



/GIT /GITHUB





/AGENDA



/01 /git

> Conceptos básicos

/02 /uso y comandos

> Inicializar repo
Comandos básicos

/03 /ramificaciones

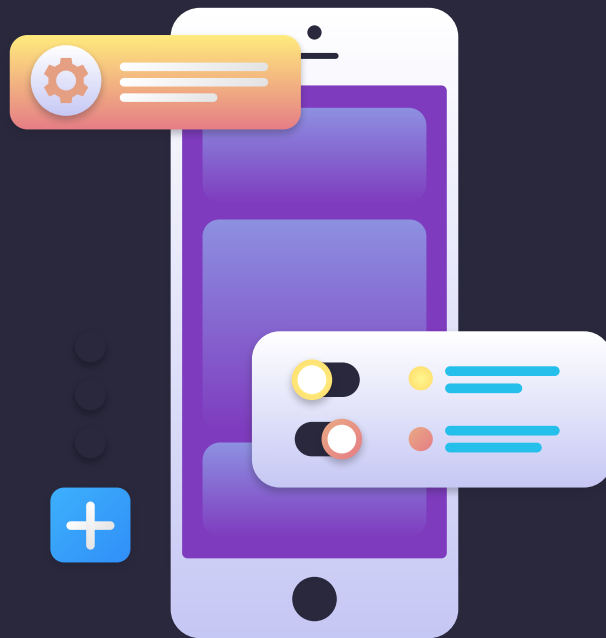
> Conceptos
Comandos básicos





/01

/git





/qué es git?



- > GIT es un DVCS (sistema de control de versiones distribuido).
- > Muy usado por desarrolladores Open Source como también sistemas comerciales.
- > Permite a cada usuario el acceso total a cada archivo, rama e iteración de un proyecto, teniendo acceso al completo historial de cambio.





/los tres estados



/modified

Se realizaron cambios pero no se ha hecho un commit



/staged

Se lo marcó para que vaya en el siguiente snapshot

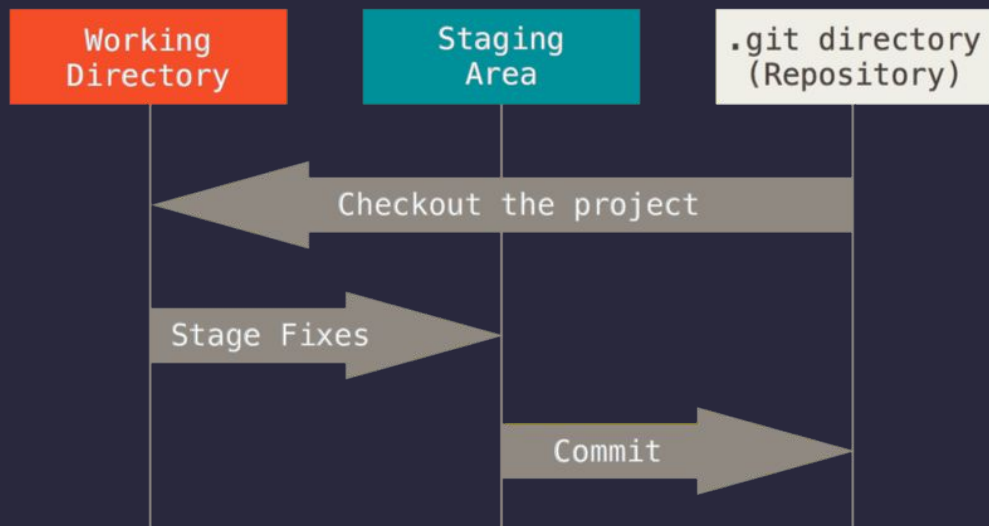


/committed

La información fue guardada localmente de forma segura.



