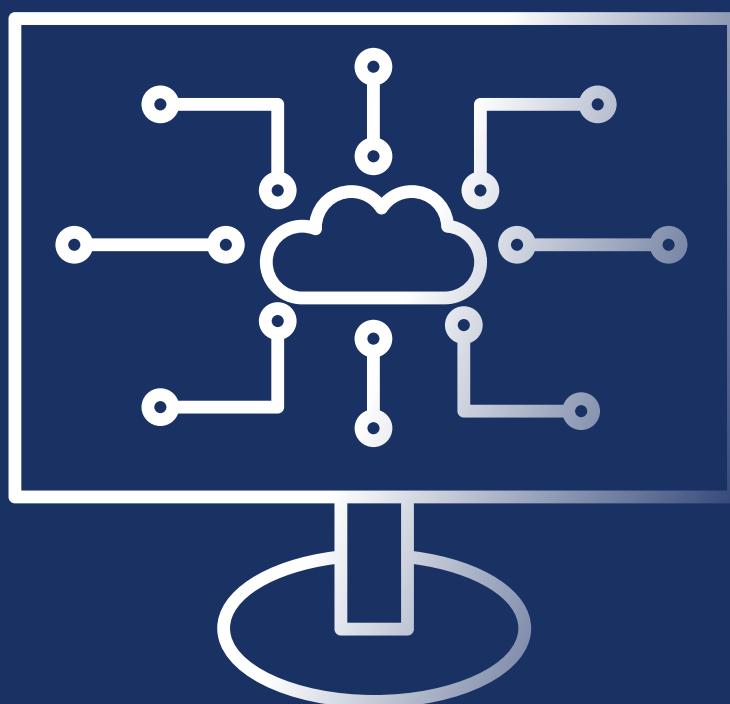


# Programación I



---

Lic. Julia Monasterio  
[marmonasterio@uade.com.ar](mailto:marmonasterio@uade.com.ar)



# Unidades

-  **Unidad 1:** Repaso de matrices, estructura, operaciones y uso en funciones
-  **Unidad 2:** Implementación y gestión en proyectos en Git
-  **Unidad 3:** Listas avanzadas, cadena de caracteres y expresiones regulares.
-  **Unidad 4:** Manipulación avanzada de diccionarios, tuplas y conjuntos.
-  **Unidad 5:** Excepciones y pruebas unitarias
-  **Unidad 6:** Archivos
-  **Unidad 7:** Recursividad

# Clase N°2

## TEMAS

- Conceptos básicos de control de versiones: repositorios locales y remotos
- Clonación de repositorios
- Seguimiento de archivos y cambios
- Operaciones: añadir, confirmar y eliminar archivos
- Integración de Git con plataformas de alojamiento como Github

# Control de versiones

10/8/2022 creo el archivo **biografia.txt** que contenga la siguiente información

 biografia.txt: Bloc de notas

Archivo Edición Formato Ver Ayuda

Fecha de Nacimiento: 17/4/1990

Lugar de Nacimiento: América - Prov. Bs As.

Peso: 70

Altura: 1.68

Lugar de Residencia: Madrid - España |

10/8/2024 modiflico el archivo con los datos recientes.

 \*biografia.txt: Bloc de notas

Archivo Edición Formato Ver Ayuda

Fecha de Nacimiento: 17/4/1990

Lugar de Nacimiento: América - Prov. Bs As.

Peso: 80

Altura: 1.67

Lugar de Residencia: Buenos Aires |

Si guardo el archivo modificado, pierdo el archivo original.  
Por eso sería ideal tener una forma de solo guardar los cambios que se van realizando

# SISTEMA DE CONTROL DE CAMBIOS

Un sistema de control de versiones es una herramienta o software que **permite gestionar los cambios realizados en archivos** a lo largo del tiempo.

Permite a los usuarios guardar diferentes versiones de los archivos, registrar **quién hizo qué cambios y cuándo, y comparar y revertir versiones si es necesario.**

**1** biografia.txt: Bloc de notas[Archivo](#) [Edición](#) [Formato](#) [Ver](#) [Ayuda](#)

Fecha de Nacimiento: 17/4/1990

Lugar de Nacimiento: América - Prov. Bs As.

Peso: 70

Altura: 1.68

Lugar de Residencia: Madrid - España |

**2** biografia.txt: Bloc de notas[Archivo](#) [Edición](#) [Formato](#) [Ver](#) [Ayuda](#)

Fecha de Nacimiento: 17/4/1990

Lugar de Nacimiento: América - Prov. Bs As.

Peso: 80

Altura: 1.67

Lugar de Residencia: Buenos Aires |

## CON SISTEMA DE CONTROL DE CAMBIOS

Se guardan solamente los cambios que se realizaron:

- 70 > 80
- 7>8
- Madrid - España > Buenos Aires

# SISTEMAS DE CONTROL DE CAMBIOS



Bazaar





Git es un software de control de versiones diseñado por **Linus Torvalds**.

Fue pensado para el mantenimiento de versiones de aplicaciones - cuando tienen un gran número de archivos de código fuente.

**Su propósito es llevar registro de los cambios en archivos y coordinar el trabajo de varias personas en archivos compartidos**



# DESARROLLO COLABORATIVO

Proporciona herramientas para que múltiples personas puedan hacer desarrollos en conjunto de una contribuyendo simultáneamente al mismo proyecto.

# GIT vs OTROS SISTEMAS DE CONTROL DE VERSIONES

La principal diferencia entre Git y otros sistemas de control de versiones (SCV) tradicionales se encuentra **en cómo almacenan y gestionan la historia** de los cambios en un proyecto.

# ENFOQUE TRADICIONAL

Almacenan la información del proyecto como una serie de diferencias o cambios entre versiones.

- **Lista de cambios (Deltas):**
  - Estos sistemas guardan la historia de un archivo registrando las diferencias (deltas) entre una versión y la siguiente.
  - Por ejemplo, si tienes un archivo A.txt en la versión 1 y luego haces un cambio para crear la versión 2, el sistema guarda lo que cambió entre la versión 1 y la versión 2.

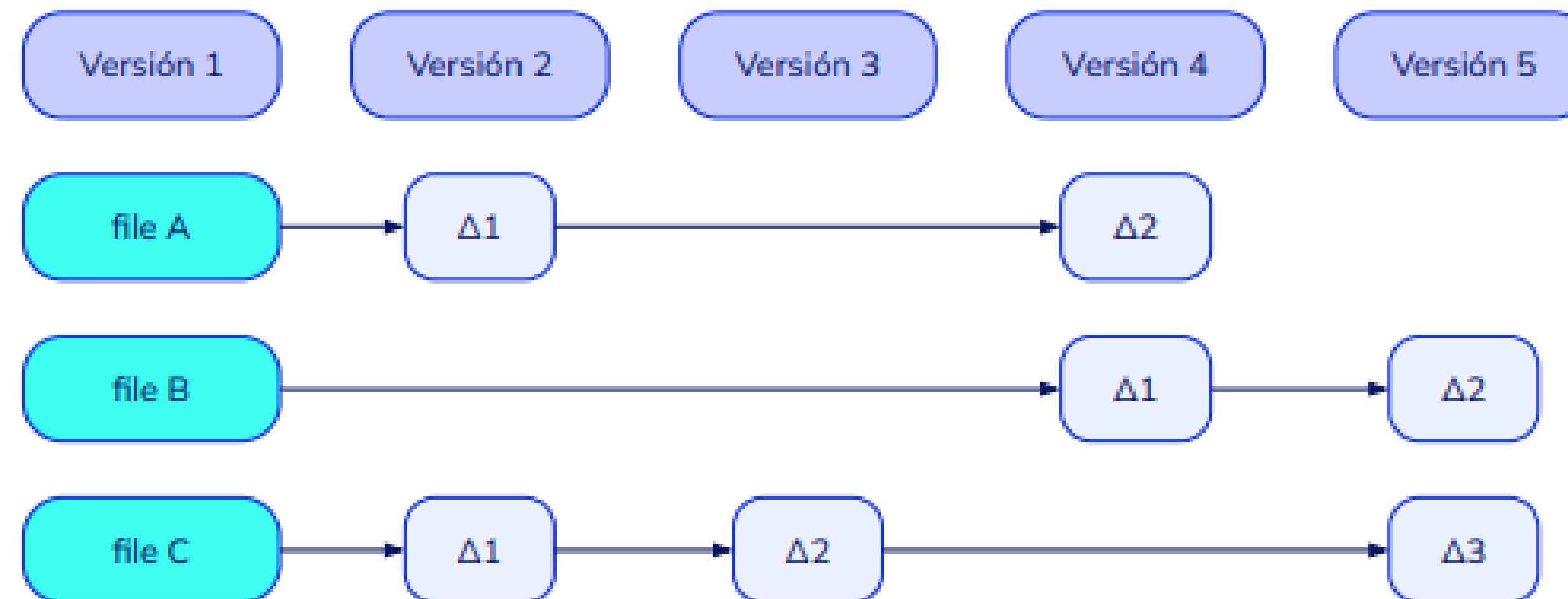
# ENFOQUE GIT

Git piensa en sus datos como un **conjunto de snapshots (instantáneas)** de un mini sistema de archivos.

Cada vez que se confirma o se guarda el estado de un proyecto en Git, básicamente se toma una fotografía de cómo se ven todos los archivos en ese momento y se almacena una referencia a esa instantánea.

**Importante:** si los archivos no han cambiado, **Git no almacena el archivo nuevamente**, solo un enlace al archivo idéntico anterior que ya ha almacenado.

# ENFOQUE GIT



# DESCARGA E INSTALACIÓN

Para **Windows** se debe ingresar a :

<https://gitforwindows.org/>

Para **MAC**:

<https://git-scm.com/download/mac>

O directamente accediendo a:

<https://git-scm.com/downloads>

## Downloads



macOS



Windows



Linux/Unix



Older releases are available and the [Git source repository](#) is on GitHub.

### GUI Clients

Git comes with built-in GUI tools ([git-gui](#), [gitk](#)), but there are several third-party tools for users looking for a platform-specific experience.

[View GUI Clients →](#)

### Logos

Various Git logos in PNG (bitmap) and EPS (vector) formats are available for use in online and print projects.

[View Logos →](#)

### Git via Git

If you already have Git installed, you can get the latest development version via Git itself:

```
git clone https://github.com/git/git
```

You can also always browse the current contents of the git repository using the [web interface](#).

