

AWAKE LAB

#programmingbootcamp

CITY GITHUB





¿QUÉ VAMOS A VER?

- ¿Git = GitHub?
- Git.
- GitHub.
- ¿Clone o Fork?
- GitHub en Android Studio.



¡VAMOS A COMENZAR!

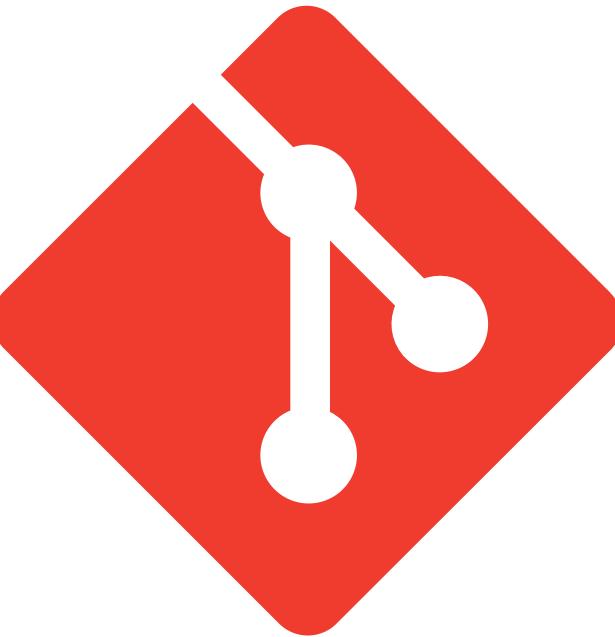




¿GIT = GITHUB?

¿GIT = GITHUB?

- Lo correcto sería **Git != GitHub** para conseguir un *True óptimo* dentro de la programación.
- Normalmente se trabajan en conjunto y no se suele nombrar uno sin el otro, aunque existan claras diferencias que analizaremos antes de saber como utilizarlos a nuestro favor.
- En la era actual el saber utilizar Git en general, es incluso tan o más importante que programar a la perfección, ya que será la forma en que puedas tener tu proyecto de manera local y compartir los cambios en tiempo real con tu equipo de trabajo.



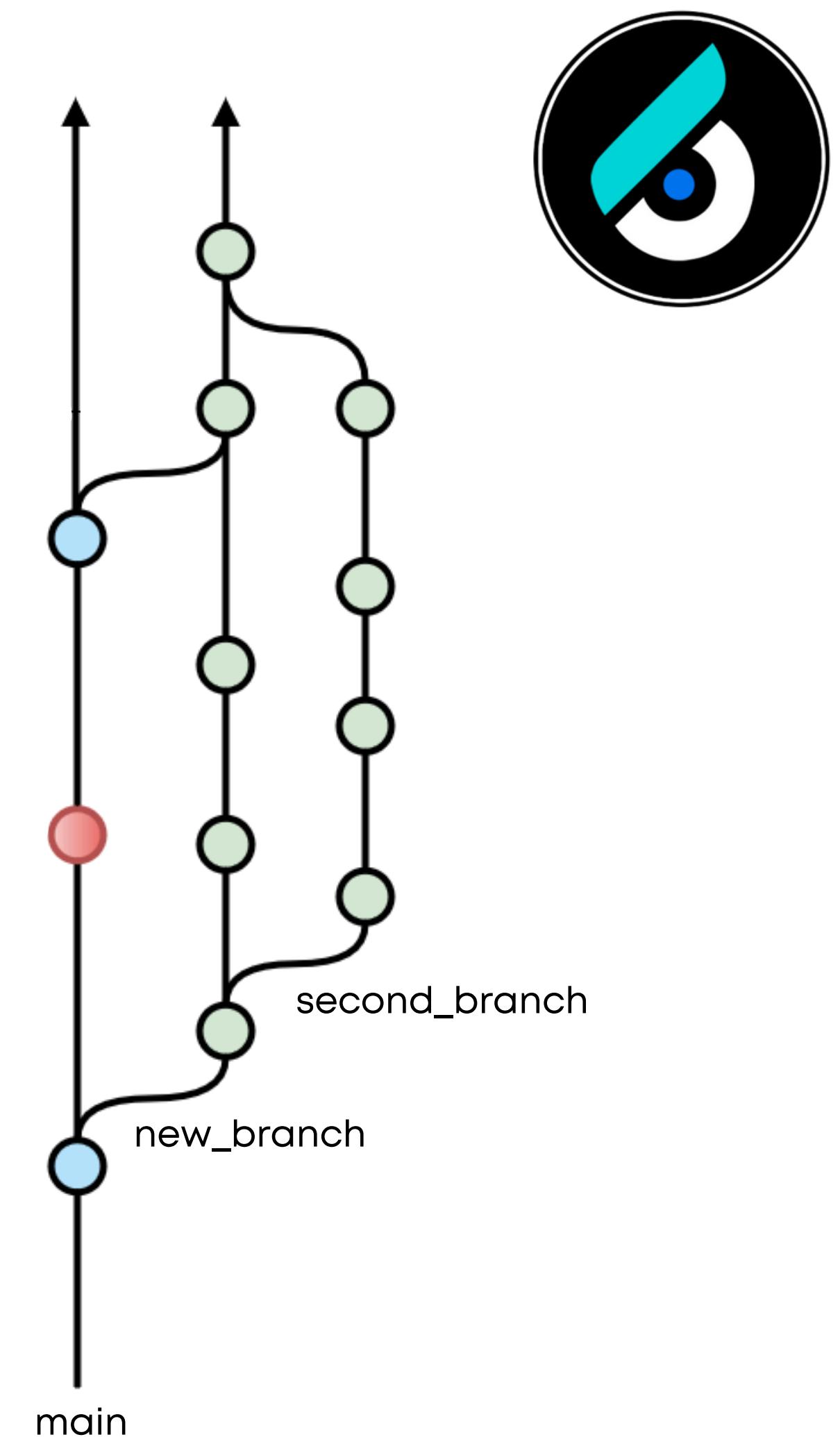


GIT



GIT

- Es un sistema de control de versiones que permite a los desarrolladores gestionar y seguir los cambios en el código fuente a lo largo del tiempo.
 - Permite crear ramas (branches) para trabajar en un línea de tiempo alterna a la versión principal.
 - Facilita la fusión (merge) de ramas y la resolución de conflictos.
 - Ofrece un manejo local (funciona sin internet), para realizar manejar cualquier código fuente o ramas que se encuentren en el sistema propio.
- Lo primordial de Git, es que no requiere de un servidor para funcionar, nos permite gestionar nuestras propias ramas de manera local sin problema.