/las tres áreas



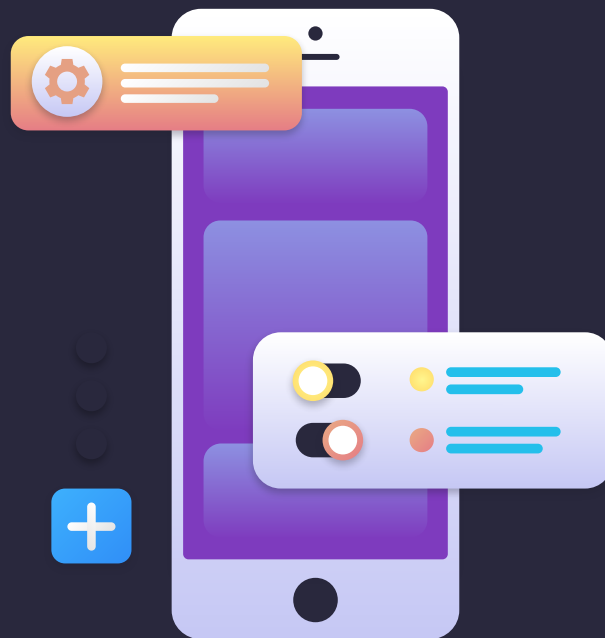


/workflow normal



/02

/uso y comandos





/cómo usar git?



/command line

Permite ejecutar todos los comandos disponibles



/GUI client

En general permiten ejecutar un subset de los comandos disponibles. Ejemplos: Source Tree, GitKraken, etc.



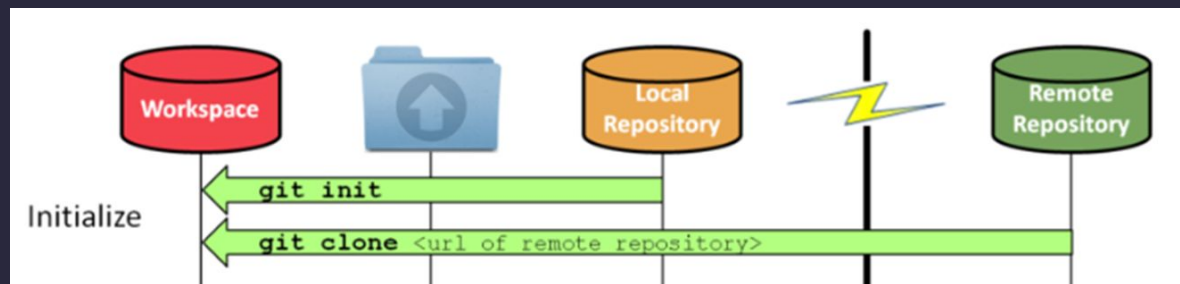


/cómo inicio un repositorio?



- Tenemos dos formas para comenzar a utilizar git.
- > Las mismas nos servirán para introducir los dos primeros comandos de la herramienta
 - Empezar con un nuevo repositorio de Git
 - Clonar un repositorio de Git ya existente







/git init



> Este comando inicializa un nuevo repositorio git dentro del directorio que deseemos versionar. Simplemente nos movemos dentro de dicho directorio y ejecutamos.

- Creamos un nuevo directorio utilizando el comando *mkdir*
- Navegamos dentro del mismo utilizando *cd*
- Luego *git init* y tenemos nuestro repositorio





/git clone



- > La otra forma para comenzar a utilizar Git es clonar un repositorio ya existente. Para ello ejecutamos

```
git clone [url_del_repositorio]
```

- > La URL del repositorio varía según el protocolo utilizado para clonar el mismo, por ejemplo

```
git clone utilizando HTTPS
```

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





/git status



> Muestra el estado de los archivos

- *git status*





/git add



> Actualiza el index usando el contenido actual del working tree. En general agrega todo el contenido excepto lo ignorado (git ignore), pero puede configurarse con los distintos parámetros. Se puede llegar a usar muchas veces antes de un commit.

- *git add*





/git add .



