

/usr/bin/git



Git es un sistema de control de versiones distribuido utilizado para el seguimiento de cambios en archivos de código fuente durante el proceso de desarrollo de software. Permite trabajar en equipo de manera colaborativa y mantener un historial de cambios en el código fuente de un proyecto.

¿Qué es GIT y por qué importa?

- Git es un sistema de control de versiones (Version Control System, VCS).
- Permite tener un histórico de archivos y los cambios realizados sobre ellos.
- Permite que varios desarrolladores trabajen en un mismo proyecto a la vez.
- Hace que volver a una versión anterior sea más fácil.

¿Qué es GitHub?

GitHub es una plataforma de alojamiento de código fuente (aunque permite almacenar todo tipo de archivos) en línea que utiliza Git como su sistema de control de versiones subyacente.

Es una herramienta muy popular para compartir y colaborar en proyectos de software de código abierto, y también se utiliza en empresas y organizaciones para la gestión de proyectos y el desarrollo de software.

Git vs GitHub

Git es el programa que permite realizar el control de versiones y el seguimiento de archivos, GitHub es el servidor para repositorios de Git.

Instalación de GIT

```
$ conda install -c anaconda git
```

(en mac)

```
$ brew install git
```

Instalamos y seguimos las instrucciones.

Una vez terminado como siempre podremos ejecutar `$git --version` para que nos muestre la versión de git instalada.

Configuración de GIT

Una vez instalado Git, es necesario configurarlo antes de poder comenzar a usarlo. Para hacerlo, abre una terminal y ejecuta los siguientes comandos, reemplazando TU NOMBRE y TU CORREO con tu nombre de usuario de GitHub y correo electrónico con el que te registraste:

```
git config --global user.name "TU NOMBRE" (tenemos que poner también las comillas dobles)
git config --global user.email "TU CORREO"
```

Tendremos que establecer nuestras claves SSH de GitHub en Git para que no nos lo pida cada vez que queramos hacer un 'push' de lo que hayamos modificado en nuestros archivos. (Veremos más adelante los términos y funcionamiento de GitHub y que es eso de un push)

GENERAR CLAVES SSH DE GITHUB:

- Comprobar si existe una clave SSH

Primero, comprueba si ya has generado las claves SSH para tu máquina. Abre una terminal e introduce el siguiente comando:

4 - GIT

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

Si las claves existen deberías tener una salida como esta:

```
-rw----- 1 usuario usuario 1766 Jul 7 2018 id_rsa
-rw-r--r-- 1 usuario usuario 414 Jul 7 2018 id_rsa.pub
-rw----- 1 usuario usuario 12892 Feb 5 18:39 known_hosts
```

Si no hay ninguna salida después de ejecutar el comando o nos dice 'No such file' quiere decir que no existe la carpeta `/.ssh`

Por lo que tendremos que crearla en nuestra raíz:

```
$mkdir ~/.ssh
```

- Generar un nuevo conjunto de claves

Sustituid la parte en *itálica* por el mail que tenéis en GitHub

```
$ssh-keygen -t rsa -b 4096 -C tu@correo.com
```

Nos indicará que va a crear el par de claves en una carpeta del tipo `home/usuario/.ssh/id_rsa...` le damos enter, que lo guarde en ese fichero

Nos pedirá que metamos una **palabra** a modo de clave, algo que sea fácil de recordar ¡por favor! que va a funcionar en muchas ocasiones como **password**, es decir, os tendréis que acordar de ella.

Se debería generar un nuevo par de claves, lo podemos comprobar con `$ls -al ~/.ssh` ó `$ls -al $HOME/.ssh`

Ahora vuestra salida debe parecerse a la imagen de más arriba.

Las claves siempre van por pares. La que tiene extensión `.pub` es la pública y es la que debemos usar cuando nos la pidan en los logueos, etc...La clave privada no debéis compartirla nunca ni introducirla en ningún logueo.

