**FUNDAMENTOS GIT**

**¿QUES ES GIT?**

es un software de control de versiones distribuido y descentralizado que permite a un equipo de desarrolladores trabajar sobre el mismo código.

Se denomina **"distribuido"** porque cada miembro del equipo dispone de una copia completa del código.

**¡CONFIGURACION INICIAL!**

git –version

# para ver la versión de git

git config --global user.name "Jonathan MirCha"

# configuramos el nombre

git config --global user.email [jonmircha@gmail.com](mailto:jonmircha@gmail.com)

# configuramos el correo electrónico

git config --global user.ui true

# asignando colores a la terminal

git config --global init.defaultBranch main

#

git config --list

# para ver la lista de las configuraciones asignadas

git config --global core.editor "code --wait"

# asignando visual studio code como editor de configuración de git

git config --global -e

# edición a la configuración global

git config --global core.autocrlf true

# para estandarizar los saltos de línea en windows

git config --global core.autocrlf input

# para estandarizar los saltos de línea en linux/mac

git config -h

# ver todas las opciones de la configuración en la terminal

git help config

# ver todas las opciones de la configuración en el navegador

**¡INICIALIZAR GIT EN UN DIRECTORIO LOCAL!**

mkdir carpeta

# creamos una carpeta

cd carpeta

# accedemos(NOS CAMBIAMOS) a esa carpeta

ls

o

ls -a

# vemos todos los archivos de esa carpeta

touch README.md

# creamos el archivo markdown

touch .gitignore

# creamos el archivo gitignore

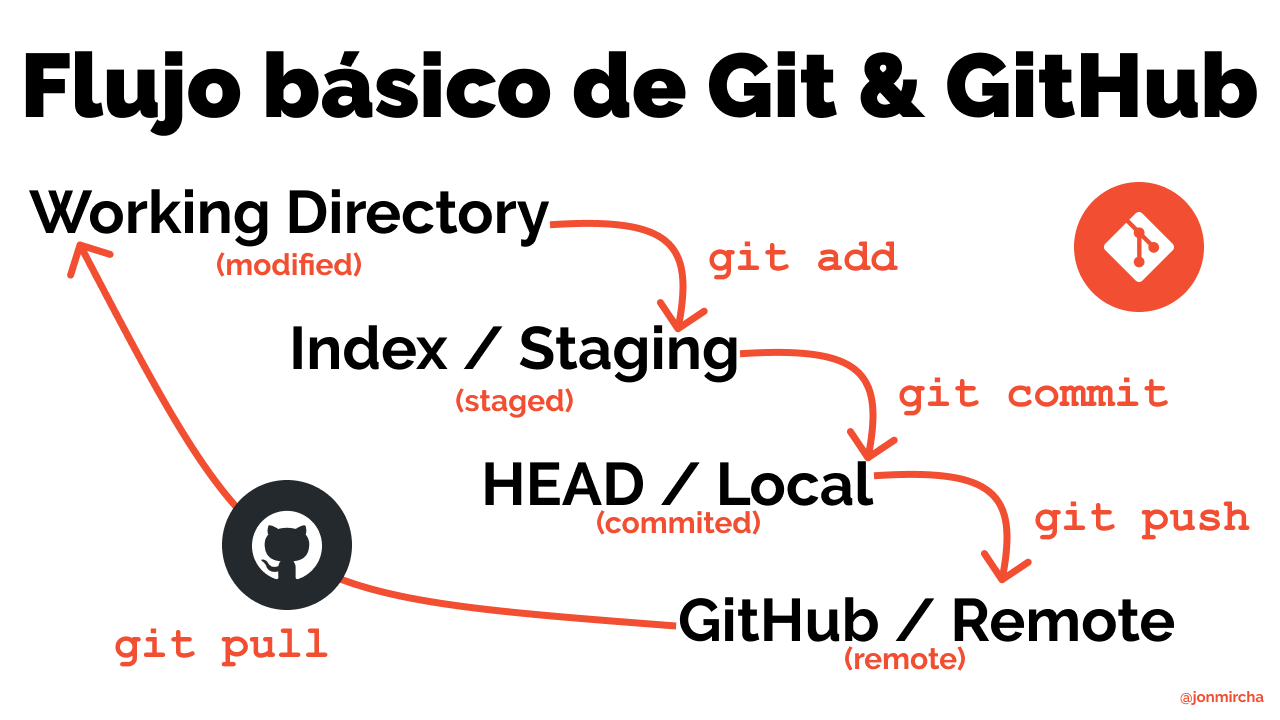
git init

# inicializamos por primera vez un repositorio en git

code .

