

GIT

Flujo básico del desarrollo de una aplicación

Inicio → Interfaz de usuario → Conexión a BD → Fin (listo para producción)

Inicio: Proyecto desde cero: planear la idea, estructuras carpetas, instalar herramientas, etc.

Interfaz de usuario (UI): lo que el usuario ve e interactúa, botones, formularios, menús, colores, fuentes (HTML, CSS, JavaScript, React, etc).

Conexión a Base de Datos: conecta la lógica y seguridad de backend a una base de datos para el flujo de datos.

Relacionales: MySQL, PostgreSQL (usan tablas como Excel).

No relacionales: MongoDB (usan estructuras tipo JSON)

Guardar tareas nuevas

Consultar tareas existentes

Eliminar o editar datos

Fin (listo para producción): después de probar y ajustar todo, la app está lista para subirla a internet (publicar) "desplegar a producción".

Git: sistema de control de versiones, gestor de repositorios, es decir almacena una copia de un código en su servidor; permite trabajar de forma colaborativa y respalda un archivo o conjunto de archivos a lo largo del tiempo.



GIT

Flujo local

- Directorio trabajo actual/**working directory**: eventualmente este contenido estará en área de preparación ya que actualmente lo sigo trabajando.

Tipo de archivos:

Untracked: no están siendo seguidos por Git aún.

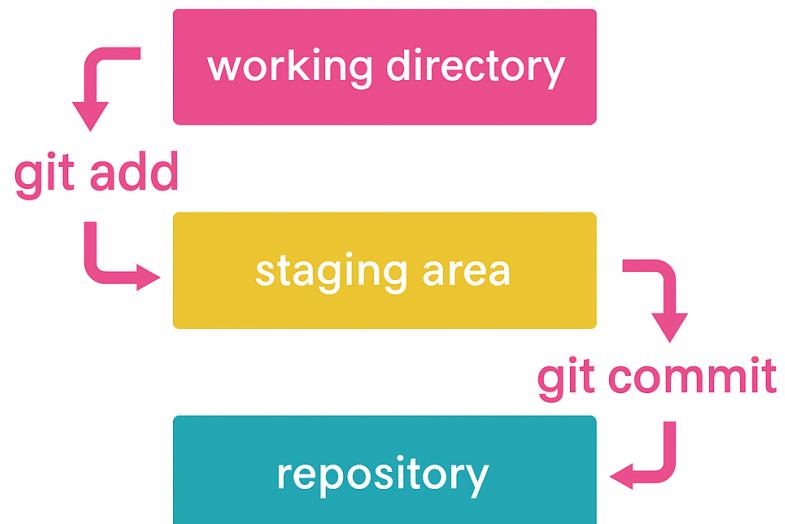
Modified: ya están bajo control, pero tienen cambios no guardados aún en Git.

- Área de preparación/**staging area**: index, área intermedia.

Uso de comando git add <file>

- Directorio/**repository**: .git directory donde se almacena todo lo que se guarda, no se toca.

Uso de comando git commit -m "mensaje"



Flujo Básico



Paso 1 modificar archivos: working directory (editor de texto o IDE), no está registrado en git.

Paso 2 preparar archivos: git add <file> git sabe de los cambios listos para guardar.

Paso 3 revisar estado: git status muestra qué archivos están modificados, cuáles ya están en staging y cuáles no están siendo rastreados.

Paso 4 confirmar los cambios: git commit -m "mensaje" guarda los cambios en el historial del proyecto, crea un punto de restauración y solo se guarda lo que estaba en staging.

GIT

IDE Integrated Development Environment (Entorno de Desarrollo Integrado)

Programa para escribir código, normalmente incluye:

- Editor de texto (para escribir código)
- Terminal integrada
- Coloreado de sintaxis
- Ayuda para autocompletar código
- A veces: depurador, administrador de versiones, plugins, etc.

Visual Studio Code (VSCode), PyCharm, IntelliJ, Eclipse

Sistema	Como funciona	Copia local con historial	Trabaja sin conexión
Descentralizado (git)	Cada persona que clona un repositorio tiene una copia completa del historial del proyecto, no solo los archivos actuales.	Si	Si
Sistemas centralizados	Un servidor único, donde todo depende de una sola fuente, los usuarios solo tienen la última versión de los archivos, no el historial completo. Subversion (SVN)	No	No

Comandos:

Todos inician con **git** seguido de un **parámetro**

Ayuda

git help
git —help
git help <command>

Inicio configuración

git config —global user.name "nombre"
git config —global user.email "email@dominio.com"

```
MacBook-Pro-:~ apple$ git config --global user.name  
Lolita  
MacBook-Pro-:~ apple$ git config --global user.email  
mail@gmail.com
```

git config —list ver los datos de configuración registrados en Git

Comillas: shift + 2 → “ ”

GIT

Iniciar repositorio

`git init` crea repositorio local (.git directory)

`git clone <url>` Clona un repositorio remoto

Cambiar el nombre de la rama principal

`git config --global init.defaultBranch <main>`

`git branch -m <main>`

(futuros proyectos)

(proyecto actual, estar dentro de la rama)

(main) rama principal actual.

