
  + [1.2 - Descargar Git](https://www.netmentor.es/entrada/introduccion-git-github#mcetoc_1ehne4tdie)
* [2 - Qué es GitHub?](https://www.netmentor.es/entrada/introduccion-git-github#mcetoc_1ehne4tdif)
  + [2.1 - Crear una cuenta en GitHub](https://www.netmentor.es/entrada/introduccion-git-github#mcetoc_1ehne4tdig)
  + [2.2 - Qué es un repositorio?](https://www.netmentor.es/entrada/introduccion-git-github#mcetoc_1ehne4tdih)
  + [2.3 - Crear un repositorio](https://www.netmentor.es/entrada/introduccion-git-github#mcetoc_1ehne4tdii)
  + [2.4 - Utilizar Git en nuestro ordenador](https://www.netmentor.es/entrada/introduccion-git-github#mcetoc_1ehne4tdij)
* [3 - Realizar cambios](https://www.netmentor.es/entrada/introduccion-git-github#mcetoc_1ehne4tdik)
  + [3.1 - Realizar un commit](https://www.netmentor.es/entrada/introduccion-git-github#mcetoc_1ehne4tdil)
  + [3.2 - Realizar un push](https://www.netmentor.es/entrada/introduccion-git-github#mcetoc_1ehne4tdim)
  + [3.3 - Incorporar cambios ajenos](https://www.netmentor.es/entrada/introduccion-git-github#mcetoc_1ehne4tdin)

1 - Qué es Git?

Git es un sistema de control de versiones. Desde una perspectiva de desarrollador es un sistema que almacena y controla los cambios sobre los ficheros que modificamos.

Además, git es un sistema de versiones distribuido, esto quiere decir que disponemos de un repositorio remoto en el servidor y un repositorio local el cual se almacena en la máquina de cada desarrollador. Lo que implica que el código esta tanto en el servidor como en cada máquina que lo modifica.

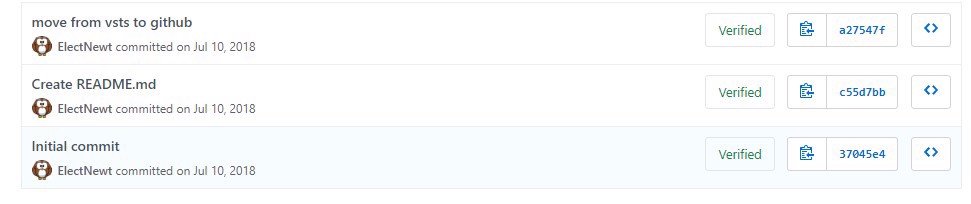
Es importante incluir toda esta información, ya que la gran mayoría de las empresas incluyen un sistema de control de versiones, comúnmente git, aunque hay más como subversión o csv.

1.1 - Para qué sirve Git?

En un entorno laboral real nos permite trabajar en paralelo con varios desarrolladores. Ya que gracias a git podemos comprobar todos los cambios del código, así como los conflictos.

Cuando realizamos un cambio decimos que hacemos un commit, el cual contiene información, como el autor, la fecha, un comentario y los ficheros que han cambiado.

Esto es un ejemplo de varios commits que tengo en una librería:



1.2 - Descargar Git

Si tenemos instalado visual studio 2019 tendremos git instalado en el nuestra máquina por defecto, pero aun así podemos comprobar si esta instalado con el siguiente comando en la terminal (cmd)

git --version

nos indicara si tenemos git instalado, en caso contrario, lo podemos descargar desde la pagina oficial para la versión que necesitemos <https://git-scm.com/downloads>

2 - Qué es GitHub?

Es un servicio de compartir y publicar código. Básicamente es como si fuera una red social de programadores. La cual utiliza git por detrás.

Comúnmente trabajaremos tanto con Git como con GitHub en paralelo. Personalmente creo que es la mejor combinación. Si por algún motivo no nos gusta la interfaz de GitHub siempre podemos utilizar otras como bitbucket.

2.1 - Crear una cuenta en GitHub

Para crear una cuenta de GitHub tan solo tenemos que acceder a su página web <https://github.com/>

Para así poder crear nuestro primer repositorio.

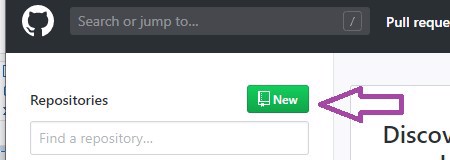
2.2 - Qué es un repositorio?

Dentro de GitHub tenemos el código dividido por secciones, por ejemplo, aplicación para subir imágenes, aplicación para escribir posts, etc. Cada una de estas aplicaciones está dentro de un repositorio, por supuesto un repositorio puede contener mas de una aplicación.

