

GIT Y GITHUB

LAS CLASES EN EL SITIO

Comandos

ls: un listado de todos los directorios que tenemos en nuestra ubicación

cd: cd + (nombre del directorio), nos ubicamos en un directorio determinado

cd+(espacio)+(punto)(punto)osea cd .. : nos sirve para volver al directorio anterior

pwd: me dice cual es la ruta donde me encuentro

mkdir "nombre carpeta": nos sirve para crear carpeta

touch "nombre archivo": sirve para crear un archivo

<https://mouredev.com/git>

introducción git – github

git: pagina oficial [Git \(git-scm.com\)](https://git-scm.com)

libro de GIT [Git \(git-scm.com\)](https://git-scm.com) <https://git-scm.com/book/es/v2>

GIT es un sistema de control de versiones distribuido, un sistema de control de versiones nos permite llevar un historial de todos los cambios de mi proyecto.

- Instalación de GIT

En la página la versión de Windows y siguiente

Para trabajar GIT se puede desde la terminal o con versiones gráficas, pero la mejor forma de entender y aprender es desde la terminal

Probar si esta instalado git, simplemente el comando (git) y enter

Para ver la versión> git –version + enter

O >git -v

Ayuda > git – -help o git -h

Lo primero para configurar git es un usuario y un mail

Tu Identidad

Lo primero que deberás hacer cuando instales Git es establecer tu nombre de usuario y dirección de correo electrónico. Esto es importante porque los "commits" de Git usan esta información, y es introducida de manera inmutable en los commits que envías:

```
$ git config --global user.name "John Doe"
```

```
$ git config --global user.email "johndoe@example.com"
```

De nuevo, sólo necesitas hacer esto una vez si especificas la opción --global, ya que Git siempre usará esta información para todo lo que hagas en ese sistema. Si quieres sobrescribir esta información con otro nombre o dirección de correo para proyectos específicos, puedes ejecutar el comando sin la opción --global cuando estés en ese proyecto.

Muchas de las herramientas de interfaz gráfica te ayudarán a hacer esto la primera vez que las uses.

Comprobando tu Configuración

Si quieres comprobar tu configuración, puedes usar el comando `git config --list` para mostrar todas las propiedades que Git ha configurado:

```
$ git config --list(doble Guion medio)
```

Tambien

```
$ git config user.name
```

```
O $ git config user.email
```

Para crear un fichero de la consola usamos `$ mkdir <nombre del fichero>`

Para crear un archivo dentro de nuestra carpeta o fichero `$ touch <nombre del archivo>`

Para que en un directorio empiece a trabajar Git debemos digitar `git init`

ya creando la carpeta `.git` ya existe el repositorio para controlar versiones

(si quiero configurar temas para la terminal [Oh My Zsh - a delightful & open source framework for Zsh](#))

Renombramos la rama master por medio de `$ git branch -m main`

Ahora ya no sale master al final si no main

\$ git status= sirve para mirar el estado o configuración de nuestro proyecto dentro de git

Para que nuestro proyecto guarde fotografías de un archivo lo debemos agregar

Por medio de add

Para abrir visual estudio code desde git \$ code .(code(espacio)(punto))

\$ git add hellogit.py(agregamos ya para que se creen fotos o imágenes de este archivo)

Si queremos agregar todo lo de la carpeta git add (seguido de punto)

Ahora hay que iniciar el commit que es lanzar nuestro proyecto local lanzando una fotografía del mismo

Para no ir al editor donde nos pide el mensaje se realiza el siguiente código

\$git commit -m "este es mi primer commit" (-m significa message) para hacer el commit con mensaje

Git log y status

\$ git log: muestra la información del commit con autor, correo y fecha de creación

\$git status: nos muestra la información de cómo está nuestro proyecto, si tengo información modificada que se deba agregar o agregados que se deba hacer commit

Variaciones de git log

\$git log --graph: muestra una gráfica de ramas,

\$git log --graph --pretty=oneline: muestra en línea todo los commit

\$git log --graph --decorate --all --oneline=muestra todo en línea :muestra los commit con sus ramas y las variaciones de ruta que ha tenido

Git checkout y reset

git checkout hellogit2.py = de esta forma volvemos al estado anterior de nuestro archivo donde realizamos el último commit para no crear una fotografía del mismo y dejarlo como estaba originalmente

git reset= es para volver a la ultima fotografia que teníamos guardada

```
PS C:\Users\user\onedrive\fernando\ademass\git-github\helloGit> git reset
```

Unstaged changes after reset:

```
M    hellogit.py
```

Le informa que ese archivo esta modificado para volver al estado anterior utilizamos checkout

```
PS C:\Users\user\onedrive\fernando\ademass\git-github\helloGit> git checkout hellogit.py
```

Updated 1 path from the index

Nos dice que se ha actualizado el archivo, no lo deja como estaba con el último commit

```
PS C:\Users\USER\onedrive\fernando\ademass\git-github\hellogit> git log --graph
```

```
* commit eded2936dd3bec9adc1b34ed9147d217c0c45289 (HEAD -> main)
```

```
| Author: Fernando Ortiz <ortizreyesfernando@gmail.com>
```

```
| Date: Sat Oct 19 06:48:37 2024 -0500
```

```
|
```

```
| se actualiza el texto print
```

```
|
```

```
* commit ffd6730f6466f224c648caf98a8b376cc4413f82
```

```
| Author: Fernando Ortiz <ortizreyesfernando@gmail.com>
```

```
| Date: Thu Oct 17 06:31:28 2024 -0500
```

```
|
```

```
| este es mi segundo commit
```

```
|
```

```
* commit 379eaa7373929a0dea990164552e4dfa60144e74
```

```
Author: Fernando Ortiz <ortizreyesfernando@gmail.com>
```

```
Date: Thu Oct 17 06:24:01 2024 -0500
```

este es mi primer commit

git log --graph nos muestra una grafica o las ramas del proyecto

otra instrucción(`git log --graph --pretty=oneline`)

nos muestra los commit en una sola linea osea como un resumen

```
PS C:\Users\USER\onedrive\fernando\ademass\git-github\hellogit> git log --graph --pretty=oneline
```

```
* eded2936dd3bec9adc1b34ed9147d217c0c45289 (HEAD -> main) se actualiza el texto print
* ffd6730f6466f224c648caf98a8b376cc4413f82 este es mi segundo commit
* 379eaa7373929a0dea990164552e4dfa60144e74 este es mi primer commit
```

GIT ALIAS

Para hacer que diferentes comandos queden en una sola palabra o apodo se utiliza esta opción

Ejemplo

```
git log --graph --decorate --all --oneline
```

```
* eded293 (HEAD -> main) se actualiza el texto print
* ffd6730 este es mi segundo commit
* 379eaa7 este es mi primer commit
```

Esta instrucción me permite ver la información de los commit en forma resumida y en una línea, pero pues es muy larga, por eso creamos un alias que va a remplazar toda la instrucción

```
$git config --global alias.tree "log --graph --decorate --all --oneline"
```

Alias.(nombre para el alias que nosotros queramos) y entre comillas las instrucciones que queremos remplazar

IGNORAR ARCHIVOS

Para hacer que no aparezcan al momento de hacer status archivos que no queremos que se guarden de los cuales no queremos hacer fotografías, para eso hacemos lo siguiente

1 creamos un archivo por medio de: `$ touch .gitignore`= esto hace que creamos en la carpeta de nuestro proyecto el fichero oculto `.gitignore`

Lo que agreguemos dentro de este fichero son las rutas, ficheros o archivos que no queremos tener en cuenta

Abrimos el archivo `.gitignore` y dentro de el escribimos el fichero que deseamos ignorar en la siguiente forma:

En gitbash me funciona asi:

```
# Ignorar archivos .log
```

```
*.log
```

```
# Ignorar carpeta node_modules
```

```
node_modules/
```

```
# Ignorar archivos temporales de OS
```

```
.DS_Store
```

```
Thumbs.db
```

Después de incluir los archivos que queremos ignorar en la carpeta .gitignore, adicionamos por medio de git add .gitignore y hacemos commit git commit -m "incluimos el gitignore"

Ahora si realizamos status debe aparecer nuestro repositorio limpio

GIT DIFF

Para buscar que he cambiado con respecto a la última fotografía o commit que he hecho antes de hacer el commit

```
@@ -1 +1 @@
```

```
-print("new Hello git")
```

```
\ No newline at end of file
```

```
+print("new Hello git whit changes!")
```

```
\ No newline at end of file
```

Dice que hemos quitado una linea y muestra la linea que quitamos y que hemos agregado otra

DESPLAZAMIENTO

Por medio de git checkout (id del commit) donde nos queremos desplazar se logra ubicarse en una de nuestras fotografías anteriores y de la misma forma podemos volver a la ultima o a la primera

```
$ git checkout 379eaa7
```

Note: switching to '379eaa7'.(cambiamos a este id que identifica el primer commit)

You are in 'detached HEAD' state. You can look around, make experimental changes and commit them, and you can discard any commits you make in this state without impacting any branches by switching back to a branch.

