GIT

Es una software de control de versiones de aplicaciones.

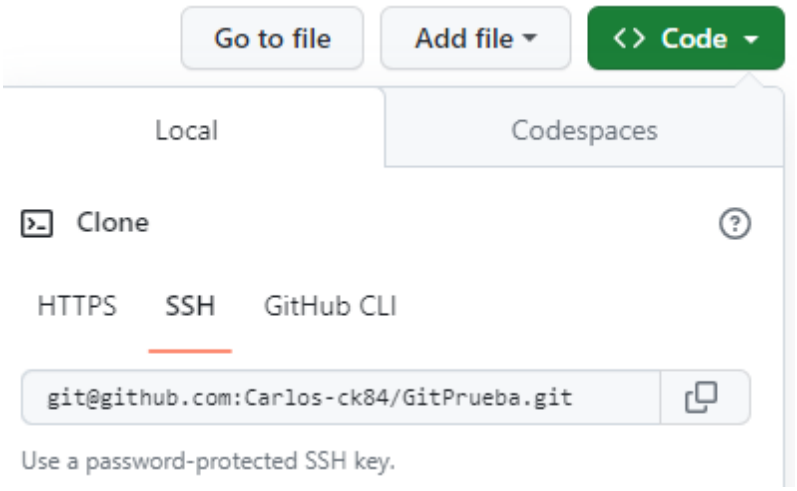
Comandos mas utilizados : https://training.github.com/downloads/es_ES/github-git-cheat-sheet/

Link descarga : https://git-scm.com/download/win

COMANDOS:

- pwd: lugar(carpeta) donde estoy ubicado
- ls: Muestra el contenido por donde puedes navegar
- mkdir “nombre”: Crea un carpeta nombre
- clear: limpiar consola
- code . : Abrir visual code
- git config --global user.name “CarlosF” : Configuración de git para toda persona que entre a mi usuario, para cualquier proyecto. Se crea el usuario CarlosF
- git config --global user.email “cfonseca.electronico@gmail.com” : Configuración del email para el archivo .gitconfig que se creó.
  NOTA: los dos comandos anteriores es la base para poder iniciar a usar git.
- git init: Con este comando se inicializa el funcionamiento de git, y el directorio donde ejecutamos el comando trabaja con un control de versiones. Se crea un rama principal de nuestro proyecto llamada master
- git branch -m main: Cambia el nombre de la rama de master a main.
- git branch: obtengo el o los nombres de las ramas existentes.
- git status: Da información de nuestro proyecto
- git add archivo.txt : preparamos archivo.txt para crearle una version
- git add.: añade todos los ficheros
- git commit -m "segundo commit”: Crea el commit para el fichero.
- git checkout archivo.txt: Se regresa a la versión anterior de la fotografía que saque, asi le haya hecho cambios al archivo.
- git reset : Revisa si existen archivo con algún cambio, para darte a elegir si lo deseas cambiar.
- git log--graph: Se visualiza las dependías o ramas.
- git config --global alias.tree “log --graph”: Se crea un alias llamado tree, el cual va tener el código entre paréntesis. Por lo tanto si ejecuto git tree=git log --graph .
- touch nombrearchivo: crea en la carpeta actual el archivo nombrearchivo
- Para ignorar archivo x de nuestra carpeta, es necesario crear un archivo .gitignore y con esto ya nunca nos preguntara si este archivo x le queremos hacer git.
  1. touch .gitignore
  2. Editamos el archivo anterior y colocamos **/archivo x
  3. Lo añadimos al git y le hacemos commit.
- git diff: Visualiza que archivo o rama y que cambios se hicieron sobre el archivo o rama.
- git checkout 7d81cfe99116607ca3e6003fa613363394d76e8b: Para este ejemplo va a volver a la primera versión de nuestro proyecto.
- git log --graph --decorate --all –oneline: Con este comando podemos ver todo el árbol, ya que lo intente con git log y solo me muestra el nodo principal cuando hice el checkout a la primera versión.
- git reset--hard id: Borra todos los commit de ese id hacia adelante, no lo incluye al id en el borrado. Pero también sirve para restablecer todo si colocamos el ultimo id del árbol.
- git reflog: Nos muestra todo el historial que hemos hecho en el git, incluso los commit borrados.
- git tag clase1: Se genera un tag para el head actual.