# GIT

1. Dentro de **/etc/gitconfig** -> contiene los valores específicos de todos los usuarios del Sistema Operativo y todos los repositorios
2. **~/.gitconfig** -> contiene los valores específicos del usuario

Para ver todas las configuraciones podemos utilizar el comando:

**git config --list --show-origin**

# GIT

## Modificar nombre de usuario:

**git config --global user.name “Nombre de usuario”**

## Modificar Correo electrónico:

**git config --global user.email “email”**

**Este mail y este usuario es el que se asociará a los commits en todos los repositorios.**

# GIT

## Opción --global vs --local

**--global:** Configura el correo electrónico para **todos los repositorios en tu sistema.** Es útil si siempre utilizas el **mismo correo para todos tus proyectos.**

**--local:** Configura el correo electrónico **solo para el repositorio actual.** Es útil si trabajas en múltiples proyectos y necesitas **usar diferentes correos para cada uno.**

# GIT

## Validar configuraciones

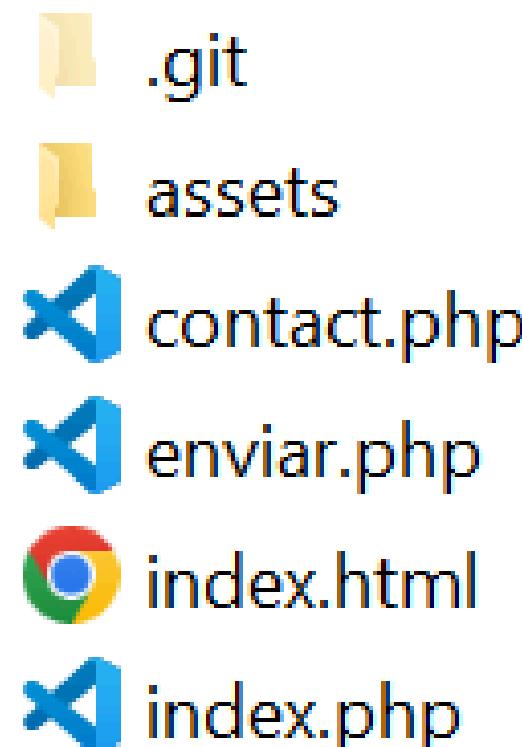
Con el comando **git config --list** se listan todas las variables con su valor y si quiero ver el valor de alguna variable en particular, por ejemplo nombre podemos ejecutar:

**git config user.name**

# Repositorios

# Repositorios

Un repositorio en git es la carpeta **.git/** dentro de un proyecto. Esta carpeta la crea git al inicializar el proyecto



Un directorio .git tiene la siguiente estructura:

	hooks
	info
	logs
	objects
	refs
	COMMIT_EDITMSG
	config
	description
	HEAD
	index

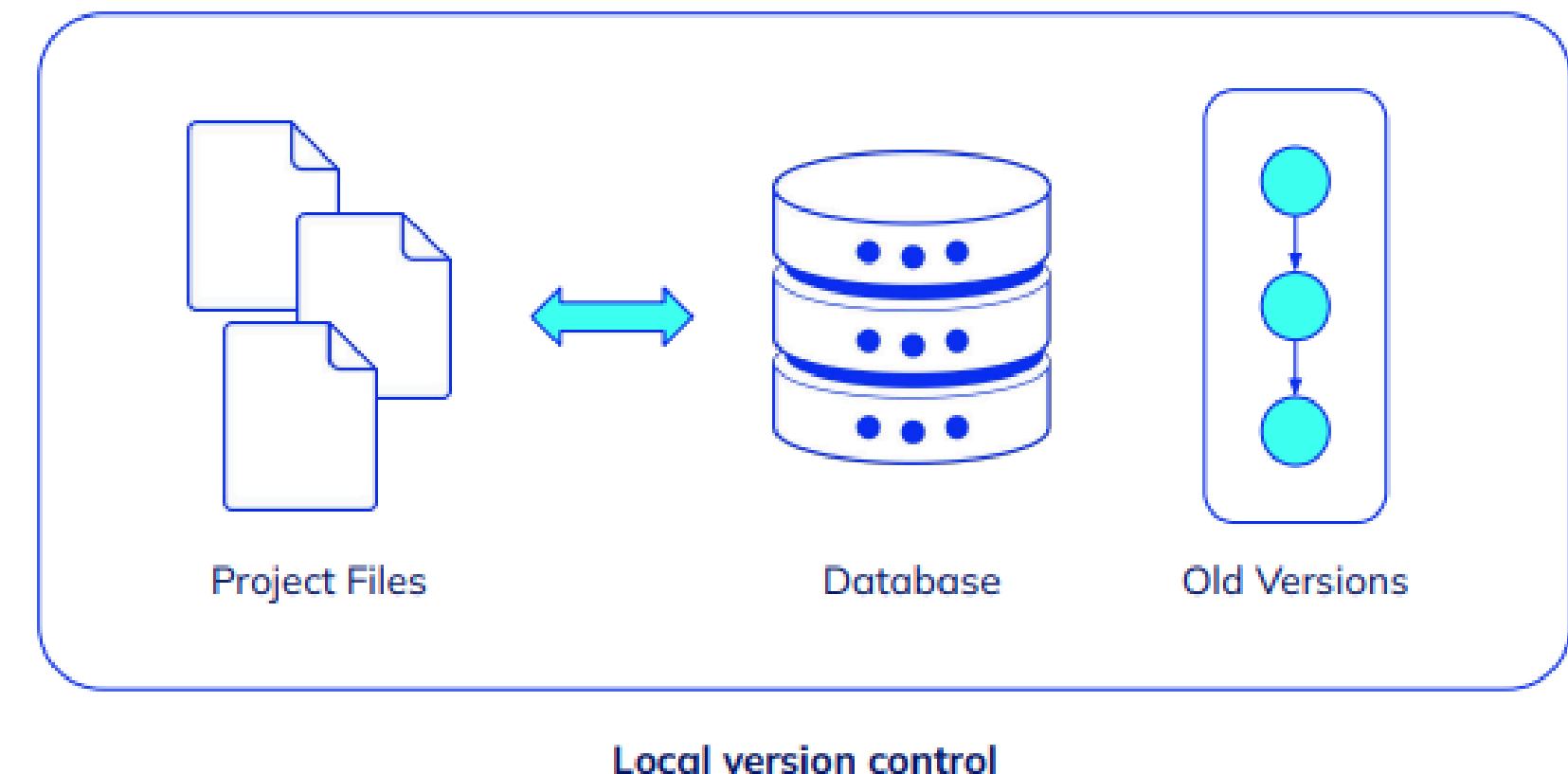
Rastrea todos los cambios realizados en los archivos de un proyecto y crea un historial a lo largo del tiempo. **Si se elimina la carpeta .git/ entonces desaparece el historial del proyecto**

# Tipos de Repositorios

## REPOSITORIOS LOCALES

Existe en el sistema de archivos de tu computadora.

Está compuesto por todos los archivos del proyecto y un subdirectorio oculto .git que contiene toda la información del historial de cambios, configuraciones, y metadatos necesarios para que Git funcione.



**No ofrece la posibilidad de compartir el código**  
en caso de que se elija trabajar en modo de  
Desarrollo Colaborativo

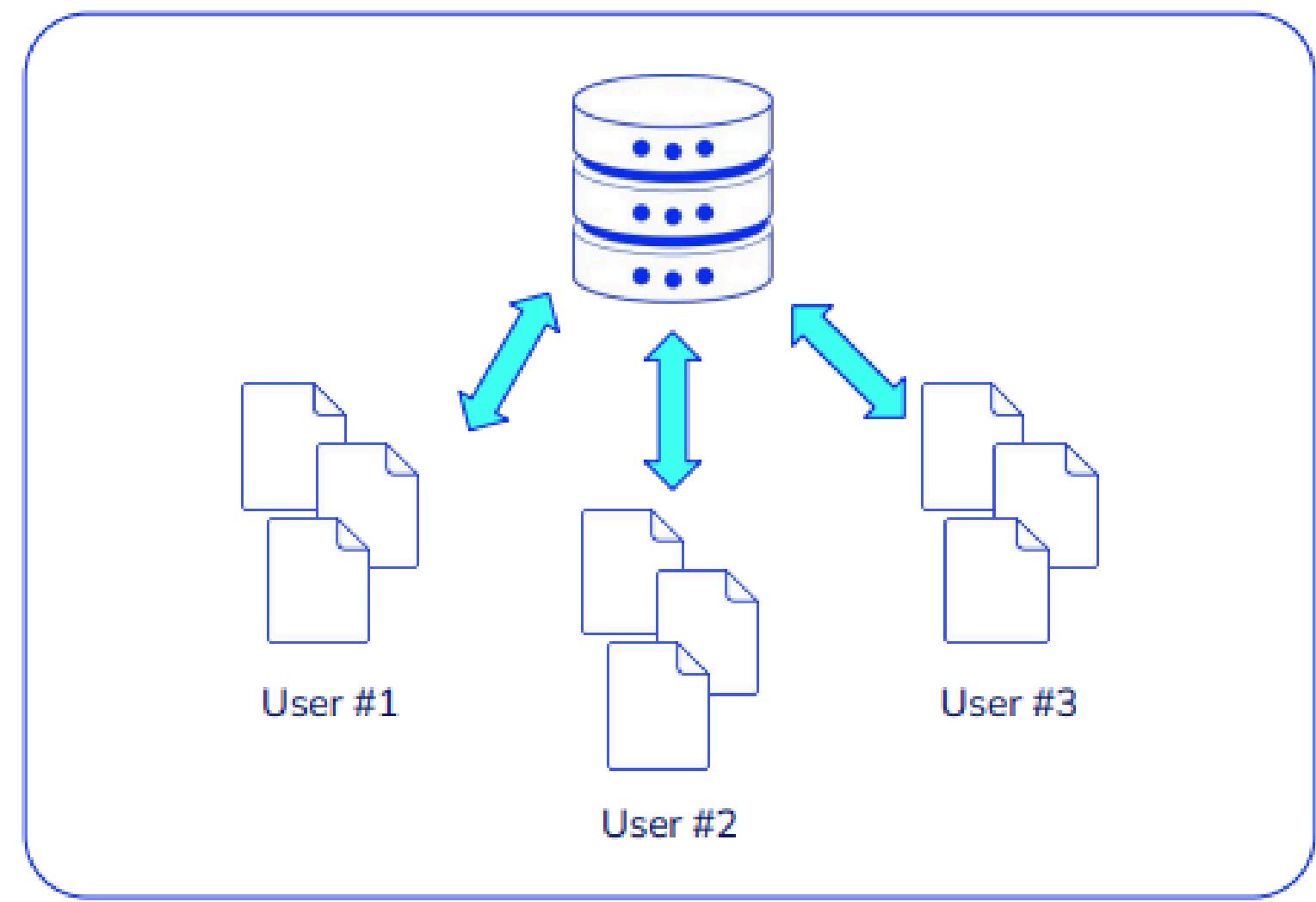
# Tipos de Repositorios

## REPOSITORIOS CENTRALIZADOS

En lugar de almacenar los cambios y versiones en el disco duro, pasaron a **guardarlas en un servidor.**

El avance frente a los sistemas de control de versiones locales fue enorme, los sistemas centralizados trajeron nuevos retos como múltiples usuarios trabajando en un mismo archivo al mismo tiempo.

Si se cortaba la conexión a Internet o al servidor no se podía seguir trabajando.