- Agrega tu clave SSH a ssh-agent

Primero, asegúrate de que ssh-agent se está ejecutando con:

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

En caso de que nos diga que no puede conectar con el Agent tiramos:

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

Si con todo y con eso no quiere funcionar:

```
$ exec ssh-agent bash  
$ ssh-add ~/.ssh/id_rsa
```

Debe salir un mensaje del tipo Agent PID XXXX, quiere decir que está ejecutando.

- Agregamos la clave privada a ssh-agent:

```
$ssh-add ~/.ssh/id_rsa  
O en su defecto  
$ssh-add $HOME/.ssh/id_rsa
```

- Copiamos la clave pública de SSH

```
$ cat ~/.ssh/id_rsa.pub
```

Nos saldrá por pantalla nuestra clave pública, básicamente estamos haciendo un cat del archivo que como ya vimos es ver el contenido de un archivo.

- Agregamos nuestra clave pública a nuestra cuenta de GitHub

Entramos a nuestra cuenta y vamos a Settings > SSH and GPG Keys > New SSH Key

Nos saldrá algo como esto

SSH keys / Add new

Title

Key type

Authentication Key ↕

Key

Begins with 'ssh-rsa', 'ecdsa-sha2-nistp256', 'ecdsa-sha2-nistp384', 'ecdsa-sha2-nistp521', 'ssh-ed25519', 'sk-ecdsa-sha2-nistp256@openssh.com', or 'sk-ssh-ed25519@openssh.com'

Add SSH key

Title: el nombre que queráis, debería servir para identificar el ordenador del que proviene.

Ej: laptop_pepe ó pc_juan ...

Key: aquí debéis pegar la clave que habéis copiado de vuestra consola

Pulsamos en *Add SSH Key* y todo está configurado correctamente!

Si todo está correcto, podemos comprobarlo en nuestra consola con:

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

Nos pedirá si queremos añadir la clave y le decimos 'yes'

Nos pedirá la palabra secreta que configuramos al crear el par de claves y nos saldrá, si todo está correcto, un mensaje diciendo *'Hi userX! You've successfully authenticated, but GitHub does not provide shell access'*

TERMINOLOGÍA GIT

Repository | repo : carpeta del proyecto donde se guardan todos los fichero y subcarpetas que lo componen

Untracked : un archivo o directorio no monitoreado por Git, no pertenece al repo. No se subirá cuando hagamos push.

Tracked : archivo que hemos añadido a un repo mediante git add. Git si que seguirá los cambios que vayamos realizando sobre él y podrá formar parte de nuestros push.

Commit : un punto en la línea temporal del repo. Es una "captura" de todos los cambios que "hacemos oficiales", el último estado de los archivos. Cada commit tiene un id único y debe tener un comentario. Cuanto más descriptivo sea el mensaje mejor. Sobre todo si queremos volver a un punto anterior de nuestro repositorio.

Local repo | repo local : un repositorio de código en el ordenador que estás usando

Remote repo : un repositorio en una máquina diferente (e.g.: Github)

Forking | Forkear : es la manera de 'clonar' un repositorio existente del que no somos los creadores y poder usarlo de punto de partida para nuestros propios proyectos sin afectar al contenido original del repositorio del creador.

Cloning | Clonar: hacer una copia local de un repositorio remoto

Push : mandar los cambios realizados en local a remoto

Pull : traer a local los cambios hechos en remoto

.gitignore : archivo de texto para registrar archivos y/o carpetas que no queremos que Git monitoree y suba a ningún repositorio aunque estén dentro de nuestro repo local, claves, info personal, tokens de acceso a APIs o simplemente datos adicionales de nuestro repo que no queremos que se puedan ver.