Los repositorios suelen estar divididos internamente por lógica de negocio

2.3 - Crear un repositorio

Para crear un repositorio únicamente tenemos que pulsar en el botón verde y ya empezaremos a trabajar con él.



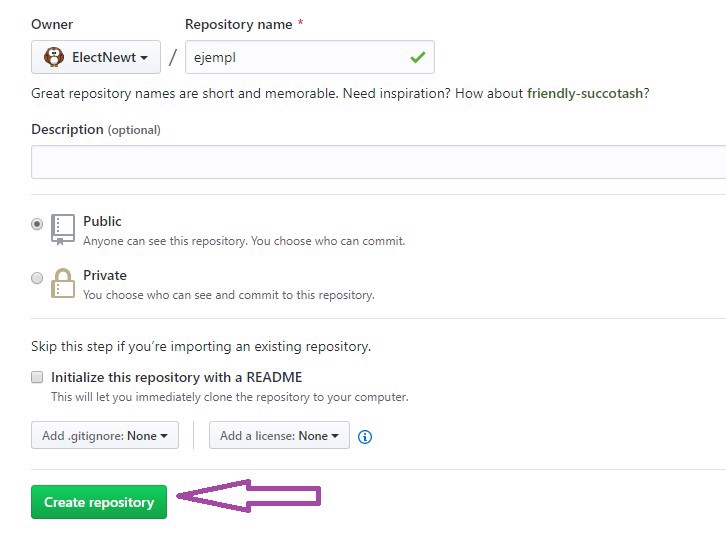
Le indicamos un nombre y pulsamos en crear repositorio.

Además nos deja indicar varias cosas como por ejemplo

Si queremos el repo público o privado

Si queremos introducir un fichero README, los ficheros readme se escriben, normalmente, en lenguaje markdown.

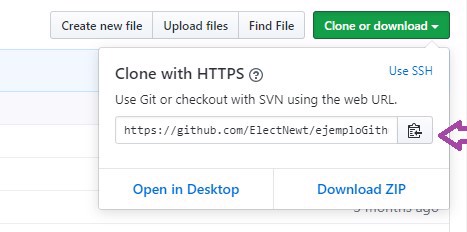
Indicar el fichero .gitignore el cual evitara que publiquemos fichero que no deseamos que se publiquen, comúnmente ponemos dentro de este todos los ficheros temporales como los de la carpeta /bin u /obj.



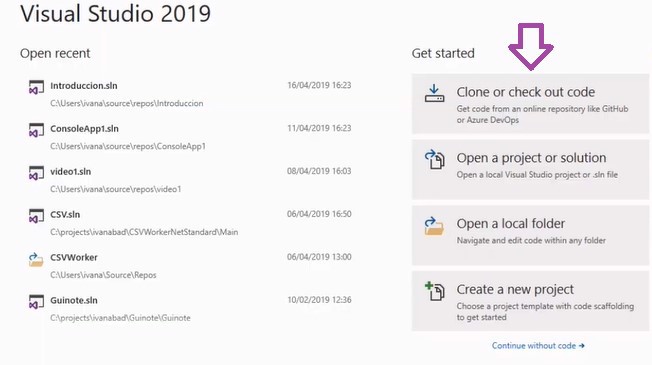
2.4 - Utilizar Git en nuestro ordenador

Para ello tendremos que poner el repositorio en nuestro equipo.

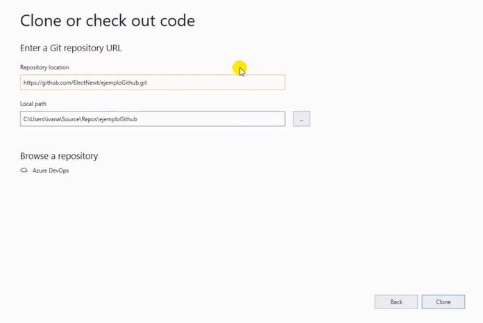
Como hemos indicado antes, en GitHub disponemos de una copia en el servidor y otra en local. Para copiar el directorio. Para ello pulsamos en el botón que dice “clonar o descargar” y después pulsar en el icono para copiar.



Posteriormente abrimos visual stuido y pulsamos en clone o checkout code



Y pegamos la url en la localización como vemos la podemos cambiar o dejar la de por defecto.



Podemos utilizar el siguiente comando

git clone git@github.com:<tuCuentadeGithub>/ejemploGitHun.git

Bien ahora ya tenemos el repositorio enlazado en nuestro ordenador. Por lo que pasamos a crear un fichero o un proyecto, o lo que deseemos, también funciona con archivos de audio, imágenes, etc. Es un control de versiones para todo tipo de fichero.