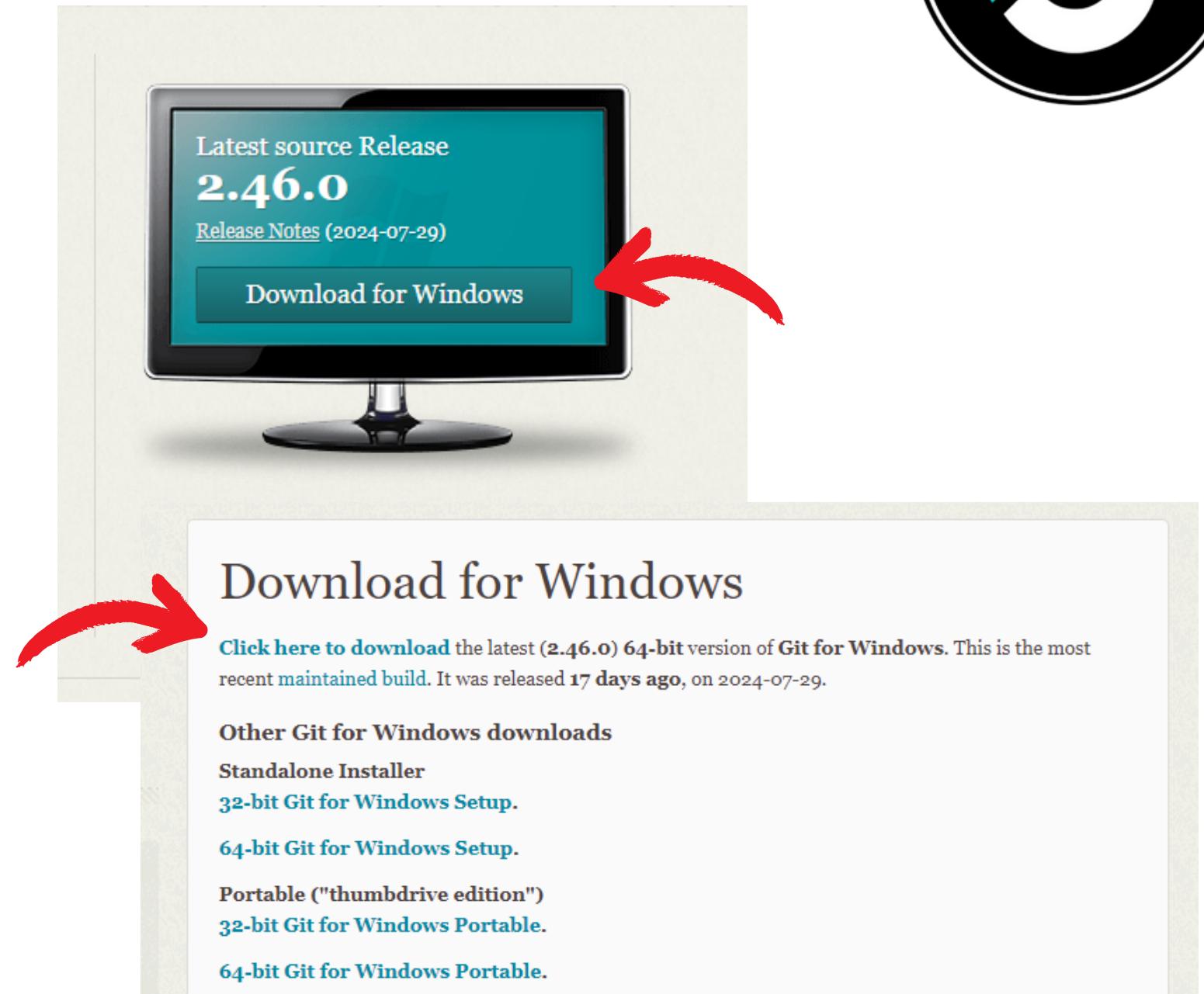
CONCEPTOS CLAVES

- **Repositorio:** Es una carpeta en tu computadora que contiene todos los archivos de tu proyecto, así como el historial de cambios.
- **Commit:** Es una instancia de los cambios realizados en tu proyecto.
- **Rama (Branch):** Es una línea alterna al desarrollo principal del proyecto. Se pueden crear múltiples ramas para trabajar en diferentes características en paralelo.
- **Fusión (Merge):** Es el proceso de combinación de los cambios realizados en una rama. Puede ser la combinación entre dos ramas o una rama con el proyecto principal.
- **Conflicto de fusión (Merge conflict):** Ocurre cuando se intenta mezclar dos ramas que han realizado cambios en el mismo código. Git proporciona una instancia de ayuda para poder resolver los conflictos manualmente.



DESCARGAR E INSTALAR GIT

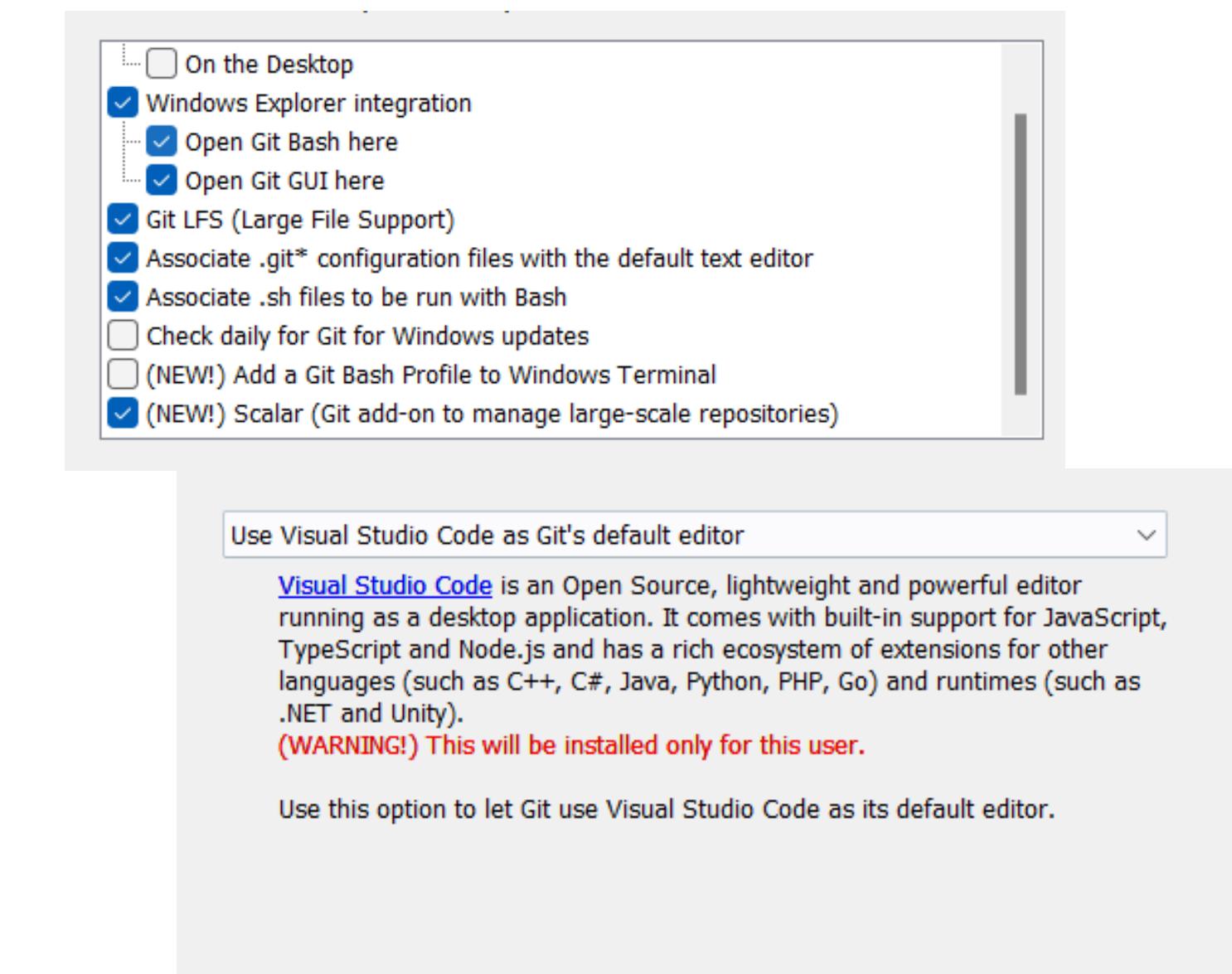
- Deben dirigirse al sitio oficial de Git:
git-scm.com
- En la página principal encontrarán un botón de descarga que reconocerá automáticamente el sistema operativo del que acceden.
- Se les abrirá una nueva pestaña en dónde podrán acceder a versiones anteriores, pero simplemente le dan clic en “**Click here to download**”
- Ahora comenzará la descarga del instalador que no durará más de unos minutos.





DESCARGAR E INSTALAR GIT

- Cuando le den comenzar al instalador, seguramente el SO les pedirá permisos, simplemente se los dan.
- En primera instancia se encontrarán con la licencia de Git, le dan a Next.
- Les mostrará la selección de componentes, simplemente dejan los que traiga por defecto.
- Ahora les va a pedir un editor de texto para enlazarlo a Git, por defecto trae seleccionado Vim. En caso de que no tengan otro, simplemente dejan ese y le dan a Next.





DESCARGAR E INSTALAR GIT

- En algunos casos puede que el siguiente punto les traiga por defecto la rama inicial antigua, o en otros les recomiende el nuevo nombre. A modo personal, les recomiendo marcar la segunda opción y darle de nombre “main”.
- Lo siguiente será especificar que tipo de PATH se utilizará, simplemente dejen la opción que Git trae como recomendado y le dan a Next.
- En la elección de SSH, le dejan la opción de “**Use bundled OpenSSH**”

The screenshot shows the configuration steps for a new Git installation:

- Step 1: Default Branch Name**
 - Options:
 - Let Git decide
 - Override the default branch name for new repositories
 - Description: Let Git use its default branch name (currently: "master") for the initial branch in newly created repositories. The Git project [intends](#) to change this default to a more inclusive name in the near future.
 - Input field: main
 - Note: This setting does not affect existing repositories.
- Step 2: PATH Configuration**
 - Options:
 - Use Git from Git Bash only
 - Git from the command line and also from 3rd-party software
 - Use Git and optional Unix tools from the Command Prompt
 - Description: This is the most cautious choice as your PATH will not be modified at all. You will only be able to use the Git command line tools from Git Bash.
 - Description: (Recommended) This option adds only some minimal Git wrappers to your PATH to avoid cluttering your environment with optional Unix tools. You will be able to use Git from Git Bash, the Command Prompt and the Windows PowerShell as well as any third-party software looking for Git in PATH.
 - Note: Both Git and the optional Unix tools will be added to your PATH.
Warning: This will override Windows tools like "find" and "sort". Only use this option if you understand the implications.
- Step 3: SSH Configuration**
 - Options:
 - Use bundled OpenSSH
 - Use external OpenSSH
 - Description: This uses ssh.exe that comes with Git.
 - Description: NEW! This uses an external ssh.exe. Git will not install its own OpenSSH (and related) binaries but use them as found on the PATH.

Buttons at the bottom: Only show new options, Back, Next, Cancel.