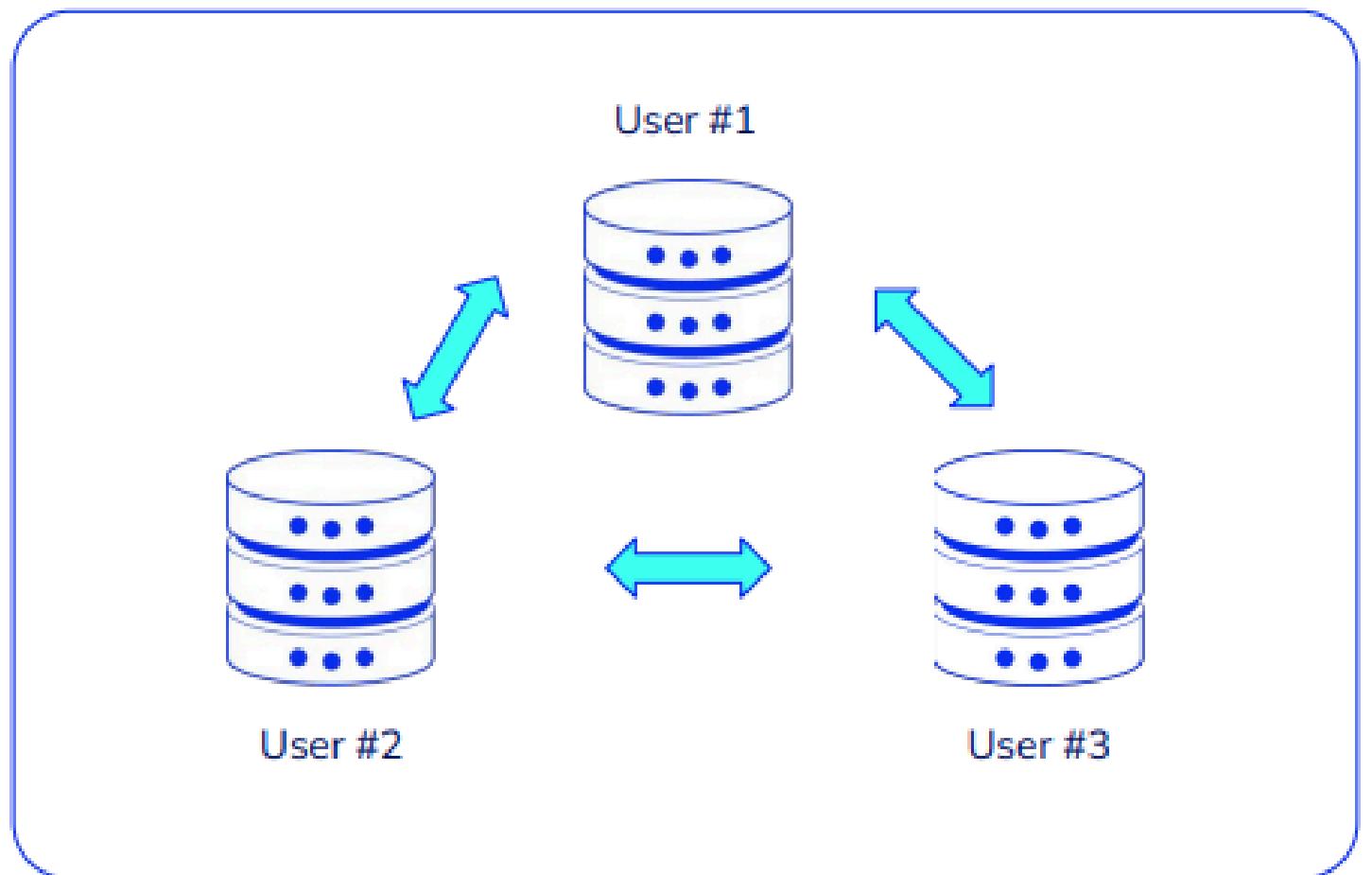
Centralized version control

# Tipos de Repositorios

## REPOSITORIOS DISTRIBUIDOS

Cada desarrollador **tiene una copia local de todo el proyecto.**

De esta manera se construyó una red distribuida de repositorios, en la que cada desarrollador podía trabajar de manera aislada pero teniendo un mecanismo de resolución de conflictos mucho más elegante que en su versión anterior

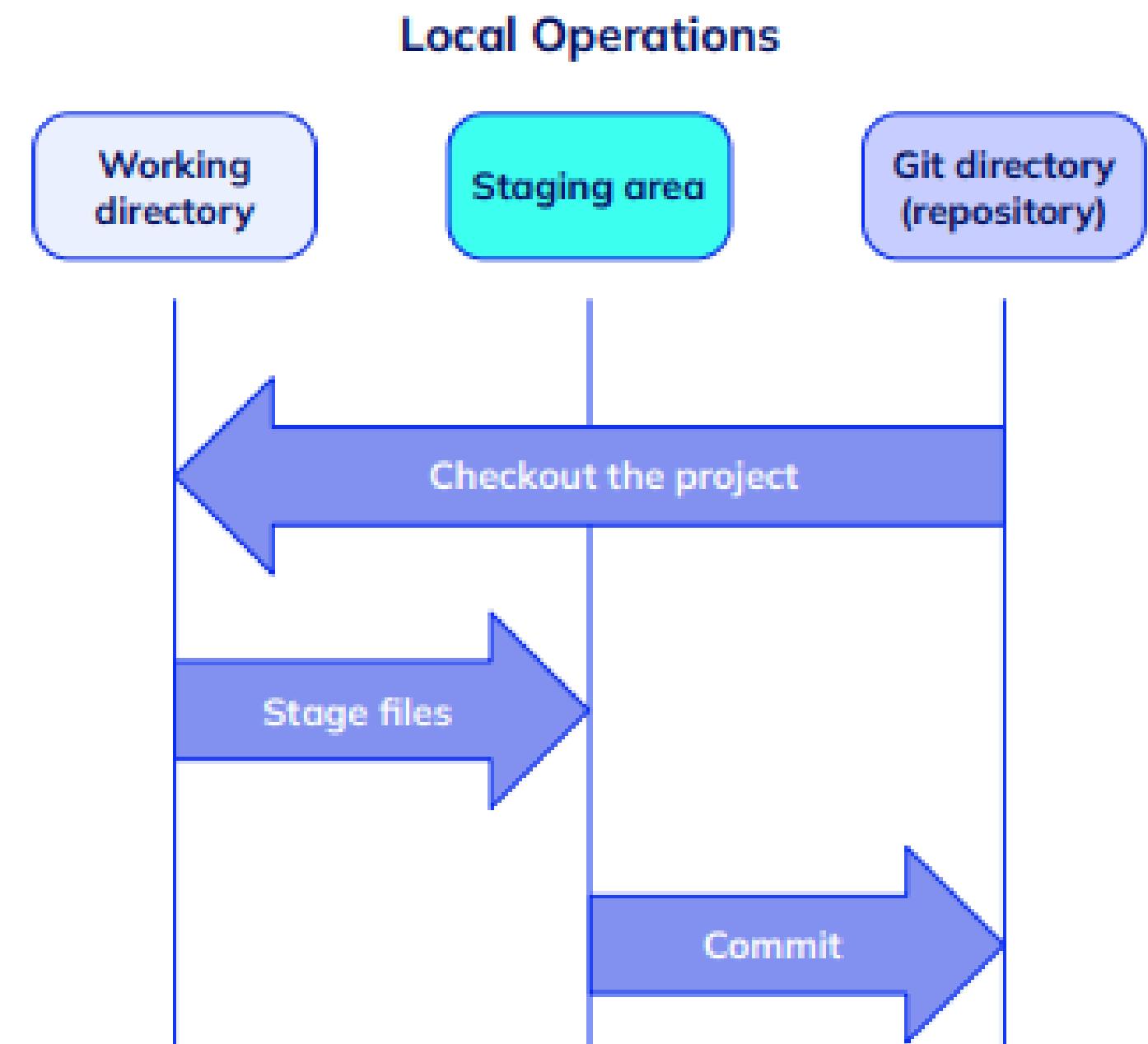


Distributed version control

# Estados de un repositorio Git

Git tiene tres estados principales:

- **Staged:** significa que se ha marcado un archivo modificado en su versión actual para pasar a su próximo snapshot de confirmación
- **Modified:** significa que se ha cambiado el archivo pero aún no se ha confirmado en su base de datos
- **Committed:** significa que los datos se almacenan de forma segura en su base de datos



# Iniciar un proyecto

# Inicializar un repositorio

Se puede obtener un repositorio git de dos maneras:

- **Convertir un directorio local** que, actualmente, no esta bajo ningún control de versión a un repositorio de Git
- **Clonar un repositorio** de Git existente desde algún otro lugar

# Convertir un directorio local en repositorio

En el caso de tener un directorio de proyecto, que actualmente no esta bajo ningún control de versión, y **desea comenzar a controlarlo con Git, debe seguir los siguientes pasos:**

1. Dirigirse al directorio que se quiere convertir en repositorio
2. Una vez en el repositorio, ejecutar **git init**
3. Esto va a crear un nuevo subdirectorio llamado **.git** que contiene todos los archivos necesarios para el repositorio - el esqueleto de un repositorio git

# Convertir un directorio local en repositorio

```
julia@DESKTOP-6D0HCII MINGW64 /d/Organizador/UADE/Segundo cuatrimestre/Programación 1/Curso Monserrat/Prueba Rep  
ositorio  
● $ git init  
Initialized empty Git repository in D:/Organizador/UADE/Segundo cuatrimestre/Programación 1/Curso Monserrat/Prueba Repositorio/.git/
```

Luego con el comando **-ls -al** se listan los directorios creados:

- **ls:** lista los archivos
- **-a:** muestra los archivos ocultos
- **-l:** muestra la lista en formato detallado

```
julia@DESKTOP-6D0HCII MINGW64 /d/Organizador/UADE/Segundo cuatrimestre/Programación 1/Curso Monserrat/Prueba Repositorio (master)  
$ ls -al  
total 0  
● drwxr-xr-x 1 julia 197609 0 ago. 11 10:01 ./  
drwxr-xr-x 1 julia 197609 0 ago. 11 10:01 ../  
drwxr-xr-x 1 julia 197609 0 ago. 11 10:01 .git/
```

# git status

Comando que muestra el estado actual de tu repositorio.

Proporciona la siguiente información:

- Indica la rama en que estas trabajando
- **Cambios no preparados:** muestra los archivos que han sido modificados en el directorio local, pero aún no han sido añadidos a la “staging area”
- **Cambios preparados:** muestra los archivos que han sido añadidos a la “**staging area**“ y estan listos para ser commitados
- **Archivos no rastreados:** lista de los archivos nuevos que git aun no esta siguiendo

# git status

Luego de inicializar el repositorio si ejecutamos **git status** se muestra lo siguiente:

```
● $ git status
On branch master
No commits yet
nothing to commit (create/copy files and use "git add" to track)
```

Nos indica que estamos trabajando sobre la **rama master**

**“No commits yet”** significa que en este repositorio aún no se ha hecho ningún commit

Por último sugiere, que para comenzar a rastrear archivos, primero debemos crear los archivos en el proyecto y luego usar el comando **git add** para añadir esos archivos al **staging area**

# git status

Creamos un nuevo archivo dentro del proyecto, por ejemplo “**info.txt**” y luego volvemos a ejecutar **git status**

```
● $ git status
○ On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    info.txt

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

El archivo **info.txt** no tiene seguimiento ya que se encuentra debajo de **Untracked files**

**Untracked** significa que git ve un archivo que no tenía en la versión anterior del proyecto (**snapshot**)

**GIT NO LO VA A INCLUIR HASTA QUE NO SE INDIQUEMOS EXPLICITAMENTE**

# git add

Para incluir y darle seguimiento a un archivo se puede usar el comando:

```
julia@DESKTOP-6D0HCII MINGW64 /d/Organizador/UADE/Segundo cuatrimestre/Programación 1/Curso Monserrat/Prueba  
Repositorio (master)  
● $ git add info.txt
```

Si volvemos a ejecutar **git status** veremos que nuestro archivo esta bajo seguimiento y listo para ser confirmado, es decir, formar parte de la nueva versión del repositorio (**commit**).

