

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>

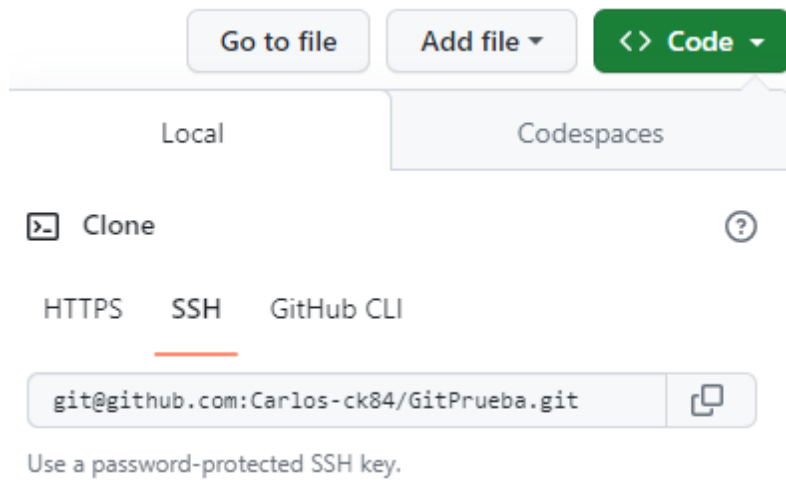
COMANDOS:

- 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 branch -r: obtengo el o los nombres de las ramas existentes incluso del repositorio remoto después de hacer el fetch.
- 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 nombreadarchivo: crea en la carpeta actual el archivo nombreadarchivo
- 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, y copia todo lo que estaba en main.
- `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 específica.
- `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>

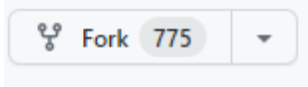
- Para comunicar github con git tenemos que hacer los siguientes pasos:
 1. Crear la carpeta .ssh en la carpeta del usuario
 2. Nos vamos al terminal git bash, nos ubicamos en la carpeta .ssh y digitamos
ssh-keygen -t ed25519 -C your_email@example.com
 3. Nos pide un nombre de archivo podemos colocar uno de los nombres de sugerencia del manual -> id_rsa.
 4. Nos pide passphrase podemos darle enter dos veces y dejarlo vacío.
 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 está 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 último 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 lo siguiente:

1. Vamos a Fork, Create New Fork



2. Creamos la carpeta en nuestro repositorio.

Create a new fork

A fork is a copy of a repository. Forking a repository allows you to freely experiment with changes without affecting the original project. [View existing forks.](#)

Owner * / Repository name *

By default, forks are named the same as their upstream repository. You can customize the name to distinguish it further.

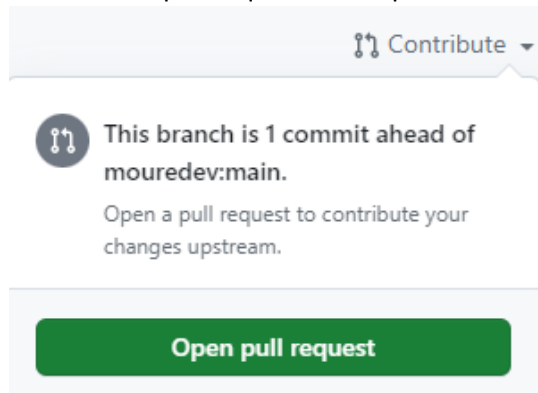
Description (optional)

☒ Copy the `main` branch only
Contribute back to mouredev/hello-git by adding your own branch. [Learn more.](#)

ⓘ You are creating a fork in your personal account.

[Create fork](#)

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

Open a pull request

Create a new pull request by comparing changes across two branches. If you need to, you can also [compare across forks](#).

base repository: mouredev/hello-git

base: main

head repository: Carlos-ck84/hello-git

compare: main

✓ Able to merge. These branches can be automatically merged.

Nuevo Usuario

Write

Preview

H B I

Ejemplo de pull request

Attach files by dragging & dropping, selecting or pasting them.

☒ Allow edits by maintainers

Create pull request

Helpful resources
[GitHub Community Guidelines](#)

- 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.
- 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.

<> Code

Issues

Pull requests

Actions

Projects

Wiki

Security

Insights

Settings

Filters

Labels Milestones

New pull request

5 Open ✓ 82 Closed

Author Label Projects Milestones Reviews Assignee Sort

Branch to add my name

#88 opened 4 minutes ago by ZentoBernabeuPerez

Update hello.md

#87 opened 8 minutes ago by Gdawso

Update hello.md

#86 opened 9 minutes ago by LizzColDev

Update hello.md

#85 opened 36 minutes ago by Kasteryo17

Update hello.md

#84 opened 38 minutes ago by danielaescandar

ProTip! Filter pull requests by the default branch with base:main.