DESCARGAR E INSTALAR GIT

- Para especificar el tipo de conexión, dejan marcado “**Use the OpenSSL library**” y le dan a Next.
- Las opciones de conversión de líneas al final de un archivo de texto, también dejan la opción que propone el asistente de instalación y le vuelven a dar Next.
- Cuando lleguen a la opción de seleccionar la funcionalidad de “git pull”, dejan seleccionada la de “**Fast-forward or merge**”.
- En la vista de las credenciales seleccionan “**Git Credential Manager**”.

The screenshot shows the configuration interface for Git. It highlights several options with red arrows:

- Use the OpenSSL library**: Selected (radio button highlighted).
- Checkout Windows-style, commit Unix-style line endings**: Selected (radio button highlighted).
- Use MinTTY (the default terminal of MSYS2)**: Selected (radio button highlighted).
- Fast-forward or merge**: Selected (radio button highlighted).

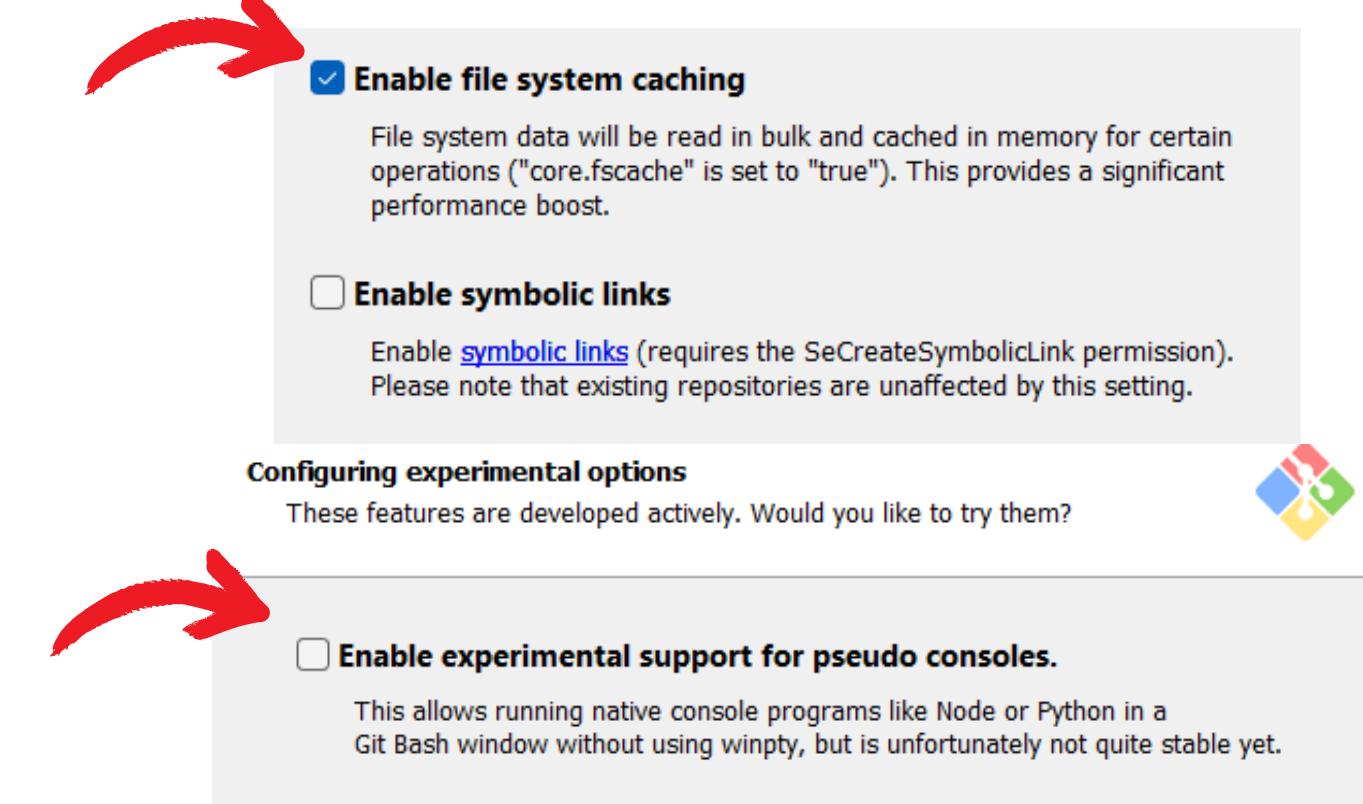
Below these, other options are shown:

- Use the native Windows Secure Channel library**
- Checkout as-is, commit Unix-style line endings**
- Use Windows' default console window**
- Rebase**
- Only ever fast-forward**



DESCARGAR E INSTALAR GIT

- Dentro de las opciones extras que saldrán luego de tantos Next, seleccionan **Enable file system caching**.
- Seguramente les saldrá una nueva opción experimental, que por ahora **no** la seleccionaremos.
- Por fin, le dan a **Install** y luego **Finish**, para terminar con el proceso de instalación.





CONFIGURACIÓN DE GIT

- Para comprobar si la instalación fue correcta, podemos abrir la **cmd** o la consola exclusiva de git (**Git Bash**) y escribir “**git --version**”
- El comando nos debería arrojar la versión de git que tenemos instalada.
- La configuración importante que debemos hacer, es especificar de manera global nuestro nombre y correo para un correcto seguimiento de los cambios.
- Cuando den enter en cada comando no aparecerá nada, por eso, para comprobar si se guardó correctamente la configuración, deben ejecutar el siguiente comando:
git config --global --list

```
carlo@ctapiadev MINGW64 ~
$ git --version
git version 2.46.0.windows.1
```

```
carlo@ctapiadev MINGW64 ~
$ git config --global user.name "Carlos Tapia"
carlo@ctapiadev MINGW64 ~
$ git config --global user.email "ctapiadev@gmail.com"
```

```
carlo@ctapiadev MINGW64 ~
$ git config --global --list
user.email=ctapiadev@gmail.com
user.name=Carlos Tapia
```



GITHUB

GITHUB



- Es una plataforma web que utiliza Git como sistema de control de versiones y proporciona herramientas adicionales para la colaboración de proyectos en línea.
- Nos va a ofrecer un repositorio remoto en dónde podemos almacenar nuestro código fuente y compartirlo con los demás, incluso, varios programadores pueden trabajar en conjunto dentro de un mismo repositorio.
- Incluye algunas funciones como el seguimiento de problemas (*issues*), la gestión de nuevo código (*pull requests*) o la misma documentación del proceso (*wiki*).
- Tenemos incluso la capacidad de crear proyectos privados, en dónde van a participar quienes nosotros invitemos.





CREANDO UNA CUENTA EN GITHUB

- github.com
- Lo primero que les pedirá la web, será un email valido y crear la contraseña.
- También les pedirá su nombre de usuario y un check de permisos para que GitHub les pueda enviar publicidad.

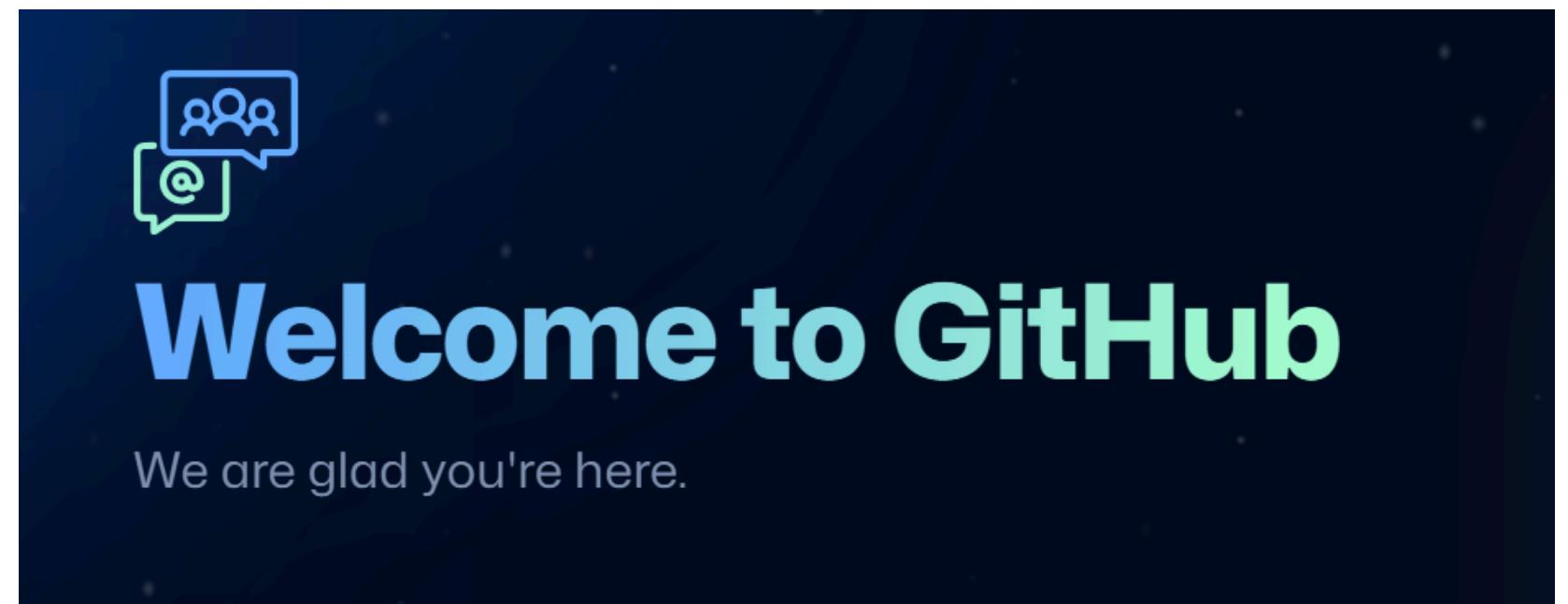
The image displays three sequential screenshots of the GitHub account creation interface:

- Screenshot 1:** Shows the first step where the user is prompted to "Enter your email*". A placeholder email "ctapiadevprueba@gmail.com" is entered, indicated by a green checkmark and the text "✓ ctapiadevprueba@gmail.com".
- Screenshot 2:** Shows the second step where the user is prompted to "Create a password*". A password has been entered, indicated by a green checkmark and the text "✓".
- Screenshot 3:** Shows the third step where the user is prompted to "Enter a username*". An empty input field is shown for the username.



CREANDO UNA CUENTA EN GITHUB

- Es importante el uso de un email valido, ya que luego de un pequeño “juego” de validación, les va a pedir introducir un código que les llegará al correo.
- Con todos esos pasos listos ya podrán acceder a su cuenta de GitHub por primera vez.
- Es probable que la primera vez que ingresen les pidan responder algunas preguntas.





CREAR E INICIAR UN PROYECTO

- En el mismo Home tendremos disponible una pequeña vista para agregar el nombre de nuestro nuevo repositorio y su estado (público o privado).
- Lo siguiente será decidir uno de los dos caminos a seguir que nos proporciona la plataforma, y eso va a depender de si estamos comenzando nuestro proyecto desde cero o queremos subir a nuestro repositorio un proyecto ya creado. En cualquier de las dos instancias, se deben dirigir a su Bash de la carpeta que quieren guardar el repositorio o tienen guardado, y ejecutar los comandos proporcionados por GitHub.

Start a new repository for bootAndroid

A repository contains all of your project's files, revision history, and collaborator discussion.

Repository name *

Public
Anyone on the internet can see this repository

Private
You choose who can see and commit to this repository

Create a new repository

...or create a new repository on the command line

```
echo "# clase_git_github" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin https://github.com/cTapiaDev/clase_git_github.git
git push -u origin main
```

...or push an existing repository from the command line

```
git remote add origin https://github.com/cTapiaDev/clase_git_github.git
git branch -M main
git push -u origin main
```



CREAR E INICIAR UN PROYECTO

- En este caso utilizaremos el camino de crear un proyecto desde cero, ya que será el repositorio que usaremos para practicar nuestros comandos e interacciones con la plataforma.
- En este caso voy a crear una carpeta vacía con el mismo nombre del repositorio y dentro daré segundo clic para abrir la **Git Bash**.
- En la consola ejecutaré el bloque de comandos proporcionados por GitHub.
- Si todo sale bien, ahora les aparecerá el nombre de la rama principal “**main**” al lado de la ruta. Además, dentro de la carpeta se creará el archivo README.md

```
carlo@ctapiadev MINGW64 ~/OneDrive/Escritorio/Bootcamp/clase_git_github
$ echo "# clase_git_github" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin https://github.com/cTapiaDev/clase_git_github.git
git push -u origin main

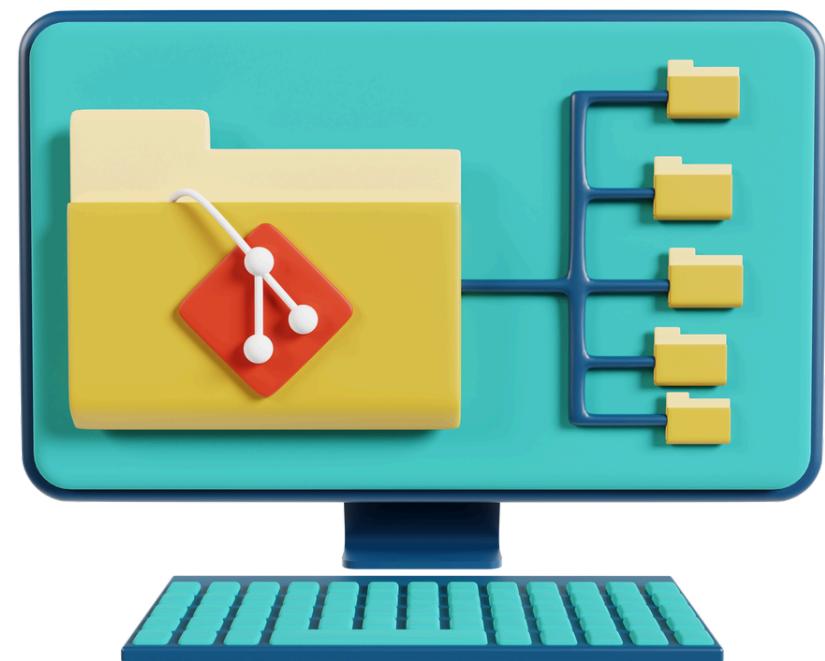
[main (root-commit) b8f558d] first commit
 1 file changed, 1 insertion(+)
 create mode 100644 README.md
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Writing objects: 100% (3/3), 231 bytes | 231.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To https://github.com/cTapiaDev/clase_git_github.git
 * [new branch]      main -> main
branch 'main' set up to track 'origin/main'.

carlo@ctapiadev MINGW64 ~/OneDrive/Escritorio/Bootcamp/clase_git_github (main)
$
```



COMANDOS PARA CREAR UNA RAMA

- **git branch nombre_rama:** Con este comando podemos crear una nueva rama en nuestro repositorio local. En caso de que el repositorio esté alojado en GitHub, la web va a reconocer nuestra rama con sus propios cambios.
- **git checkout nombre_rama:** Este comando nos permite posicionarnos en una rama para comenzar a trabajar en ella.
- **git branch:** Nos permite visualizar las ramas existentes en el repositorio.
- **git checkout -b nombre_rama:** Con este comando podemos agilizar el uso de la herramienta, ya que al mismo tiempo que creamos una nueva rama, nos posicionamos en ella inmediatamente.





COMANDOS BÁSICOS

- **git clone:** Nos sirve para clonar un repositorio de manera remota. Para que funcione, nos queda a nosotros proporcionarles la url referida al proyecto que queremos clonar en nuestro sistema local.

```
carlo@ctapiadev MINGW64 ~
$ git clone https://github.com/cTapiaDev/repo_bootcamp_android_diurno.git|
```

- **git add:** Este comando se va a utilizar para agregar nuevos archivos y/o carpetas a un área de preparación, en dónde los nuevos cambios se preparan para ser confirmados. Tenemos dos formas de proceder, la primera es agregando cada nombre de los archivos que deseamos agregar, y la segunda es agregar un punto, para adherir todos los cambios realizados hasta ese momento.

```
carlo@ctapiadev MINGW64 ~
$ git add .
```





COMANDOS BÁSICOS

- **git commit:** Una vez tengamos los cambios agregados o preparados, los debemos confirmar por medio de un commit, es más, lo recomendado es agregar un mensaje descriptivo para indicar que tipo de cambios se están subiendo al repositorio.

```
carlo@ctapiadev MINGW64 ~
$ git commit -m "Mensaje descriptivo del commit"
```

- Para que el mensaje sea considerado valido, se le agrega una “flag” antes de agregarlo, que en este caso es **-m**
- **git status:** Este comando nos va a mostrar el estado actual de nuestro repositorio, incluyendo los archivos modificados, preparados y commits realizados.

```
carlo@ctapiadev MINGW64 ~/OneDrive/Escritorio/Bootcamp
pertino (ctapiadev)
$ git status
On branch ctapiadev
Your branch is up to date with 'origin/ctapiadev'.

nothing to commit, working tree clean
```



COMANDOS BÁSICOS

- **git log:** Con este comando tendremos acceso a los registros de commits realizados en nuestro repositorio, incluyendo mensaje, fecha y autor.

```
carlo@ctapiadev MINGW64 ~/OneDrive/Escritorio/Bootcamp/repo_bootcamp_android_verse
pertino (ctapiadev)
$ git log
commit 798cd179366cad923ce570767d1bf73d09e62802 (HEAD -> cta
pertino (ctapiadev)
Author: Carlos Tapia <ctapiadev@gmail.com>
Date:   Wed Aug 14 20:43:05 2024 -0400

    transacciones

commit 85bd6095a2256fe8da5fe9ad6acb5476868cfcac
Author: Carlos Tapia <ctapiadev@gmail.com>
Date:   Tue Aug 13 22:28:31 2024 -0400

    clase 7
```

- **git pull:** Este comando se utiliza para obtener los cambios más recientes de un repositorio remoto y fusionarlos con tu rama local (En la que estés posicionado en ese momento).