(master) rama principal antigua.

git status

Estado de los archivos: modificados, en staging, sin seguimiento, etc.

git add <file>

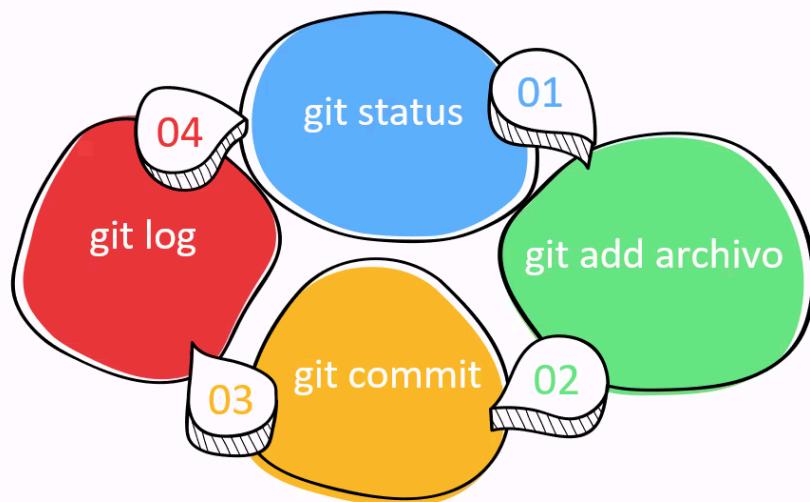
Agrega un archivo modificado al área de preparación (*staging area*).

git commit

Guarda los cambios agregados a staging, con mensaje.

-m mensaje para mencionar de que fue el cambio

Sin -m, me lleva vi (nano, vsc o el predeterminado)



git commit --amend -m ""

Cambia el mensaje del ultimo commit

git restore <file>

Deshace los cambios en un archivo modificado (lo regresa al último commit).

git restore --staged <file>

Saca el archivo del área de staging (vuelve al estado modificado pero no preparado).

git log

Muestra la historia de commits con autor, fecha (orden cronológico) y hash.

git show <commit>

Muestra los detalles del commit (cambios línea por línea), preliminar

GIT

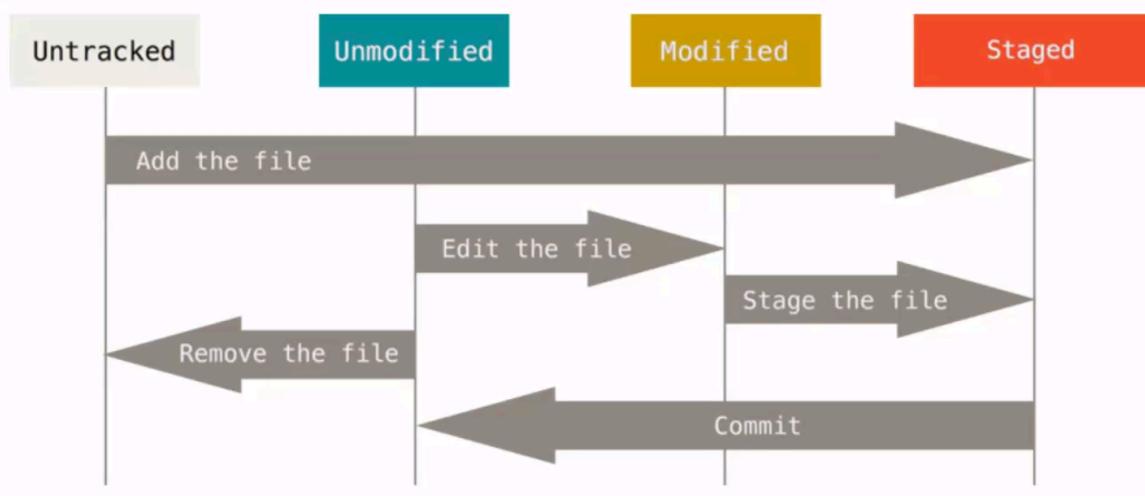
+++ archivo nuevo o modificado

--- archivo eliminado o versión anterior

/dev/null se refiere a que no existía antes (archivo nuevo)

git checkout <commit>

Cambia el proyecto al estado que tenía en ese commit (modo *detached HEAD*).