 3 - Realizar cambios

Cuando realizamos un cambio, ya sea modificar un fichero, añadir un fichero, borrar un fichero o moverlo, este cambio quedará reflejado en el controlador de versiones.

Por ello creamos un fichero de texto plano (podemos utilizar el README) y añadimos texto en el.

Como podremos imaginar esos cambios están pendientes de ser enviados al servidor, para ello tenemos que realizar dos pasos

3.1 - Realizar un commit

\*Nota: en el video todos los pasos son realizados en el entorno gráfico de Visual Studio, aquí en el post, utilizaremos línea de comandos.

El primero de los pasos es realizar un commit, el cual es un comando que indicara al servidor de versiones que esa versión del fichero es la versión actual.

Para ello utilizando línea de comandos o powershell vamos a la carpeta donde se encuentra el fichero y escribimos lo siguiente:

git commit -m “mensaje descripción”

como vemos el comando contiene git commit y la opción -m la cual permite escribir un comentario. Es recomendable, y debería ser obligatorio añadir un comentario cuando se hace un commit, así puedes ver de una forma mas rápida sobre que es el cambio.

3.2 - Realizar un push

Como hemos indicado, el cambio esta en el controlador de versiones, pero esta en el controlador de versiones de tú máquina, para que este se mueva al servidor tenemos que hacer un push con el siguiente comando.

git push

Con ello los cambios serán movidos al servidor. En caso de que otro desarrollador durante el tiempo que hemos estado trabajando en el fichero haya realizado algún cambio, al realizar el push nos indicará que hay un conflicto y tendremos que resolverlo. Pero lo veremos en el siguiente post.

3.3 - Incorporar cambios ajenos

Supongamos que nos hemos ido de vacaciones, y alguien se ha puesto en nuestro proyecto a arreglar problemas durante este tiempo. Eso quiere decir que en nuestra maquina tenemos una versión inferior a la versión real de la aplicación.

Lo que tenemos que conseguir ahora es actualizar la versión de nuestra máquina.

Para ello realizaremos los siguientes dos comandos:

git fech

git merge

los cuales actualizaran la versión de la aplicación con los últimos cambios.

Otra opción es realizar ambos en un solo comando

git pull

pero puede haber algún problema con conflictos, lo mas seguro es realizar tanto git fetch como git merge.

<https://www.netmentor.es/entrada/trabajo-equipo-git-github>

Trabajo en equipo con Git y GitHub

Índice

* [1 - Qué es un branch?](https://www.netmentor.es/entrada/trabajo-equipo-git-github#mcetoc_1egi9bevrm)
  + [1.1 - como crear un branch](https://www.netmentor.es/entrada/trabajo-equipo-git-github#mcetoc_1egi9bevrn)
* [2 - Qué es un merge?](https://www.netmentor.es/entrada/trabajo-equipo-git-github#mcetoc_1egi9bevro)

  + [2.1 - Cómo hacer un merge?](https://www.netmentor.es/entrada/trabajo-equipo-git-github#mcetoc_1egi9bevrq)
  + [2.2 - Cómo borrar un branch?](https://www.netmentor.es/entrada/trabajo-equipo-git-github#mcetoc_1egi9bevrr)
* [3 - Pull request](https://www.netmentor.es/entrada/trabajo-equipo-git-github#mcetoc_1egi9bevrs)
  + [3.1 - Qué es un pull request?](https://www.netmentor.es/entrada/trabajo-equipo-git-github#mcetoc_1egi9bevrt)
* [3.2 - Cómo hacer un pull request?](https://www.netmentor.es/entrada/trabajo-equipo-git-github#mcetoc_1egi9bevru)
  + [3.3 - Revisar el código](https://www.netmentor.es/entrada/trabajo-equipo-git-github#mcetoc_1egi9bevrv)
  + [3.4 - Merge de un pull request](https://www.netmentor.es/entrada/trabajo-equipo-git-github#mcetoc_1egi9bevr10)