> Es un shortcut que nos permite mover todos los
archivos untracked y modified a staged sin tener
que hacerlo uno por uno

- `git add .`





/git commit



> Para guardar un nuevo snapshot con los archivos staged

- `git commit`





/git commit -m “mensaje”



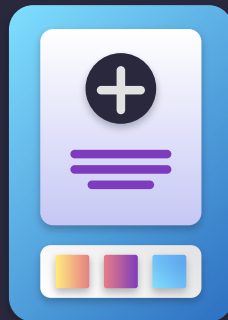
> Shortcut que evita abrir la consola de mensaje y ya pasarle el mensaje de commit

- `git commit -m “mensaje”`





<REPO
REMOTO>





/git pull



> Incorpora los cambios desde el repositorio remoto
a la rama actual

- *git pull*





/git fetch



> Chequea si hay cambios disponibles para traer a la rama. No realiza transferencia de datos.

- *git fetch*





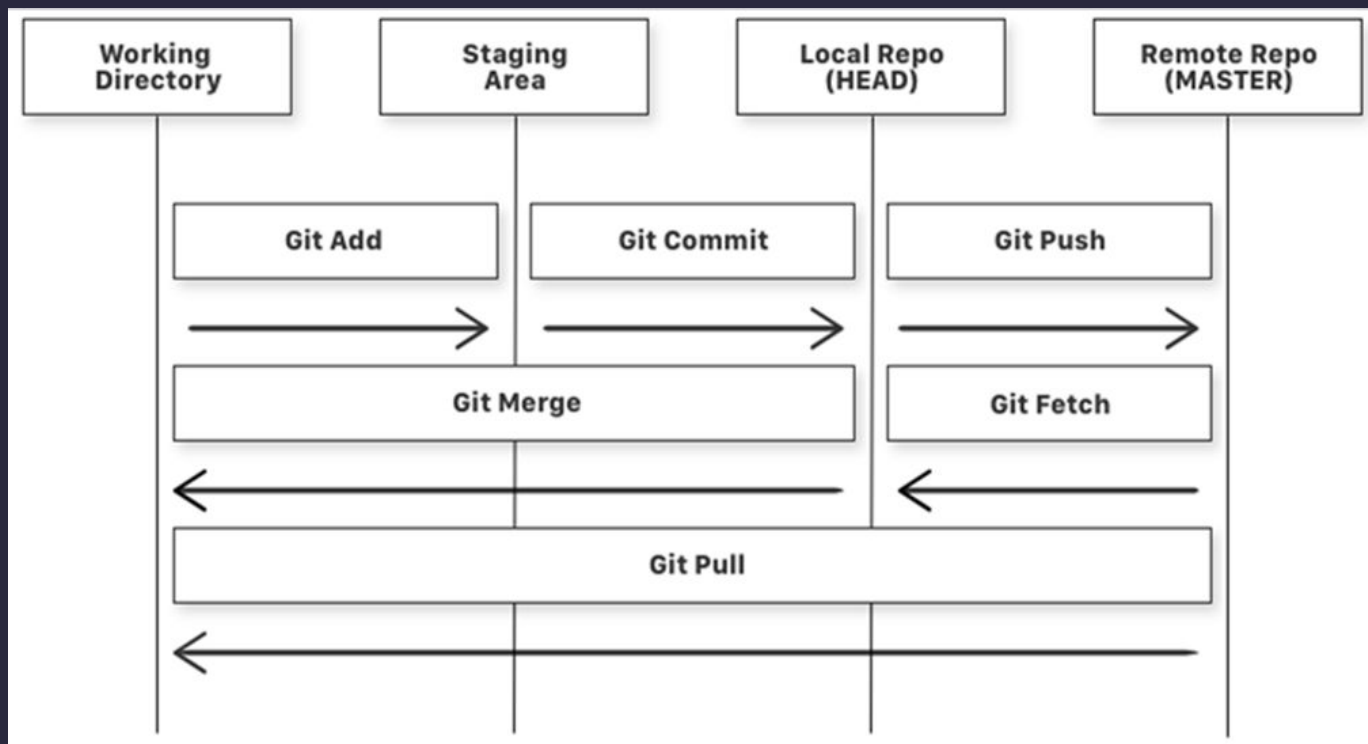
/git push



> Actualiza las referencias remotas usando las
locales

- *git push*

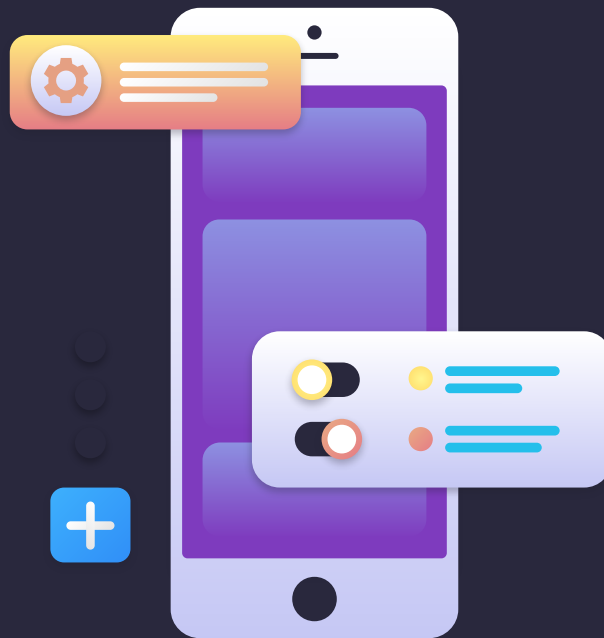






/03

/ramificaciones





/qué es una branch?



Una branch en Git no es más que un puntero a un commit. Por defecto la branch inicial siempre se > denomina master o main, aunque no difiere en su comportamiento con ninguna otra branch. Se pueden crear por múltiples motivos:

- Organización interna del equipo.
- Aislar un grupo.
- Variante del producto para un cliente





/git branch



- > Crea una nueva branch que referencia al commit en el que estábamos posiciones

- `git branch <name>`

- > Si no le pasamos un nombre, el comando nos muestra la lista de todas las ramas.

- > Si agregamos “-d” antes de especificar un nombre, eliminará dicha rama.

- `git branch -d <name>`





/git checkout



> Para movernos de branch

- `git checkout <name>`

> Existe un shortcut que nos permite crear una nueva rama y automáticamente cambiarnos a esa rama

- `git checkout -b <name>`





/merge & rebase