```
julia@DESKTOP-6D0HCII MINGW64 /d/Organizador/UADE/Segundo cuatrimestre/Programación 1/Curso Monserrat/Prueba  
Repositorio (master)  
● $ git status  
On branch master  
  
No commits yet  
  
Changes to be committed:  
(use "git rm --cached <file>..." to unstage)  
  new file:   info.txt
```

# git add

El comando **git add** puede aceptar el nombre de:

- archivo
- directorio



Si es un directorio, el comando agrega todos los archivos en ese directorio

Si tengo más de un archivo que quiero agregar, puedo ejecutar **git add .**

# git add

Si luego de hacer el **git add info.txt** modiflico el archivo y vuelvo a realizar un **git status** se va a visualizar la siguiente información:

```
$ git status
On branch master
No commits yet

Changes to be committed:
(use "git rm --cached <file>..." to unstage)
  new file:  archivo1.txt
  new file:  archivo2.txt
  new file:  info.txt

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:  info.txt
```

El archivo **info.txt** aparece en la sección **Changes to be committed** y dentro de **Changes not staged for commit**.

Si hacemos un commit en este momento los últimos cambios sobre el archivo no se van a agregar.

# git commit

Ahora que el archivo se encuentra en el área de preparación (**stage area**) estamos listos para **confirmar los cambios**, es decir realizar un **commit**.

Podemos ejecutarlo con:  
**git commit**

Al hacer esto se visualizará en el editor elegido el siguiente mensaje:

```
Prueba Repositorio > .git > COMMIT_EDITMSG
1 |
2 # Please enter the commit message for your changes. Lines starting
3 # with '#' will be ignored, and an empty message aborts the commit.
4 #
5 # On branch master
6 #
7 # Initial commit
8 #
9 # Changes to be committed:
10 #   new file: archivo1.txt
11 #   new file: archivo2.txt
12 #   new file: info.txt
13 #
14
```

# git commit

Coloco el comentario en la primer linea,  
se debe colocar si #, porque las lineas  
que comienzan con # son ignoradas.

Luego al cerrar el archivo se procederá a  
realizar el commit

```
Prueba Repositorio > .git > ✏ COMMIT_EDITMSG
 1  Ultimo cambio
 2  # Please enter the commit message for your changes. Lines starting
 3  # with '#' will be ignored, and an empty message aborts the commit.
 4  #
 5  # On branch master
 6  # Changes to be committed:
 7  #   modified:   info.txt
 8  #
```

# git commit

Puede que falte configurar el editor por defecto, para eso ejecutamos:

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



# git commit

Otra alternativa es ejecutando el comando de la siguiente forma

**git commit -m “Comentario”**

Luego de la opción -m se debe agregar el comentario

```
julia@DESKTOP-6D0HCII MINGW64 /d/Organizador  
ositorio (master)  
● $ git commit -m "Ultimos cambios"  
[master 9ddd3b8] Ultimos cambios  
 1 file changed, 1 insertion(+)
```

La salida del comando nos informa sobre que rama lo ejecuto (**master**), el identificador del commit **9ddd3b8** y la cantidad de archivos modificados, y la estadística sobre las líneas que fueron insertadas o eliminadas

# MALA PRACTICA

**NO COLOCAR MENSAJES EN LOS COMMIT**

# BUENA PRACTICA

**COLOCAR COMENTARIOS EN LOS COMMIT QUE DESCRIBAN  
QUE SE AGREGA O MODIFICA**

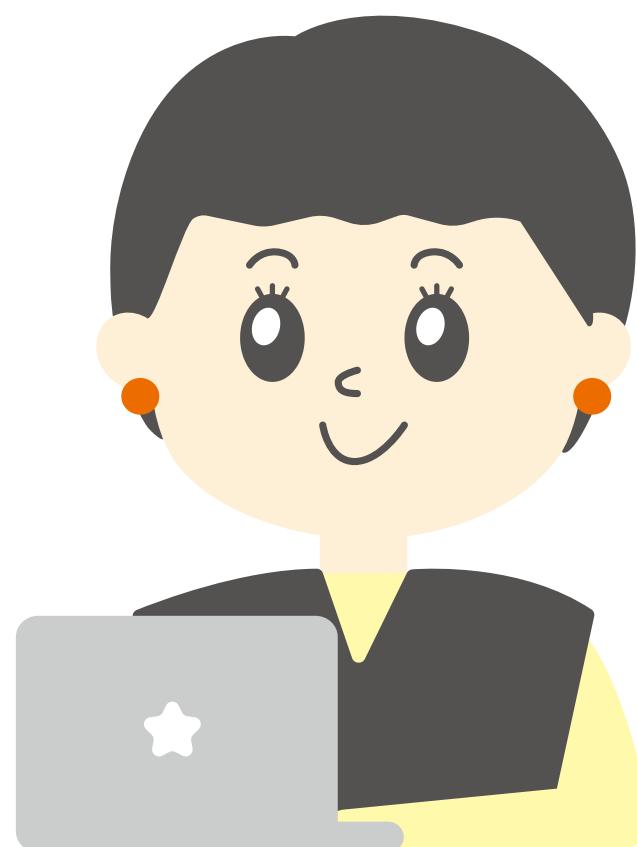
# Atajo

Si queremos realizar el **add** y el **commit** con un solo comando lo podemos hacer de la siguiente forma:

**git commit -a -m “Comentario”**

De esta manera se puede omitir el comando  
**git add**

**FUNCIONA SIEMPRE QUE LOS ARCHIVOS  
NO SEAN NUEVOS.  
ES DECIR HACER ADD DE  
MODIFICACIONES**



# git commit --amend

Con este comando podemos realizar lo siguiente:

- Modificar el mensaje del último commit
- Agregar nuevos archivos al commit anterior

```
PS D:\Organizador\UADE\Segundo cuatrimestre\Programación 1\Curso Monserrat\Clase 2\Prueba> git commit --amend -m "Agrego archivo"
● [master d02683b] Agrego archivo
  Date: Sun Aug 11 17:53:07 2024 -0300
  2 files changed, 1 insertion(+)
  create mode 100644 info.txt
  create mode 100644 otroarchivo.txt
```

# git log

Con el comando **git log** podemos ver hacia atrás todo lo que ha sucedido en el repositorio.

Muestra el más reciente primero.

```
PS D:\Organizador\UADE\Segundo cuatrimestre\Programación 1\Curso Monserrat\Clase 2\Prueba> git log
● commit 17a6854589eeebd9ae296c3cef4648d258c3c7a8 (HEAD -> master)
  Author: BEETS <beetsftb@gmail.com>
  Date:   Sun Aug 11 18:02:54 2024 -0300
```

Segundo commit

```
commit d02683bdf39c38f150db9bf9068bba2faebe71c7
  Author: BEETS <beetsftb@gmail.com>
  Date:   Sun Aug 11 17:53:07 2024 -0300
```

Agrego archivo

# git log

**git log --oneline** muestra la información más abreviada de cada commit

```
PS D:\Organizador\UADE\Segundo cuatrimestre\Programación 1\Curso Monserrat\Clase 2\Prueba> git log --oneline
17a6854 (HEAD -> master) Segundo commit
● d02683b Agrego archivo
```

**git log nombre\_archivo** para ver el log de un archivo en particular

# git show

Muestra los detalles completos de un solo commit, o si no se envían parámetros muestra la información del último commit

```
PS D:\Organizador\UADE\Segundo cuatrimestre\Programación 1\Curso Monserrat\Clase 2\Prueba> git show info.txt
● commit ce9f62ad948012a80b7e8e6cc689f70b434c112b (HEAD -> master)
Author: BEETS <beetsftb@gmail.com>
Date:   Sun Aug 11 18:25:45 2024 -0300