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

ElHereje commented 5 hours ago

First-time contributor

Añado usuario de GitHub a la lista.
Un saludo.

Añadido ElHereje (Almadieis)

5cdcdc8

Add more commits by pushing to the main branch on ElHereje/hello-git.

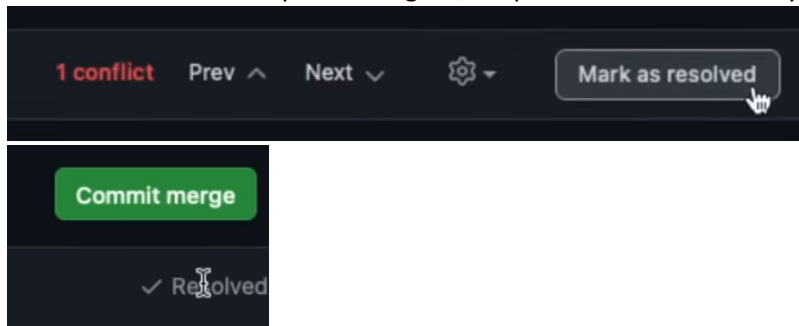
This branch has conflicts that must be resolved
Use the [web editor](#) or the [command line](#) to resolve conflicts.
Conflicting files
hello.md

resolve conflicts

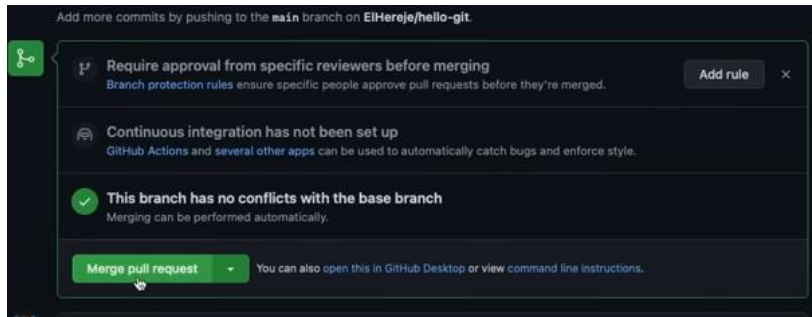
Merge pull request

You can also open this in [GitHub Desktop](#) or [view command line instructions](#).

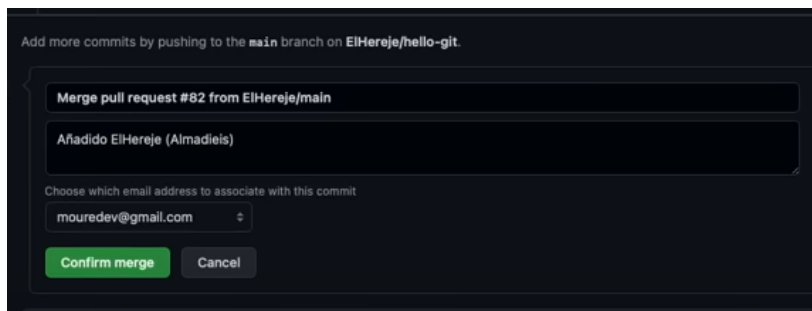
10. Nos muestra un editor para corregirlo, después Mark as resolved y luego Commit merge.



11. Después tenemos que actualizar cuando ya se corrigió.

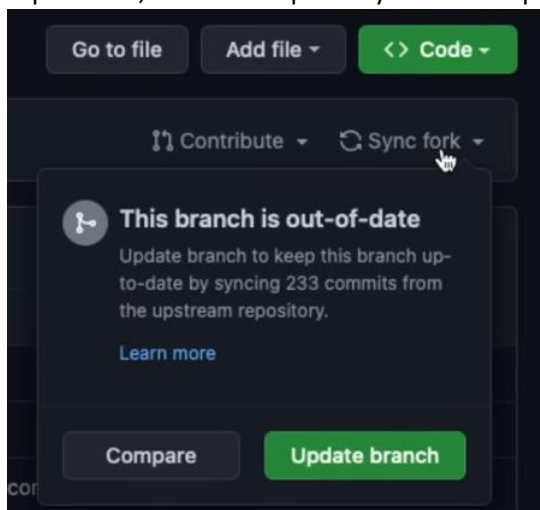


12. Por ultimo confirmamos.



13. Ya con esto se actualizo el repositorio que teníamos, aceptando los cambios del usuario específico.

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.