(Te encuentras en estado de «CABEZA desprendida». Puede echar un vistazo, hacer cambios experimentales y confirmarlos, y puede descartar cualquier confirmación que haga en este sin afectar a ninguna rama cambiando de nuevo a una rama.)

If you want to create a new branch to retain commits you create, you may do so (now or later) by using -c with the switch command. Example:

(Si desea crear una nueva rama para retener los commits que cree, puede hacerlo (ahora o más tarde) usando -c con el comando switch. Ejemplo:)

```
git switch -c <new-branch-name>
```

Or undo this operation with:

```
git switch -
```

Turn off this advice by setting config variable advice.detachedHead to false

HEAD is now at 379eaa7 este es mi primer commit

(Desactive este consejo estableciendo la variable de configuración advice.detachedHead en false)

(HEAD está ahora en 379eaa7 este es mi primer commit)

Ok, ahora si queremos que nuestra cabecera del proyecto cambie (aunque en la prueba que hice el HEAD apenas cambie de rama también se ubica en esa rama sin necesidad de hacer la siguiente instrucción)

Git checkout HEAD (que dice que es para ubicar la HEAD de nuestras ramas del proyecto)

Si vemos la información con nuestro alias tree

\$ git tree

* 6e56f58 (**main**) archivos ignorados incluidos .txt

* 48ccb46 agregue el gitignore

* eded293 se actualiza el texto print

* ffd6730 este es mi segundo commit

* 379eaa7 (**HEAD**) este es mi primer commit

No muestra que el main esta en la ultima rama y HEAD en el primer commit, para volver a la ultima rama o a otra solo debemos digitar \$ git checkout (id de la rama)6e56f58

GIT RESET HARD Y REFLOG

El reset es para dejar nuestro proyecto como estaba sin guardar los cambios en el ultimo commit y el reset --hard es para eliminar fotografías para que nuestra rama inicie en un punto que nosotros elijamos donde nos parezca que nuestro proyecto esta mejor o que de en ese estado es donde vamos a continuar

\$ git reset --hard (id de Nuestro commit donde queremos ubicarnos ahora y obviar todo lo que hemos hecho)

\$ git reset --hard 48ccb46

HEAD is now at 48ccb46 agregue el gitignore (nos muestra ahora donde debemos continuar el proyecto)

Ahora si queremos volver a un commit que ya no aparece con git-log lo buscamos con git-reflog y con reset hard podemos volver de nuevo a ese commit

Pero si queremos deshacer el reset hard, también se puede 1:24

GIT REFLOG

Me muestra todos los movimientos que se han hecho en los commit creados

Si queremos volver a nuestro punto de trabajo podemos hacer git reset --hard (y el id de nuestra rama)

USER@DESKTOP-0KP4BHJ MINGW64 ~/onedrive/fernando/ademass/git-github/hellogit
(main)

\$ git log

commit 379eaa7373929a0dea990164552e4dfa60144e74 (HEAD -> main)

Author: Fernando Ortiz <ortizreyesfernando@gmail.com>

Date: Thu Oct 17 06:24:01 2024 -0500

este es mi primer commit

estamos ubicado en el primer commit ahora digitamos \$ git reflog, para ver todo el historial buscamos el número id de nuestro último commit y digitamos \$ git reset --hard (id)

GIT TAG

Para crear etiquetas para nuestros commit

\$ git tag clase_1, nombramos nuestro commit con clase_1

Se pueden crear diferentes Tag en tu proyecto como señal de que son puntos importantes y para poder ver la lista de tags simplemente digitas \$ git tag

Y para ubicarse en ese tag se hace un checkout así:

\$ git checkout tags/clase_1 (clase_1 o el nombre del tag al cual te quieras desplazar)

GIT BRANCH SWITCH

Cuando se quiere hacer algo en paralelo pero que no va a estar ligado con nuestro main o rama principal creamos otra rama un Branch \$ git Branch (nombre de la rama)

Por ejemplo vamos a hacer una función que sirva para logearse o identificarse va a ser la rama login

\$ git Branch login

Y para cambiar a la rama login utilizamos switch \$ git switch login

GIT MERGE

Sirve para combinar lo que se está haciendo en otra rama diferente que donde estamos ubicados, o sea si estoy en la rama login y ha habido cambios en la rama main que no se reflejan en mi rama con el comando merge se hace eso

Git merge main (me manda al editor de texto que le coloque un mensaje para salir se digita

:wq

Merge made by the 'ort' strategy.

hellogit3.py | 2 +/-

1 file changed, 1 insertion(+), 1 deletion(-)

Me dice que se cambio un archivo o que hubo una inserción, actualizo y modifíco los archivos de esta rama con lo que había en la otra rama y automáticamente realiza un nuevo commit

CONFLICTOS EN GIT

```
$ git merge main
```

Auto-merging hellogit3.py

CONFLICT (content): Merge conflict in hellogit3.py

Automatic merge failed; fix conflicts and then commit the result.

Al hacer merge desde la rama login para ver el contenido de la rama main Git nos detecta un problema indicando que hay un conflicto En el archivo hellogit3 y dice que falla el merge nos muestra directamente en el editor la línea con conflicto

```
<<<<<<< HEAD
```

```
print("Hello git 3 v login;")
```

```
=====
```

```
print("Hello git 3 v3;")
```

```
>>>>>>> main
```

Y pregunta en el editor con cual rama se debe trabajar pero antes hay un aviso En el editor, se arreglan las líneas en conflicto y después como se hizo un cambio se adiciona luego se hace commit

GIT STASH

El comando stash nos sirve para guardar algo sin hacer commit pues porque es un código incompleto o al cual le realizamos cambio y no queremos subirlo todavía

para hacer un stash se digita `$git stash`

para ver la lista de stash `$git stash list`

\$ git switch main(estando en la rama login)

error: Your local changes to the following files would be overwritten by checkout:

login.py

Please commit your changes or stash them before you switch branches.

Aborting

Git no deja cambiar de rama si no has hecho commit

Al hacer stash ya permite salir de la rama cuando volvemos debemos hacer (git stash pop) para que nos recupere lo que teníamos o lo que estábamos haciendo

Ahora si queremos borrar el stash que teníamos guardado \$git stash drop

REINTEGRACION EN GIT (2:10)

Con el merge se puede integrar lo de una rama con la otra, pero antes de hacer merge podemos mira si la otra rama tiene cambio o no a traves de (git diff nombre de la rama)

\$git diff login

diff --git a/.gitignore b/.gitignore

index 5ea7b47..b2d5aec 100644

--- a/.gitignore

+++ b/.gitignore

@@ -1,2 +1,3 @@

#ignono los archivos .txt

***.txt**

+querida.css

Muestra los cambios que hay en la otra rama

Ahora para traer lo que hay en login para main se hace merge

\$git merge login (con este comando integramos lo de login en main)

ELIMINACION DE RAMAS

Ya se integro lo de la rama login en la rama main, ya no se necesita la rama login pues ya esta integrado todo el codigo para eliminar la rama que ya no se necesita

\$git branch -d login (comando para eliminar la rama login, debemos esta ubicados en la rama principal ose main para nuestro ejemplo)

Deleted branch login (was e37c126).

GITHUB

Credenciales: ortizreyesfernando@hotmail.com clave: Tortolo900218

2:22 introduccion github

Tarea: crear cuenta git hub

Documentacion de git-hub [GitHub Docs](#)

SINCRONIZAR CAMBIOS

LOCAL Y REMOTO

AUTENTICACION SSH

2:50

[Configuración de Git - Documentación de GitHub](#)

Verificar si tengo claves ssh

Antes de generar una nueva clave SSH, debes comprobar la máquina local en busca de claves existentes.

1. Abra Git Bash.
2. Escriba `ls -al ~/.ssh` para ver si hay claves SSH existentes.
3. `$ ls -al ~/.ssh`
4. # Lists the files in your .ssh directory, if they exist
5. Comprueba la lista de directorio para ver si ya tiene una clave SSH pública. De manera predeterminada, los nombres de archivo de claves públicas admitidas para GitHub son uno de los siguientes.
 - `id_rsa.pub`
 - `id_ecdsa.pub`
 - `id_ed25519.pub`

Tip

Si recibes un error que indica que `~/.ssh` no existe, no tienes un par de claves SSH en la ubicación predeterminada. Puedes crear un par de llaves SSH nuevas en el siguiente paso.

6. Puedes ya sea generar una llave SSH nueva o cargar una existente.
 - Si no tienes un par de llaves pública y privada compatibles o si no quieres utilizar cualquiera que esté disponible, genera una llave SSH nueva.
 - Si ves un par de claves pública y privada existente (por ejemplo, `id_rsa.pub` y `id_rsa`) que le gustaría usar para conectarse a GitHub, puede agregar la clave a `ssh-agent`.