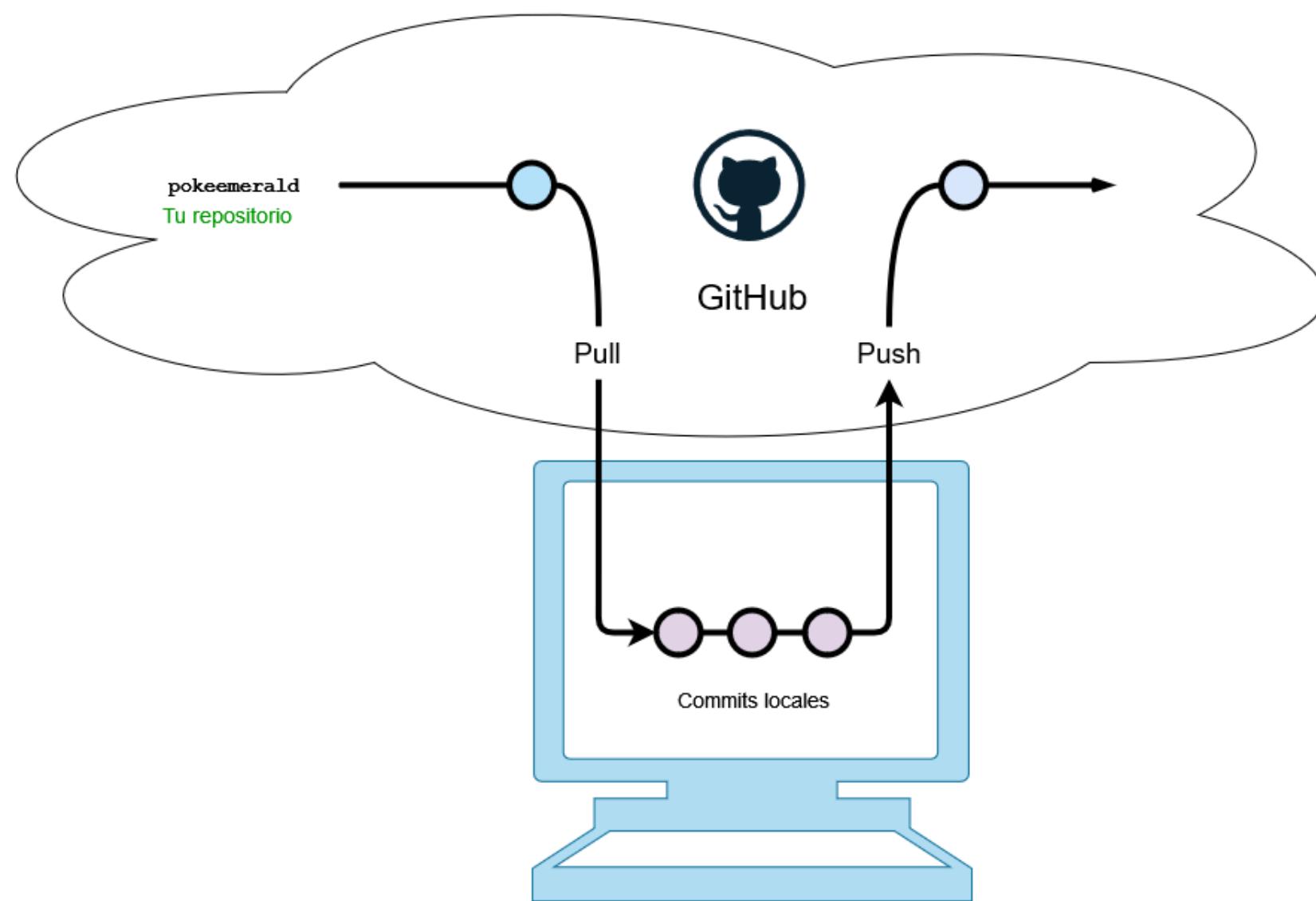
```
carlo@ctapiadev MINGW64 ~/OneDr
pertino (ctapiadev)
$ git pull origin nombre_rama|
```



COMANDOS BÁSICOS

- **git push:** Este es el comando a utilizar para enviar los cambios locales a un repositorio remoto. En conjunto se suele utilizar la “flag” **-u** (`--set-upstream`) para marcar el seguimiento a la rama remota.

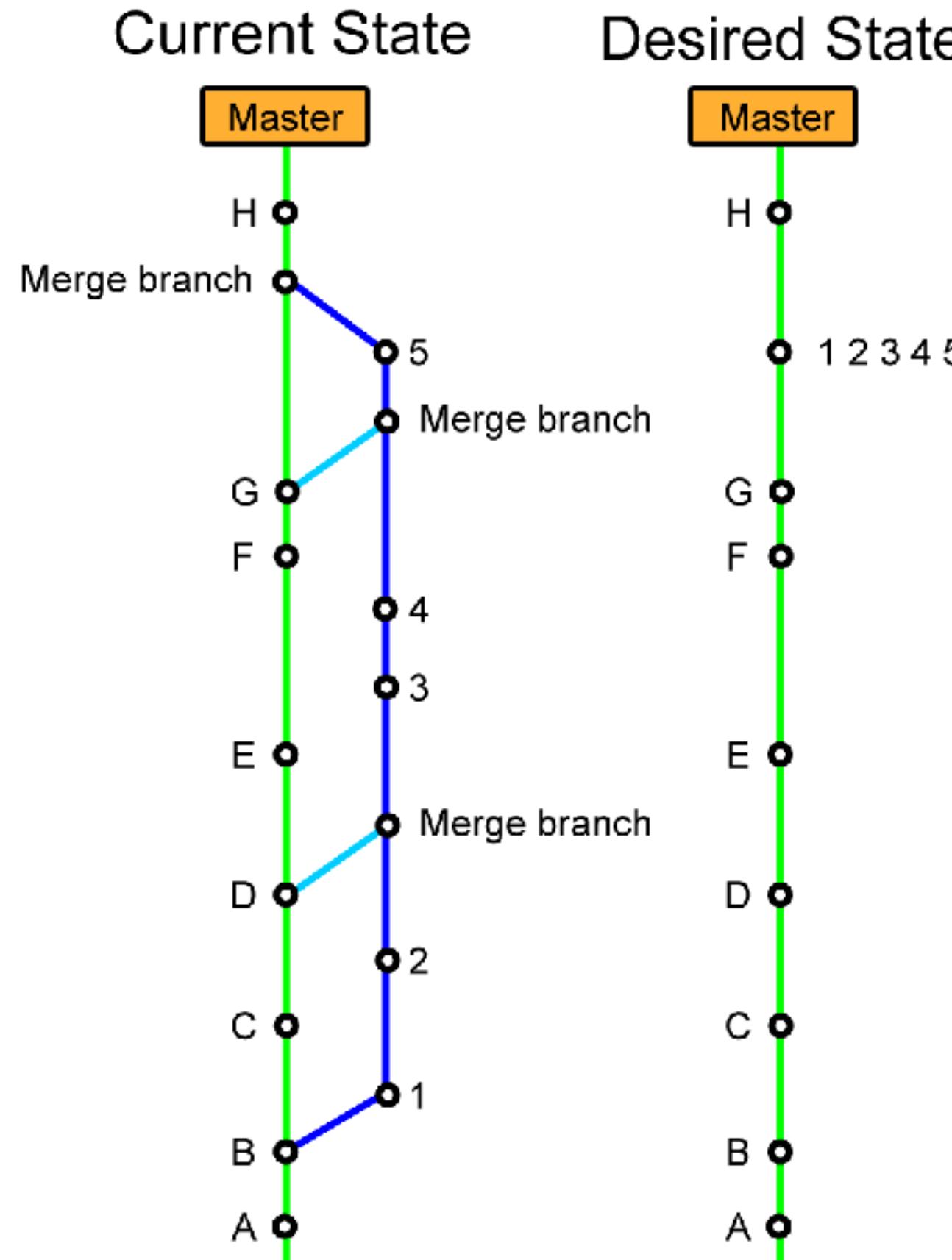
```
carlo@ctapiadev MINGW64 ~/OneDrive/Escriptino (ctapiadev)
$ git push -u origin nombre_repositorio
```





COMANDOS BÁSICOS

- **git merge nombre_rama:** Nos permite fusionar cambios locales, ya que desde la rama que “llamamos”, nos traemos sus cambios a nuestra rama.
- Es verdad estos no son todos los comandos existentes, la gama es muy amplia, al igual que las diferentes formas de trabajar dentro de un proyecto.





