

Git y control de versiones



git

Juan Rodrigo Leños Bermejo

juan.leanos@upa.edu.mx

Introducción a Git y control de versiones

- **Conceptos clave:**
 - **Git** es un sistema de control de versiones distribuido, que permite registrar cambios en archivos y colaborar con otros usuarios en el mismo proyecto.
 - **Control de versiones:** Es una práctica que permite llevar un registro histórico de los cambios realizados en los archivos a lo largo del tiempo.



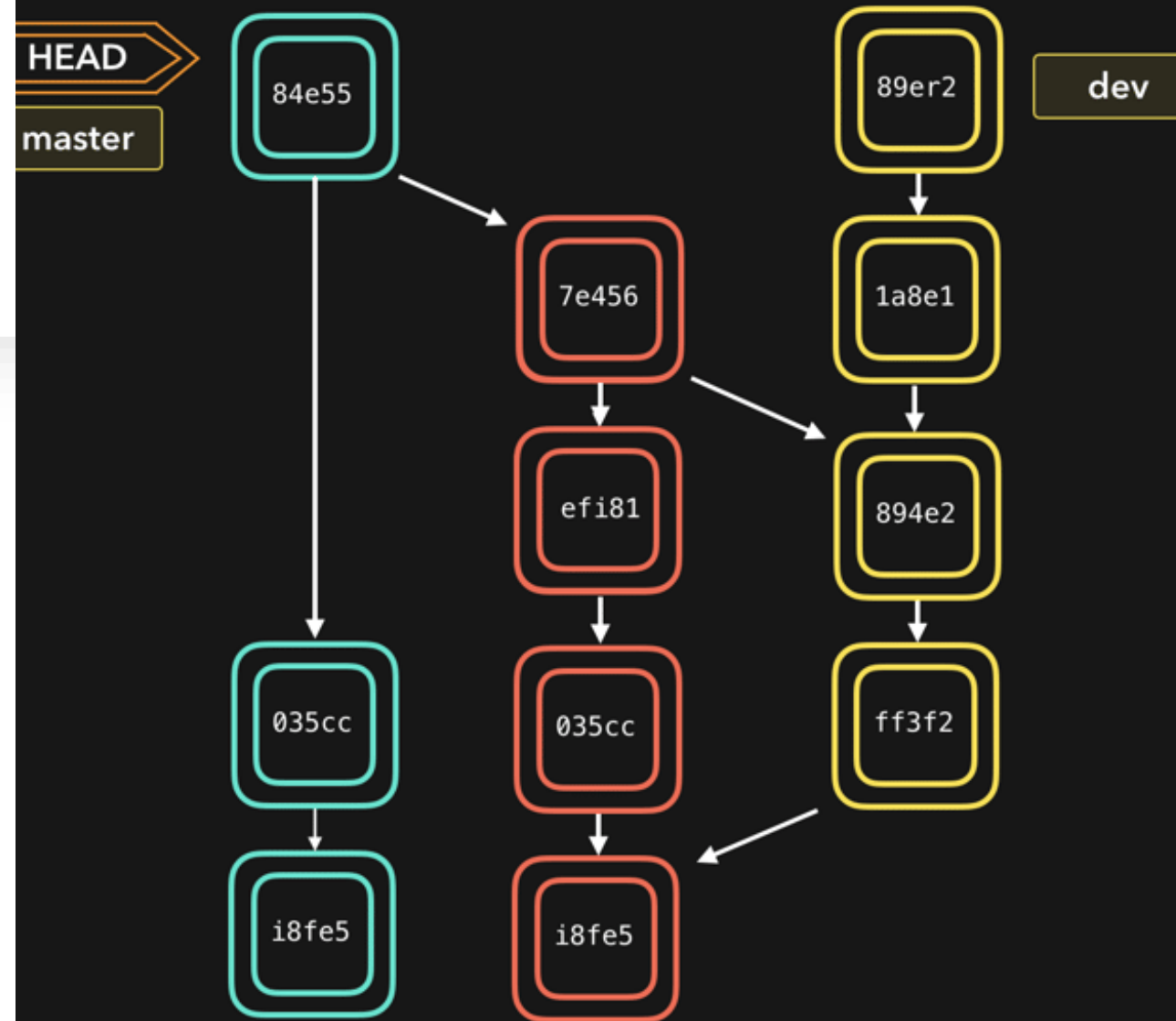
Introducción a Git y control de versiones