- git checkout tags/clase1: Mueve el head al tag clase1
- git checkout main: Mueve el head al main
- git branch login: Puede crear otra rama para hacer una funcionalidad de login en mi app
- git switch login: cambio la rama de main a login.
- git merge main: si seguimos trabajando en el main, la rama del login se des actualiza con la información del main, con este comando el login va a tener la última versión del main.
- git stash: Guarda una versión temporales que puede tener errores, pero sin generarle id ni commit. Porque por buena practica siempre se crea un commit de algo que este funcionando.
- git stash list: Muestra las versiones temporales pendientes.
- git stash pop: Si me cambio de rama y vuelvo a ingresar a la que le hice el stash me va a cargar por defecto el ultimo commit, pero yo tengo una versión temporal que quiero cargar, con este comando lo puedo hacer.
- git stash drop: borra el stash creado previamente.
- git branch -d login: Elimino toda la rama login.
- git fetch: Descarga el historial de cambios de GitHub pero sin descargar los cambios
- `git push -u origin main: Sube los archivos y toda la información a GitHub.`
- git pull: se descargar el historial y los cambios también.
- git clone url: se descarga todo el proyecto que viene de GitHub en la carpeta que queramos de nuestro pc, Por la configuración ssh establecida anteriormente. La url es la que aparece en GitHub en la carpeta del proyecto -> Code->ssh



- brew install git-flow: instala un software de Flujo Git. Solo en Linux
- git flow init: Se usa para iniciar el flujo de nuestro proyecto
- git flow feature start correo: Inicia una rama dentro de feature llamada correo.
- git flow feature finish correo: Cierra una rama dentro de feature llamada correo y hace el merge con la rama develop.
- git flow release start 1.0: Inicia una rama que va a tener una versión de develop que después se va a subir a main, en esta rama no se añade nuevas funciones, se crean soluciones a errores o documentación.
- git flow release finish 1.0: Cierra la rama y hace el merge con main.
- git cherry-pick id: Traemos solo un commit de una rama a otra rama especifica.
- git rebase: Traernos una rama a un punto concreto. Se pondría al final de la rama y se perdería lo que se ha hecho en esa rama su fuera distinto.

GIT HUB

https://github.com/Carlos-ck84/carlos-ck84

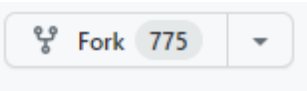
- Para comunicar github con git tenemos que hacer los siguientes pasos:
  1. Crear la carpeta .ssh en la carpeta del usuario
  2. Nos vamos al terminal git bash, nos ubicamos en la carpeta .ssh y digitamos
     `ssh-keygen -t ed25519 -C your_email@example.com`
  3. Nos pide un nombre de archivo podemos colocar unos de los nombres de sugerencia del manual -> id_rsa.
  4. Nos pide passphrase podemos darle enter dos veces y dejarlo vacio.
  5. Digitamos -> eval "$(ssh-agent -s)", para saber si ya tenemos un id que se debe mostrar en la línea siguiente de la terminal.
  6. Agregamos la llave privada -> ssh-add ~/.ssh/id_rsa
  7. Vamos a github -> https://github.com/settings/ssh/new
  8. Abrimos el archivo id_rsa.pub y copiamos el código que esta adentro.
  9. Este código lo copiamos en el campo key del link de github que abrimos y añadimos la key.
  10. Vamos a verificar si se puede establecer la conexión ->
      `ssh -T git@github.com`

      Nos pregunta si queremos continuar conectado le decimos SI
  11. Con lo anterior ya podemos subir nuestro git a github.
  12. Creamos un nuevo repositorio con el mismo nombre de la carpeta del proyecto si queremos
  13. Ahora configuramos para que se conecte con el repositorio creado anteriormente, para eso en el terminal vamos a la carpeta del proyecto git y después digitamos ->
      `git remote add origin https://github.com/Carlos-ck84/GitPrueba.git`
  14. Por ultimo subimos el git y se apunta a la rama principal que es main
      `git push -u origin main`

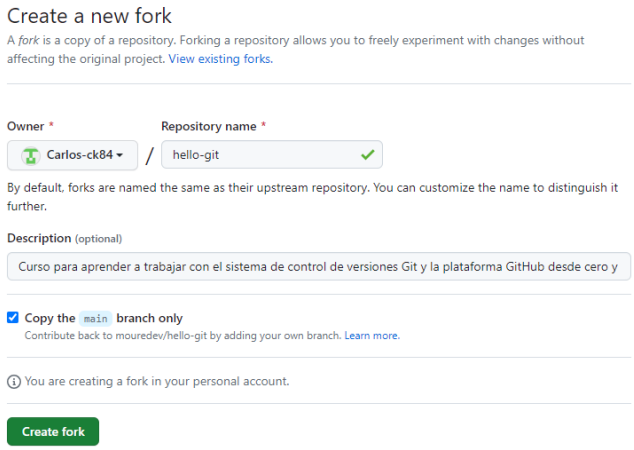
Ambiente colaborativo GitHub

Si queremos trabajar con un código de un usuarios al cual no tenemos permiso de subir el código modificado tenemos que hacer los siguiente:

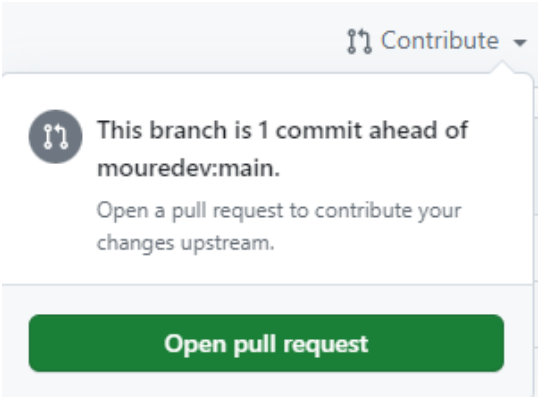
1. Vamos a Fork, Create New Fork



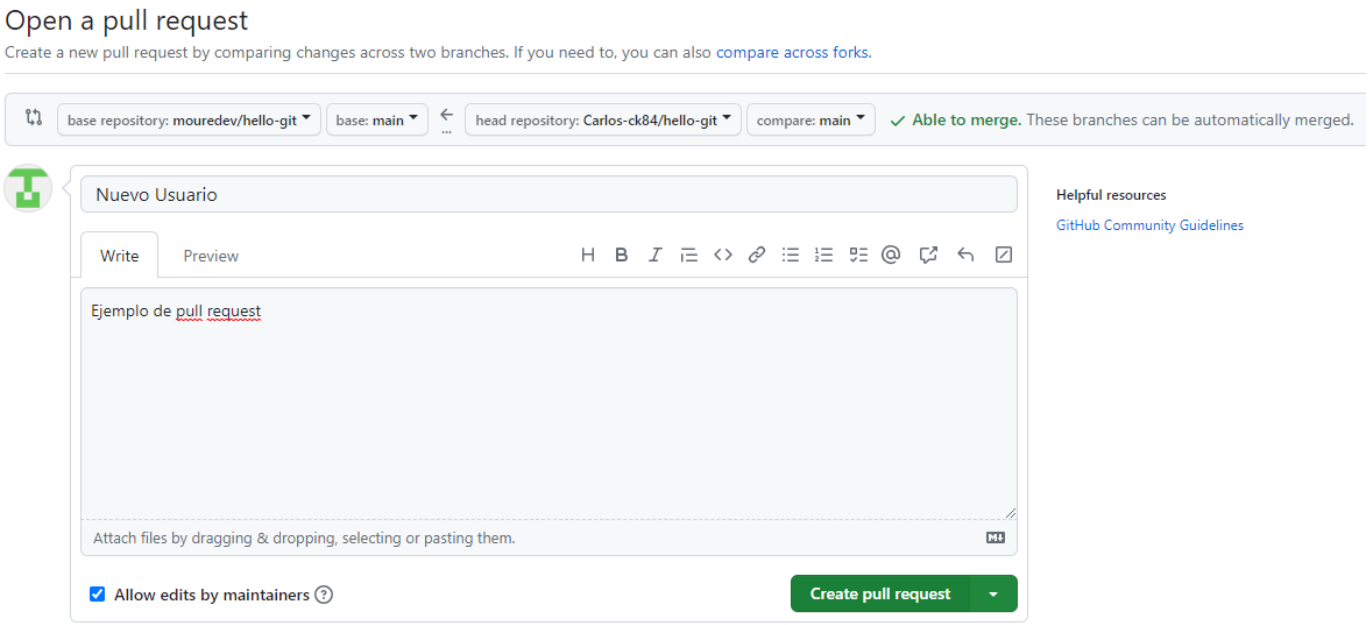
2. Creamos la carpeta en nuestro repositorio.