¿CLONE O FORK?



¿CLONE O FORK?

- La respuesta siempre será... **depende**.
- Podemos tener varios factores externos que afecten a la decisión de que camino tomar.
- ¿El proyecto es personal y solamente voy a trabajar yo?, ¿Lo estoy realizando con unos amigos y ninguno es más “dueño” que el otro? Además, confiamos entre nosotros.
- ¿Es un amplio equipo de trabajo? Quizás incluso de programadores nuevos que apenas se conocen... por aquí ya las cosas cambian un poco y es necesario tener un mayor control de lo que está pasando dentro del proyecto.



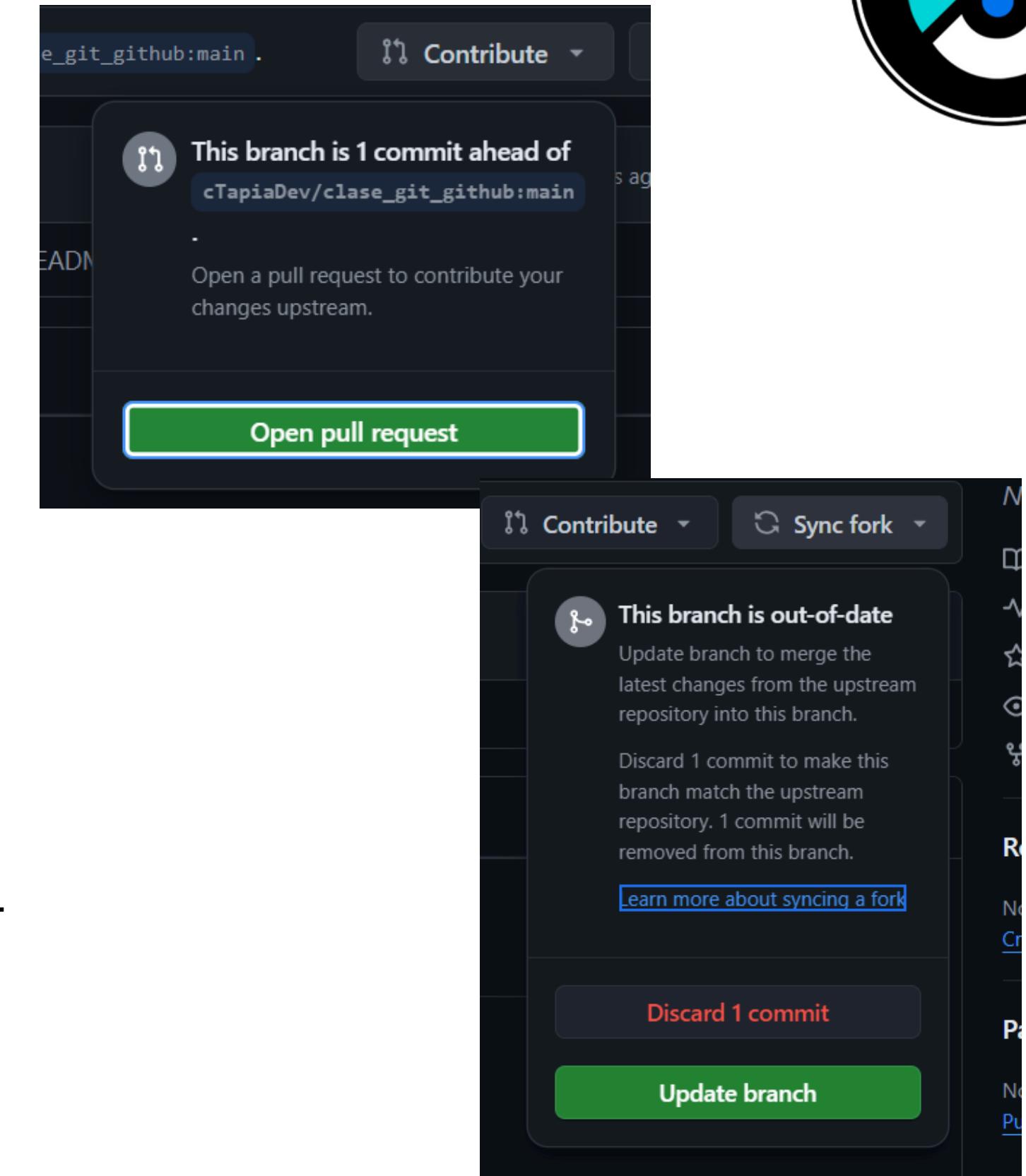
¿QUÉ ES CLONE?

- Ya les puede parecer conocido al ver los comandos de git, y es que hace referencia directa a git clone.
- Dentro de un proyecto se pueden asignar “Colaboradores” que tendrán gran acceso a las cosas/opciones del repositorio, pudiendo incluso mezclar directamente sobre la rama principal.
- **Settings > Collaborators > Add people**
- ¿En qué momento no podemos evitar dar el acceso de Colaborador a quién vaya a participar del proyecto? Cuando estemos trabajando con un proyecto Privado, en ese caso es necesario el acceso para poder visualizarlo, aunque eso no evita la decisión de tomar otro camino y no el de mezclar directamente las ramas.



¿QUÉ ES FORK?

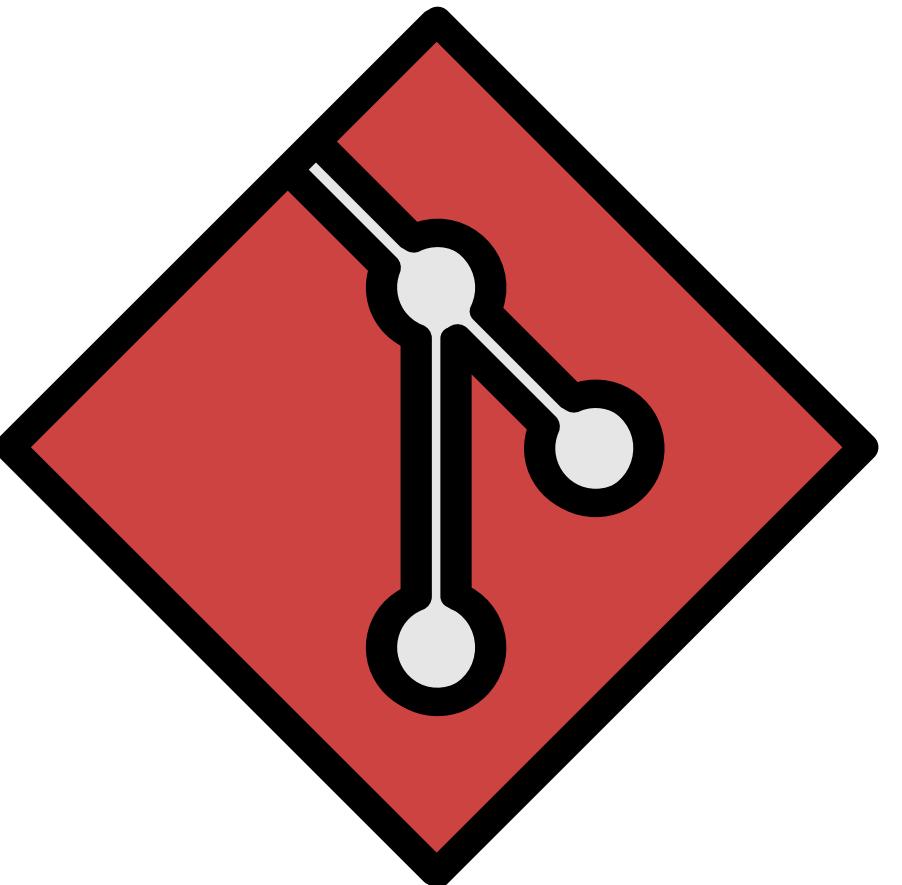
- En palabras simples, es hacer una copia del repositorio, en tu propio GitHub.
- Esto te permite de igual manera clonar la copia a tu espacio local y hacer todos los cambios que estimes convenientes dentro del proyecto.
- Es más, puedes mantener tu repositorio actualizado el tiempo que tu quieras, y GitHub te da la opción de Contribuir con el repositorio original por medio de un pull-request o incluso, traer a tu repositorio los cambios que se hayan realizado en el real por medio de una sincronización.





ENTONCES... ¿CUÁL ES MEJOR?

- Ninguna es mejor que otra, la respuesta a la primera pregunta sigue siendo “depende”.
- Aunque, la recomendación es siempre preferir el trabajo por medio de Forks, esto simplemente para evitar malos entendidos en la rama principal.
- De esa manera, cada cambio que se quiera agregar al main original, deberá ser revisado de que no tenga ningún conflicto en su merge, y obviamente aprobado por el líder del proyecto o algún encargado para realizar esa tarea.