Para más información sobre la generación de una nueva clave SSH o la adición de una clave existente al agente SSH, consulta "[Generación de una nueva clave SSH y adición al agente SSH](#)".

Una vez que has comprobado las claves SSH existentes, puedes generar una nueva clave SSH para usarla para la autenticación y luego agregarla al `ssh-agent`.

Generar clave ssh

Puedes generar una nueva clave SSH en el equipo local. Después de generar la clave, puede agregar la clave pública a la cuenta en GitHub.com para habilitar la autenticación para las operaciones de Git a través de SSH.

1. Abra Git Bash.
2. Pega el texto siguiente, reemplazando el correo electrónico usado en el ejemplo por tu dirección de correo electrónico de GitHub. (`ortizreyesfernando@gmail.com`)
3. `ssh-keygen -t ed25519 -C "your_email@example.com"`

Nota: Si usas un sistema heredado que no admite el algoritmo Ed25519, usa:

```
ssh-keygen -t rsa -b 4096 -C "your_email@example.com"
```

Esto crea una llave SSH utilizando el correo electrónico proporcionado como etiqueta.

> Generating public/private ALGORITHM key pair.

Cuando se te pida: "Introduce un archivo en el que se pueda guardar la clave", teclea **Enter** para aceptar la ubicación de archivo predeterminada. Ten en cuenta que si ya creaste claves SSH anteriormente, `ssh-keygen` puede pedirte que vuelvas a escribir otra clave. En este caso, se recomienda crear una clave SSH con nombre personalizado. Para ello, escribe la ubicación de archivo predeterminada y reemplaza `id_ALGORITHM` por el nombre de clave personalizado.

> Enter file in which to save the key (/c/Users/YOU/.ssh/id_ALGORITHM):[Press enter]

4. Cuando se le pida, escriba una frase de contraseña segura. Para obtener más información, vea «[Trabajar con contraseñas de clave SSH](#)».
5. > Enter passphrase (empty for no passphrase): [Type a passphrase]

> Enter same passphrase again: [Type passphrase again]

Yo escribo esta(Tortolo900218**)

Ahora al comprobar si tengo clave ssh sale esto

USER@DESKTOP-0KP4BHJ MINGW64 ~ (master)

```
$ ls -al ~/.ssh
```

```
total 18
```

```
drwxr-xr-x 1 USER 197121  0 Dec  3 06:38 ./
```

```
drwxr-xr-x 1 USER 197121  0 Dec  3 06:32 ../
```

```
-rw-r--r-- 1 USER 197121 464 Dec  3 06:38 id_ed25519
```

```
-rw-r--r-- 1 USER 197121 110 Dec  3 06:38 id_ed25519.pub
```

Ose que ya tengo una clave SSH en mi equipo

Inicio el agente sh en segundo plano

[Generación de una nueva clave SSH y adición al agente SSH - Documentación de GitHub](#)

2:55

Agregar tu clave SSH al ssh-agent

1. En una nueva ventana de PowerShell con *privilegios elevados de administrador*, asegúrate de que el agente ssh se esté ejecutando. Puede usar las instrucciones de "Auto-lanzamiento ssh-agent" en "[Trabajar con contraseñas de clave SSH](#)" o iniciarla manualmente:

```
# start the ssh-agent in the background
```

```
Get-Service -Name ssh-agent | Set-Service -StartupType Manual
```

```
Start-Service ssh-agent
```

```
PS C:\Users\USER> ssh-add c:/Users/User/.ssh/id_ed25519
```

```
Enter passphrase for c:/Users/User/.ssh/id_ed25519 (aca escribe: Tortolo900218**)
```

```
Identity added: c:/Users/User/.ssh/id_ed25519 (ortizreyesfernando@gmail.com)
```

```
PS C:\Users\USER>
```

Agregar una clave SSH nueva a tu cuenta de GitHub


Para configurar tu cuenta en GitHub.com a fin de utilizar tu clave SSH nueva (o existente), también necesitarás agregar la clave a tu cuenta.

Después de generar un par de claves SSH, debes agregar la clave pública a GitHub.com para habilitar el acceso SSH en la cuenta.

1. Entramos a Github,
2. En nuestro usuario en la imagen parte superior derecha, click y luego click en settings
3. Entramos en (SSH and GPG keys)
4. Click en new ssh key
5. Abrimos con bloc de notas la clave que esta en la carpeta .ssh id_ed25519.pub
6. Copiamos todo el contenido en el portapapeles y lo agregamos en el espacio donde dice (key)
7. Click en (add ssh key)
8. Obtenemos esto:

This is a list of SSH keys associated with your account. Remove any keys that you do not recognize.

Authentication keys

 **ortizreyesfernando@gmail.com**
SHA256: bkgiGudU5v1pPC5MB0uchqOE0RZ+jMhgWvIpOg9oUQI
Added on Dec 5, 2024
Never used — Read/write

Delete

Check out our guide to [connecting to GitHub using SSH keys](#) or troubleshoot [common SSH problems](#).

Probar tu conexión SSH

Después de configurar la clave SSH y agregarla a GitHub.com, puede probar la conexión.

1. Abra Git Bash.
2. Escriba lo siguiente:

Shell

```
ssh -T git@github.com
```

```
# Attempts to ssh to GitHub
```

Puedes ver una advertencia como la siguiente:

- > The authenticity of host 'github.com (IP ADDRESS)' can't be established.
- > ED25519 key fingerprint is SHA256:+DiY3wvvV6TuJJhbpZisF/zLDA0zPMSvHdkr4UvCOqU.
- > Are you sure you want to continue connecting (yes/no)?

3. Compruebe que la huella digital del mensaje que ve coincide con [la huella digital de clave pública de GitHub](#). En caso afirmativo, escriba yes:
4. > Hi USERNAME! You've successfully authenticated, but GitHub does not
5. > provide shell access.

Note

El comando remoto debe cerrarse con el código 1.

6. Comprueba que el mensaje resultante contenga tu nombre de usuario. Si recibes un mensaje de "permiso denegado", consulta "[Error: Permiso denegado \(publickey\)](#)".
7. USER@DESKTOP-0KP4BHJ MINGW64 ~ (master)
8. \$ ssh -T git@github.com
9. The authenticity of host 'github.com (140.82.114.3)' can't be established.
10. ED25519 key fingerprint is
SHA256:+DiY3wvV6TuJJhbpZisF/zLDA0zPMSvHdkr4UvCOqu.
11. This key is not known by any other names.
12. Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
13. Warning: Permanently added 'github.com' (ED25519) to the list of known hosts.
14. Enter passphrase for key '/c/Users/USER/.ssh/id_ed25519': Tortolo900218**
15. Hi FenanoOrtiz! You've successfully authenticated, but GitHub does not provide shell access.

La autenticación ha sido un éxito

Ya se encuentra mi pc conectado a github, si tengo mas computadores donde quiere trabajar con un repositorio o mi cuenta debo hacer el mismo proceso en cada pc

REPOSITORIO PROYECTO

Git Remote

Creemos un nuevo repositorio en git-hub, con el nombre hello-git puede ser cualquier nombre, luego nos ubicamos en nuestra pc en la carpeta donde estamos trabajando el proyecto de ejemplo hellogit

La siguiente instrucción se coloca en git-bash para conectar el repositorio de github a nuestra carpeta hellogit

```
git remote add origin https://github.com/FenanoOrtiz/hello-git.git
```

al ejecutar no debe mostrar ningún error ni mostrar ningún mensaje, pero internamente ya queda emparejado mi proyecto local con el repositorio que cree en github

```
git push -u origin main
```

luego realizamos esta instruccion para cargar todo el proyecto al repositorio de git-hub

USER@DESKTOP-0KP4BHJ MINGW64 ~/onedrive/fernando/ademass/git-github/hellogit (main)

```
$ git push -u origin main
```

info: please complete authentication in your browser...

Enumerating objects: 47, done.

Counting objects: 100% (47/47), done.

Delta compression using up to 12 threads

Compressing objects: 100% (32/32), done.

Writing objects: 100% (47/47), 4.06 KiB | 831.00 KiB/s, done.

Total 47 (delta 14), reused 0 (delta 0), pack-reused 0 (from 0)

remote: Resolving deltas: 100% (14/14), done.

To https://github.com/FenanoOrtiz/hello-git.git

* [new branch] main -> main

branch 'main' set up to track 'origin/main'.