# abre la carpeta en vscode

**¡FLUJO BASICO!**



**El flujo de Git, consta de tres estados locales, es decir en la computadora donde se está trabajando y uno más de forma remota cuando accedemos al código centralizado en plataformas como GitHub, Gitlab, Bitbucket, etc.**

**Dichos estados son modified, staged, committed y remote. A cada uno de ellos le corresponde un área de trabajo:**

1. **Working Directory: Es el área correspondiente al estado modified y es la carpeta local de tu computadora donde almacenas los archivos de tu proyecto.**
2. **Staging Area: Es el área correspondiente al estado staged también se le llama índex porque es el área donde git indexa y agrega los cambios realizados en los archivos previos a comprometerlos en su registro.**
3. **Local Repository: Es el área correspondiente al estado committed, donde los cambios ya se han registrado en el repositorio de git también se le llama HEAD por que indica en qué cambio se encuentra el puntero del repositorio.**
4. **Remote Repository: Es el área correspondiente al estado remote y es el directorio remoto donde almacenamos los archivos del proyecto en alguna plataforma web como GitHub, GitLab, BitBucket. Git denomina origin al repositorio remoto.**

# agregar los cambios de un archivo al staged

git add archivo/directorio

# agregar todos los cambios de todos los archivos al staged

git add .

# los cambios son comprometidos en el repositorio

# debes escribir el mensaje del cambio

# cuando se abra el archivo de configuración

# al terminar guarda y cierra el archivo

# para que los cambios tengan efecto

git commit

# es un shortcut del comando anterior

# escribes y confirmas el mensaje del cambio en un sólo paso

git commit -m "mensaje descriptivo del cambio"

# se agrega el origen remoto de tu repositorio de GitHub

git remote add origin https://github.com/usuario/repositorio.git

# la primera vez que vinculamos el repositorio remoto con el local

git push -u origin master

# para las subsecuentes actualizaciones, sino cambias de rama

git push

#para descargar los cambios del repositorio remoto al local

git pull

**¡De *master* a *main!***

**Con los desafortunados acontecimientos del 25 de mayo de 2020 en los Estados Unidos que culminaron con el asesinato del afroamericano**[***George Floyd***](https://es.wikipedia.org/wiki/Muerte_de_George_Floyd)**a manos de policias de la ciudad de *Mineápolis*, se intensificó de manera global el movimiento**[***#BlackLivesMatter***](https://es.wikipedia.org/wiki/Black_Lives_Matter)**.**

**Con dicho movimiento muchas industrias y empresas comenzaron a tomar acciones para erradicar el racismo.**

**En la industria de la tecnología por años se han empleado palabras como *master*, *slave*, *whitelist*, *blacklist* entre otras que actualmente no son bien vistas por el contexto y la semántica que implican.**

**Al respecto *Microsoft* empresa propietaria de *GitHub* decidió** comenzar una campaña para reemplazar el nombre de la rama principal de los repositorios de master a main; como lo han explicado en este [documento](https://github.com/github/renaming):

"El 1 de octubre de 2020, cualquier nuevo repositorio que crees utilizará 'main' como la rama por defecto, en lugar de 'master'. Este cambio no afecta a ninguno de tus repositorios existentes: los repositorios existentes continuarán teniendo la misma rama por defecto que tienen ahora".

**Este cambio implica agregar una par de líneas de comandos adicionales para crear la rama '*main*' y hacerla principal en el repositorio.**

**Entonces el flujo básico quedaría de la siguiente manera:**

**¡Para repositorios nuevos!**

git init

git touch README.md

git add .

git commit -m "Primer commit"

git branch -M main

git remote add origin https://github.com/usuario/repositorio.git

git push -u origin main

**¡Para repositorios existentes!**

git branch -M main

git remote add origin https://github.com/usuario/repositorio.git

git push -u origin main

**¡Para reemplazar la rama *master* por *main* en *GitHub!***

# Paso 1

# Crea la rama local main y pásale el historial de la rama master

git branch -m master main

# Paso 2

# Haz un push de la nueva rama local main en el repositorio remoto de GitHub

git push -u origin main

# Paso 3

# Cambia el HEAD actual a la rama main

git symbolic-ref refs/remotes/origin/HEAD refs/remotes/origin/main

Paso 4

**Cambia la rama *default* de *master* a *main* en tu repositorio de *GitHub* .**

# Paso 5

# Elimina la rama master del repositorio remoto

git push origin --delete master

**¡Para reemplazar la rama master por main en Git!**

git config --global init.defaultBranch main

**¡Ayuda!**

# ayuda en la terminal

git comando -h

# ayuda en el navegador

git help comando

**¡Ignorar archivos!**

**En el archivo .gitignore incluimos todo lo que NO queramos incluir en nuestro repositorio. Lo podemos crear manualmente o con**[**gitignore.io**](https://www.gitignore.io/)**.**

