DOCUMENTACIÓN GIT EXAMEN

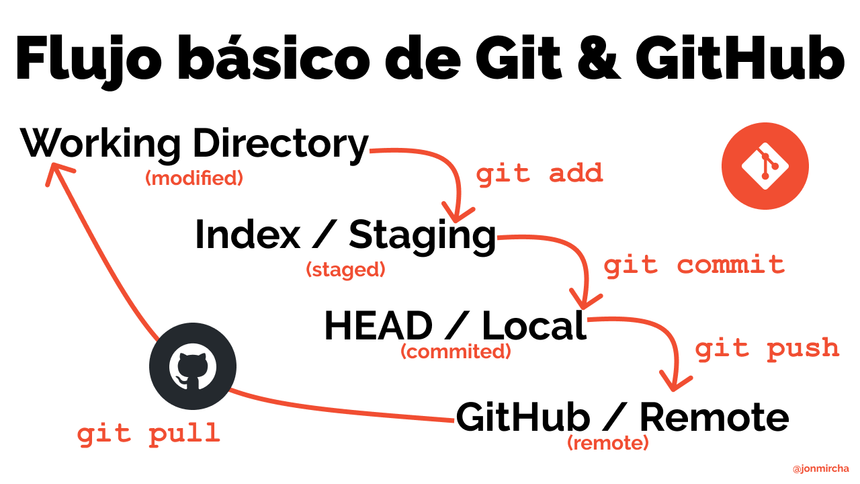




# 1. Introducción a Git

Git es una herramienta de control de versiones que permite registrar los cambios en un proyecto, recuperar versiones anteriores y colaborar con otros usuarios.

Permite mantener el código organizado y evita perder trabajo al realizar cambios importantes.



## Conceptos clave

|  |  |
| --- | --- |
| Concepto | Descripción |
| Repositorio | Carpeta donde Git guarda el historial del proyecto. |
| Commit | Versión o guardado de los cambios realizados. |
| Branch | Rama de desarrollo independiente del proyecto principal. |
| Merge | Fusión de dos ramas en una sola. |
| Remote | Repositorio remoto alojado en un servidor (GitHub, GitLab...). |

# Instalación y configuración de Git

Instalación en diferentes sistemas operativos:

Windows: descargar desde https://git-scm.com  
Linux: sudo apt install git  
MacOS: brew install git

Configuración inicial del usuario:

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

Archivo .gitignore:

node\_modules/  
\*.log  
\*.tmp

# Modelo de Git y comandos principales

El modelo de Git se basa en tres áreas: directorio de trabajo, área de preparación y repositorio local.

|  |  |
| --- | --- |
| Comando | Descripción |
| git init | Crea un repositorio nuevo. |
| git clone <url> | Clona un repositorio remoto. |
| git add <archivo> | Añade archivos al área de preparación. |
| git commit -m 'mensaje' | Guarda los cambios. |
| git status | Muestra el estado de los archivos. |
| git log | Muestra el historial de commits. |
| git diff | Muestra las diferencias entre versiones. |
| git branch | Crea o lista ramas. |
| git checkout | Cambia de rama. |
| git merge | Fusiona ramas. |
| git remote | Gestiona repositorios remotos. |
| git pull | Descarga cambios del remoto. |
| git push | Sube cambios al remoto. |
| git revert | Deshace un commit anterior. |
| git stash | Guarda temporalmente los cambios. |

# Ramas y flujo de trabajo

Las ramas permiten desarrollar nuevas funciones sin afectar la versión principal del proyecto.

git branch nueva-funcion  
git checkout nueva-funcion  
git merge nueva-funcion  
git branch -d nueva-funcion

Ejemplo visual del flujo de trabajo:

main ───●────●───────────●───────────  
 \ \  
 \── feature ─────●── merge ───

# Buenas prácticas

- Usa mensajes de commit claros y descriptivos.  
- No subas archivos grandes o temporales.  
- Usa ramas para desarrollar nuevas funciones.  
- Sincroniza el proyecto frecuentemente.

# Reproducción y resolución de un conflicto

1. Crear repositorio:

git init ConflictoGit  
cd ConflictoGit  
echo 'Hola Mundo' > index.txt  
git add .  
git commit -m 'Primer commit'

2. Crear rama feature y modificar el archivo:

git branch feature  
git checkout feature  
echo 'Hola desde feature' > index.txt  
git add .  
git commit -m 'Cambio en feature'

3. Modificar la misma línea en main:

git checkout main  
echo 'Hola desde main' > index.txt  
git add .  
git commit -m 'Cambio en main'

4. Provocar el conflicto:

git merge feature

5. El conflicto aparece así:

<<<<<<< HEAD  
Hola desde main  
=======  
Hola desde feature  
>>>>>>> feature

6. Resolver el conflicto editando el archivo y dejando una versión final:

Hola desde ambas ramas

7. Después ejecutar:

git add index.txt  
git commit -m 'Conflicto resuelto'