Salió una Ventana de git bash pidiendo autorización acepté y listo

SUBIDA PROYECTO

3:07

Dentro de github voy hacer el readme para el repositorio con cualquier texto, y vamos a git-bash y realizamos cambios en un fichero, luego adicionamos y realizamos commit, luego intentamos realizar la instrucción git push, git hub no nos permite subir porque hay otros cambios en el repositorio que no tenemos actualizado en el pc, se va a utilizar las siguientes instrucciones

```
USER@DESKTOP-0KP4BHJ MINGW64 ~/onedrive/fernando/ademass/git-github/hellogit (main)
$ git push
To https://github.com/FenanoOrtiz/hello-git.git
 ! [rejected]        main -> main (fetch first)
error: failed to push some refs to 'https://github.com/FenanoOrtiz/hello-git.git'
hint: Updates were rejected because the remote contains work that you do not
hint: have locally. This is usually caused by another repository pushing to
hint: the same ref. If you want to integrate the remote changes, use
hint: 'git pull' before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
```

GIT FETCH Y PULL

GIT FECH

Descarga a mi pc el historial sin los cambios que hay en el repositorio

USER@DESKTOP-0KP4BHJ MINGW64 ~/onedrive/fernando/ademass/git-github/hellogit
(main)

\$ git fetch

remote: Enumerating objects: 4, done.

remote: Counting objects: 100% (4/4), done.

remote: Compressing objects: 100% (2/2), done.

remote: Total 3 (delta 1), reused 0 (delta 0), pack-reused 0 (from 0)

Unpacking objects: 100% (3/3), 931 bytes | 58.00 KiB/s, done.

From https://github.com/FenanoOrtiz/hello-git

a8c7a38..fbd6ba8 main -> origin/main

```
USER@DESKTOP-0KP4BHJ MINGW64 ~/onedrive/fernando/ademass/git-github/hellogit (main)
$ git tree
* 1df8d75 (HEAD -> main) hello git-hub
| * fbd6ba8 (origin/main) Create README.md
|/
| * a8c7a38 modifique el gitignore
|/
| * e37c126 modifique el gitignore al agregar querida.css
| * 918b96e Merge branch 'login'
|/
| * b0b7498 login v2
```

Ya nos muestra en git el readme que se creo en github, ahora si estamos de acuerdo con esos cambio descargamos el historial con los cambios con git pull

Al digitar git pull nos abre un cuadro de dialogo indicando que se va hacer un merge y que coloque un mensaje (se digito (:.)osea dos puntos y después punto) y realizo merge sobre nuestro main

Merge made by the 'ort' strategy.

README.md | 1 +

1 file changed, 1 insertion(+)

create mode 100644 README.md

```

USER@DESKTOP-OKP4BHJ MINGW64 ~/onedrive/fernando/ademass/git-github/hellogit (main)
$ git tree
*   ffb7e57 (HEAD -> main) Merge branch 'main' of https://github.com/FenanoOrtiz/hello-git
| \
|  * fbd6ba8 (origin/main) Create README.md

```

Ya tenemos agregados los cambios de github a nuestro pc

GIT PUSH

Ya sincronizamos lo que habia en github en nuestro pc ahora tenemos que subir los cambios del pc al ordenador

```

USER@DESKTOP-OKP4BHJ MINGW64 ~/onedrive/fernando/ade
$ git status
On branch main
Your branch is ahead of 'origin/main' by 2 commits.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean

```

Nos dice que usemos git push para publicar lo que tenemos en el pc o el commit local

```

USER@DESKTOP-OKP4BHJ MINGW64 ~/onedrive/fernando/ademass/git-github/hello
$ git push
Enumerating objects: 9, done.
Counting objects: 100% (8/8), done.
Delta compression using up to 12 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (5/5), 535 bytes | 535.00 KiB/s, done.
Total 5 (delta 2), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (2/2), completed with 1 local object.
To https://github.com/FenanoOrtiz/hello-git.git
   fbd6ba8..ffb7e57  main -> main

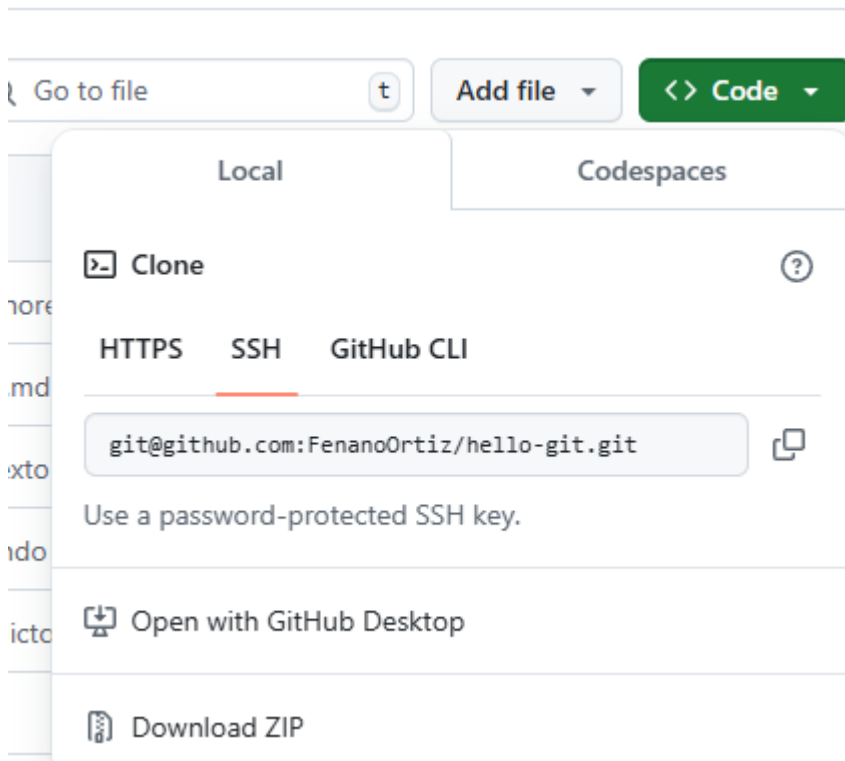
```

Ahora revisamos si nuestro repositorio en github se actualizo con los cambios hechos en el pc

GIT CLONE

Nos permite traer un proyecto completo desde github a nuestro pc para trabajar realizar cambios y después se subiría esos cambios al github

El repositorio que queramos clonar en github en la pestaña code nos muestra opciones para clonar utilizamos la opción SSH que fue la que configuramos en el computador



Copiamos esa direccion para clonar y vamos a gitbash nos ubicamos en la carpeta donde deseamos descargar el proyecto y digitamos lo siguiente

```
USER@DESKTOP-0KP4BHJ MINGW64 ~/onedrive/fernando/ademass/git-github/proyecto (master)
$ git clone git@github.com:FenanoOrtiz/hello-git.git
Cloning into 'hello-git'...
Enter passphrase for key '/c/Users/USER/.ssh/id_ed25519':
remote: Enumerating objects: 50, done.
remote: Counting objects: 100% (50/50), done.
remote: Compressing objects: 100% (20/20), done.
remote: Total 50 (delta 15), reused 46 (delta 14), pack-reused 0 (from 0)
Receiving objects: 100% (50/50), 4.94 KiB | 843.00 KiB/s, done.
Resolving deltas: 100% (15/15), done.
```

Como yo le cree frase de seguridad SSH siempre tengo que digitarla (Tortolo900218**)

Hemos creado una copia del repositorio en nuestro ordenador

GITHUB FORK

Sirve para descargar una copia de todo un proyecto en tu cuenta de github, que de esa forma ya puedes descargar a la computadora y realizar las modificaciones que quieras y volver a subir a tu cuenta de github

Hacemos el git clone del repositorio y vamos a crear un nuevo archivo que después vamos a subir de nuevo a github

FLUJO COLABORATIVO

Al realizar el clone del repositorio de mouredev no me sirvió con SSH por tal motivo realice el clone con HTTPS me funciono bien

Ya teniendo el fichero de mouredev agrego mi usuario al archivo hello.md y gurardo

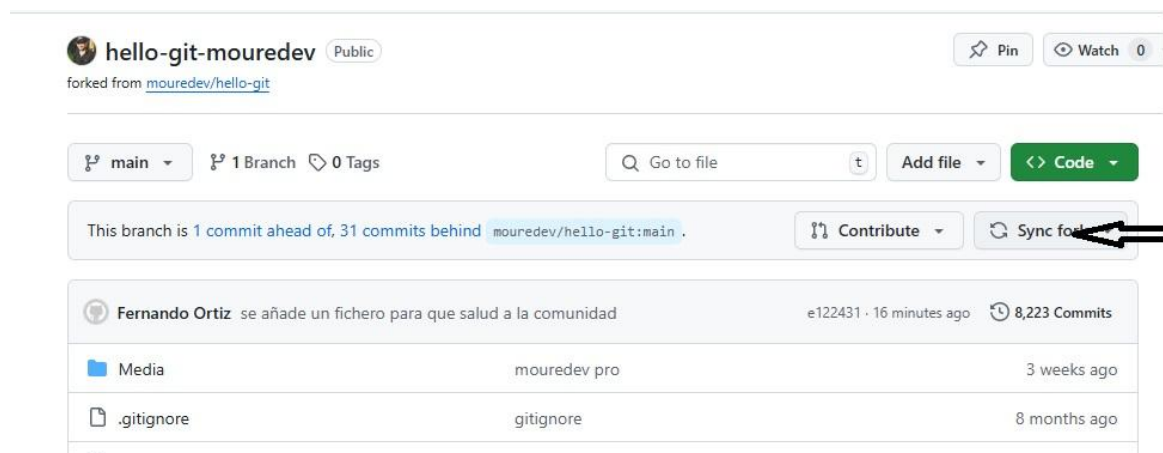
Realizado en gitbasch git add . para adicionar el archivo luego realizo el commit