- **Diferencia entre Git y otros sistemas:** Git es distribuido, lo que significa que cada colaborador tiene una copia completa del proyecto, en lugar de depender de un servidor central.
- **Beneficios de usar Git:**
 - Mantener un historial detallado de los cambios.
 - Poder revertir versiones si es necesario.
 - Facilitar la colaboración en equipo.



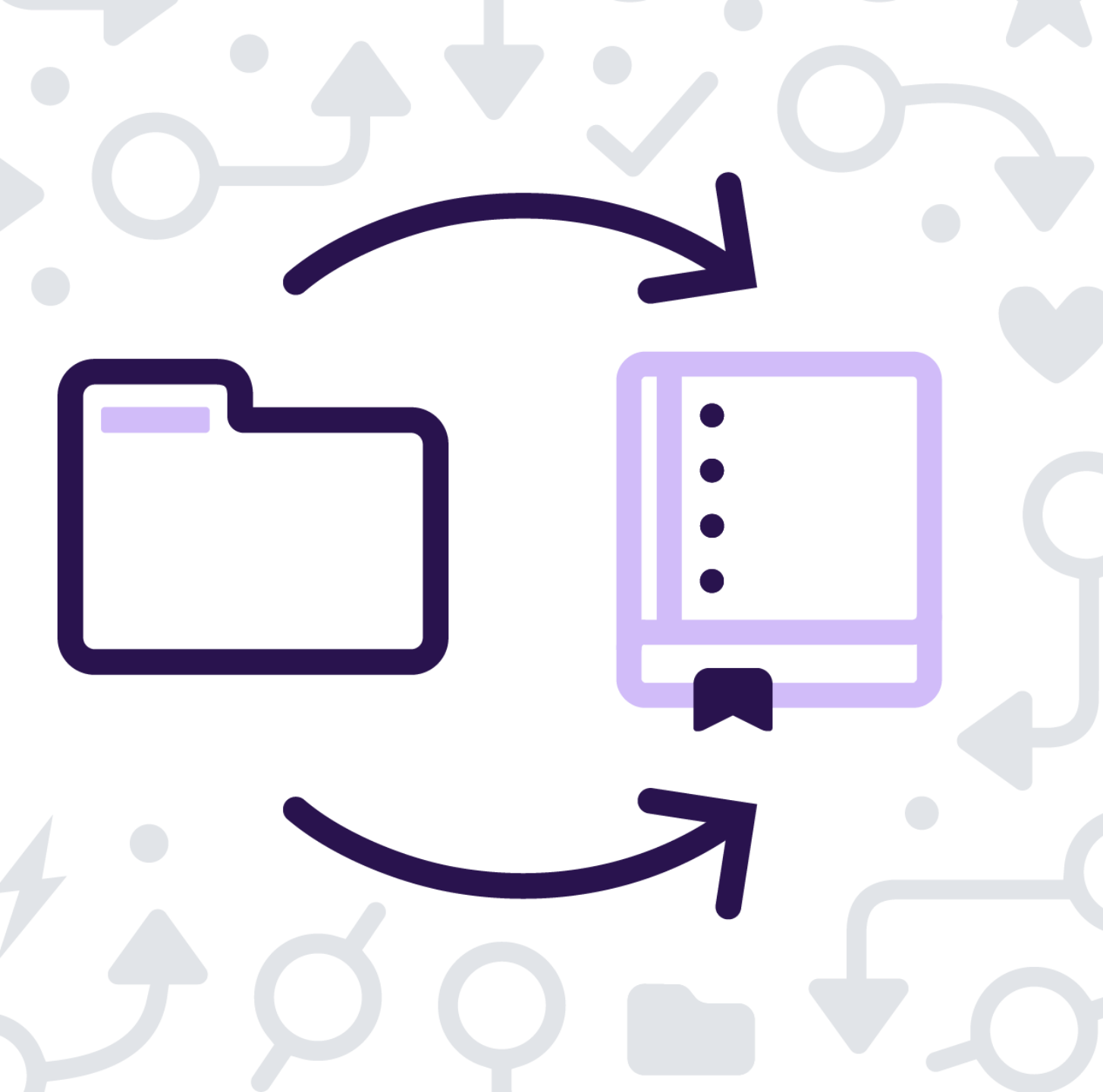
Instalación y configuración

```
git config --global user.name "Tu Nombre"  
git config --global user.email  
"tuemail@ejemplo.com"
```