GIT

bash

Primer commit

```
MacBook-Pro-:ejercicio 1 apple$ git commit -m "creacion de doc"
[main (root-commit) 45f9338] creacion de doc
 1 file changed, 4 insertions(+)
 create mode 100644 lista.txt
```

Inicio de repositorio

```
MacBook-Pro-:ejercicio 1 apple$ git init
Initialized empty Git repository in /Users/apple/direjercicios/control
de versiones/ejercicio 1/.git/
```

```
MacBook-Pro-:ejercicio 1 apple$ git status
On branch main
```

No commits yet

```
nothing to commit (create/copy files and use "git add" to track)
```

Creación de doc con vi, cat para revisar contenido

```
MacBook-Pro-:ejercicio 1 apple$ vi lista.txt
MacBook-Pro-:ejercicio 1 apple$ cat lista.txt
lista de compras
1
2
```

```
MacBook-Pro-:ejercicio 1 apple$ git status
On branch main
```

No commits yet

Untracked files:

```
(use "git add <file>..." to include in what will be committed)
  lista.txt
```

```
nothing added to commit but untracked files present (use "git add" to
track)
```

Inicio de comandos para guardar

```
MacBook-Pro-:ejercicio 1 apple$ git add lista.txt
MacBook-Pro-:ejercicio 1 apple$ git status
On branch main
```

No commits yet

Changes to be committed:

```
(use "git rm --cached <file>..." to unstage)
```

GIT

```
new file: lista.txt
```

```
MacBook-Pro-:ejercicio 1 apple$ git commit -m "creación de doc"
[main (root-commit) 45f9338] creación de doc
 1 file changed, 4 insertions(+)
 create mode 100644 lista.txt
```

Después de modificar un archivo

```
MacBook-Pro-:ejercicio 1 apple$ git status
On branch main
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
    (use "git restore <file>..." to discard changes in working
     directory)
      modified: lista.txt
```

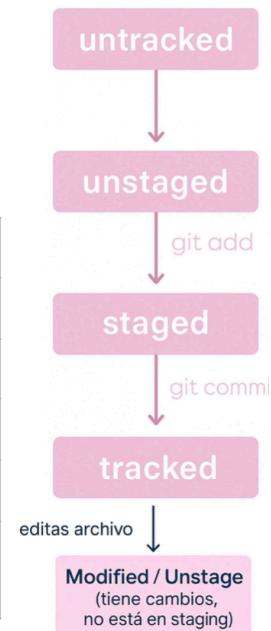
```
no changes added to commit (use "git add" and/or "git commit -a")
```

Me pregunta si se restaura o continuar con comando git add

GIT

Tipo de archivos

Archivo	Como funciona
New	archivo nuevo
Untracked	archivo nuevo, Git no lo está rastreando todavía.
Unstaged	archivo modificado, pero no agregado aún al staging area.
Staged	archivo listo para ser confirmado (commit).
Tracked	archivo ya conocido por Git enviado al repositorio con cambios definitivos (en algún commit pasado).



+ comandos

Comando	Función	Uso local / remoto
<code>git reset HEAD <file></code>	Quita archivos del staging area (como git restore —staged) de staged a unstaged	Local
<code>git stash</code>	Guarda temporalmente cambios sin hacer commit. Ideal si se necesita cambiar de rama rápido, sin perder tu trabajo. <code>git stash pop</code> para recuperar	Local
<code>git reset --hard <hash></code>	Volver el repo al estado exacto de un commit anterior (! destructivo)	Local
<code>git rm <file></code>	Elimina un archivo del proyecto y lo marca para ser eliminado en el próximo commit de tracked a untracked	Local
<code>git clean</code>	git clean -n # muestra qué archivos se eliminarán git clean -f # elimina los archivos no rastreados git clean -fd # fuerza eliminación e incluye carpetas ! elimina archivos físicamente, no los manda a la papelera.	Local
<code>git branch</code>	Muestra ramas locales, o crea una nueva rama (<code>git branch <nueva></code>)	Local
<code>git branch -r</code>	Muestra ramas remotas	Remoto
<code>git checkout <rama></code>	Cambia de rama	Local
<code>git checkout -</code>	Cambia de rama anterior	Local