Con mensaje “se añade un fichero para que salude a la comunidad”

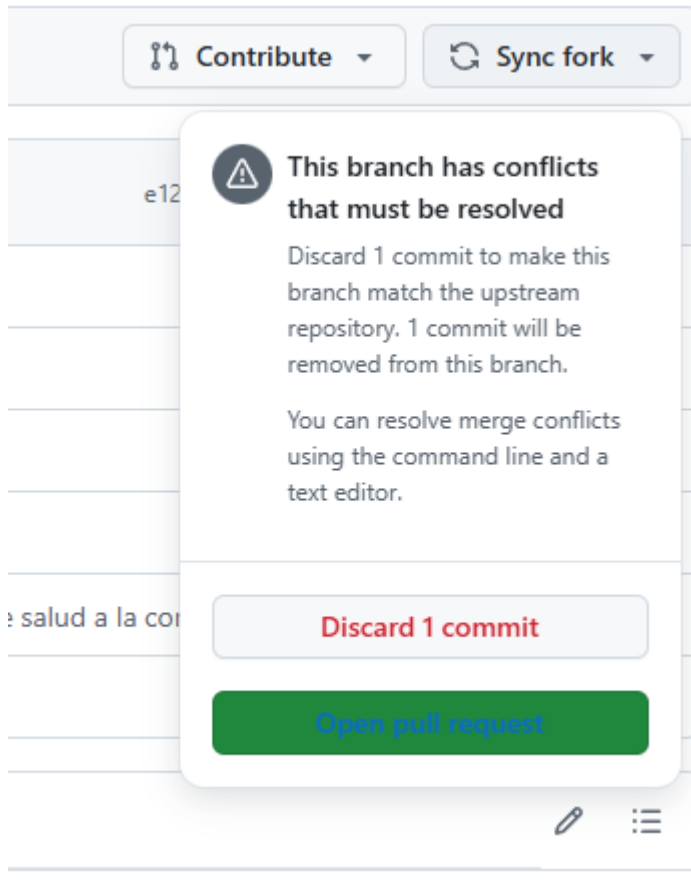
Ahora vamos a subir a github por medio de git push, después de hecho el push vamos a github y en el repositorio vemos que se ha realizado el cambio que hemos hecho en nuestro pc

Ahora para que el repositorio origina tenga los cambio que hemos hechos se realiza el siguiente paso

Lo primero es realizar sync fork



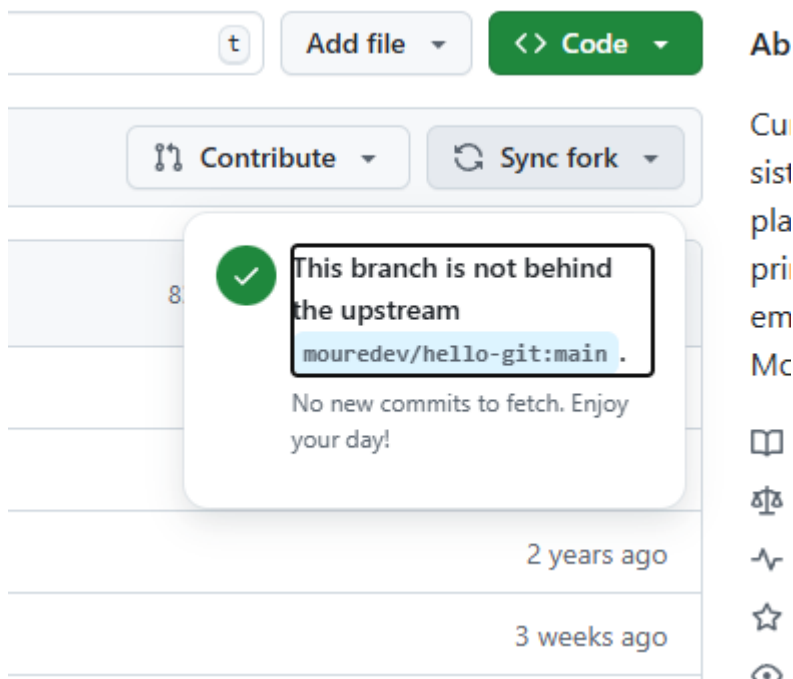
Es una especie de merge entre repositorios de usuarios distintos que busca cambios entre los dos repositorios el original y e que tenemos en nuestra cuenta y nos dice si esta todo correcto si no existen conflictos para poder solicitar agregar nuestros cambios al repositorio original



Existían cambio en la rama original que habían hecho otros usuarios

Descargue el commit,

En gitbash realice un git pull, Para actualizar todo el contenido depues fui a visual y acepte los camibos que descargue para eliminar conflictos guarde y realice el git pusck ahora voy de nuevo a Sync fork



PULL REQUEST

Vamos a la opción contribuir y luego open pull request

practica pull request #3073

Open FenanoOrtiz wants to merge 3 commits into `mouredev:main` from `FenanoOrtiz:main`

Conversation 0 Commits 3 Checks 0 Files changed 1

FenanoOrtiz commented now

ejemplo practica curso git-github

Fernando Ortiz added 3 commits 35 minutes ago

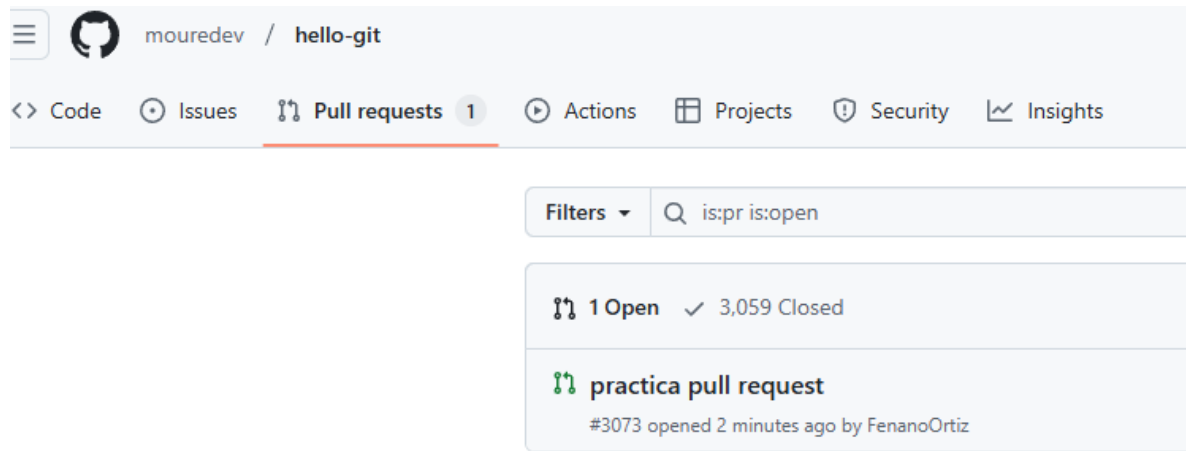
- se añade un fichero para que salud a la comunidad e122431
- modificacion de repositorio 6e2fc35
- cambios realizados en el repo 83e0c1f

✓ This branch has no conflicts with the base branch
Only those with [write access](#) to this repository can merge pull requests.