Iniciando un repositorio

- **Conceptos clave:**
 - Un **repositorio** es donde Git guarda el historial de los cambios de un proyecto.
 - Un repositorio puede ser **local** (en tu computadora) o **remoto** (por ejemplo, en GitHub).



Comandos

Crear un repositorio:


```
bash
```

 Copy code

```
git init
```

Clonar un repositorio:

```
bash
```

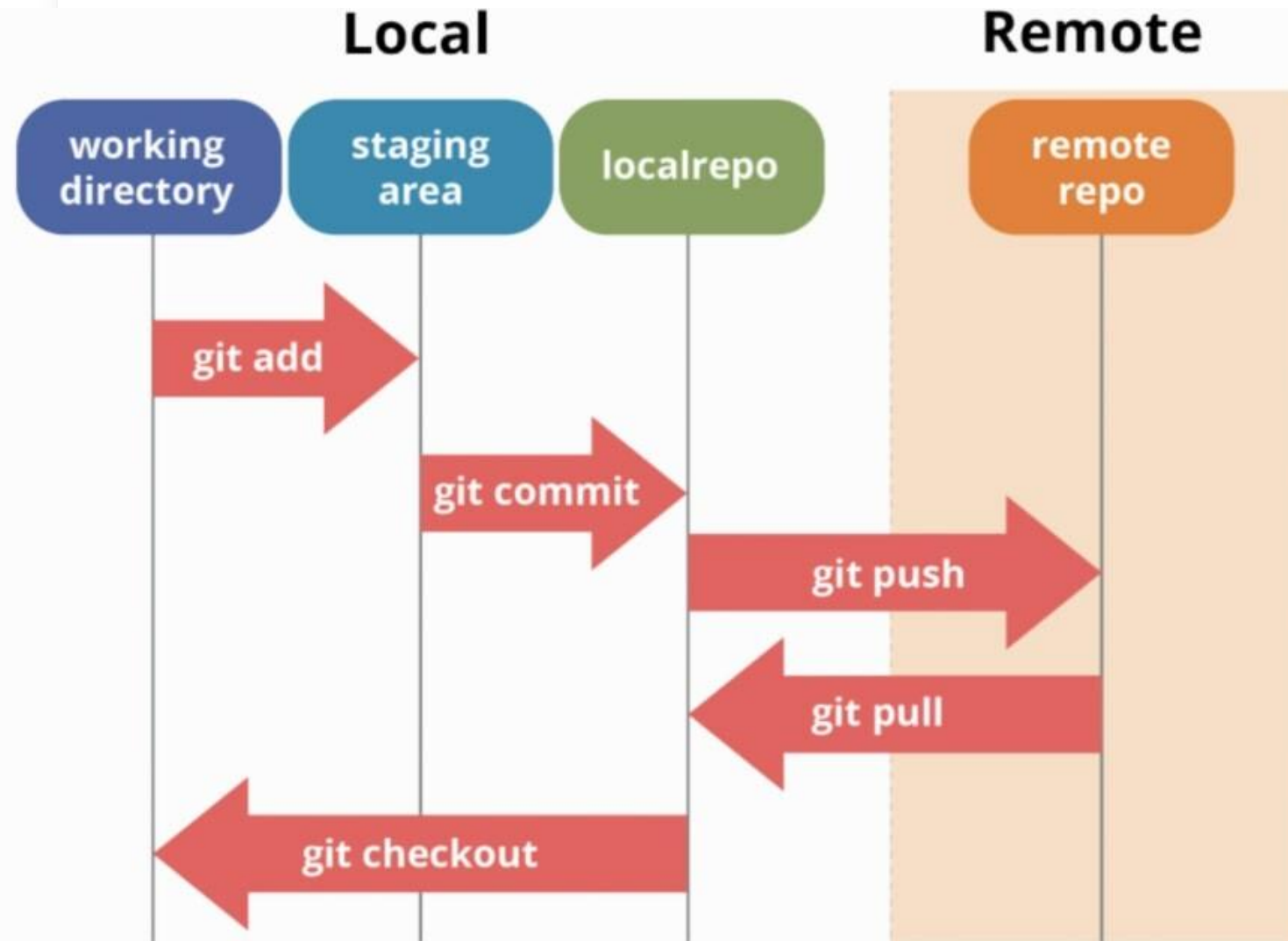
 Copy code

```
git clone <URL_del_repositorio>
```