Ej: Si dentro del archivo .gitignore de nuestro repo escribimos

```
logo.jpg # git ignorará archivos con ese nombre específico
develop_test/ # git ignora el directorio con ese nombre y todo su contenido.
*.jpg # git ignora todos los archivos con esa extensión
.gitignore #no sube nuestro fichero gitignore a nuestros repos.
```

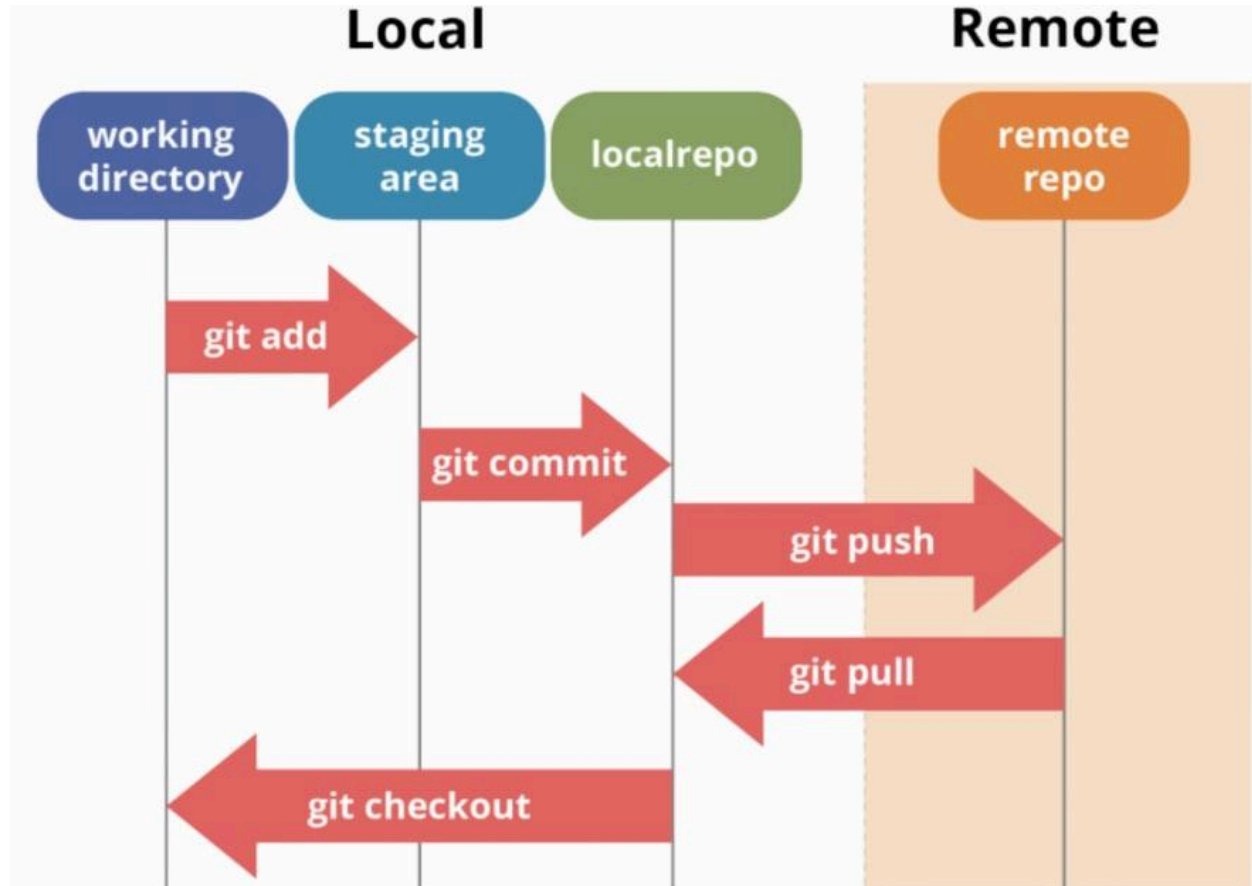
Existen cantidad de ejemplos de plantillas disponibles en internet aunque muchas veces crear las nuestras es lo más rápido.