[Try the new merge experience](#)

Vemos los commit que realice y los cambios que se van a enviar para colaborar con el repositorio original

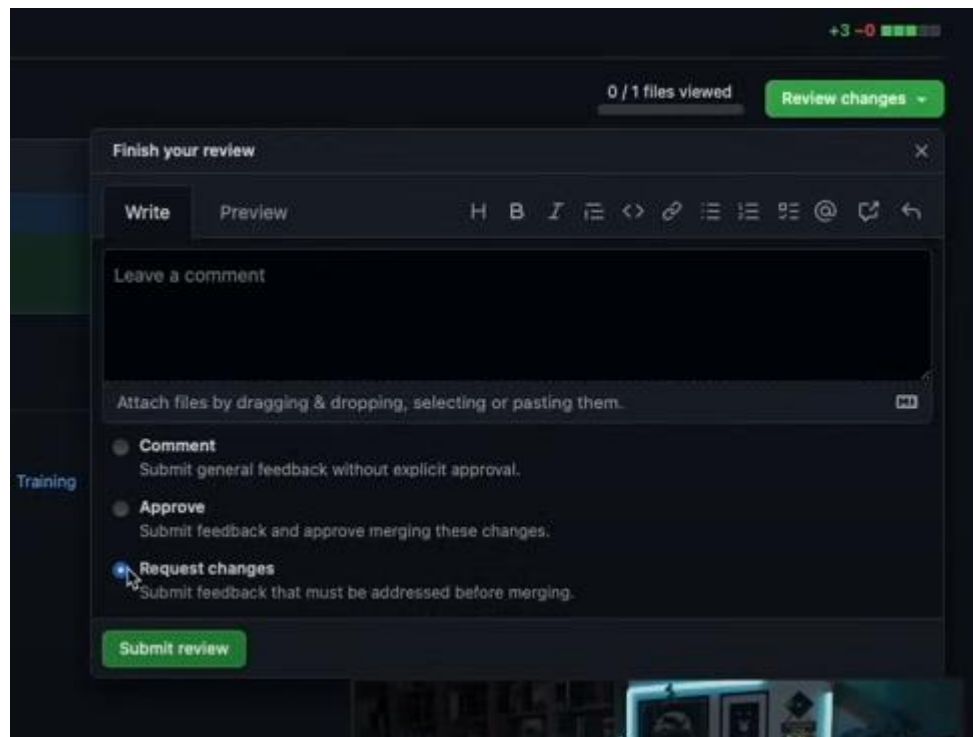
En la cuenta de mouredev le debe aparecer la solicitud de pull request



Pr

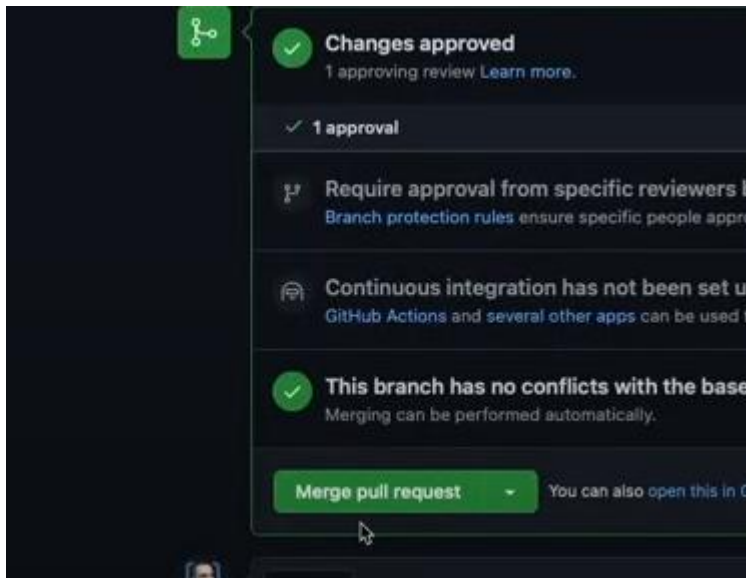
En el fichero de mouredev aparece lo siguiente, la solicitud y el usuario

Desde la cuenta original se puede ver lo cambios que se quieren agregar y después realizar una revisión si se decide aceptar los cambios



puede en la pestaña de revisar cambios, enviar un comentario a quien realizo la solicitud de pull request, aprobar los cambios propuestos o rechazar y se envia un mensaje para continuar con el flujo colaborativo

ahora después de revisar y aprobar se hace MERGE PULL REQUEST



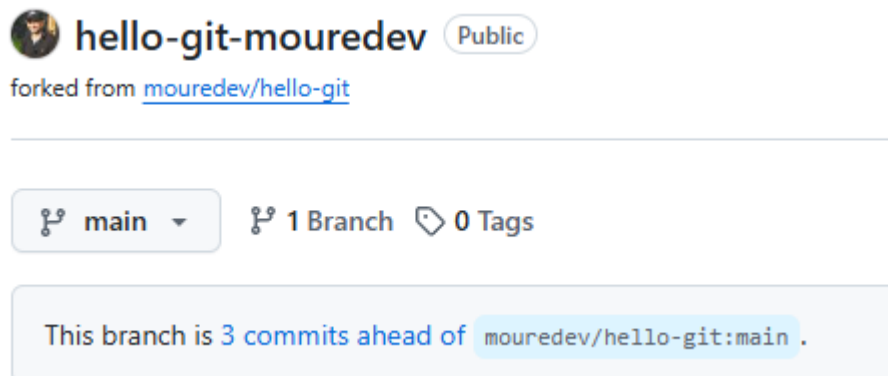
Esto desde la pagina donde esta el repositorio original

Luego confirmar

EJERCICIO PRACTICO, FUE lo que hice con el archivo hello.md que se realizado el fork, clone, push, pull request

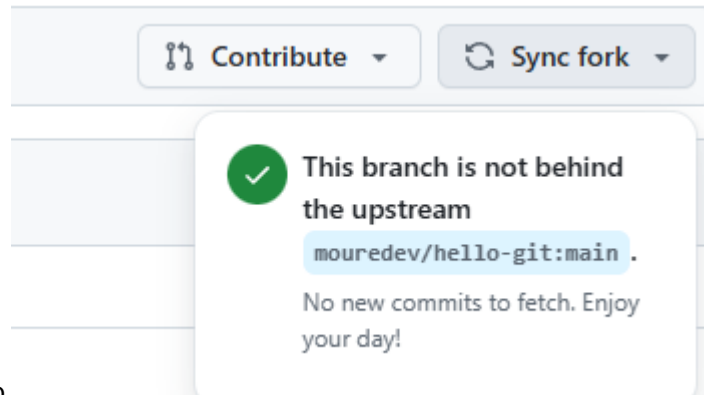
3:37—3:53

SINCRONIZACION FORK



Github me informa que la rama tiene 3 commit desde que yo hice el fork

Utilizamos la opción SYNC FORK para comparar y para actualizar los cambios que tenga el repositorio



En nuestro ejemplo
hay cambios o nuevos commits que debamos actualizar o revisar

nos dice que no

GITHUB MARKDOWN

MARKDOWN es un editor de texto, para hacer mejores presentaciones dentro de github

[Sintaxis de escritura y formato básicos - Documentación de GitHub](#)

HERRAMIENTAS GRAFICAS

4:02

GITHUB DESKTOP (oficial de github)

GITKRAKEN

SOURCETREEAPP

GIT-FORK

NOTA: practicar totalmente el flujo de git con el terminal, utilizando comandos y después probar las dos últimas herramientas gráficas que explica el expositor

GIT & GITHUB FLOW

Flujo de trabajo recomendado



<https://gitkraken.com/learn/git/best-practices/git-branch-strategy>



<https://atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow>

[Flujo de trabajo de Gitflow | Atlassian Git Tutorial](#) (en español)

GIT-FLOW

Estrategia de rama de Git Flow

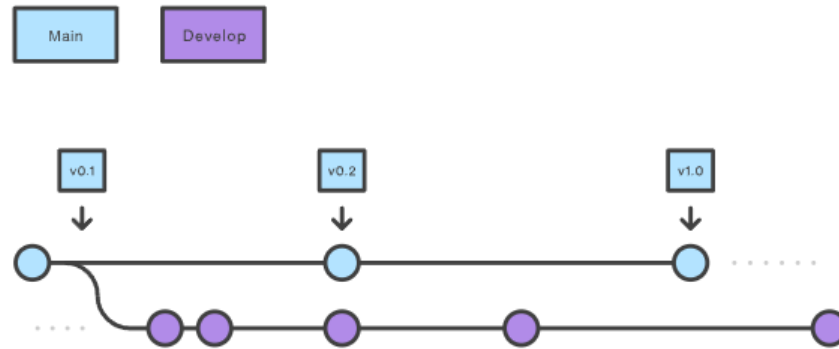
La idea principal detrás [de la estrategia de bifurcación de flujo de Git](#) es aislar tu trabajo en diferentes tipos de ramas. Hay cinco tipos de ramas diferentes en total:

- Principal (MAIN)
- Desarrollar (DEVELOP)
- Característica (FEATURE)
- Lanzamiento (RELEASE)
- Revisión (HOTFIX)

Las dos ramas principales del flujo de Git son *main* y *develop*. Hay tres tipos de ramas auxiliares con diferentes propósitos previstos: *característica*, *versión* y *revisión*.