GITLAB

- PARA SUBIR UN PROYECTO NUEVO A GITLAB

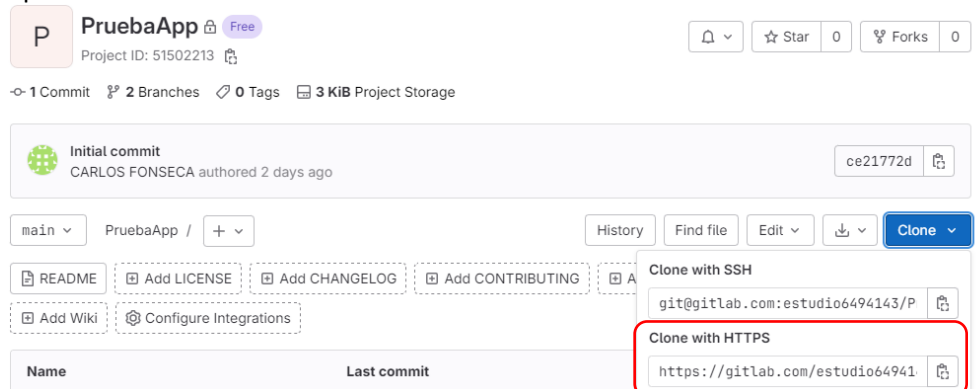
1. Primero vamos a gitlab y creamos un proyecto en blanco, y le colocamos un nombre.
2. En la carpeta de nuestro proyecto iniciamos GitBash
3. Configuramos usuario, esta información la podemos obtener tan pronto creamos el proyecto

```
git config --global user.name "CARLOS FONSECA"
git config --global user.email "cfonseca.electronico@gmail.com"
```

4. En GitBash ejecutamos:
git init
5. Debe ejecutar git add . y git commit -m "mensaje", por lo menos una vez.
6. En GitBash ejecutamos:
git remote add origin <https://gitlab.com/estudio6494143/gitlab2.git>
7. En GitBash ejecutamos:
git push -u origin main
8. Y ya tenemos nuestro proyecto en GitLab, esta es la configuración básica cuando el usuario principal eres tú.

- PARA DESCARGAR UN PROYECTO EXISTENTE Y TRABAJAR SOBRE ÉL.

1. Entramos al proyecto en GitLab, vamos a la opción de clone y copiamos en este caso la url que aparece en HTTPS.



2. Ahora en nuestro pc, creamos una carpeta que va a contener el proyecto, iniciamos dentro de esta carpeta el GitBash.
3. En GitBash ejecutamos
git clone + url que aparece en HTTPS
4. Nos ubicamos en la carpeta creada en GitBash.
cd + nombreproyecto
5. Configuramos usuario, esta información la podemos obtener tan pronto creamos el proyecto

```
git config --global user.name "CARLOS FONSECA"
git config --global user.email "cfonseca.electronico@gmail.com"
```

6. Con la configuración anterior ya podemos comenzar a trabajar con nuestro proyecto de GitLab.

DOCUMENTACION BASICA PARA TRABAJAR CON GITLAB

```
git config --global user.name "CARLOS FONSECA"
git config --global user.email "cfonseca.electronico@gmail.com"
```

Create a new repository

```
git clone https://gitlab.com/estudio6494143/gitlab2.git
cd gitlab2
git switch --create main
touch README.md
git add README.md
git commit -m "add README"
git push --set-upstream origin main
```

Push an existing folder

```
cd existing_folder
git init --initial-branch=main
git remote add origin https://gitlab.com/estudio6494143/gitlab2.git
git add .
git commit -m "Initial commit"
git push --set-upstream origin main
```

Push an existing Git repository

```
cd existing_repo
git remote rename origin old-origin
git remote add origin https://gitlab.com/estudio6494143/gitlab2.git
git push --set-upstream origin --all
git push --set-upstream origin --tags
```

- PARA SUBIR UNA RAMA NUEVA EJECUTAMOS EN GITBASH
 1. `git push -f origin nombrerama:nombrerama`
NOTA: PARA SUBIR UNA RAMA DEBEMOS ESTAR EN ELLA EN GITBASH, ESTA RAMA SE CREA EN EL GITLAB ASI TENGA O NO ARCHIVOS, O COMMITS.
- PARA DESCARGAR UNA RAMA EXISTENTE EJECUTAMOS EN GITBASH
 1. `git pull origin nombrerama`
NOTA: SI EJECUTAMOS ESTE CODIGO Y ESTAMOS PARADOS EN OTRA RAMA. NOS VA A SALIR UN CONFLICTO SI TENEMOS ARCHIVOS EN COMUN EN LAS DOS RAMAS CON DATA DISTINTA, YA QUE POR DEFECTO ESTE HACE UN MERGE AUTOMATICO. ASI RESOLVAMOS EL CONFLICTO NOS TOCA HACER NUEVAMENTE EL PULL PERO EN LA RAMA INDICADA O PODEMOS HACER UN MERGE CON LA RAMA CORREGIDA PERO ANTES HAY QUE ANALIZAR LOS ARCHIVOS QUE TOCAREMOS
- EN EL EJEMPLO ANTERIOR TUVIMOS UN ERROR POR DIFERENCIA EN LOS ARCHIVOS POR LO TANTO UNA FORMA DE SOLUCIONAR ES EJECUTAR EN GITBASH SOLAMENTE GIT PULL