Control de cambios y seguimiento

- **Área de staging:** Donde colocas archivos que están listos para ser confirmados en el repositorio.
- **Commit:** Es una instantánea de los archivos en el repositorio, una forma de "guardar" el estado actual de tu trabajo.




Comandos

Ver estado del repositorio:

Esto te dice qué archivos han sido modificados y cuáles están listos para confirmar

bash

 Copy code

```
git status
```

Agregar archivos al área de staging:


bash

 Copy code

```
git add <nombre_del_archivo>    # para un archivo en específico  
git add .                        # para todos los archivos modificados
```

Hacer un commit:

bash


 Copy code

```
git commit -m "Mensaje descriptivo del cambio"
```


Comandos

Hacer un commit:


bash

 Copy code

```
git commit -m "Mensaje descriptivo del cambio"
```

Subir cambios a GitHub (suponiendo que ya configuraste el repositorio remoto):

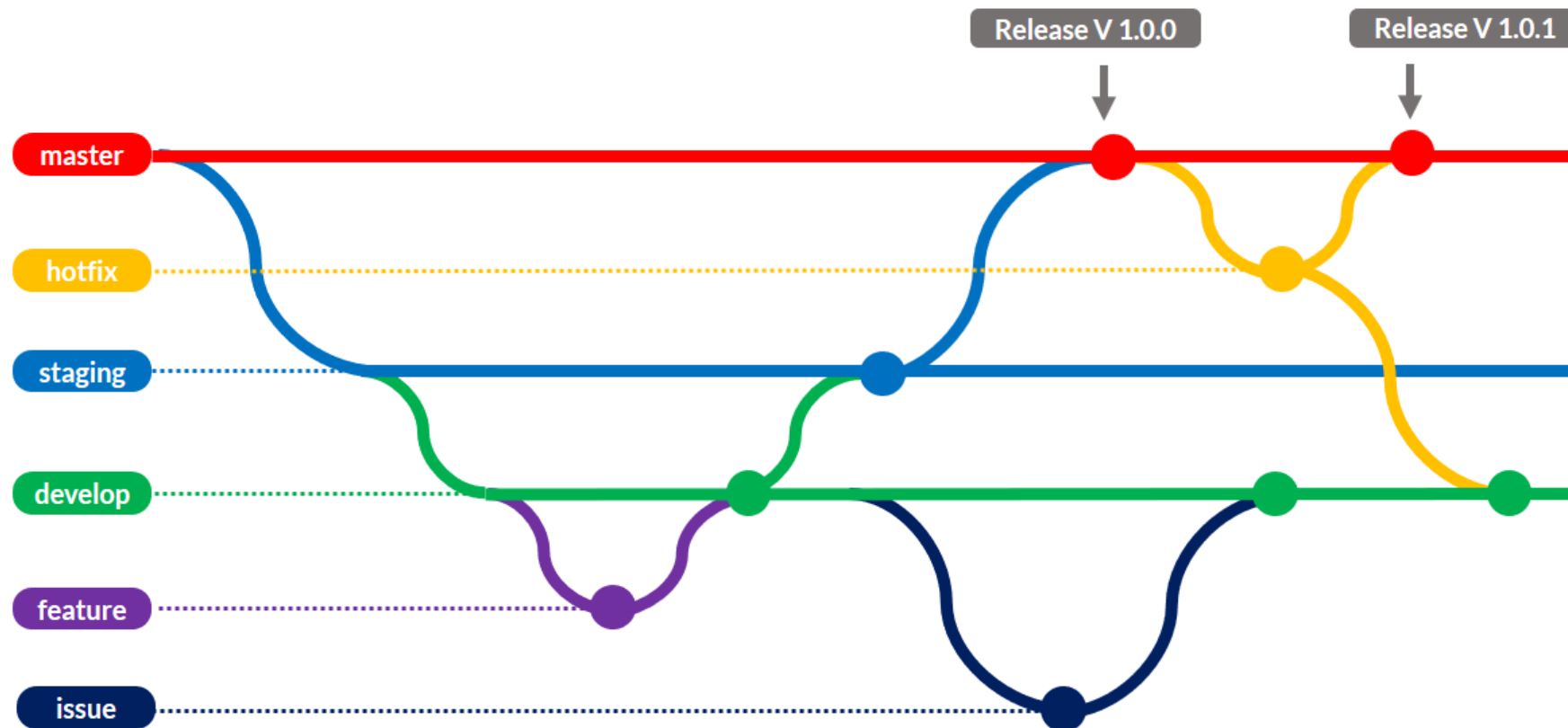
bash

 Copy code