GIT

<code>git checkout -b</code>	Crea y cambia a esa nueva rama	Local
<code>git merge <rama></code>	Fusiona la rama indicada con la actual	Local
<code>git cherry-pick <hash></code>	Traer un solo cambio específico (un commit) desde otra rama a la actual sin traer todo lo demás (merge completo).	Local
<code>git rebase</code>	Reaplica commits de una rama sobre otra (útil para historia limpia)	Local
<code>git diff</code>	Muestra diferencias entre archivos (cambios sin commitear o entre ramas)	Local
<code>git log -p</code>	Muestra historial de commits con diferencias línea por línea	Local
<code>git log --oneline</code>	Ver historial de commits resumido (una línea por commit)	Local
<code>git log --pretty=oneline</code>	Ver historial de commits condensado (una línea por commit)	Local
<code>git log --pretty=format:"%h - %an, %ar : %s"</code>	Ver historial de commits id, usuario, tiempo transcurrido, cambio (git commit -m)	Local
<code>git log --stat</code>	Ver historial de commits con modificaciones específicas	Local
<code>git log origin/main</code>	Muestra los commits nuevos en el remoto	Remoto
<code>git log main..origin/main</code>	Muestra diferencias entre rama local y la remota	Remoto
<code>git reflog</code>	Ver historial de cambios recientes, incluso los que ya no aparecen con log	Local
<code>git tag</code>	Crear marcas (etiquetas) para versiones importantes	Local
<code>git blame <archivo></code>	Ver quién hizo cada línea de un archivo	Local
<code>git fetch</code>	Trae los últimos cambios del remoto sin mezclarlos <code>git remote show origin</code>	Remoto
<code>git remote -v</code>	Muestra la conexión con el remoto vinculado (como GitHub)	Remoto

Notas:

Remoto verdadero: solo son `git clone`, `git push`, `git pull` y `git fetch` porque intercambian datos entre el dispositivo e internet.

Git guarda una copia de las ramas remotas en el dispositivo bajo el prefijo `origin/`.

GIT

Resumen de comandos locales que no requieren internet

Comando	Función
<code>git init</code>	Crea un repositorio en dispositivo local
<code>git add</code>	Prepara archivos para guardar
<code>git commit</code>	Guarda los cambios en el historial
<code>git status, git log, git show, git diff</code>	Muestran información sobre cambios
<code>git reset, git restore, git rm, git clean</code>	Manipulan archivos o cambios localmente
<code>git branch, git checkout, git merge, git cherry-pick, git rebase</code>	Trabajan con ramas y versiones locales
<code>git stash, git tag, git blame, git reflog</code>	Herramientas para organización, debugging o respaldo

Git branch

Ramas dentro de git

Permiten trabajar en paralelo sin modificar la rama principal (`main`).

Se usan para desarrollar nuevas funciones, probar ideas o corregir errores.

Cada colaborador suele tener su propia rama.

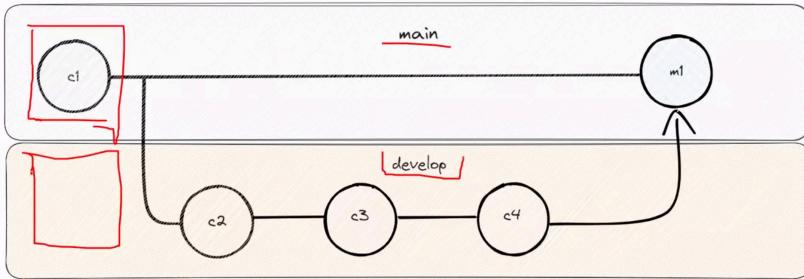
Al terminar, se hace un **merge** a `main` para integrar los cambios.



GIT

Ramas

Main: rama **principal** o base del proyecto. Se mantiene limpia, a partir de aquí se trabajan (commits) en solo ramas (develop/test, 1 por persona). Solo recibe **cambios estables al final**, mediante un **merge**.



.gitignore archivo de texto: Archivo de texto que lista archivos o carpetas que Git debe ignorar (no rastrear ni guardar en el historial).

- Se escribe en la raíz del proyecto.
- Puedes ignorar por nombre (archivo.txt), por extensión (*.log) o por carpeta (/ carpeta/).
- No afecta archivos que ya estaban siendo rastreados. En ese caso, quitarlos del seguimiento con: git rm --cached archivo.txt

.git: Carpeta oculta que se crea automáticamente al ejecutar git init.