Branch | Rama :las diferentes líneas temporales de un repositorio. Permite desarrollo en paralelo

Merge : unión de diferentes ramas en una sola. Es lo opuesto de forking, en vez de separar, se unen las diferentes líneas temporales

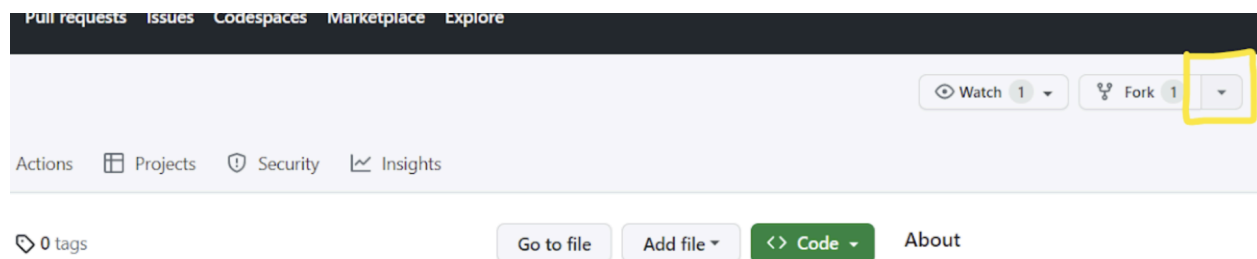
4 - GIT

Pull request : es el acto de pedir al propietario del repositorio que acepte los cambios que tú has realizado. Es literalmente hacer la petición de que haga pull con tus cambios.S



FORKEANDO UN REPO REMOTO

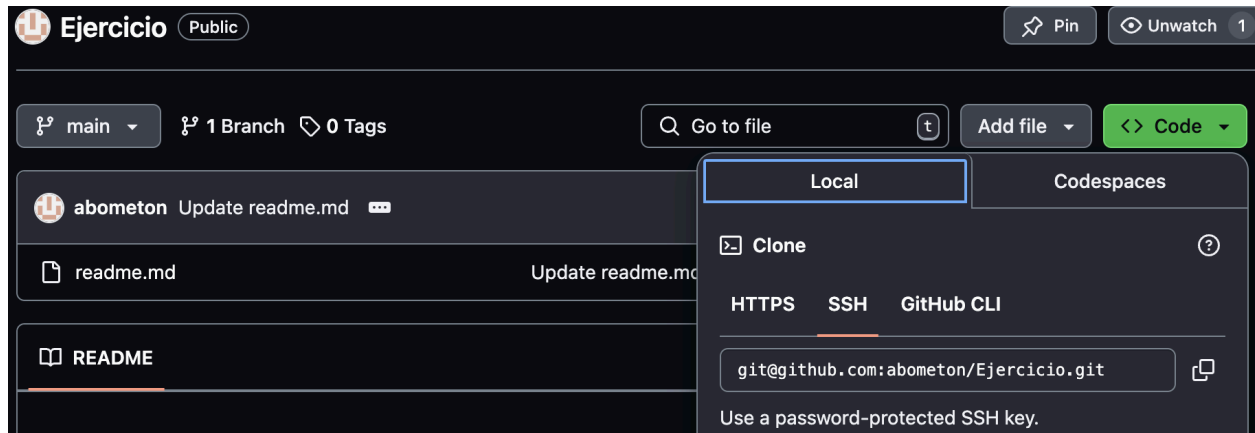
En nuestro caso vamos a forkear el repo <https://github.com/abometon/Ejercicio> que será básicamente donde podremos encontrar todo el contenido de Python que vayamos viendo a lo largo de estas semanas. Accedemos a la URL y en la parte superior vamos a



Create new Fork

La bifurcación de ese repositorio pasará a formar parte de los repos de nuestro perfil. Cuando forkeemos el repo, pasaremos a estar en la versión de ese repo pero en nuestro perfil.

Pinchamos en el bloque verde desplegable que pone **< > Code** y copiamos la ruta **SSH**



Una vez copiado nuestro forkeo del repo, lo podremos clonar en nuestro local. Abrimos la terminal.