    ultimo commit

diff --git a/info.txt b/info.txt
index a5eedee..f778431 100644
--- a/info.txt
+++ b/info.txt
@@ -1,2 +1,4 @@
 Prueba prueba!
-prueba prueba
\ No newline at end of file
+prueba prueba
+
+taca cambie!
\ No newline at end of file
```

En **diff** muestra las diferencias de los cambios realizados en el commit

# git tag

Se utiliza para crear etiquetas (tags) que son referencia a puntos específicos en la historia del proyecto.

Se suelen utilizar para marcar versiones específicas del software, como lanzamientos (releases).

## git tag v1.0

Para ver las diferentes etiquetas podemos utilizar **git tag**

# Versiones

4 . 7 . 6

Primer numero **Major version** cambios mayores, que no es compatible con la version anterior. Por ejemplo modificar la firma de una funcion, eliminar una API, o agregar una nueva.

Segundo numero **Minor version** mejora en las funcionalidades pero no alteran las funcionalidades existentes.

Tercer numero **Patches** se incrementa cada vez que implementamos un hotfix

# Práctica en clase

1. Crear un directorio en la máquina local e iniciar un repositorio git en el
2. Configurar git con los datos de usuario de manera global para el sistema operativo
3. Crear un archivo en ese directorio y darle seguimiento en GIT
4. Confirmar el archivo dentro de nuestro historial de commits.
5. Crear un nuevo archivo en el mismo directorio, editar el archivo que se creo en el punto anterior con cualquier contenido.
6. Verificar el estado actual del repositorio para tener un panorama general de los pasos siguientes que podemos realizar sobre el.
7. Incorporar en el stage area los cambios del punto 5 y confirmar los cambios

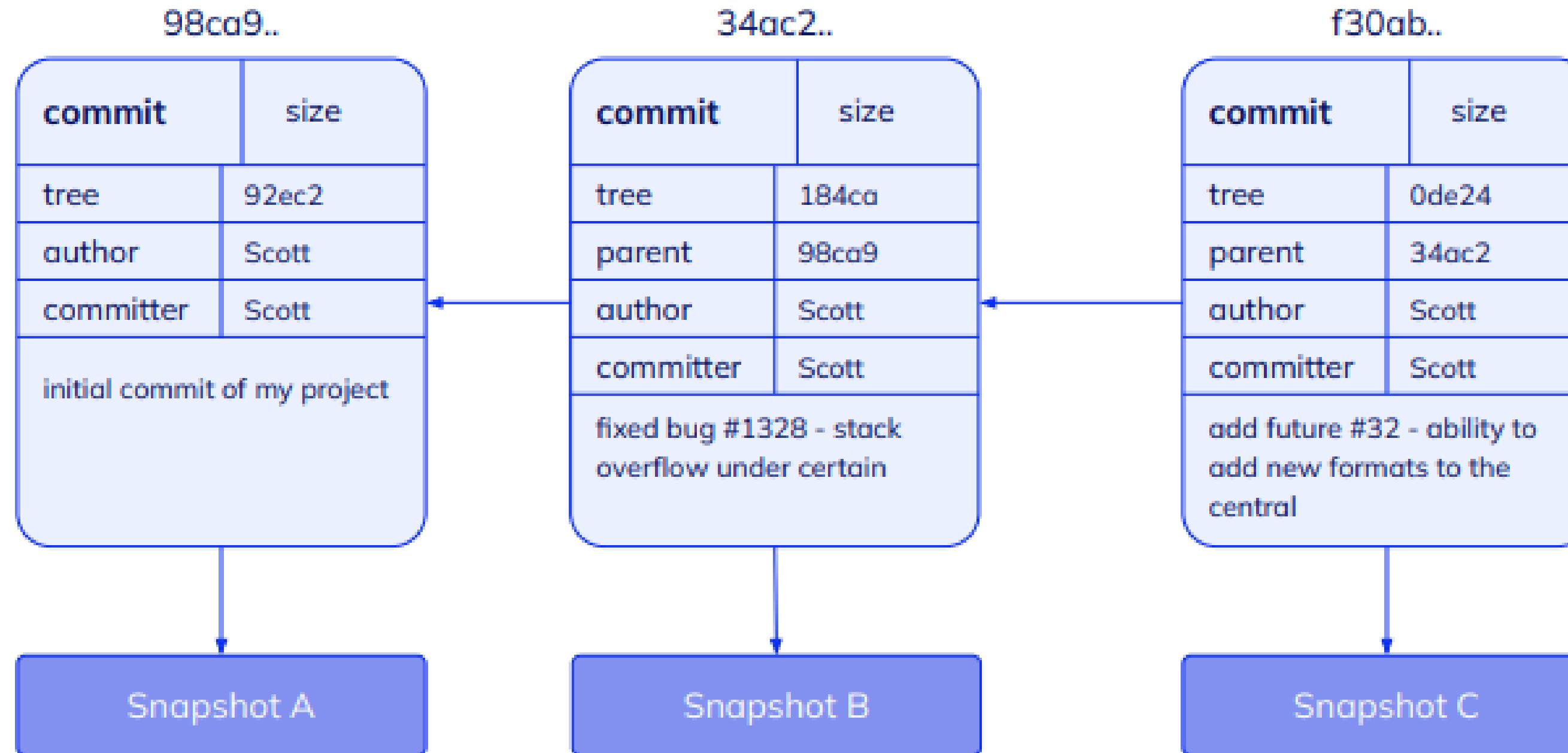


# Crear ramas

# Ramas o Branches

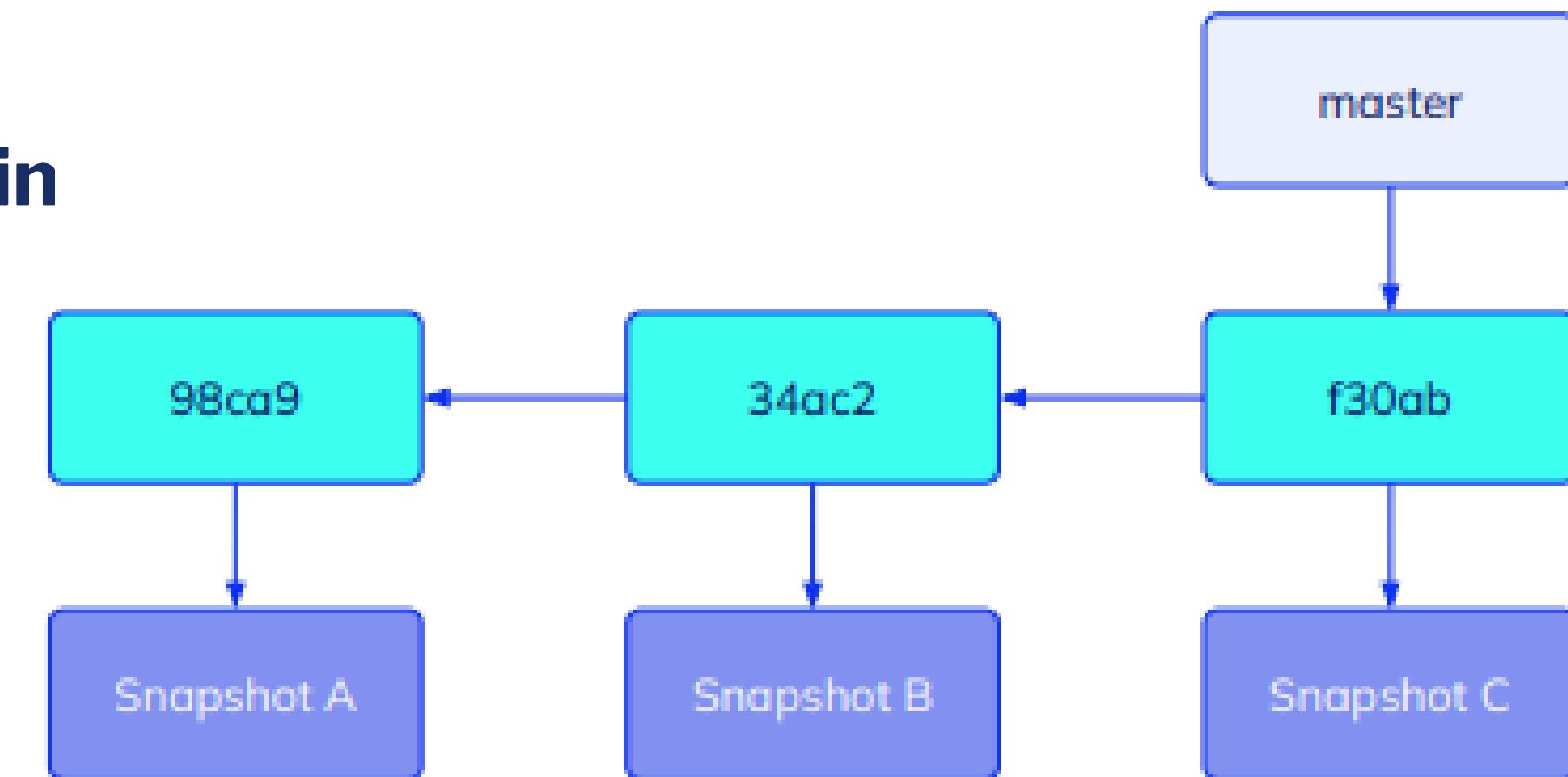
- **Git** almacena como una serie de snapshots (copias puntuales de los archivos completos, tal y como se encuentran en ese momento).
- En cada confirmación de cambios (commit), Git almacena un punto de control que conserva:
  - Un apuntador a la copia puntual de los contenidos preparados (staged).
  - Metadatos con el autor y el mensaje explicativo.
  - Uno o varios apuntadores a las confirmaciones (commit) que sean padres directos de esta.

# Ramas o Branches



# Branch

- Un **branch** o una **rama** es un **apuntador móvil** que apunta a una de esas confirmaciones o commit.
- **La rama por defecto de Git es master o main**
- En cada commit de cambios que hagamos, la rama irá avanzando automáticamente.
- **La rama master apuntará siempre a la última confirmación realizada**

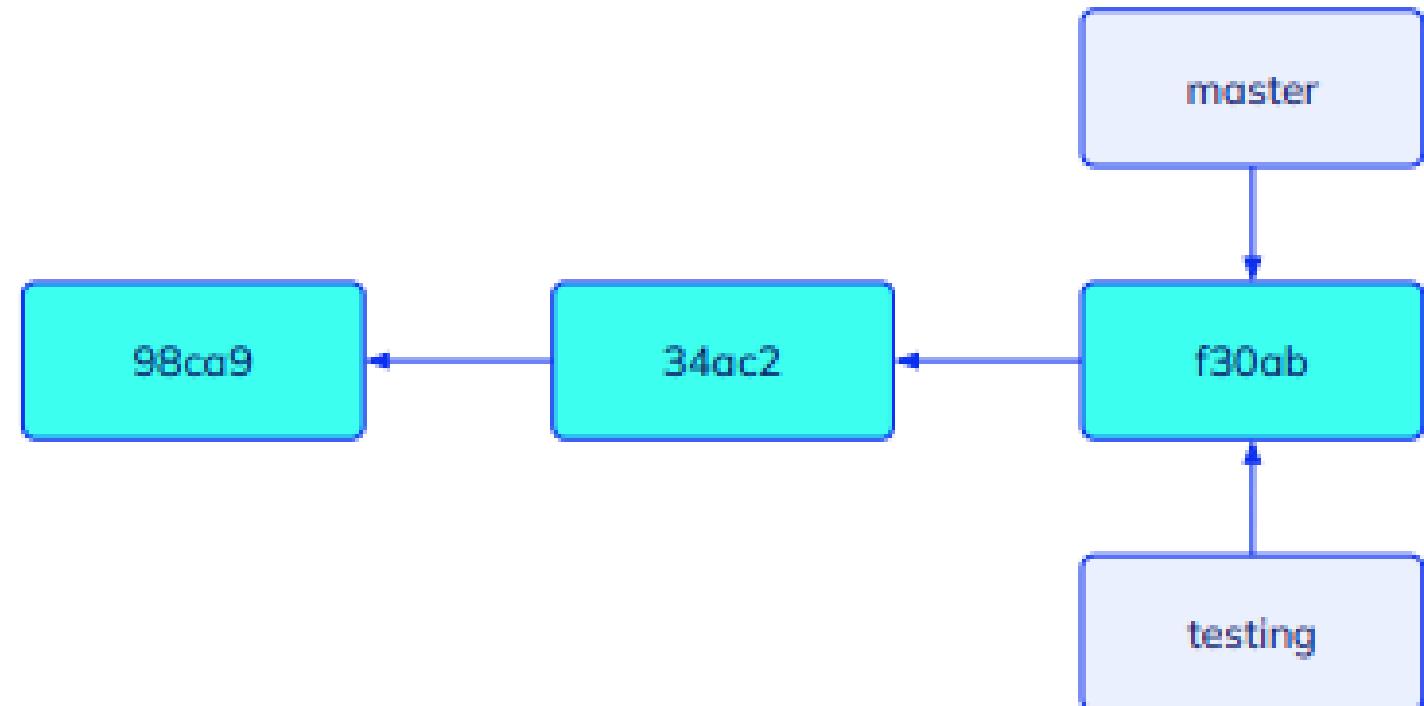


# Branch

- Cuando se crea una rama o un branch, **se crea un nuevo apuntador**
- Es conveniente pensarla como una “**bifurcación**” o “**copia del código**” en la que podemos trabajar sin afectar otras ramas.

- Para crear una nueva rama se utiliza el comando

**git branch nombre\_branch**



**Esto creará un nuevo apuntador a la misma confirmación en la que estamos. ES IMPORTANTE DESTACAR QUE CREA LA RAMA, PERO NO SALTA A LA MISMA**

# Branch

```
PS D:\Organizador\UADE\Segundo cuatrimestre\Programación 1\Curso Monserrat\Clase 2\Prueba> git branch develop
```

Este comando crea una nueva rama, a partir de **master** (porque estamos en master), y como nombre le pone **develop**

# ¿Como sabe Git en que rama estas en este momento?

- Mediante un **apuntador especial denominado HEAD**
- **HEAD** es un apuntador a la rama local en la que estamos en un momento.

# git checkout

- Para **saltar de una rama a otra**, utilizamos el comando **git checkout**
- **Git checkout** mueve el apuntador **HEAD** a la rama que deseamos dirigirnos

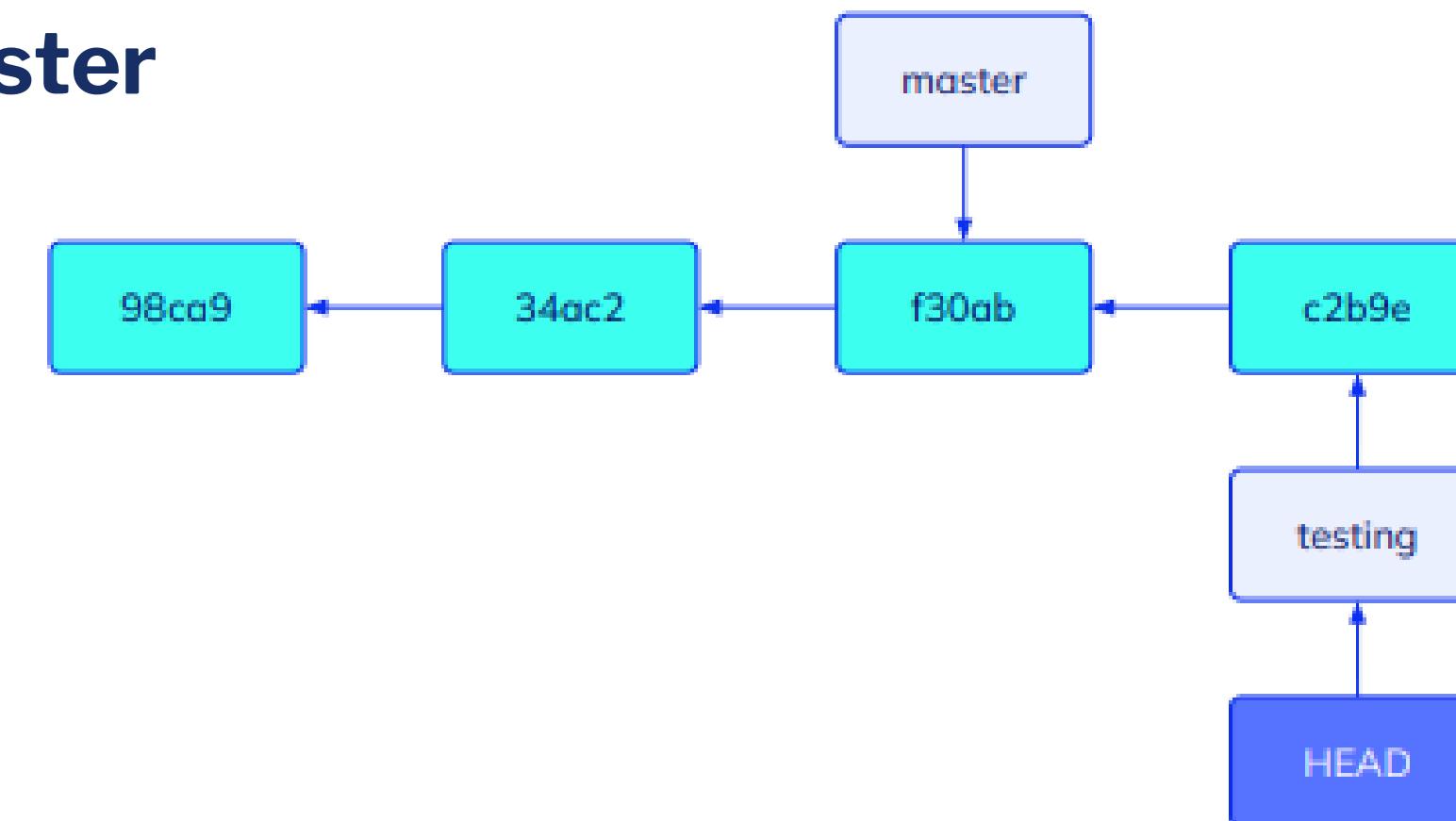
**git checkout nombre\_rama**