- Contiene toda la base de datos de Git (commits, ramas, configuración, etc.).
- No debe tocarse ni editarse manualmente.

bash para git branch

```
MacBook-Pro-:ejercicio 1 apple$ pwd  
/Users/apple/direjercicios/control de versiones/ejercicio 1  
MacBook-Pro-:ejercicio 1 apple$ git branch  
* main
```

```
*  
MacBook-Pro-:ejercicio 1 apple$ git branch develop
```

```
MacBook-Pro-:ejercicio 1 apple$ git branch  
    develop  
* main
```

```
MacBook-Pro-:ejercicio 1 apple$ git checkout develop  
M  lista.txt  
Switched to branch 'develop'
```

```
MacBook-Pro-:ejercicio 1 apple$ touch resumen.txt
```

```
MacBook-Pro-:ejercicio 1 apple$ ls -al  
total 8  
drwxr-xr-x  5 apple  staff  160 Jul 11 10:50 .  
drwxr-xr-x  4 apple  staff  128 Jul  4 10:52 ..  
drwxr-xr-x 12 apple  staff  384 Jul 11 10:46 .git  
-rw-r--r--@ 1 apple  staff   38 Jul  4 17:10 lista.txt  
-rw-r--r--  1 apple  staff     0 Jul 11 10:50 resumen.txt
```

GIT

```
MacBook-Pro-:ejercicio 1 apple$ git add resumen.txt
```

```
MacBook-Pro-:ejercicio 1 apple$ git status
On branch develop
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    new file:   resumen.txt
```

```
MacBook-Pro-:ejercicio 1 apple$ git commit -m "anexo resumen"
[develop b1d84d7] anexo resumen
 2 files changed, 2 insertions(+), 2 deletions(-)
 create mode 100644 resumen.txt
```

```
MacBook-Pro-:ejercicio 1 apple$ git checkout main
Switched to branch 'main'
```

IMPORTANTE Forzosamente usar checkout del develop para ver el archivo

```
MacBook-Pro-:ejercicio 1 apple$ git checkout develop
Switched to branch 'develop'
```

Fusionar ramas (develop y main)

```
MacBook-Pro-:ejercicio 1 apple$ git checkout main
Switched to branch 'main'
```

```
MacBook-Pro-:ejercicio 1 apple$ git merge develop
Updating 319caa4..99009fe
Fast-forward
  lista.txt    | 9 +++++-----
  resumen.txt  | 1 +
  2 files changed, 5 insertions(+), 5 deletions(-)
  create mode 100644 resumen.txt
```

+ es cambio y - es un eliminado

GIT

Llaves SSH

SSH: Secure Shell, protocolo de red

Llave pública y privada; método seguro para autenticar equipo con GitHub u otros servidores sin usar usuario y contraseña cada vez.

Fork: bifurcación del repositorio, copia completa de un repositorio de GitHub para realizar modificaciones local. Uso colaborativo

```
ssh-keygen -t ed25519 -C "email@example.com"
```

Para vincularla con GitHub:

Subir **llave pública** (generalmente en `~/.ssh/id_ed25519.pub`)

En perfil de GitHub → *Settings > SSH and GPG keys*.

Proceso

No importa en qué carpeta estés. Por convención y orden, las llaves se guardan en carpeta personal (también llamada `home`) `/Users/apple/`.

```
ssh-keygen -t ed25519 -C "email@example.com"
```

va a preguntar dónde guardar la llave. Presionar `Enter` para aceptar la ruta por defecto `/Users/apple/.ssh/id_ed25519`

Pedirá una **frase de seguridad (passphrase)** — dejarla en blanco o poner una contraseña adicional para más seguridad.

Se generarán **dos archivos** dentro de `~/.ssh/`:

`id_ed25519` → **llave privada**

`d_ed25519.pub` → **llave pública** (se sube a GitHub)

```
eval "$(ssh-agent -s)"
```

```
ssh-add --apple-use-keychain ~/.ssh/id_ed25519
```

```
cat ~/.ssh/id_ed25519.pub
```

Copiar la salida e ir a GitHub

Una vez logueada en la cuenta de Github

Github->configuraciones->SSH and GPG Keys -> New SSH Key

Pegar el contenido ahí, darle nombre y guardar.