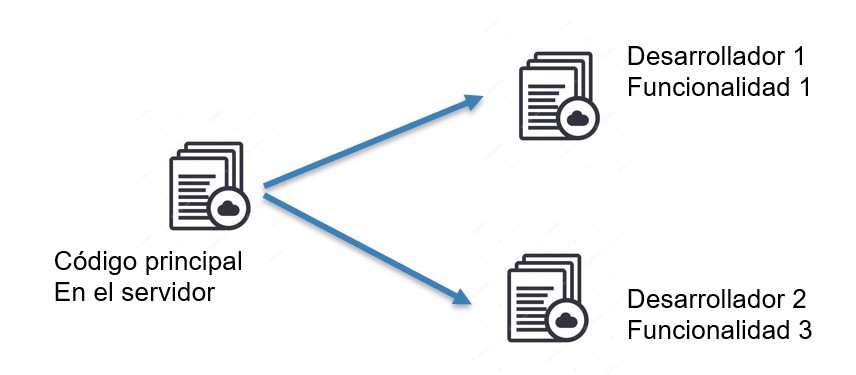
Este es el segundo video del curso Git y github que continua al anterior en el que aprendimos a instalar y configurar git así como a crear una cuenta de github, para ello puedes[acceder aquí](https://www.netmentor.es/curso/git-y-github/introduccion-git-github)

Digamos que ese tutorial era cuando trabajamos solos en un proyecto, hacemos cambios y los publicamos para mantener un registro de los mismos.

En este segundo tutorial veremos cosas algo más complejas que serán las que utilizamos cuando trabajamos en equipo.

1 - Qué es un branch?

Git funciona por ramas, esto quiere decir que tenemos el código principal en la rama A. Nosotros queremos hacer un cambio, para realizar ese cambio no lo realizamos directamente en la rama A, sino que creamos una rama paralela a raíz de la rama A.



Esto lo realizamos porque así permitimos a otros desarrolladores trabajar y tanto ellos como nosotros no vamos a tener el código del otro, el cual puede no estar funcional al 100% todo el tiempo o contener errores.

Comúnmente estos branches se crean porque tenemos una tarea o una funcionalidad que añadir, para ello GitHub tiene su propia pestaña en el repositorio.

1.1 - como crear un branch

Supongamos que tenemos la tarea EJ-0001 y queremos arreglarla.

Para crear un Branch tenemos dos opciones

1. Ir a GitHub y crear un Branch como podemos ver en el video
2. Utilizando la línea de comandos utilizando el siguiente comando

git branch [nombre]

Este comando creara un Branch con el nombre que le indicamos. Posteriormente nos tenemos que posicionar sobre él, lo cual lo haremos con el comando checkout.

git checkout EJ-0001

Ambos comandos se pueden fusionar en uno solo, el cual nos creara un Branch y posicionara el código sobre ese Branch.

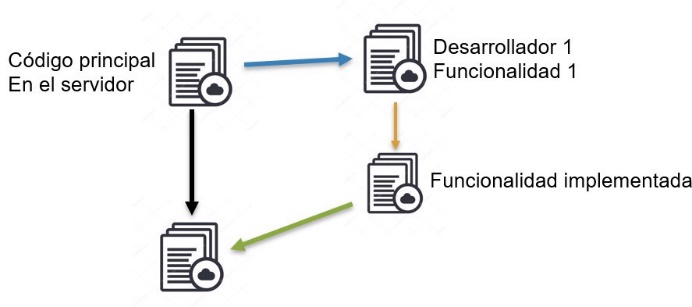
git checkout -b EJ-0001

A partir de ahora cuando hagamos un commit al código este se realizará sobre el Branch EJ-0001 y no sobre el principal.

Cuando terminemos todos los cambios en el código y queramos ponerlos en el Branch principal, ya que están listos para producción, realizaremos un merge.

2 - Qué es un merge?

Cuando queremos implementar los cambios que hemos realizado dentro de la rama principal del repositorio realizamos una acción denominada merge, la cual combinara nuestro código con el principal.



2.1 - Cómo hacer un merge?

En GitHub es muy sencillo ya que nos da un botón para realizar el merge.

En línea de comandos nos tenemos que ubicar sobre la rama principal del repositorio, y entonces escribir el comando de la siguiente manera:

git merge EJ-0001

Y si no hay ningún conflicto (siguiente video) el código se combinara con el de la rama principal.

2.2 - Cómo borrar un branch?

Para borrar un branch únicamente neceistamos el siguiente comando

git Branch -d EJ-0001

3 - Pull request

Como acabamos de ver, juntar nuestro código con la rama principal es muy fácil y sencillo, lo que puede llevar a algún problema si no hay ninguna “doble comprobación”. Aquí viene por qué GitHub fue tan popular cuando salió en comparación con sus rivales.

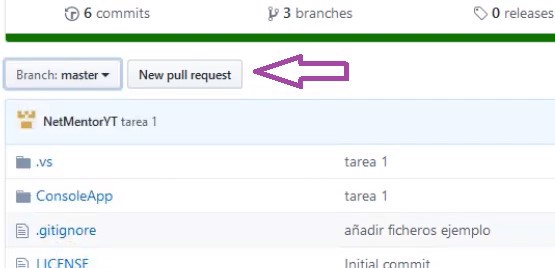
3.1 - Qué es un pull request?