La rama principal se utiliza solo para las versiones (main)

La rama DEVELOP es donde se montan todos los desarrollos que se van realizando que ya están probados y listos para desplegar, cuando ya este todo aprobado se monta en main que sería como la nueva versión del programa o app que se esté realizando



En la rama main solo existen los puntos donde se despliega la app en ciertos periodos de tiempo,

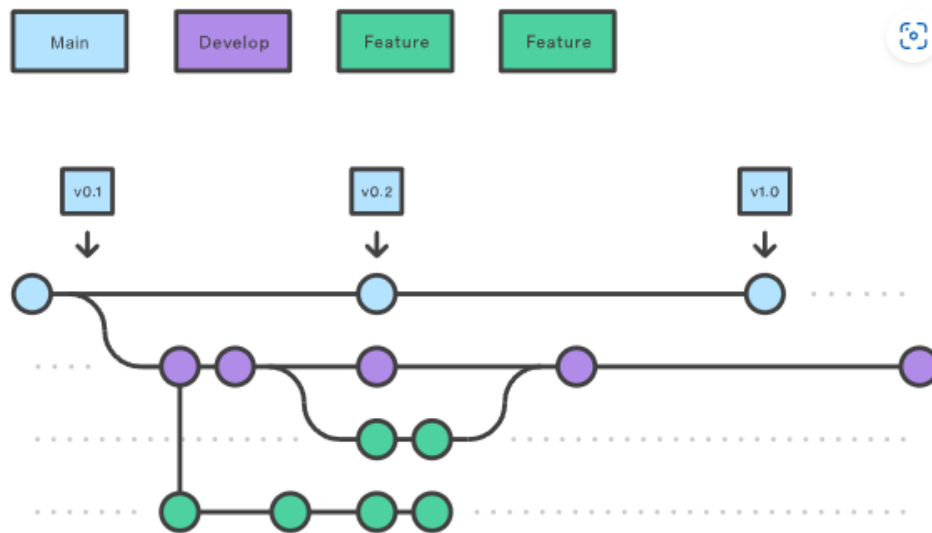
Feature

Es la rama donde se desarrollan los programas y se agregan a la rama Develop y pueden ser diferentes equipos de trabajo

Ramas de función

Paso 1. Crea el repositorio

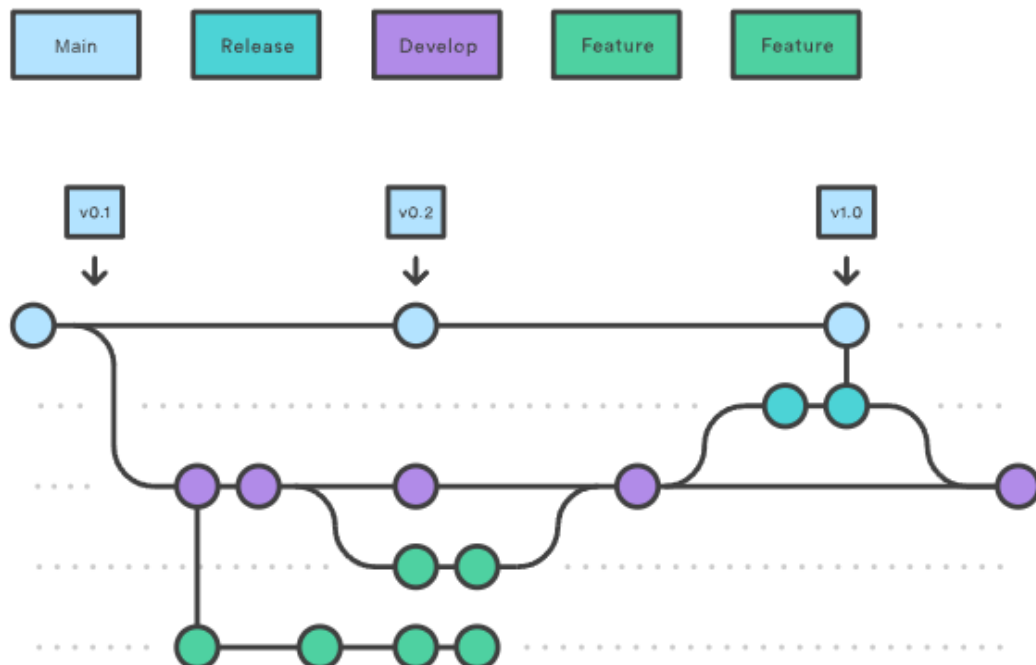
Todas las funciones nuevas deben residir en su propia rama, que se puede [enviar al repositorio central](#) para copia de seguridad/colaboración. Sin embargo, en lugar de ramificarse de main, las ramas feature utilizan la rama develop como rama primaria. Cuando una función está terminada, [se vuelve a fusionar en develop](#). Las funciones no deben interactuar nunca directamente con main.



Ten en cuenta que las ramas feature combinadas con la rama develop conforman, a todos efectos, el flujo de trabajo de ramas de función. Sin embargo, el flujo de trabajo Gitflow no termina aquí.

Las ramas feature suelen crearse a partir de la última rama develop.

Ramas de publicación

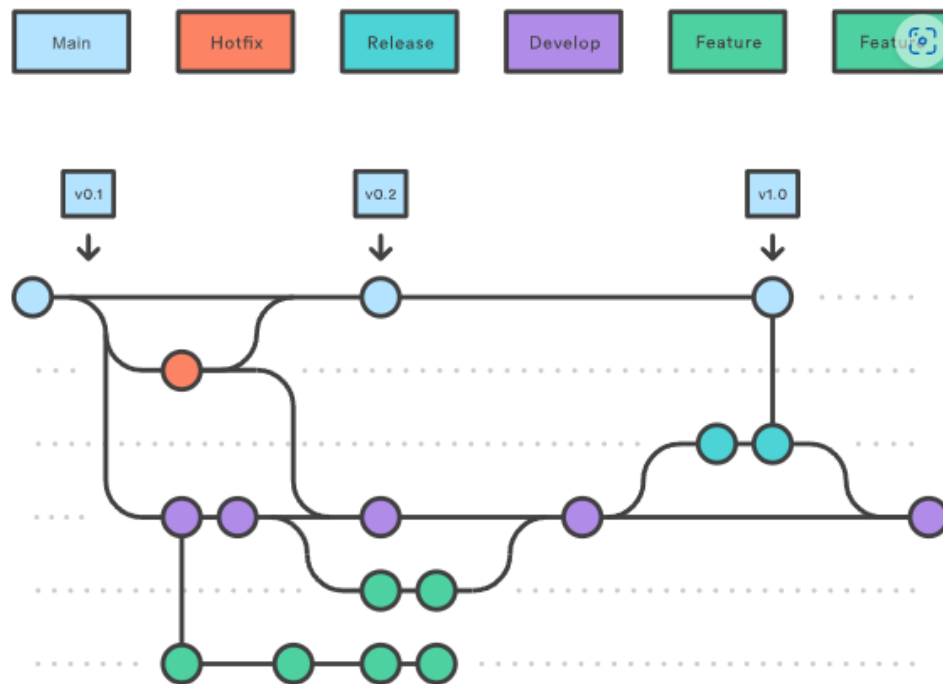


Cuando develop haya adquirido suficientes funciones para una publicación (o se acerque una fecha de publicación predeterminada), debes bifurcar una rama release (o de

publicación) a partir de develop. Al crear esta rama, se inicia el siguiente ciclo de publicación, por lo que no pueden añadirse nuevas funciones una vez pasado este punto (en esta rama solo deben producirse las soluciones de errores, la generación de documentación y otras tareas orientadas a la publicación). Cuando está lista para el lanzamiento, la rama release se fusiona en main y se etiqueta con un número de versión. Además, debería volver a fusionarse en develop, ya que esta podría haber progresado desde que se iniciara la publicación.

Utilizar una rama específica para preparar publicaciones hace posible que un equipo perfeccione la publicación actual mientras otro equipo sigue trabajando en las funciones para la siguiente publicación. Asimismo, crea fases de desarrollo bien definidas (por ejemplo, es fácil decir: "Esta semana nos estamos preparando para la versión 4.0" y verlo escrito en la estructura del repositorio).

RAMA DE CORRECCION O HOTFIX BRANCHES



Las ramas de mantenimiento, de corrección o de hotfix sirven para reparar rápidamente las publicaciones de producción. Las ramas hotfix son muy similares a las ramas release y feature, salvo por el hecho de que se basan en la rama main y no en la develop. Esta es la única rama que debería bifurcarse directamente a partir de main. Cuando se haya terminado de aplicar la corrección, debería fusionarse en main y develop (o la rama release actual), y main debería etiquetarse con un número de versión actualizado.

EJEMPLO DE GITFLOW

En gitbasch no tuve que instalar git Flow, parece que está instalado, al digitar \$git flow; me salen las opciones de git Flow

Available subcommands are:

init Initialize a new git repo with support for the branching model.

feature Manage your feature branches.

bugfix Manage your bugfix branches.

release Manage your release branches.

hotfix Manage your hotfix branches.

support Manage your support branches.

version Shows version information.

config Manage your git-flow configuration.

log Show log deviating from base branch.

Try 'git flow <subcommand> help' for details.