GIT

bash

Generar la llave SSH (solo una vez)

```
MacBook-Pro:~ apple$ ssh-keygen -t ed25519 -C "email@example.com"
Generating public/private ed25519 key pair.
```

Enter file in which to save the key (/Users/apple/.ssh/id_ed25519):

Enter passphrase (empty for no passphrase):

Enter same passphrase again:

Your identification has been saved in /Users/apple/.ssh/id_ed25519

Your public key has been saved in /Users/apple/.ssh/id_ed25519.pub

The key fingerprint is:

S JCA0IHA256:9uSsLlf3U9c2TJIYGXC JRz0nbkyZtg email@example.com

The key's randomart image is:

+-- [ED25519 233] ---

```
+-----+  
| [SHA233] |-----+
```

Inicia el agente SSH, proceso que guarda las llaves en memoria

```
MacBook-Pro:~ apple$ eval "$(ssh-agent -s)"
```

Agent pid 3545

Agregar llave al llavero de Apple (macOS)

```
MacBook-Pro:~ apple$ ssh-add --apple-use-keychain ~/ssh/id_ed25519
```

Enter passphrase for /Users/apple/.ssh/id_ed25519:

Identity added: /Users/apple/.ssh/id_ed25519 (email@example.com)

Copiar la llave pública

```
MacBook-Pro-:~ apple$ cat ~/.ssh/id_ed25519.pub
```

ssh-ed25519 cniwdcpwacosc/ nodscnweosnvqpHV9C email@example.com

Flujo distribuido o flujo remoto (distributed workflow)

GIT

Git permite que cada colaborador tenga su repositorio completo, y se sincroniza con un servidor remoto como GitHub (push/pull).

Repositorio remoto: Versión del repositorio que se encuentra alojada en un servidor online como GitHub, al que se puede acceder por internet. Se usa para compartir código con otros usuarios y trabajar en equipo.

Comandos:

`git push` → enviar cambios al remoto, sube commits locales al repo remoto

`git push origin main` → enviar cambios a la rama principal

`git pull` → recibir cambios del remoto, baja y aplica los commits del repo remoto a la rama

`git pull origin main` → traer del remoto sin ser admin

`git pull nombre_rama` → traer del remoto creado en una rama paralela

`git clone <url>` → Clonar, trae una copia de un repo remoto (GitHub) al dispositivo local

Crear cambios localmente

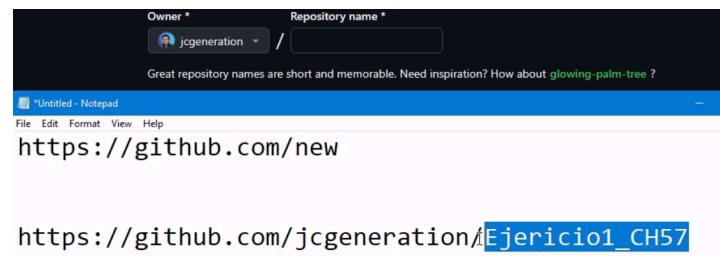
(1) Editas archivos -> (2) git add -> (3) git commit -> (4) git push

GIT

Como subir un archivo local a remoto

Crear repositorio remoto en GitHub

Repositorios > nuevo > nombre del repositorio
> MIT license / README / publico >



Dentro del repositorio en git Code elegir SSH y copiar para usarlo con git remote add origin

```
MacBook-Pro-:ejercicio 1 apple$ git checkout main
Switched to branch 'main'
```

Para revisar que repositorios existen vinculados

```
MacBook-Pro-:ejercicio 1 apple$ git remote -v
```

```
MacBook-Pro-:ejercicio 1 apple$ git remote add origin
git@github.com:Lolita/Ejercicio1_CH57.git
```

Validar que el origen está vinculado con el url SSH para descargar y/o subir cambios

```
MacBook-Pro-:ejercicio 1 apple$ git remote -v
origin  git@github.com:Lolita/Ejercicio1_CH57.git (fetch)
origin  git@github.com:Lolita/Ejercicio1_CH57.git (push)
```

Git pull para enviarlo a remoto

```
MacBook-Pro-:ejercicio 1 apple$ git pull
The authenticity of host 'github.com (140.82.100.3)' can't be
established.
ED25519 key fingerprint is SHA233:+ncuiqenvieo/cnei2qovnpw.
This key is not known by any other names
Are you sure you want to continue connecting (yes/no/[fingerprint])?
yes
Warning: Permanently added 'github.com' (ED25534) to the list of known
hosts.
Enter passphrase for key '/Users/apple/.ssh/id_ed25519':
remote: Enumerating objects: 4, done.
remote: Counting objects: 100% (4/4), done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 4 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
Unpacking objects: 100% (4/4), 1.54 KiB | 315.00 KiB/s, done.
From github.com:Lolita/Ejercicio1_CH57
 * [new branch]      main      -> origin/main
There is no tracking information for the current branch.
Please specify which branch you want to merge with.
See git-pull(1) for details.
```