```
PS D:\Organizador\UADE\Segundo cuatrimestre\Programación 1\Curso Monserrat\Clase 2\Prueba> git checkout develop
Switched to branch 'develop'
```

En este momento estamos dentro de la rama **develop**. Es decir todo lo que se desarrollará no afectará a **master** u otra rama.

# git checkout

- Si creamos un archivo en la rama **develop** , subimos los cambios.
- Luego nos volvemos a la rama **master**, vamos a ver que el último archivo no va a estar.
- Basicamente lo que va a suceder es que la rama, en la imagen de abajo **“testing”** va a estar más delante de **master**



# git checkout

- **git checkout -b <nombre\_rama>** permite crear la rama y además saltar a la nueva rama de forma inmediata

# Integrar cambios

- Una vez que el código de un branch ya esta estable, en un ambiente real pasa las pruebas **vamos a tener que integrarlos con la rama master**
- Para esto podemos utilizar el comando **git merge** desde el destino donde queremos traer los cambios de otra rama.
- Por ejemplo si a **master** queremos traer los cambios de **Testing**, desde la rama master vamos a ejecutar **git merge Testing**

# Integrar cambios

```
julia@Julia MINGW64 ~/OneDrive/Desktop/Organizador/UADE/Programacion 1/Clase 2/Prueba repositorio (master)
● $ git merge Testing
Updating 24e00c4..c576f74
Fast-forward
  cambios.txt | 0
  1 file changed, 0 insertions(+), 0 deletions(-)
  create mode 100644 cambios.txt
```

- Si lo que intentamos fusionar es una rama que esta mas adelantada, va a realizarlo sin problema y se visualizara el mensaje “Fast-forward”

# Listar branch

- Para listar las ramas disponibles o los branch que tenemos se puede ejecutar el comando
  - **git branch --list**

```
julia@Julia MINGW64 ~/OneDrive/Desktop/Organizador/UADE/Programacion 1/Clase 2/Prueba repositorio (master)
● $ git branch --list
  Testing
* master
```

- En color verde se visualiza el branch en el que nos encontramos

# Resolución de conflictos

- Cuando modifique en diferentes ramas los mismos archivos. Al hacer **git merge** se visualizará que existen conflictos.
- Se visualizará el siguiente error:

```
+ file changed, + insertion(+)

julia@Julia MINGW64 ~/OneDrive/Desktop/Organizador/UADE/Programacion 1/Clase 2/Prueba repositorio (master)
④ $ git merge Testing
Auto-merging archivo1.txt
CONFLICT (content): Merge conflict in archivo1.txt
Automatic merge failed; fix conflicts and then commit the result.
```

- Si ejecutamos el comando **git status** nos listará bajo **unmerged** todo aquello conflictivo.

```
julia@Julia MINGW64 ~/OneDrive/Desktop/Organizador/UADE/Programacion 1/Clase 2/Prueba repositorio (master|MERGING)
● $ git status
On branch master
You have unmerged paths.
  (fix conflicts and run "git commit")
  (use "git merge --abort" to abort the merge)

Unmerged paths:
  (use "git add <file>..." to mark resolution)
    both modified:  archivo1.txt

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

# Resolución de conflictos

- Además git añade unos marcadores especiales para indicar las diferencias entre ambos ramas.

```
git diff --staged --name-only
Accept Current Change | Accept Incoming Change | Accept Both Changes | Compare Changes
1 <<<<< HEAD (Current Change)
2 Tambien la modifique en master
3 =====
4 Modificacion 1 en testing
5 >>>>> Testing (Incoming Change)
6
```

- Para resolverlo, se debe elegir manualmente con que versión nos queremos quedar y luego realizar un git commit.
- Es importante destacar que si elegimos quedarnos con la versión de master, esos cambios no van a quedar en el branch.

# Eliminar un branch

- Para eliminar un branch se tiene que utilizar el comando **git branch -d <nombre\_rama>**

```
julia@Julia MINGW64 ~/OneDrive/Desktop/Organizador/UADE/Programacion 1/Clase 2/Prueba repositorio (master)
● $ git branch -d "otro_branch"
Deleted branch otro_branch (was 6779317).
```

# Práctica en clase

1. Crear una nueva rama llamada “testing” y mostrar las ramas del repositorio
2. Crear una nueva rama llamada navbar y mostrar las ramas del repositorio
3. Crear una nueva rama llamada accounting y mostrar las ramas del repositorio

Todas las ramas nacen de master

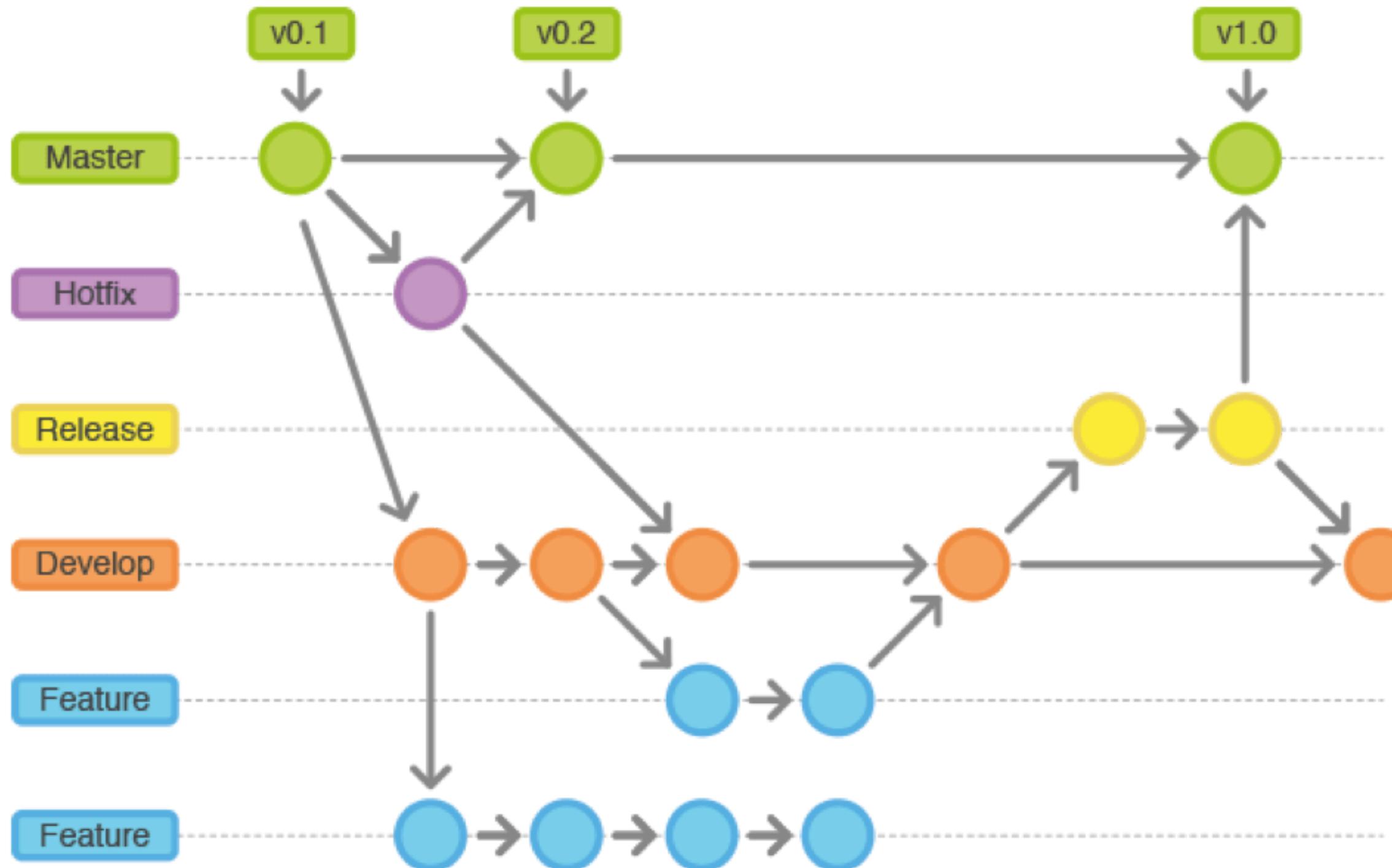


# GITFLOW

# GitFlow

- Es un flujo de trabajo de Git que nos permite **gestionar ramas e implementaciones en un proyecto**
- Fue popularizado por Vincent Driessen en 2010
- Hoy en día, la mayoría de los equipos de desarrollo la utilizan como una estrategia estructurada para gestionar versiones y colaboraciones en proyectos

# GitFlow



# GitFlow - Ramas Principales

- **master** es la rama principal y estable del proyecto. Contiene el código que esta en producción, es decir el código que esta siendo utilizado por los usuarios finales. **Solo se realiza merge a master cuando se lanza una nueva versión estable**
- **develop** es la rama principal para el desarrollo. Aqui es donde se integran las nuevas caracteristicas y correcciones de errores antes de lanzar la versión.

# GitFlow - Ramas de Soporte

- **branches** se crea una nueva rama por cada **feature** o nuevas características que se están desarrollando. Una vez que está completo el desarrollo de la nueva característica, se fusiona con **develop**. Generalmente estas ramas se nombran como **feature/nombre-característica**
- **releases** se crean a partir de **develop**. Y son ramas donde se hacen pruebas de todas las nuevas características antes de ser lanzadas. Generalmente se llaman **releases/x.x.x** donde x.x.x es el número de versión

# GitFlow - Ramas de Soporte

- **hotfix** es cuando ocurre algún error en Producción y se debe arreglar rápidamente. Esta rama se crea a partir de master, una vez resuelto se deben llevar los cambios tambien a develop y releases (si es que existiesen)

# GITHUB

# GitHub

- Es una plataforma de desarrollo colaborativo que permite a los desarrolladores **alojar, gestionar, y compartir proyectos de software.**
- Es popular para proyectos que utilizan Git
- GitHub se utiliza para gestionar el historial de versiones de un proyecto y facilitar la colaboración entre múltiples desarrolladores.