Lo primero va a ser navegar hasta el directorio de archivos en el que queremos clonar nuestro repositorio.

```
$ git clone clave_que_acabamos_de_copiar
```

Con esto ya tendremos en local el repo de apuntes clonado.

ACTUALIZANDO CAMBIOS DE REPOSITORIOS REMOTOS

Un repositorio remoto (remote) es un repositorio online desde el que descargar nuestros cambios en el repositorio local. Al clonar un repositorio, se crea un remoto llamado **origin**, que corresponde a nuestro **fork**. Para poder disponer de los cambios que se hagan en el repositorio principal, podemos añadir un remoto upstream que apunte al repositorio original: (debemos ejecutar los comandos dentro de la carpeta local de nuestro repo en cuestión)

```
$git remote add upstream <url_del_repo_original>
```

Hemos añadido como fuente el repositorio original que nos hemos forkeado. Lo podemos comprobar desde nuestra terminal:

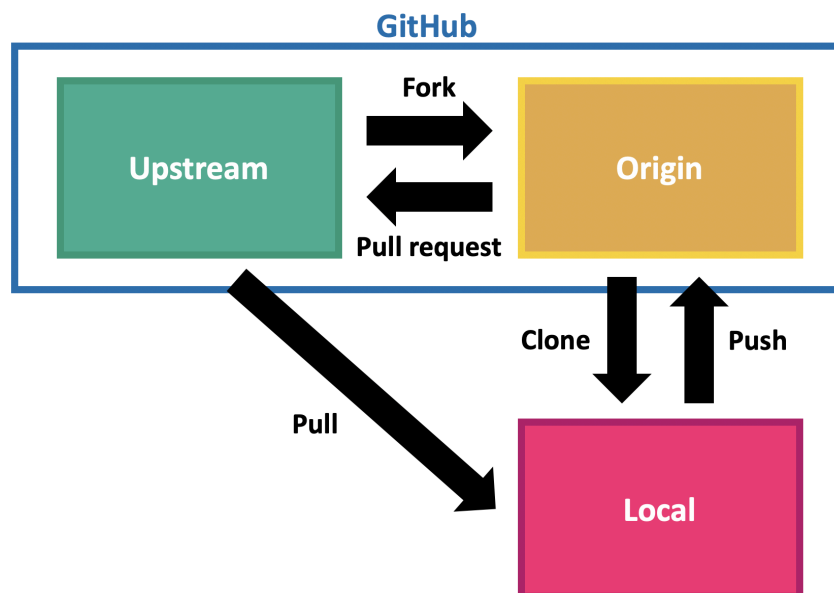
```
$git remote -v
```


Tenemos una fuente origin (repo en nuestro GitHub) y una fuente upstream (repo donde os cuelgo y actualizo los apuntes)

Una vez añadido, podemos bajar cambios del repositorio original con un pull:

```
$ git pull upstream <rama_a_descargar> / en nuestro caso la rama va a ser main  
$ git pull upstream main
```

De esta manera tendremos los apuntes actualizados con los últimos cambios que se hayan realizado.



HACER UNA PULL REQUEST:

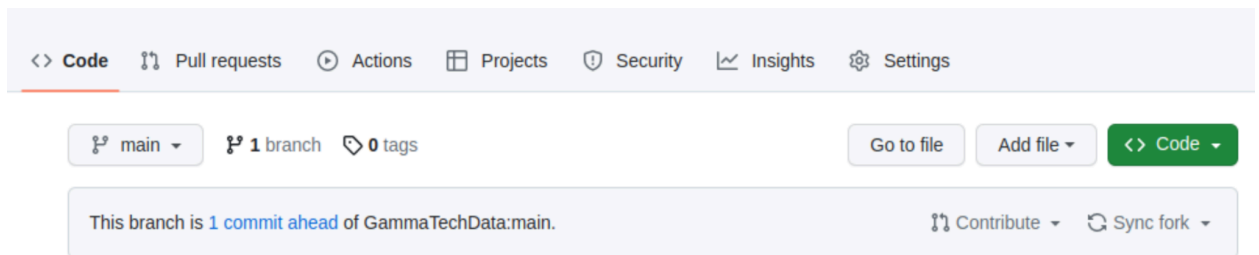
Una *pull request* es una petición para que se acepten los cambios que has realizado sobre un repositorio, desde tu **origin** hasta el **upstream** (ver *diagrama anterior*).