`git pull <remote> <branch>`

GIT

If you wish to set tracking information for this branch you can do so with:

```
git branch --set-upstream-to=origin/<branch> main
```

Si hay divergencias de ramas unificar ramas de bash y GitHub (main) usar
git pull origin main --no-rebase --allow-unrelated-histories

```
MacBook-Pro-:ejercicio 1 apple$ git pull origin main --no-rebase --  
allow-unrelated-histories  
Enter passphrase for key '/Users/apple/.ssh/id_ed25519':  
From github.com:Lolita/Ejercicio1_CH57  
 * branch           main      -> FETCH_HEAD  
Merge made by the 'ort' strategy.  
 LICENSE    | 21 ++++++  
 README.md |  2 ++  
 2 files changed, 23 insertions(+)  
 create mode 100644 LICENSE  
 create mode 100644 README.md
```

Subir cambios con git push origin main

```
MacBook-Pro-:ejercicio 1 apple$ git status  
On branch main  
nothing to commit, working tree clean
```

```
MacBook-Pro-:ejercicio 1 apple$ git push origin main  
Enter passphrase for key '/Users/apple/.ssh/id_ed25519':  
Enumerating objects: 25, done.  
Counting objects: 100% (25/25), done.  
Delta compression using up to 4 threads  
Compressing objects: 100% (13/13), done.  
Writing objects: 100% (24/24), 2.21 KiB | 1.10 MiB/s, done.  
Total 24 (delta 1), reused 0 (delta 0), pack-reused 0  
remote: Resolving deltas: 100% (1/1), done.  
To github.com:Lolita/Ejercicio1_CH57.git  
 bae1533..41df07  main -> main
```

GIT

Como subir un archivo local a remoto

Crear repositorio remoto en GitHub

Repositorios > nuevo > nombre del repositorio > MIT license / README / publico >

git clone [URL sacada de git hub]

Crea en automático la carpeta con el mismo nombre del repo

Copiamos archivos a este nuevo repo clonado

cp -r ~/Desktop/archivos/ ~/Desktop/nuevo-repo/

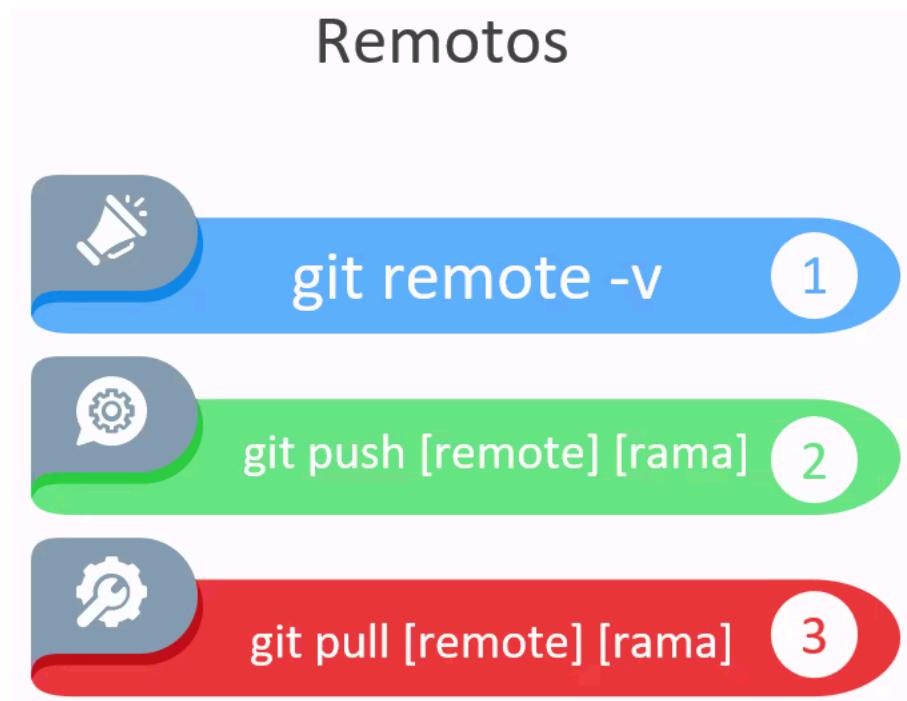
Nos metemos a la carpeta del repo nuevo clonado