# GitHub - Crear una cuenta gratuita

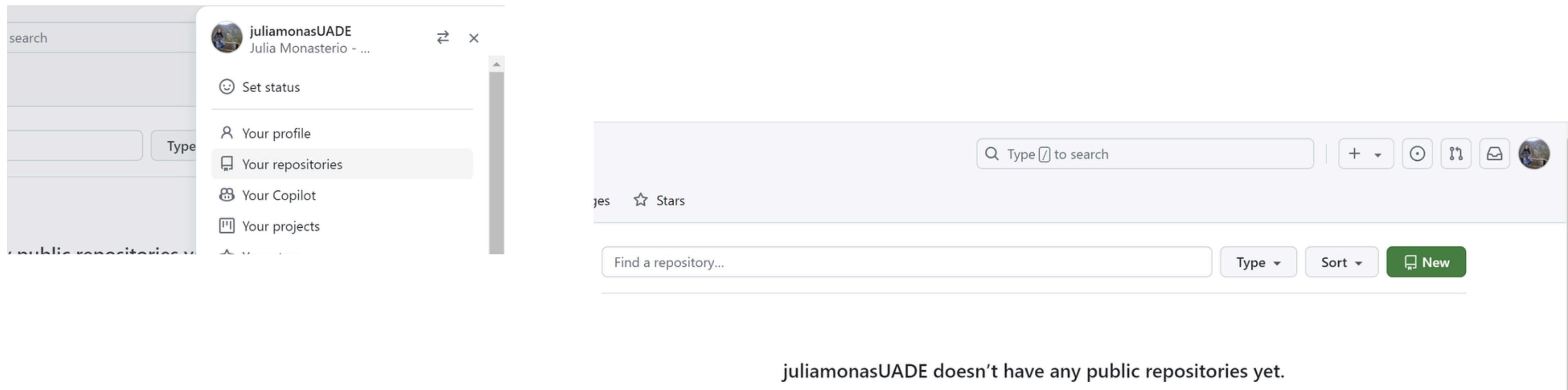
- Ingresar a **github.com** y presionar en **Sign up** para crear una cuenta
- Se deberá ingresar un correo electrónico y luego una password

# GitHub - Crear un repositorio remoto

- Un repositorio remoto contiene versiones de nuestros proyectos que estan siendo almacenados en internet.
- Pueden existir repositorios remotos que son solo de **lectura** y otros de **lectura/escritura**

# GitHub - Crear un repositorio remoto

- 1) Hacemos click en nuestro Perfil y seleccionamos la opción **Your Repositories**
- 2) Presionamos **New**



# GitHub - Crear un repositorio remoto

Esto nos va a llevar a un formulario como el siguiente:

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Required fields are marked with an asterisk (\*).

Owner \* Repository name \*

juliamonasUADE /

Great repository names are short and memorable. Need inspiration? How about super-duper-umbrella ?

Description (optional)

 Public Anyone on the internet can see this repository. You choose who can commit.  
  Private You choose who can see and commit to this repository.

Initialize this repository with:

Add a README file This is where you can write a long description for your project. [Learn more about READMEs.](#)

Add .gitignore

.gitignore template: None

Choose which files not to track from a list of templates. [Learn more about ignoring files.](#)

Choose a license

License: None

A license tells others what they can and can't do with your code. [Learn more about licenses.](#)

ⓘ You are creating a public repository in your personal account.

Vamos a seleccionar:

- Público
- Colocarle un nombre
- Una descripción opcional
- Marcar Add a Readme file
- **Presionar Create Repository**

# GitHub - Crear un repositorio remoto

The screenshot shows a GitHub repository page for 'Programacion1Belgrano'. The repository is public and has one branch ('main') and one commit ('Initial commit' by 'juliamonasUADE'). The README file contains the text 'Programacion1Belgrano'. The repository has 1 watching and 0 forks.

**Code** | **Issues** | **Pull requests** | **Actions** | **Projects** | **Wiki** | **Security** | **Insights** | **Settings**

**Programacion1Belgrano** Public

**Code** | **Pin** | **Unwatch** 1 | **Fork** 0 | **Star** 0

**main** | 1 Branch | 0 Tags | Go to file | Add file | **Code**

juliamonasUADE Initial commit · 1 minute ago · 1 Commit

README.md · Initial commit · 1 minute ago

README

**About**

No description, website, or topics provided.

Readme | Activity | 0 stars | 1 watching | 0 forks

**Releases**

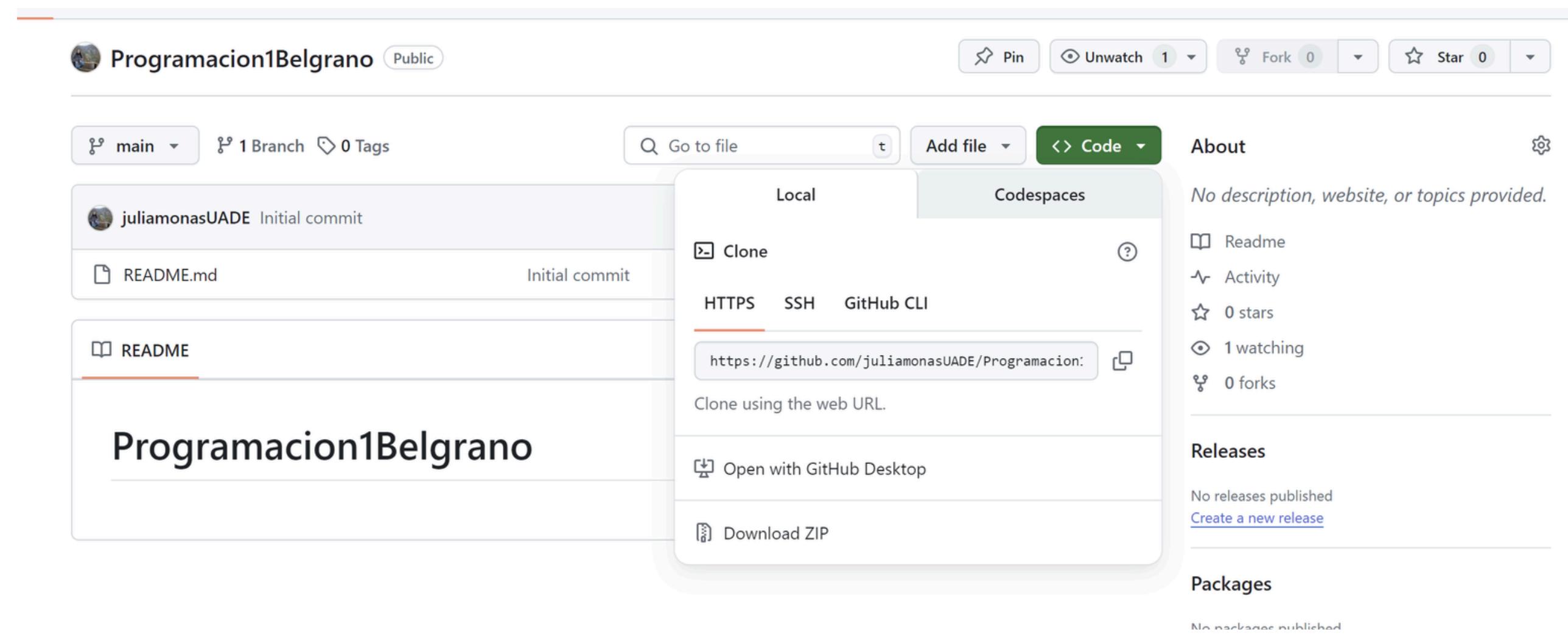
No releases published | Create a new release

**Packages**

No packages published | Publish your first package

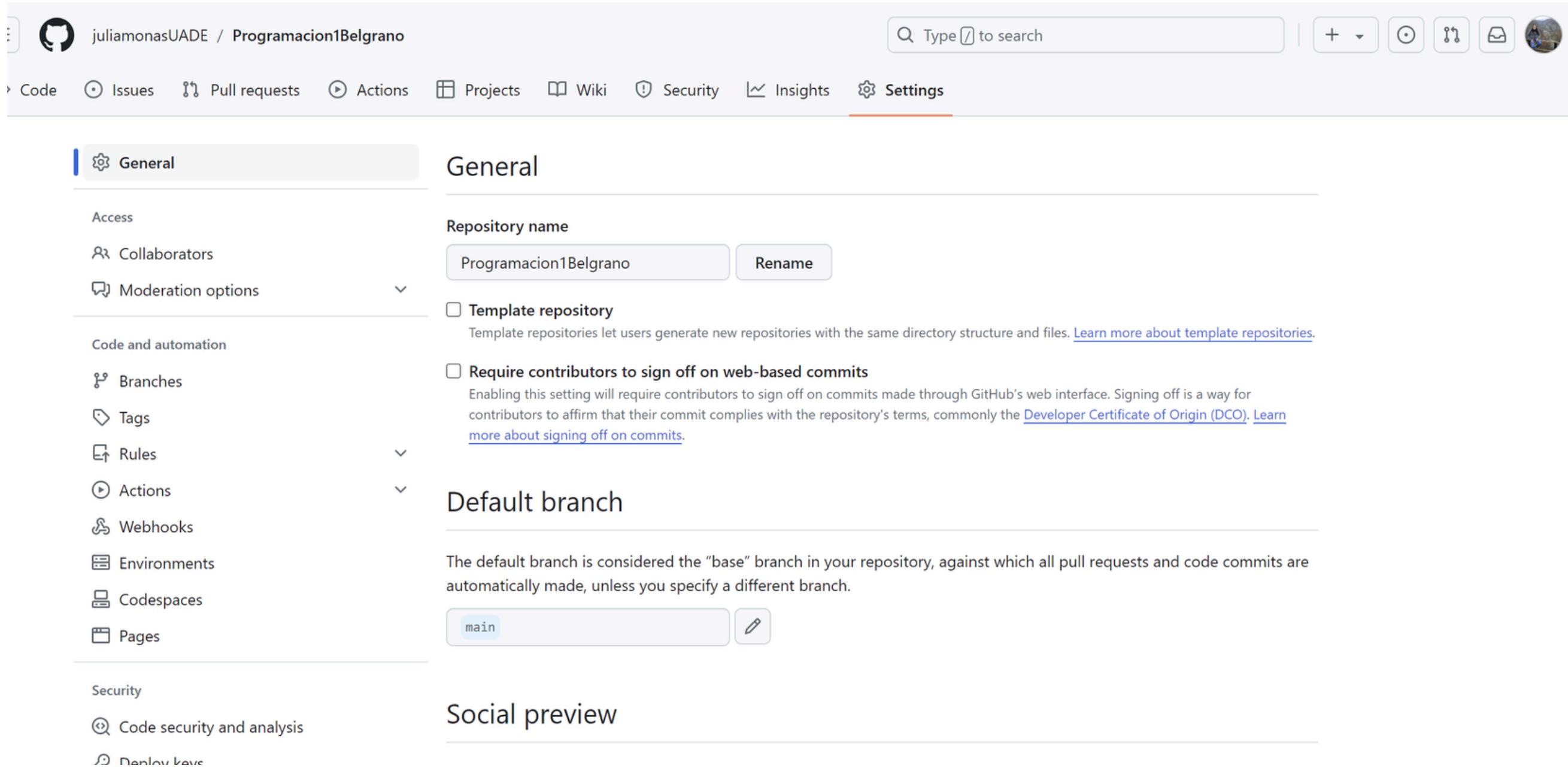
# GitHub - Crear un repositorio remoto

Cada proyecto en GitHub es accesible a través de HTTPS con la URL o a través de SSH



# GitHub - Configuraci[on

Una vez dentro del repositorio, se dirigen a Settings y ahí van a poder modificar la configuración del remoto



The screenshot shows the GitHub repository settings interface for the repository "juliamonasUADE / Programacion1Belgrano". The "General" tab is selected. On the left, there is a sidebar with various configuration sections: Access, Collaborators, Moderation options, Code and automation (Branches, Tags, Rules, Actions, Webhooks, Environments, Codespaces, Pages), and Security (Code security and analysis, Deploy keys). The main content area displays the "General" settings. It includes fields for "Repository name" (Programacion1Belgrano) with a "Rename" button, checkboxes for "Template repository" and "Require contributors to sign off on web-based commits", and a section for the "Default branch" set to "main". At the bottom, there is a "Social preview" section.