# esto es un comentario

archivo.ext

carpeta

/archivo\_desde\_raiz.ext

# ignorar todos los archivos que terminen en .log

\*.log

# excepto production.log

!production.log

# ignorar los archivos terminados en .txt dentro de la carpeta doc,

# pero no en sus subcarpetas

doc/\*.txt

# ignorar todos los archivos terminados en .txt dentro de la carpeta doc

# y también en sus subcarpetas

doc/\*\*/\*.txt

**¡Clonar repositorios!**

git clone https://github.com/usuario/repositorio.git

**¡Ramas!**

**Una rama nos permite aislar una nueva funcionalidad en nuestro código que después podremos añadir a la versión principal.**

# crear rama

git branch nombre-rama

# cambiar de rama

git checkout nombre-rama

# crear una rama y cambiarte a ella

git checkout -b rama

# para que la rama exista en el repositorio remoto

git push –u origin nombre-rama

# eliminar rama

git branch -d nombre-rama

# eliminar ramas remotas

git push origin --delete nombre-rama

#eliminar rama (forzado)

git branch -D nombre-rama

# listar todas las ramas del repositorio

git branch

# lista ramas no fusionadas a la rama actual

git branch --no-merged

# lista ramas fusionadas a la rama actual

git branch --merged

# rebasar ramas

git checkout rama-secundaria

git rebase rama-principal

**¡Fusiones!**

**Une dos ramas. Para hacer una fusión necesitamos:**

1. **Situarnos en la rama que se quedará con el contenido fusionado.**
2. **Fusionar.**

**Cuando se fusionan ramas se pueden dar 2 resultados diferentes:**

* ***Fast-Forward*:** La fusión se hace automática, no hay conflictos por resolver.
* ***Manual Merge*:** La fusión hay que hacerla manual
* , para resolver conflictos de duplicación de contenido.

# nos cambiamos a la rama principal que quedará de la fusión

git checkout rama-principal

# ejecutamos el comando merge con la rama secundaria a fusionar

git merge rama-secundaria

#le pegamos todo a la rama principal

git push –u origin main / o git push main

#eliminar ramas ya funsionadas

git branch –d nombre-rama

# eliminar ramas remotas (GitHub)

git push origin --delete nombre-rama

**¡Cambios!**

**Puedes agregar modificaciones al último cambio**

# sin editar el mensaje del último commit

git commit --amend --no-edit

# editando el mensaje del último commit

git commit --amend -m "nuevo mensaje para el último commit"

# eliminar el último commit

git reset --hard HEAD~1

**Podemos desplazarnos en el historial del repositorio hacia atrás o adelante en cambios o ramas, sin afectar el repositorio como tal.**

# cambiar a una rama

git checkout nombre-rama

# cambiar a un commit en particular

git checkout id-commit

**¡Registro del historial!**

**git log nos permite conocer todo el historial de un proyecto, con la información de la fecha, el autor y id de cada cambio.**

git log

# muestra en una sola línea por cambio

git log --oneline

# guarda el log en la ruta y archivo que especifiquemos

git log > commits.txt

# muestra el historial con el formato que indicamos

git log --pretty=format:"%h - %an, %ar : %s"

# cambiamos la n por cualquier número entero y mostrará los n cambios recientes

git log -n

# muestra los cambios realizados después de la fecha especificada

git log --after="2019-07-07 00:00:00"

# muestra los cambios realizados antes de la fecha especificada

git log --before="2019-07-08 00:00:00"

# muestra los cambios realizados en el rango de fecha especificado

git log --after="2019-07-07 00:00:00" --before="2019-07-08 00:00:00"

# muestra una gráfica del historial de cambios, rama y fusiones

git log --oneline --graph --all

# muestra todo el registro de acciones del log

# incluyendo inserciones, cambios, eliminaciones, fusiones, etc.

git reflog

# diferencias entre el Working Directory y el Staging Area

git diff

**¡Reseteo del historial!**

**Podemos eliminar el historial de cambios del proyecto hacia adelante con respecto de un punto de referencia.**

#nos muestra el listado de archivos nuevos (untracked), borrados o editados

git status

# borra HEAD

git reset --soft

# borra HEAD y Staging

git reset --mixed

# borra todo: HEAD, Staging y Working Directory

git reset --hard

# deshace todos los cambios después del commit indicado, preservando los cambios localmente

git reset id-commit

# desecha todo el historial y regresa al commit especificado

git reset --hard id-commit

**¡Resetear un repositorio!**

**Si en algún momento tienes la necesidad de resetear el historial de cambios de un repositorio para que quede como si lo acabarás de crear ejecuta esta serie de comandos:**

cd carpeta-repositorio

mv .git/config ~/saved\_git\_config

rm -rf .git

git init

git branch -M main

git add .