```
git push origin main
```

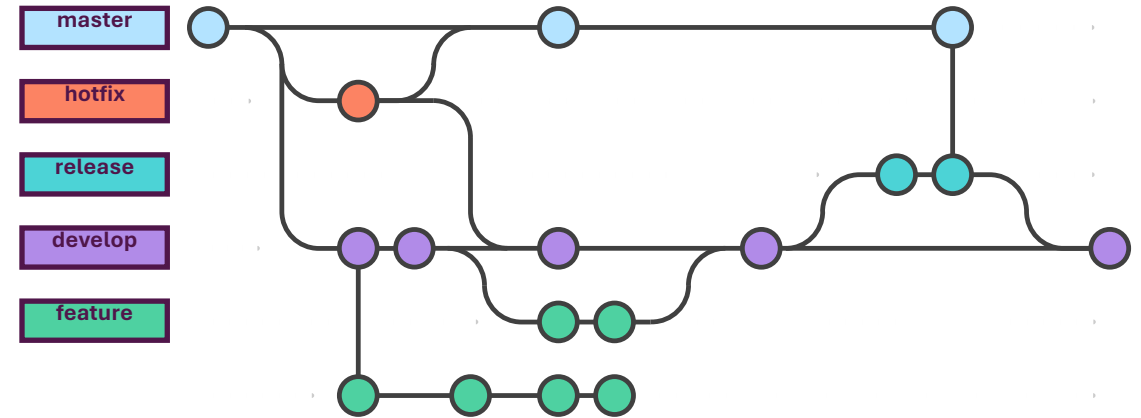
Ramas y flujo de trabajo en Git

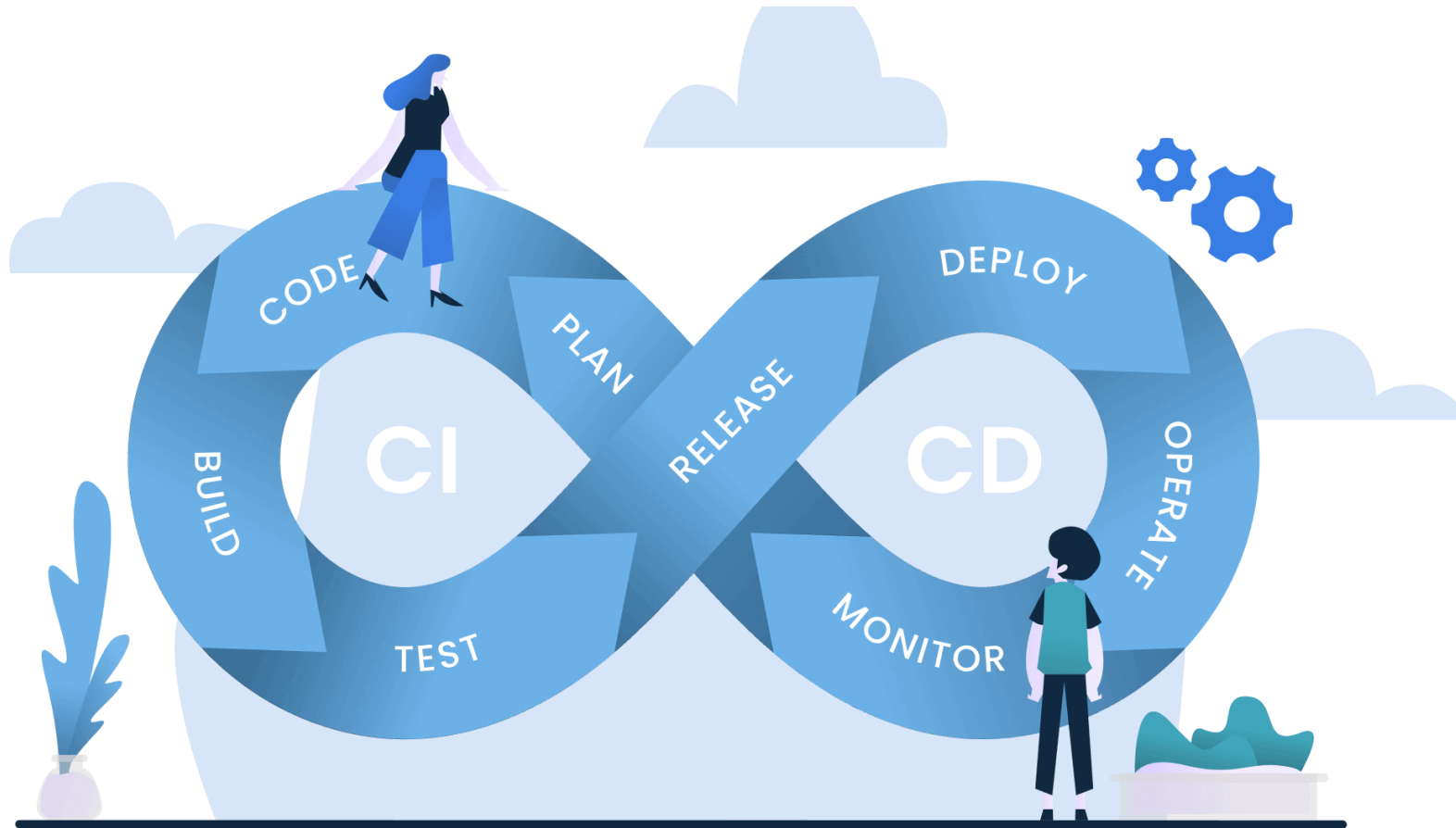
Ramas (branches) permiten que diferentes personas o procesos trabajen en paralelo sin interferir entre sí.



GitFlow

- GitFlow es un flujo de trabajo que es utilizado de como estándar a nivel global.
- Es ideal para orquestar a través de él la ejecución de prácticas de integración continua y despliegue continuo CI & CD.