Es la manera de pedir cambios en el repositorio original con las modificaciones que hayamos introducido.

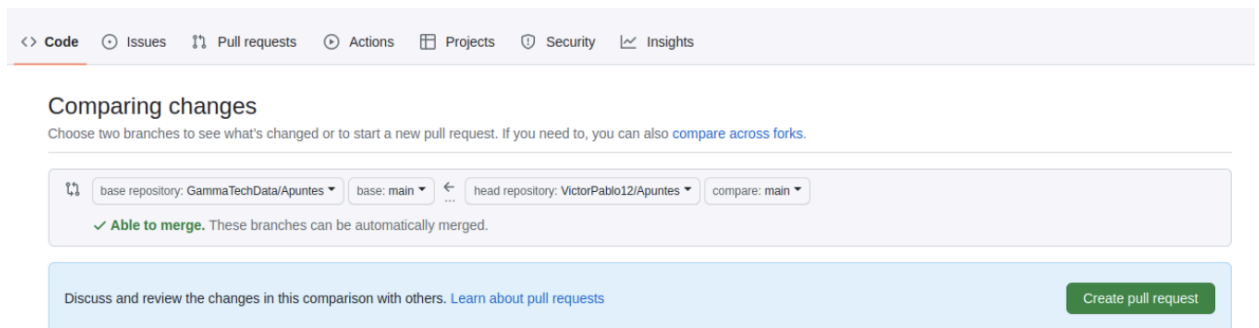
PASOS de UNA PR (pull request):

1. Debemos tener un repo forkeado y clonado en nuestro local.

2. Debemos haber realizado algún cambio sobre nuestro repo local, para que GIT detecte que hay diferencias entre nuestro repo local y remoto.
3. En la carpeta de nuestro repo local y desde nuestra terminal ejecutamos:
 - a. `git status` (nos dirá las diferencias de contenido de nuestros repos)
 - b. `git add nombre_archivo/s_cambiado/s` Podemos añadir entre 1 y n archivos en función de los que queramos añadir a staging o no. Con `$git add .` añadimos todos los archivos cambiados de una sola vez.
 - c. `git commit -m "mensaje de lo que he cambiado"` (debemos poner un mensaje que nos sirva para identificar los cambios que se han introducido en el commit, debe ser breve)
 - d. `git push origin main` (subimos todos los cambios que hemos adjuntado en staging a nuestro repo remoto)
4. Ahora nuestro repo de GitHub y el que nos forkeamos originariamente son distintos.
5. Abrimos GitHub y vamos a nuestro repo.
6. Vemos que estamos por delante del repo original