1. Git flow init
2. \$ git flow init
- 3.
4. which branch should be used for bringing forth production releases?
5. - main
6. Branch name for production releases: [main]
7. Branch name for "next release" development: [develop]
- 8.
9. How to name your supporting branch prefixes?
10. Feature branches? [feature/]
11. Bugfix branches? [bugfix/]
12. Release branches? [release/]
13. Hotfix branches? [hotfix/]
14. Support branches? [support/]
15. Version tag prefix? []
16. Hooks and filters directory? [C:/Users/USER/onedrive/fernando/ademass/git-github/hellogit/.git/hooks]
- 17.

Pregunta si queremos dejar los nombres de las ramas con esos nombres por defecto o los queremos cambiar, en todo se dejó por defecto con los nombres que asigna git Flow

Y de una vez nos ubica en develop

2. ahora si queremos empezar a desarrollar o si somos desarrolladores debemos trabajar sobre una rama (feature) \$git flow feature start (nombre de la rama)

\$ git flow feature start 2auth

Switched to a new branch 'feature/2auth'

Summary of actions:

- A new branch 'feature/2auth' was created, based on 'develop'

- You are now on branch 'feature/2auth'

Now, start committing on your feature. When done, use:

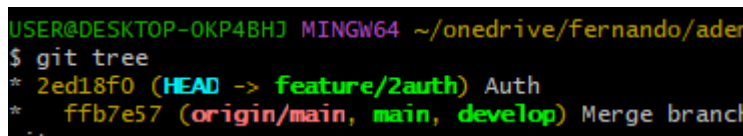
```
git flow feature finish 2auth
```

crea la rama feature llamada 2auth y nos cambia directamente a esa rama

vamos a visualizar creamos un fichero llamado auth.py, dentro de el fichero creamos un print, ahora volvemos a gitbash para continuar con el ejemplo

hacemos un status

hice commit y ahora vemos el grafico, nuestro commit lo marca en una rama feature/2auth



```
USER@DESKTOP-OKP4BHJ MINGW64 ~/onedrive/fernando/ade
$ git tree
* 2ed18f0 (HEAD -> feature/2auth) Auth
* ffb7e57 (origin/main, main, develop) Merge branch
```

Después de verificar todo el contenido de la rama de desarrollo que ya esta terminado hacemos finish

```
$git flow feature finish 2auth
```

Summary of actions:

- The feature branch 'feature/2auth' was merged into 'develop'
- Feature branch 'feature/2auth' has been locally deleted
- You are now on branch 'develop'

Pregunta sobre error al borrar un directorio y se le dice que no, luego realiza merge sobre develop y elimina la rama 2auth, al tiempo no ubica en develop

Rama reléase

Ahora vamos a entrar con una rama release o de publicación

```
$git flow release start 1.0 (esta es la versión)
```

```
$ git flow release start 1.0
```

Switched to a new branch 'release/1.0'

Summary of actions:

- A new branch 'release/1.0' was created, based on 'develop'
- You are now on branch 'release/1.0'

Follow-up actions:

- Bump the version number now!
- Start committing last-minute fixes in preparing your release
- When done, run:

```
git flow release finish '1.0'
```

nos ubica en la rama release 1.0 para empezar a revisar para publicar

```
$git flow release start 1.0
```

Después de revisar cambios que existen en develop y aceptar todas las modificaciones del proyecto se procede a finalizar el release

```
$git flow release finish 1.0
```

Me mando a la ventana de escribir mensaje para commit digite :q dos veces me saco de la venta y me arrojo el siguiente resultado

Switched to branch 'main'

Your branch is up to date with 'origin/main'.

Merge made by the 'ort' strategy.

```
auth.py | 1 +
```

```
1 file changed, 1 insertion(+)
```

```
create mode 100644 auth.py
```

Already on 'main'

Your branch is ahead of 'origin/main' by 2 commits.

(use "git push" to publish your local commits)

fatal: no tag message?

Fatal: Tagging failed. Please run finish again to retry.

fatal: ¿no hay mensaje de etiqueta?

Fatal: Error de etiquetado. Por favor, ejecute finish de nuevo para reintentarlo.

Hice de nuevo \$git flow release finish 1.0

Digite :q para salir de la ventana

Y me pregunta lo siguiente

Branches 'main' and 'origin/main' have diverged.

And local branch 'main' is ahead of 'origin/main'.

Previous HEAD position was fa33766 Merge branch 'release/1.0'

Switched to branch 'develop'

Merge made by the 'ort' strategy.

Deletion of directory '.git/logs/refs/heads/release' failed. Should I try again? (y/n)

traduccion

Las ramas 'principal' y 'origen/principal' han divergido.

Y la rama local 'main' está por delante de 'origin/main'.

La posición HEAD anterior era fa33766 Fusionar rama 'release/1.0'

Cambiado a la rama 'develop'

Fusión realizada por la estrategia 'ort'.

La eliminación del directorio '.git/logs/refs/heads/release' ha fallado. ¿Debería intentarlo de nuevo? (sí/no)

Parece que borro la rama según el informe que genera

Deleted branch release/1.0 (was 2ed18f0).

Summary of actions:

- Release branch 'release/1.0' has been merged into 'main'
- The release was tagged '1.0'
- Release tag '1.0' has been back-merged into 'develop'
- Release branch 'release/1.0' has been locally deleted
- You are now on branch 'develop'

Realize el merge de develop en main

Y nos vuelve a ubicar en develop, que son las ramas de desarrollo

```

$ git tree
*   ab2a264 (HEAD -> develop) Merge tag '1.0' into develop
| \
|  *   fa33766 (tag: 1.0, main) Merge branch 'release/1.0'
|  | \
|  |  /
|  / \
| *   2ed18f0 Auth
|  /
| *   ffb7e57 (origin/main) Merge branch 'main' of https://github.com/FenanoOrtiz/hello-git

```

Ahora vamos a subir los cambios locales al repositorio de github

Por medio de git push

No funciona porque no estaba configurado git para subir ramas a el origen, se configuro y se mando de nuevo

USER@DESKTOP-0KP4BHJ MINGW64 ~/onedrive/fernando/ademass/git-github/hellogit
(develop)

```
$ git config --global push.autoSetupRemote true(se configure para subir ramas remotas)
```

```
USER@DESKTOP-0KP4BJ MINGW64 ~/onedrive/fernando/ademass/git-github/hellogit
(develop)
```

```
$ git push
```

Enumerating objects: 6, done.

Counting objects: 100% (6/6), done.

Delta compression using up to 12 threads

Compressing objects: 100% (4/4), done.

Writing objects: 100% (5/5), 629 bytes | 314.00 KiB/s, done.

Total 5 (delta 3), reused 0 (delta 0), pack-reused 0 (from 0)

remote: Resolving deltas: 100% (3/3), completed with 1 local object.

remote:

remote: Create a pull request for 'develop' on GitHub by visiting:

```
remote: https://github.com/FenanoOrtiz/hello-git/pull/new/develop
```

remote:

To <https://github.com/FenanoOrtiz/hello-git.git>

```
* [new branch]    develop -> develop
```

branch 'develop' set up to track 'origin/develop'.

USER@DESKTOP-0KP4BHJ MINGW64 ~/onedrive/fernando/ademass/git-github/hellogit
(develop)

Ya quedo la informacion en github, ahora hay que hacer el pull-request

En github, para que quede todo actualizado

GIT CHERRY-PICK Y REBASE

CHERRY-PICK permite traer un commit en concreto a la rama que nosotros queramos

Git rabase es un comando delicado puede modificar el historial de commit

Pues se coloca en el head y queda como la rama principal

GUTHUB PAGES Y ACTION

GITHUB PAGES

[GitHub Pages | Websites for you and your projects, hosted directly from your GitHub repository. Just edit, push, and your changes are live.](#)

Desplegar paginas gratis, leer y practicar

Github action

Acciones automáticas que se pueden utilizar en github

[GitHub Actions](#)

4:50

Estado Abreviado

Si bien es cierto que la salida de `git status` es bastante explícita, también es verdad que es muy extensa. Git ofrece una opción para obtener un estado abreviado, de manera que puedas ver tus cambios de una forma más compacta. Si ejecutas `git status -s` o `git status --short`, obtendrás una salida mucho más simplificada.

```
$ git status -s
```

```
M README
```

```
MM Rakefile
```

```
A lib/git.rb
```

```
M lib/simplegit.rb
```

```
?? LICENSE.txt
```

Historial de commit

Es probable que hayas ejecutado un comando como `git log > end`, que redirige la salida del historial de commits a un archivo de texto llamado "end". Es una forma útil de guardar el historial de commits en un archivo para su posterior revisión.

Y cuando me bloqueeo debo oprimir (q)

Markdown link de curso en github

[skills/communicate-using-markdown: Organize ideas and collaborate using Markdown, a lightweight language for text formatting.](#)