# GitHub - Configuraci[on

Dentro de Danger Zone van a poder modificar la visibilidad del repositorio, como eliminar el repositorio

## Danger Zone

The screenshot shows the GitHub Danger Zone interface with five options:

- Change repository visibility**: This repository is currently public. **Change visibility** button.
- Disable branch protection rules**: Disable branch protection rules enforcement and APIs. **Disable branch protection rules** button.
- Transfer ownership**: Transfer this repository to another user or to an organization where you have the ability to create repositories. **Transfer** button.
- Archive this repository**: Mark this repository as archived and read-only. **Archive this repository** button.
- Delete this repository**: Once you delete a repository, there is no going back. Please be certain. **Delete this repository** button.

# GitHub - Crear un repositorio remoto

Para agregar un nuevo remoto a nuestro repositorio Git, debemos ejecutar el siguiente comando:

**git remote add <name> [https://github.com/<usuario>/<nombre\\_del\\_proyecto>](https://github.com/<usuario>/<nombre_del_proyecto>)**

En name tengo que colocar el nombre con el que quiero conocer al repositorio remoto, **comunmente se coloca origin**

```
julia@Julia MINGW64 ~/OneDrive/Desktop/Organizador/UADE/Programacion 1/Clase 2/Prueba repositorio (master)
● $ git remote add origin https://github.com/juliamonasUADE/Programacion1Belgrano.git
```

# GitHub - Subir nuestros cambios

Para poder subir al repositorio remoto los cambios locales, debemos utilizar el comando:

**git push <remote> <branch>**

Este comando va a funcionar solo si:

- Se cuenta con permiso de escritura en ese cliente remoto
- Nadie subio cambios mientras realizabamos e intentabamos subir nuestros cambios

Si alguien mas hizo un **push** con anterioridad, la operacion sera rechazada.

Primero se deben incorporar los cambios nuevos subidos por los demás antes de poder hacer un push

# GitHub - Subir nuestros cambios

```
julia@Julia MINGW64 ~/OneDrive/Desktop/Organizador/UADE/Programacion 1/Clase 2/Prueba repositorio (master)
● $ git push origin master
info: please complete authentication in your browser...
Enumerating objects: 14, done.
Counting objects: 100% (14/14), done.
Delta compression using up to 20 threads
Compressing objects: 100% (10/10), done.
Writing objects: 100% (14/14), 1.17 KiB | 1.17 MiB/s, done.
Total 14 (delta 4), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (4/4), done.
remote:
remote: Create a pull request for 'master' on GitHub by visiting:
remote:     https://github.com/juliamonasUADE/Programacion1Belgrano/pull/new/master
remote:
To https://github.com/juliamonasUADE/Programacion1Belgrano.git
 * [new branch]      master -> master
```

The screenshot shows a GitHub repository page for the 'master' branch. At the top, it displays the branch name 'master', 2 branches, 0 tags, and a link to 'Go to file'. Below this, a message states 'This branch is 6 commits ahead of, 1 commit behind main.' On the right, there's a 'Contribute' button. The main area lists recent commits:

Author	Commit Message	Date	Commits
beetsftb	Merge branch 'Testing'	59e088b · 12 hours ago	6 Commits
	archivo1.txt	12 hours ago	Cambios en master
	cambios.txt	12 hours ago	Incorpore nuevo archivo en testing
	otro_branch.txt	12 hours ago	Otro branch

# GitHub - Subir nuestros cambios

Para poder obtener los cambios de un repositorio remoto se utiliza el comando **git pull**

**git pull** descargara todos los cambios nuevos que no tenemos en nuestro repositorio local desde el repositorio remoto.

# GitHub - git Clone

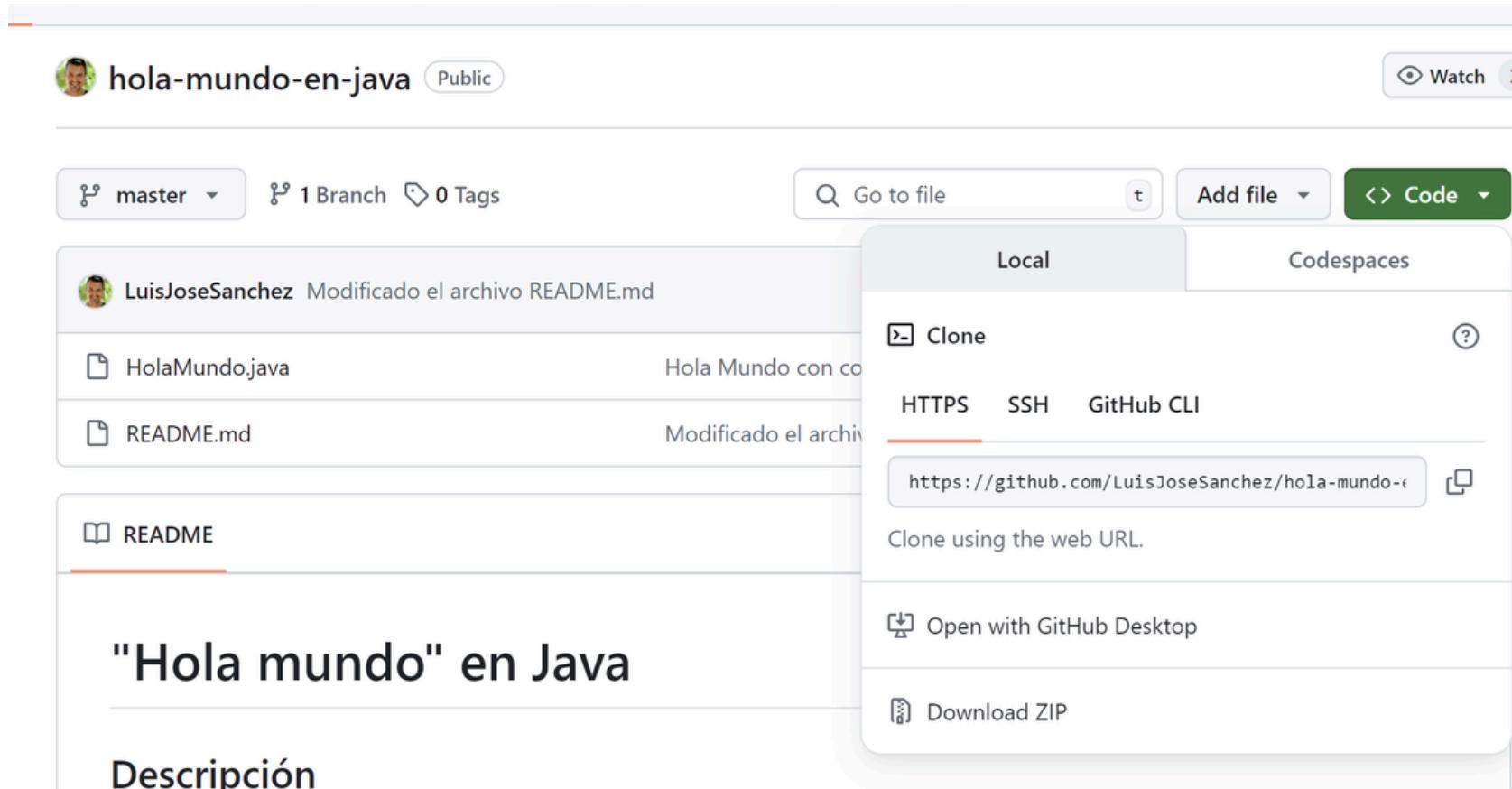
Cuando queremos colaborar en un proyecto que ya esta subido, y que tiene código desarrollado, no es necesario crear un repositorio vacío sino debemos obtener una copia del repositorio remoto.

Por ejemplo tomo un repositorio remoto que tiene un hola mundo en java:

<https://github.com/LuisJoseSanchez/hola-mundo-en-java>

# GitHub - git Clone

1) Ingreso al repositorio remoto, y copio la url



2) Luego me posiciono en mi local donde quiero que se clone el repositorio y ejecuto **git clone <url>**

# GitHub - git Clone

Este comando va a realizar lo siguiente:

- Crea un directorio con el mismo nombre que el repositorio del cliente
- Inicializa un directorio .git dentro de el
- Trae toda la información de ese repositorio, muestra su ultima versión

Si deseo colocarle un nombre diferente al que tiene en el remoto, puedo ejecutar:

**git clone <url> <nombre\_directorio\_nuevo>**

# GitHub - git Clone

Una vez que se clona un repositorio>

- Git crea **punteros** que le dan seguimiento a cada **branch** remoto dentro del repositorio local clonado (los branches remotos se pueden ver con **git branch -r**)

# GitHub - git Clone - SSH

SSH es un protocolo de red seguro que se utiliza para conectar de manera segura a dos dispositivos.

SSH garantiza que la conexión entre el cliente (generalmente nuestra PC) y el servidor (remoto) esté protegida contra accesos no autorizados.

# GitHub - git Clone - SSH

1) Se debe generar una clave SSH a la cuenta github, para eso tenemos que ejecutar en la terminal el siguiente comando:

**ssh-keygen -t rsa -b 4096 -C  
“marmonasterio@uade.edu.ar”**

Esto generará una clave privada y una clave pública

```
julia@Julia MINGW64 ~/OneDrive/Desktop/Organizador/UADE/Programacion 1/Clase 2/Prueba repositorio (master)
$ ssh-keygen -t rsa -b 4096 -C "marmonasterio@uade.edu.ar"
Generating public/private rsa key pair.
Enter file in which to save the key (/c/Users/julia/.ssh/id_rsa):
Created directory '/c/Users/julia/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /c/Users/julia/.ssh/id_rsa
Your public key has been saved in /c/Users/julia/.ssh/id_rsa.pub
The key fingerprint is:
SHA256:hCLGAexnQuBLqBer4ij6wgiwybiL6tPRr0kAd0wDnMU "marmonasterio@uade.edu.ar"
The key's randomart image is:
+---[RSA 4096]---+
|=.o.
|+++E .
|+B=o. ..
|=+=* ..
|*+*oo S
|*+ ..o
|*o. +
|Xo.+
|@*+.
+---[SHA256]---
```

# GitHub - git Clone - SSH

2) Esto generará la clave privada dentro de la carpeta  
(C:\Users\usuario\.ssh) con el nombre **id\_rsa.pub**

Copiar el contenido, y luego dirigirse a **Settings > SSH and GPG keys > New SSH key**

Copiar el contenido pegado

# Resumen de la clase

- Conceptos básicos de control de versiones: repositorios locales y remotos
- Git: Operaciones: añadir, confirmar y eliminar archivos
- Github: Clonación de repositorios
- Seguimiento de archivos y cambios



Muchas gracias!

Consultas?