git commit -m "Commit inicial"

mv ~/saved\_git\_config .git/config

git push --force origin main

**¡Remotos!**

# muestra los orígenes remotos del repositorio

git remote

# muestra los orígenes remotos con detalle

git remote -v

# agregar un orígen remoto

git remote add nombre-orígen https://github.com/usuario/repositorio.git

# renombrar un orígen remoto

git remote rename nombre-viejo nombre-nuevo

# eliminar un orígen remoto

git remote remove nombre-orígen

# descargar una rama remota a local diferente a la principal

git checkout --track -b rama-remota origin/rama-remota

**¡Etiquetas!**

**Con esta opción *git* nos permite versionar nuestro código, librería o proyecto.**

# listar etiquetas

git tag

# crea una etiqueta

git tag numero-versión

# eliminar una etiqueta

git tag -d numero-versión

# mostrar información de una etiqueta

git show numero-versión

# sincronizando la etiqueta del repositorio local al remoto

git add .

git tag v1.0.0

git commit -m "v1.0.0"

git push origin numero-versión

# generando una etiqueta anotada (con mensaje de commit)

git add .

git tag -a "v1.0.0" -m "Mensaje de la etiqueta"

git push --tags

***¡GitHub Pagés!***

[gh-pages](https://pages.github.com/)**es una rama especial para crear un sitio *web* a tu proyecto alojado directamente en tu repositorio de *GitHub*.**

* ***URL* del repositorio:**https://github.com/usuario/repositorio
* ***URL* del sitio:**https://usuario.github.io/repositorio

**Para crear esta rama especial en *GitHub* ejecutamos los siguientes comandos:**

git branch gh-pages

git checkout gh-pages

git remote add origin https://github.com/usuario/repositorio.git

git push origin gh-pages

# para descargar los cambios del repositorio remoto al local

git pull origin gh-pages

***¡Colaboración en*GitHub!**

Para poder colaborar en proyectos alojados en *GitHub* necesitamos hacer uso de los *forks* y *pull requests*, herramientas que nos ofrece la plataforma para dicho objetivo.

A continuación, describo el proceso de colaboración en ***GitHub***.

1. *Forkea* el repositorio en el que quieras colaborar, para hacerlo, sigue las instrucciones de este [enlace](https://docs.github.com/en/get-started/quickstart/fork-a-repo).
2. Una vez *forkeado* el repositorio en tu cuenta de *GitHub*, clónalo en tu equipo de cómputo.
3. En el repositorio local hay que configurar los orígenes remotos de tu nueva copía para tener ambos remotos, los originales (*origin*) y los de tu copia, para hacerlo, sigue las instrucciones de este [enlace](https://docs.github.com/en/pull-requests/collaborating-with-pull-requests/working-with-forks/configuring-a-remote-for-a-fork).
4. Crea una rama nueva en tu *fork* local para hacer tu colaboración, y sincrónizala con tu repositorio remoto, para hacerlo, sigue las instrucciones de este [enlace](https://docs.github.com/en/pull-requests/collaborating-with-pull-requests/working-with-forks/syncing-a-fork).
5. Configura tu repositorio para que acepté cambios (*pull requests*), para hacerlo, sigue las instrucciones de este [enlace](https://docs.github.com/en/pull-requests/collaborating-with-pull-requests/working-with-forks/allowing-changes-to-a-pull-request-branch-created-from-a-fork).
6. Crea una *pull request*, para hacerlo, sigue las instrucciones de este [enlace](https://docs.github.com/en/pull-requests/collaborating-with-pull-requests/proposing-changes-to-your-work-with-pull-requests/creating-a-pull-request).
7. Espera a que el dueño del repositorio original, acepte tus cambios.
8. Una vez que acepten tu *pull request*, es recomendable que borres la rama en la que trabajaste el cambio y actualices tu repositorio *forkeado*, con los cambios del repositorio original.

Anexo un resumen de los comandos a ejecutar para colaborar en un repositorio de *GitHub*:

# forkear repositorio

git clone https://github.com/usuario/repositorio.git

git remote -v

git remote rename origin fork

git remote add origin https://github.com/usuario/repositorio.git

git checkout -b rama-nueva

git push fork rama-nueva

# solicitar el pull request

# aceptar el pull request

git checkout main

git pull origin main

git push fork main

git branch -d rama-nueva

git push fork --delete rama-nueva