GITHUB EN ANDROID STUDIO



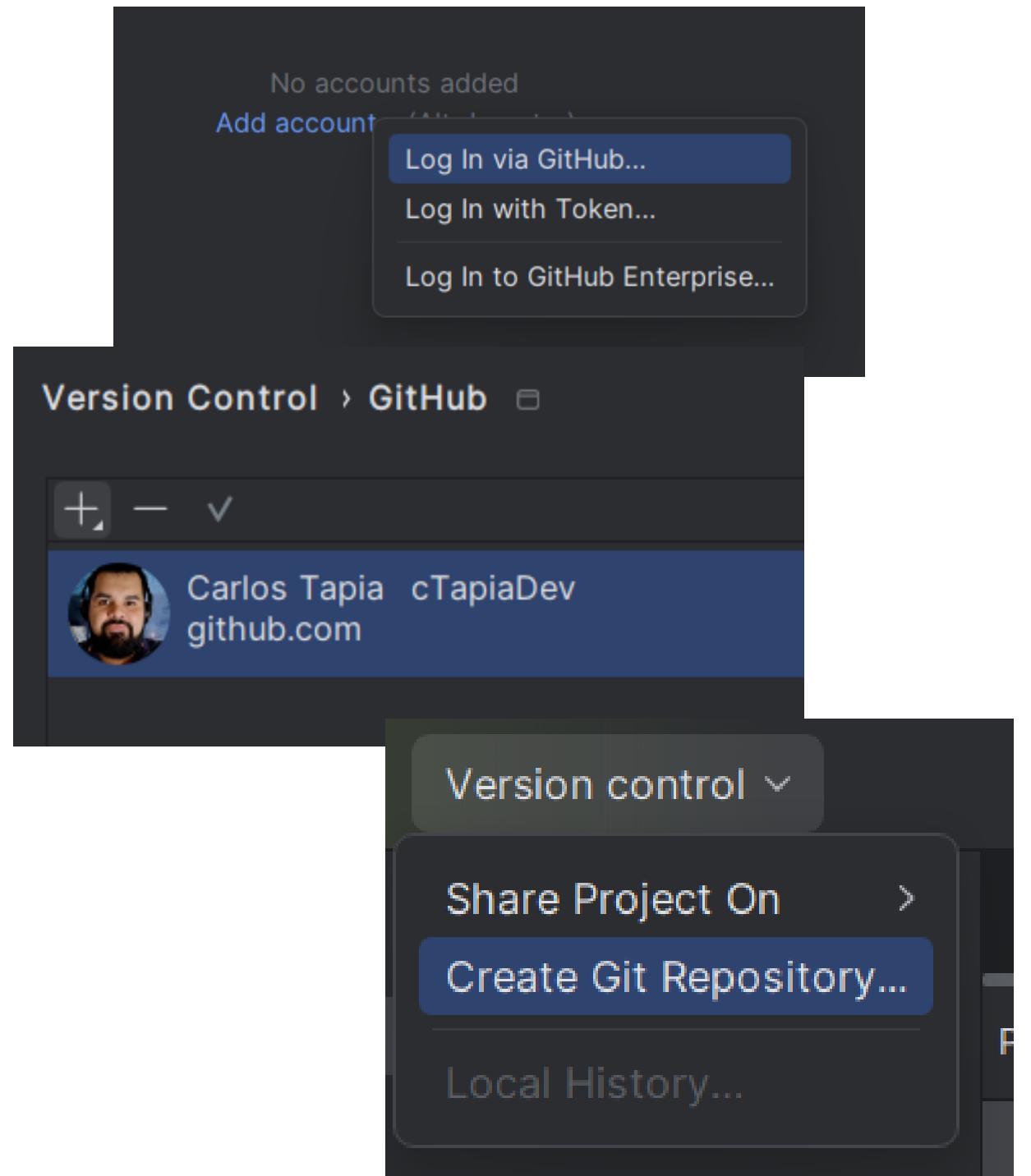
GITHUB EN ANDROID STUDIO

- **¿Puedo gestionar mis proyectos por medio de la consola de comandos?** Sin problema, es más, de entrada es mi recomendación personal para ustedes, ya que se tiene un control más amplio de las ramas, pero aún así existe la opción de gestionar el repositorio a través de Android Studio.
- Lo primero que debemos hacer es comprobar si Android Studio Koala está reconociendo automáticamente la ruta de Git.
- **File > Settings > Version Control > Git**
- Le aparecerá la ruta en la cual debería reconocerse el control de versiones, y pueden dar clic a Test para comprobar si existe o no una conexión.



GITHUB EN ANDROID STUDIO

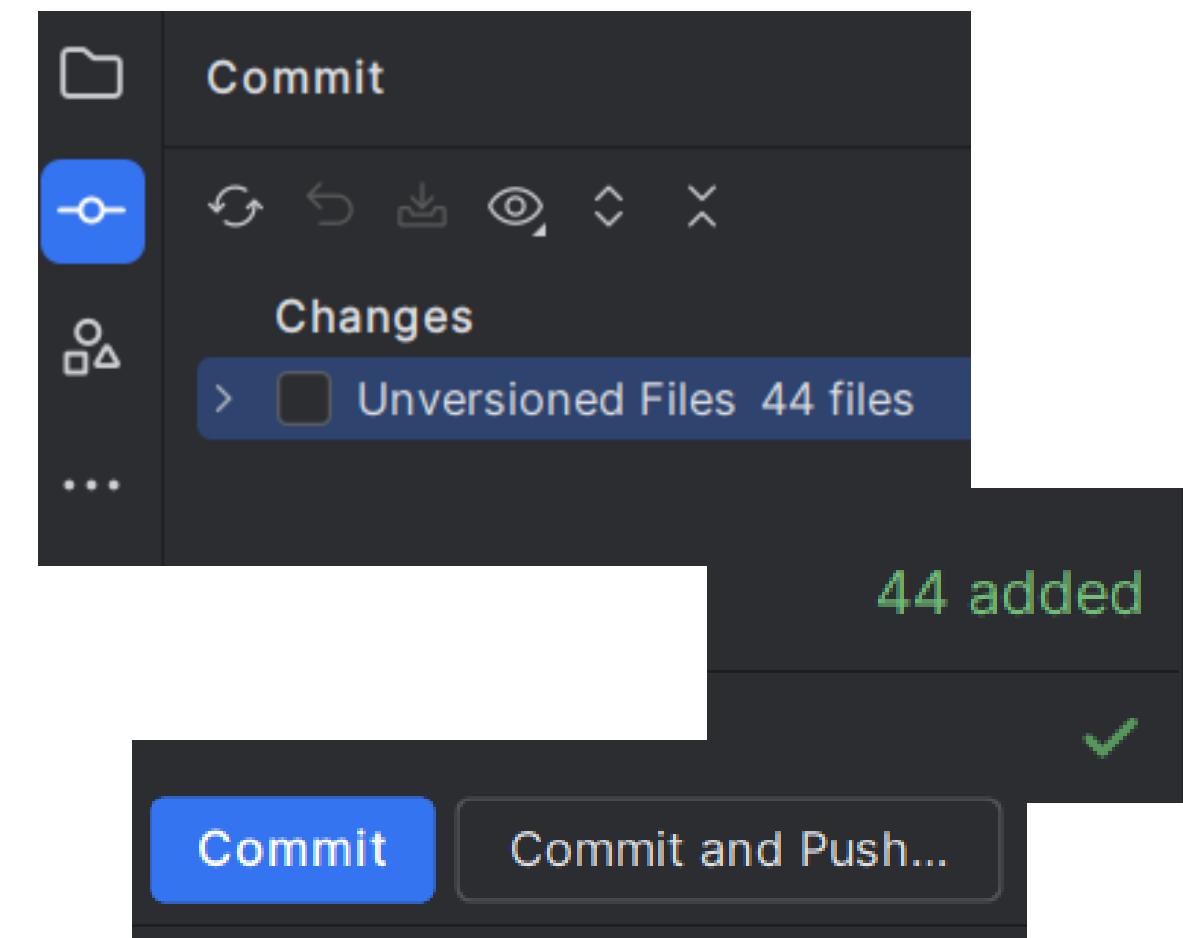
- Ahora debemos enlazar nuestra cuenta de GitHub directamente con Android Studio, para eso se van a:
- **File > Settings > Version Control > GitHub**
- Una vez ahí, le dan en **add account...** y luego en **Log In via GitHub...**
- Les pedirá autenticar su cuenta, lo hacen y ya les saldrá su usuario enlazado. Con eso listo, le dan en **apply** y **ok**.
- Con la cuenta ya enlazada, se van a la parte superior en **Version control**, y le dan a **Create Git Repository...** Se les abrirá una ventana en dónde deben seleccionar la ruta del mismo repositorio que quieren subir.