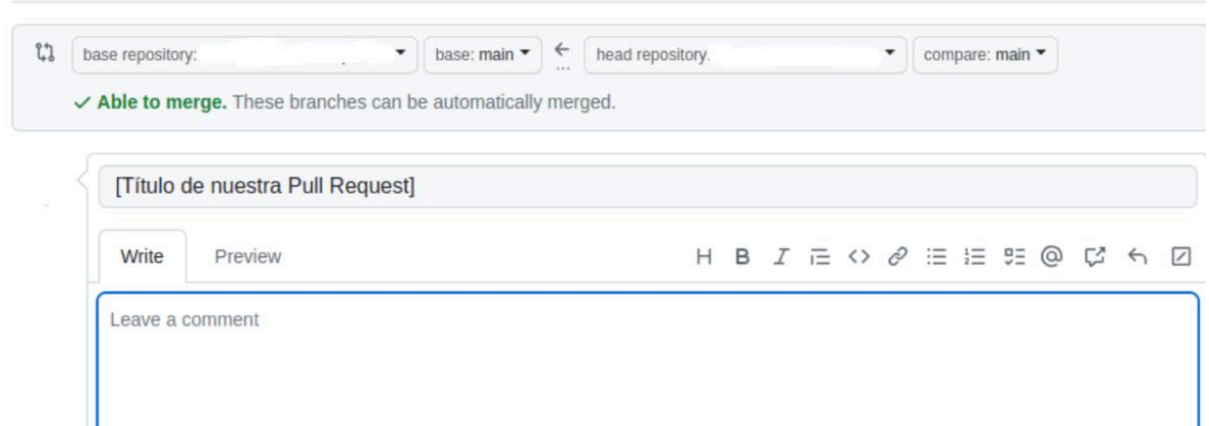
7. Pinchamos en *Pull requests* , luego en *New Pull Request*
8. Cambiará de página y pincharemos en *Create Pull Request*



9. Nos saltará un formulario con los detalles de nuestra PR

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](#).



10. El título de nuestra PR debe tener siempre el mismo formato.

nombre_del_ejercicio[nombre apellido]

Una vez hecha la PR, podéis seguir trabajando haciendo cambios sobre vuestro código sin problema.

Os recomiendo que cada vez que hagáis cambios o tengáis avances en vuestro código vayáis haciendo commits ya que son gratis y nos evitaremos problemas de perder trabajo.

EXTRAS

GESTION DE RAMAS

En Git, una rama (o branch en inglés) es una bifurcación del proyecto en la que puedes trabajar de forma independiente sin afectar a la rama principal (generalmente llamada **main** o **master**). Cada rama es una línea de desarrollo separada que te permite aislar cambios y nuevas características hasta que estés listo para integrarlas en el proyecto principal.

Conceptos Clave sobre las Ramas en Git

1. Aislamiento del Trabajo:

- Una rama te permite trabajar en nuevas características, corregir errores o experimentar con nuevas ideas sin interferir con el código en la rama principal.

2. Creación de una Rama:

- Para crear una nueva rama, usas el comando **git branch nombre-de-la-rama**. Luego, puedes cambiar a esa rama con **git checkout nombre-de-la-rama** o usar **git checkout -b nombre-de-la-rama** para crear y cambiar a la nueva rama en un solo paso.

3. Integración de Ramas:

- Una vez que has terminado de trabajar en una rama y has probado los cambios, puedes integrar esos cambios en otra rama (por ejemplo, `main`) usando el comando `git merge nombre-de-la-rama`.
4. **Flujo de Trabajo Típico con Ramas:**
- Crear una nueva rama: `git checkout -b nombre-de-la-rama`
 - Hacer cambios y commits en la nueva rama: `git add .`, `git commit -m "Mensaje del commit"`
 - Cambiar de vuelta a la rama principal: `git checkout main`
 - Fusionar la rama con la rama principal: `git merge nombre-de-la-rama`
 - Eliminar la rama (opcional): `git branch -d nombre-de-la-rama`
5. **Ramas Remotas:**
- Puedes compartir tu rama con otros desarrolladores subiéndola a un repositorio remoto como GitHub usando `git push origin nombre-de-la-rama`.

RESUMEN

1. Navegar al lugar donde vas a clonar el repositorio
2. Clonar repositorio

```
git clone git@github.com:tu-usuario/nombre-del-repositorio.git
```

3. Navegar al directorio del repo clonado

```
cd nombre-del-repositorio
```

4. Hacer cambios en archivos
5. Commit de los cambios

```
git add nombre-del-archivo
```

```
git commit -m "Mensaje del commit describiendo los cambios"
```

6. Subida al repositorio

```
git push origin main
```

*main es el nombre de la rama en el que estás trabajando, si es otra deberás cambiarlo por el nombre correspondiente.