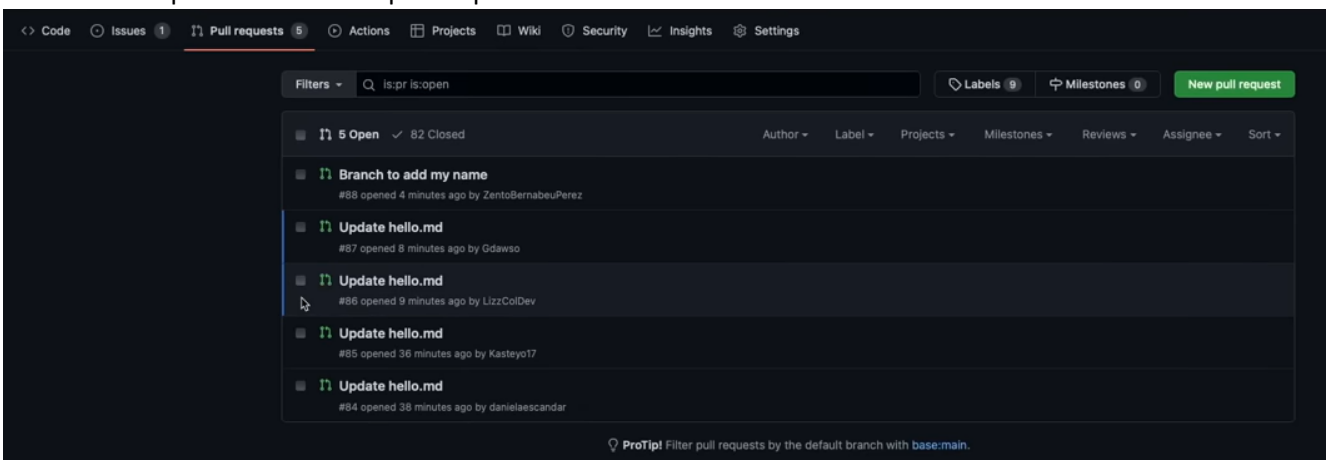
3. Descargamos el código con el método clone en nuestra terminal.
4. Modificamos los archivos que sean necesarios y aplicamos los métodos para que quede listo para hacer el push a nuestro repositorio que se creó con el fork.
5. Hacemos un pull request en la opción de Contribute



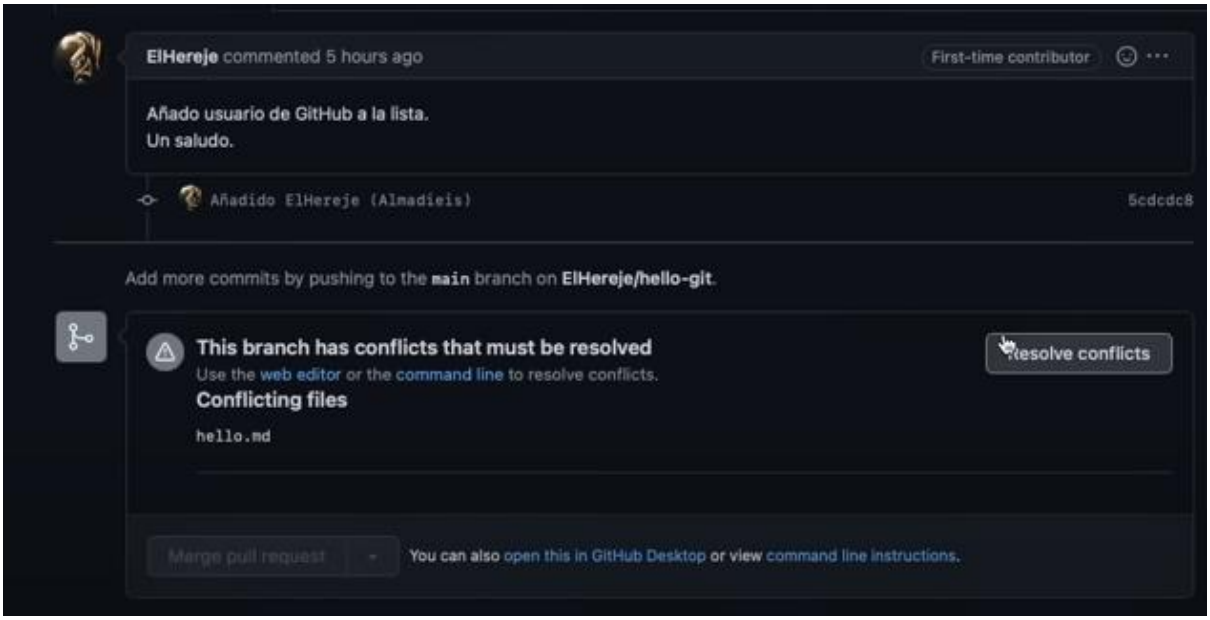
6. Creamos el pull request, importante revisar el destinatario