GITHUB EN ANDROID STUDIO

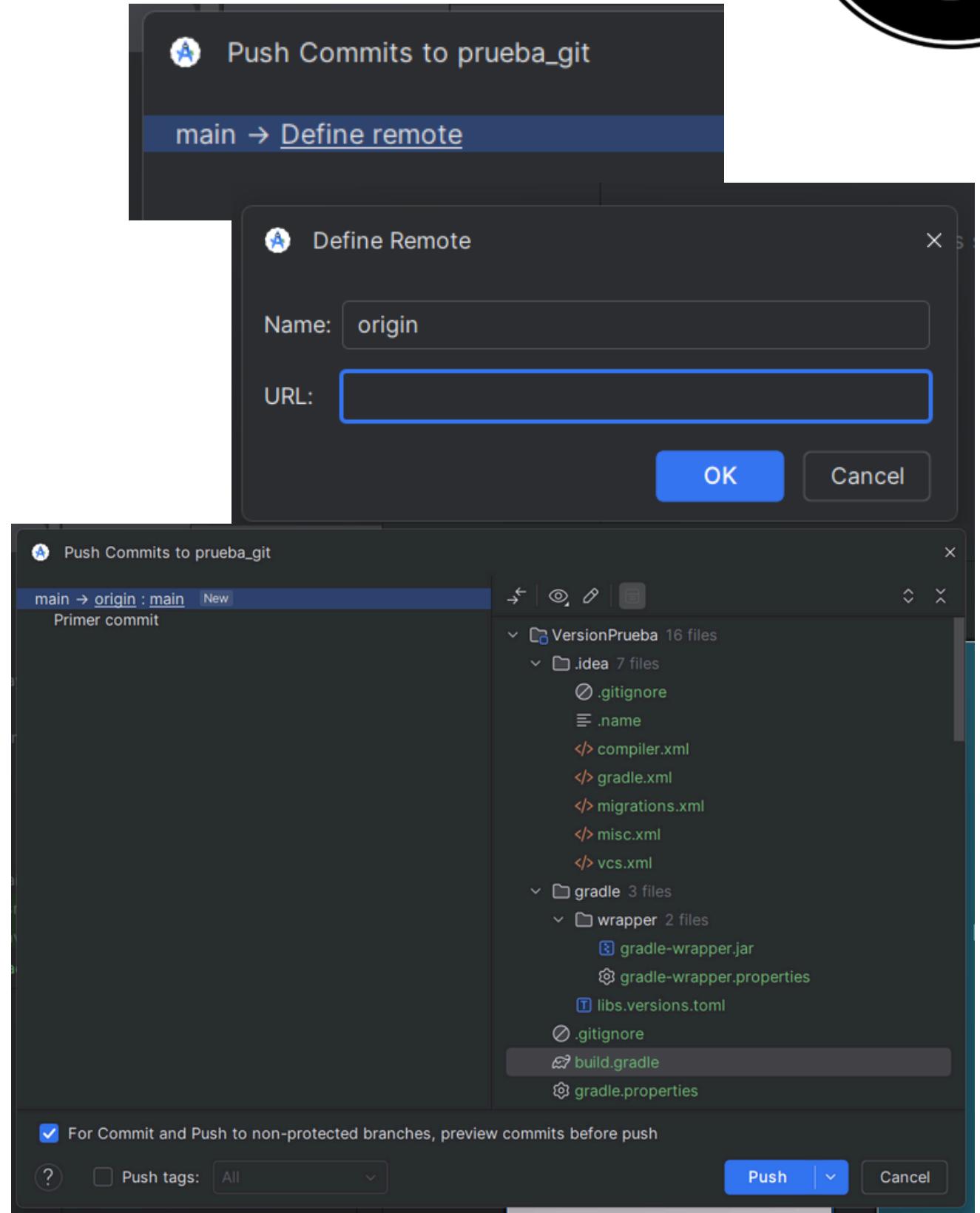
- Para hacer el primer commit puede ir a la barra lateral y hacer clic sobre el icono, o presionar **alt + 0** o **ctrl + k**
- Les aparecerán todos los cambios disponibles, aquí pueden elegir cuales agregar al commit o simplemente le dan clic a todo.
- Ahora les queda decidir si quieren hacer el commit de manera local, o directamente un commit + push para subirlo a GitHub. Aunque no se olviden de especificar el mensaje en la casilla del commit.
- De seguro les saldrá algún warning o mensaje de alerta y les pedirá confirmar el commit. Esto ocurre porque Android Studio reconoce como error hasta el warning más mínimo que se encuentre.





GITHUB EN ANDROID STUDIO

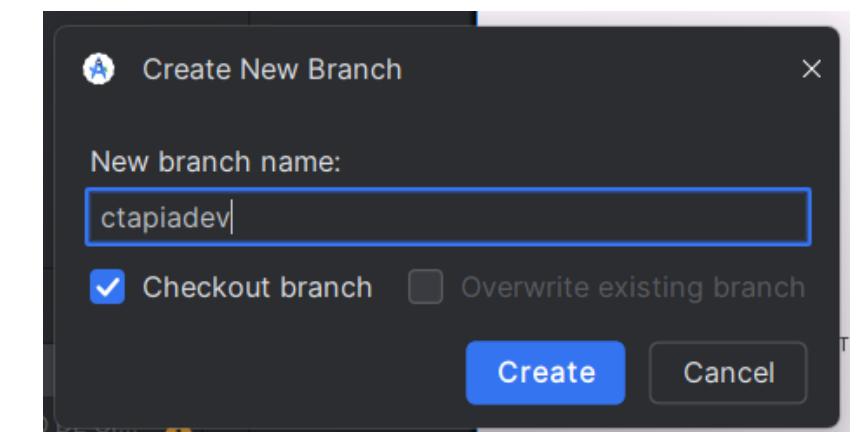
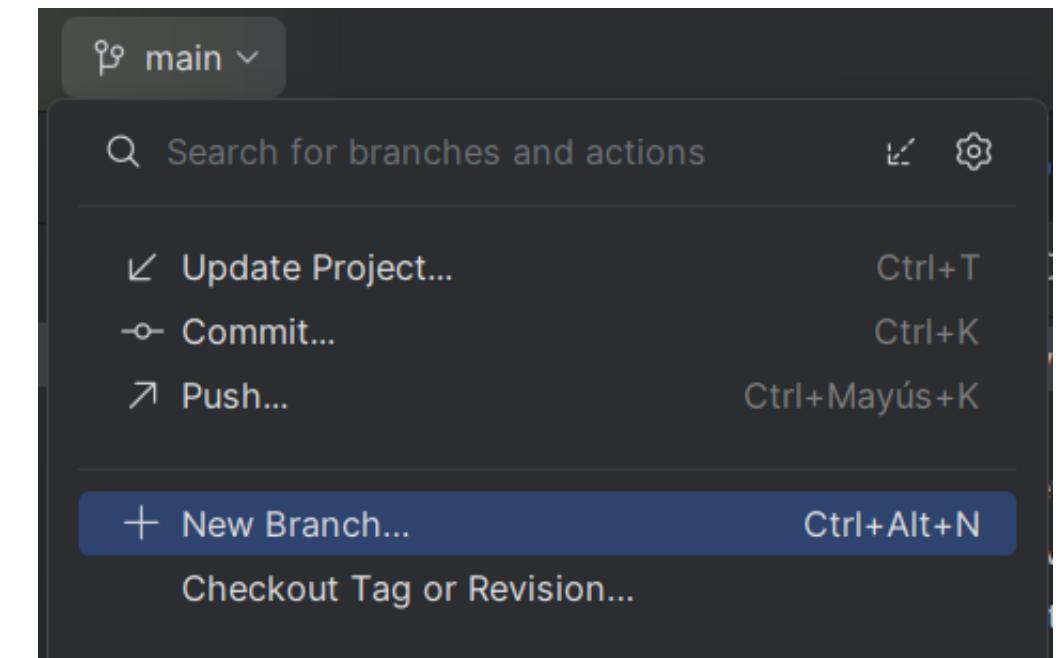
- Se les abrirá una pestaña que les pedirá definir el repositorio remoto. Para eso, nos vamos a GitHub y creamos un nuevo repositorio que alojará nuestro código fuente local.
- Con el link **HTTPS** que proporciona GitHub, creamos el enlace y nos aparece el desglose de código/carpetas que están listas para subirse. Simplemente le damos a **Push**.
- Una vez termine de cargar, pueden actualizar su GitHub y ver como el proyecto ya se encuentra arriba. Lo siguiente sería utilizar nuevamente el menú superior **Version Control** para crear una nueva rama y no trabajar directamente en el **main**, pero eso ya queda a criterio de cada uno.





GITHUB EN ANDROID STUDIO

- Para crear una nueva rama desde Android Studio, le dan a **New Branch...** y se abrirá una pequeña pestaña que les pedirá el nombre de su rama, es más, por defecto traerá seleccionada la opción de moverse directamente a esa rama.
- Solo les queda darle a **Create**, y ya tendrán su nueva rama para trabajar en ella sin afectar al código fuente principal de su proyecto.





**GRACIAS POR
LA ATENCIÓN**