DevOps – CI & CD

Comandos

Crear una nueva rama:

bash

 Copy code

```
git branch <nombre_de_la_rama>
```

Cambiar de rama:


bash

 Copy code

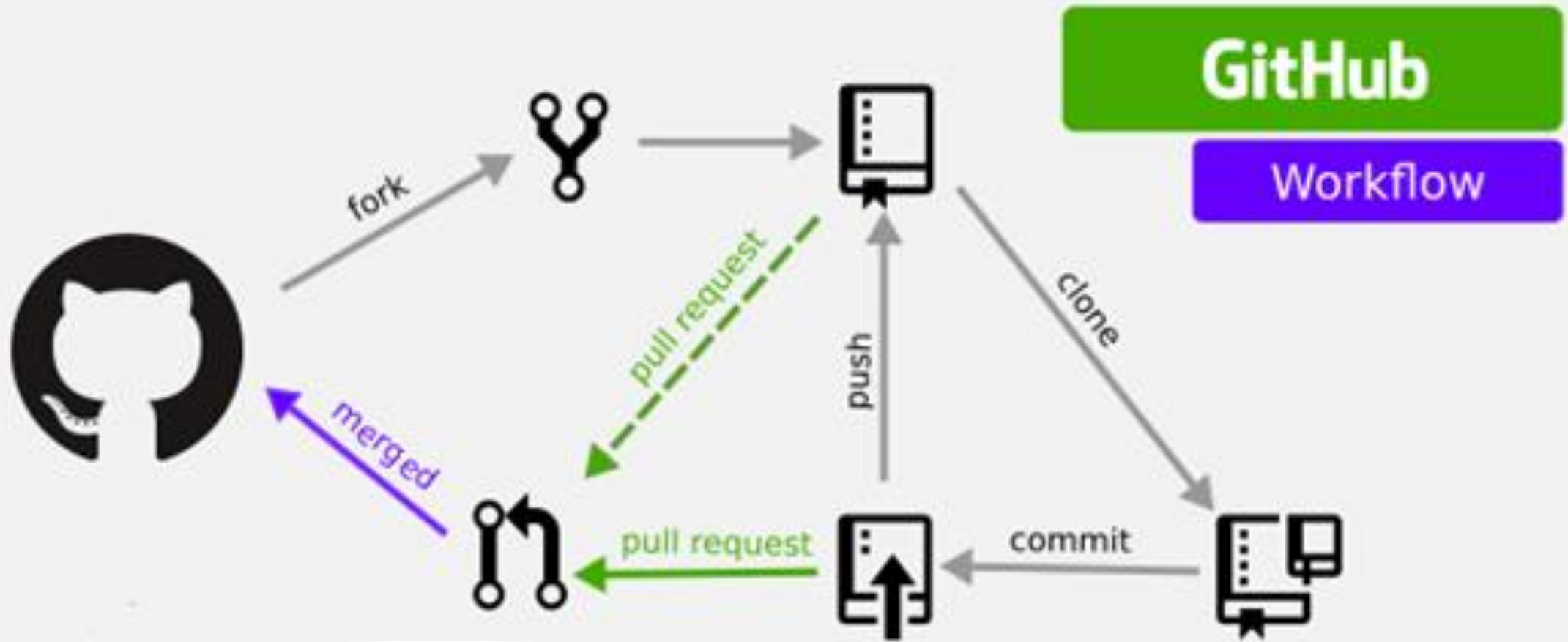
```
git checkout <nombre_de_la_rama>
```

Fusionar ramas:

bash

 Copy code

```
git merge <nombre_de_la_rama>
```



Colaboración con GitHub

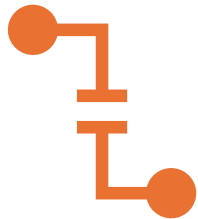


GitHub es un servicio que proporciona alojamiento para repositorios remotos de Git.



Forks y **pull requests** son formas comunes de colaborar en proyectos de código abierto.

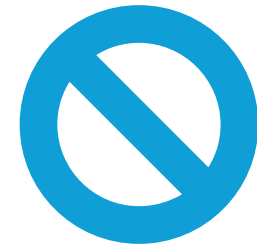
Buenas prácticas con Git



Commits pequeños y descriptivos: Cada commit debe reflejar un conjunto lógico de cambios.



Ramas específicas: Mantén las ramas bien organizadas: main, develop, feature/nueva_característica, hotfix/arreglo.



Frecuencia de commits: No guardes todo tu trabajo en un solo commit. Guarda tu progreso de manera regular.