- > Cuando trabajamos con múltiples branches en git, muchas veces queremos integrar el trabajo de una branch en otra.
- > O simplemente queremos que la branch que contiene nuestra funcionalidad esté incluida en master.
- > Tenemos dos formas para integrar los commits de una branch en otra:
 - Mergear ambas branches
 - Aplicar los commits de una branch encima de otra (rebase)





/git merge



- `git merge <name>`

- > Suponiendo el caso en el que HEAD se encuentre apuntando a master, cuando ejecutemos el comando

- `git merge testing`

- > Git tratará de mergear testing en master. Es decir que luego de ejecutar el comando, si todo salió exitosamente, master tendrá incluidos los commits de testing.





/git rebase



- `git rebase <name>`

- > Suponiendo el caso en el que HEAD se encuentre apuntando a una branch testing, cuando ejecutemos el comando

- `git rebase master`

- > Git rebobinará la branch testing hasta llegar al primer ancestro en común que tenga con master. Luego aplicará los commits restantes de testing encima de los cambios generados en master. Al finalizar la ejecución testing tendrá incluidos los cambios de master.





/ramas remotas



- > Las ramas remotas son referencias al estado de las ramas en tus repositorios remotos.
- > Si un compañero hace push a la rama remota, tu rama local no se modifica hasta que traigas los cambios. Esto quiere decir que las ramas pueden avanzar de formas distintas.





/manejar conflictos



> Cuando en dos ramas diferentes hay modificaciones dispares sobre una misma porción de código, Git no es capaz de manejarlo y hay que hacerlo manualmente.

Git automáticamente muestra unos marcadores que te ayudan a corregir los conflictos del archivo una vez que lo abras manualmente.