cd ~/Desktop/mi-repo

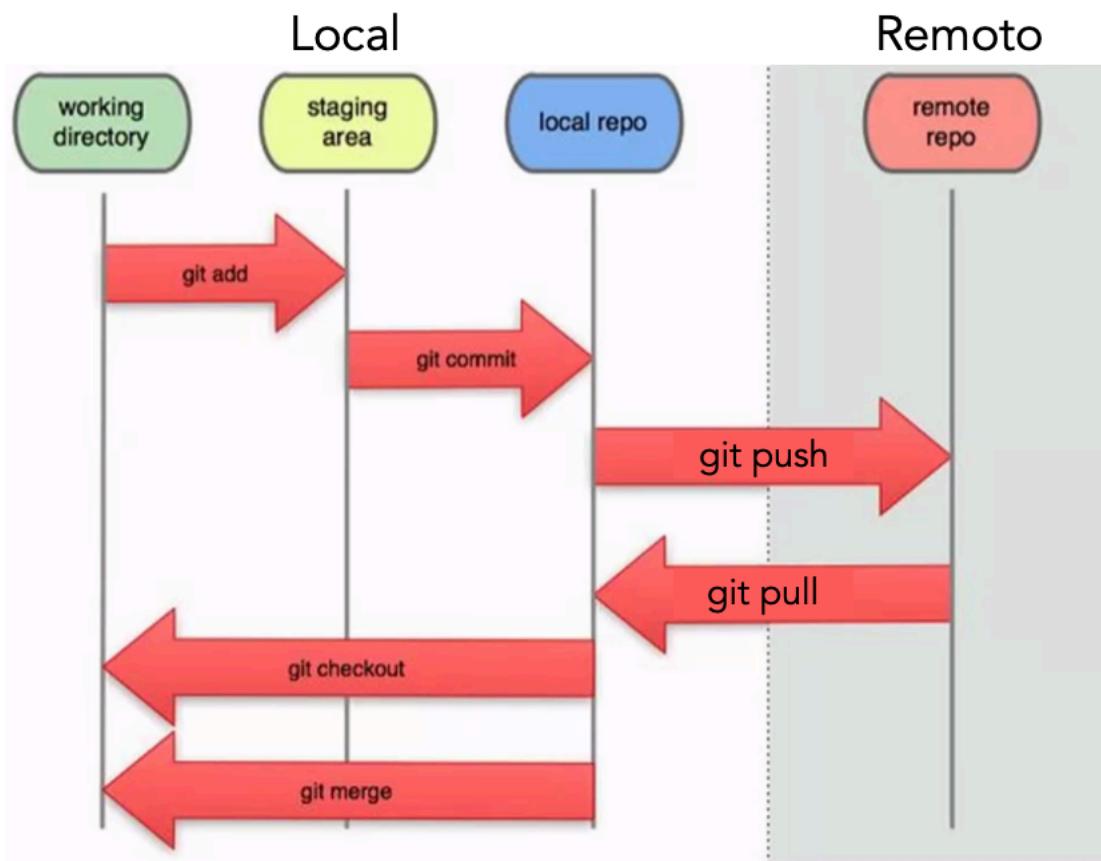
git status

Una vez con los archivos aquí continuamos con git add, git commit - m ""

Realizamos git push origin main



GIT



GIT

Como hacer un repo colaborativo

- Crear el repo con GitHub (centralizar proyecto) <https://github.com/new>
- Crear rama develop desde git hub (para evitar trabajar directamente en main)
- Agregar colaboradores: Setting->Collaborators->Add people /aceptar solicitudes (por correo)
- Abrir bash en la carpeta donde se va a guardar el repo clonado
- Sacar SSH keys de GitHub pegarlas con **git clone [URL]**
- **cd directorioNuevo** para entrar a este nuevo directorio clonado
- **git checkout nuevaRamaClonada** (develop)
- Crear rama propia con **git branch MIRAMA** (nombre e iniciales) para trabajar de forma ordenada, individual sin afectar el repo
- Cambiarme a mi nueva rama **git checkout MIRAMA** comenzar a trabajar
- Crear/copiar archivos **cp/touch/tee/echo**
- **git add** miarchivo.css
- **git commit -m ""** guarda cambios de forma local
- **git push MIRAMA** sube la rama a git hub

- Ya esta la info en GitHub ✓
- Tomar turnos para ir bajando y subiendo ramas/archivos

- **git checkout develop** (rama colaborativa)
- **git pull origin develop**, trae los cambio de las demás colaboradoras
- **git merge MIRAMA** fusiona mi info con la de los colaboradores
- Resolver conflictos si hay

- **git push origin develop** enviar el resultado de la fusión de ramas
- Ya esta la info fusionada en GitHub ✓

Ultimo paso

- **git pull** para traer ramas actualizadas
- **git branch -a** visualizo las ramas
- **git checkout** para poder cambiarme a las nuevas ramas