Como hemos indicado es un método de “doble comprobación” eso quiere decir que utilizaremos a una segunda persona de nuestro equipo para revisar que todo el código esta bien y cumple con los estándares puestos.

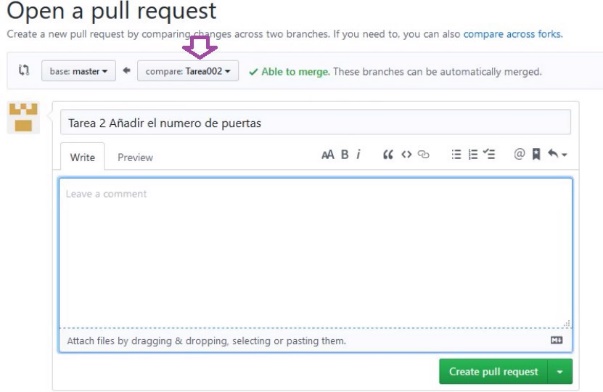
3.2 - Cómo hacer un pull request?

En este caso no hay forma de realizarla utilizando línea de comandos, por lo que tenemos que utilizar GitHub, aunque también esta disponible en bitbucket y en gitlab (merge request)

Para ello lo que tenemos que hacer es, dentro del repositorio pulsar en el botón nuevo pull request



Una vez pulsamos, veremos que nos abre una ventana como la siguiente, en la que tenemos que indicar primero el Branch principal al que vamos a hacer el merge (la rama A) y después el Branch de nuestra tarea



Como vemos después de la tarea 2 tenemos un check verde que indica que el código no tiene ningún conflicto, en el siguiente tutorial veremos como lidiar con conflictos.

Finalmente, también debemos escribir un título y una descripción que ayduaran a la persona que realice el code review.

3.3 - Revisar el código

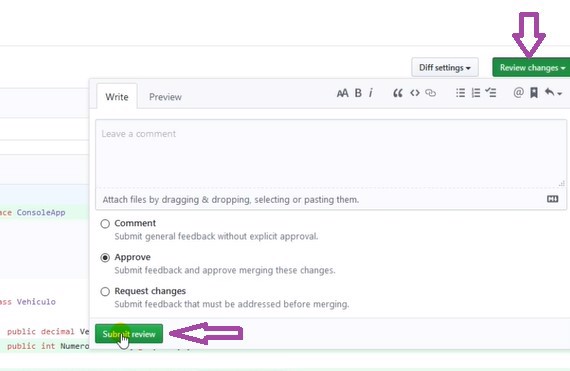
Una vez tenemos la pull request creada el usuario que hemos indicado recibirá o una alerta o un email, depende de su configuración, y pasará a revisar el código.

Lo que realizara es abrir la tarea inicial y ver si los cambios se corresponden con lo que se pide.

También comprobara si sigue todas las normas del equipo como pueden ser el nombre de las variables, espacios o tabuladores, etc. Si no esta de acuerdo con el código por cualquier motivo escribirá en la pull request que necesita cambios, y se deberán arreglar y volver a hacer otro commit.

Este commit estará incluido automáticamente en el pull request.

Para aceptar un pull request tan solo tenemos que pulsar el botón de aceptar

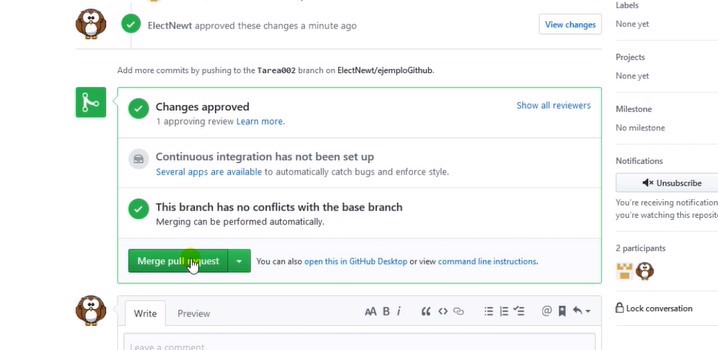


3.4 - Merge de un pull request

Una vez todos los cambios están aceptados, solo queda hacer el merge con el Branch principal.

El desarrollador que ha desarrollado la funcionalidad es el que debe realizar el merge, ya que comúnmente no hay problema de conflictos, pero puede ser que salte alguno, y el revisor no debe de encargarse de arreglarlos.

Para ello tan solo tenemos que pulsar el botón merge pull request



FIN