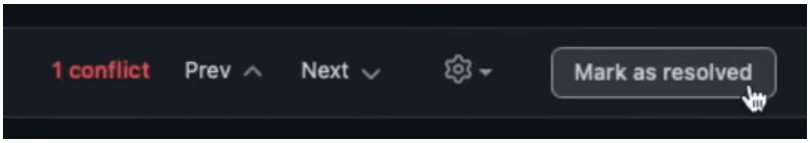
7. Ya con esto, solo dependemos del dueño del repositorio que nos puede aprobar y hacer el merge, nos puede hacer comentarios. Y con esto se acaba el flujo.
8. En caso de que nosotros seamos el admin del repositorio, podemos aprobar el repositorio subido por el usuario en el menú superior donde dice pull request.

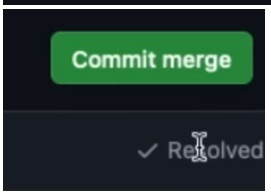


9. Si existe algún error github nos lo deja corregir por resolver conflicto si no es muy complicado el error.

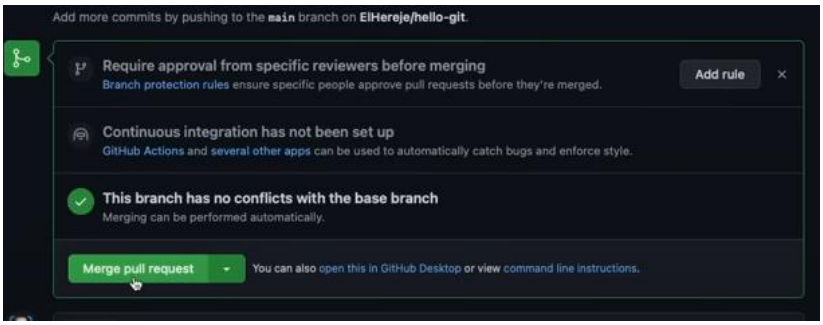


10. Nos muestra un editor para corregirlo, después Mark as resolved y luego Commit merge.

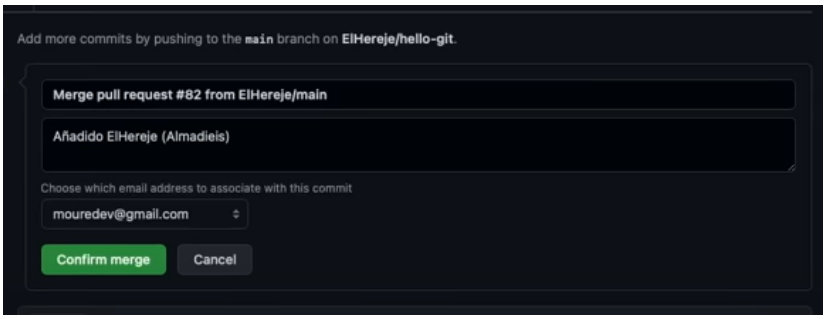




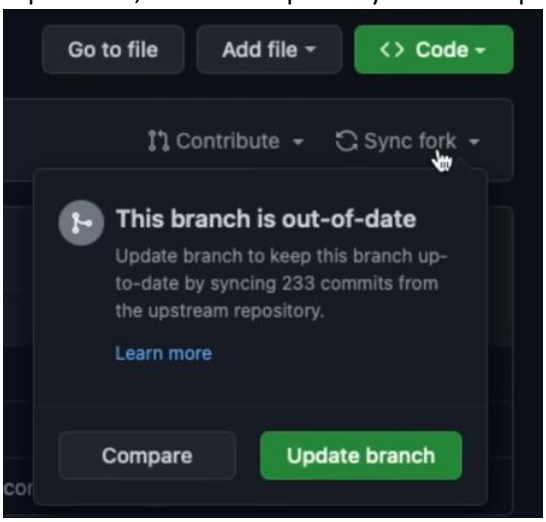
11. Después tenemos que actualizar cuando ya se corrigió.



12. Por ultimo confirmamos.



13. Ya con esto se actualizo el repositorio que teníamos, aceptando los cambios del usuario especifico.
14. Si queremos actualizar nuestro repositorio con respecto al repositorio del usuario principal, vamos a nuestro repositorio, click en la opción sync fork -> update branch



15. El archivo README.md siempre por defecto se va a mostrar (Visible en la pagina su contenido) en el repositorio de GitHub.