COMILLAS
UNIVERSIDAD PONTIFICIA
ICAI

Introducción a Git

¿Qué es Git?

- Git es un sistema de control de versiones distribuido
- Permite a los desarrolladores trabajar en proyectos de forma colaborativa
- Controla los cambios realizados en el código fuente de un proyecto
- Facilita la colaboración y el seguimiento de versiones a lo largo del tiempo

¿Para qué sirve Git?

- **Colaboración**: Facilita el trabajo en equipo en un mismo proyecto
- **Control de versiones**: Permite registrar y gestionar diferentes versiones del código
- **Seguimiento**: Puedes revisar quién hizo qué cambios y cuándo
- **Deshacer errores**: Facilita la reversión de cambios no deseados en el código.

Plataformas para alojar Git

- Existen varias plataformas populares para alojar repositorios Git
  - GitHub:
    - La más utilizada, ideal para proyectos open source y privados
  - GitLab:
    - Ofrece integración continua y gestión de proyectos
  - Bitbucket
    - Integración con herramientas de Atlassian como Jira



Áreas de Git

- Git organiza el trabajo en tres áreas principales
  - **Working Directory** (Directorio de trabajo):
    - Es donde editas y trabajas en tus archivos localmente
  - **Staging Area** (Área de preparación):
    - Donde colocas los archivos que deseas confirmar
  - **Repository** (Repositorio):
    - Almacena los commits confirmados de forma permanente
- Resumen visual:

Working Directory → git add → **Staging Area** → git commit → **Repository**

Configuración inicial de Git

- Antes de usar Git, debes configurar tu nombre y correo para que tus commits se identifiquen correctamente
  - **git config --global user.name "Tu Nombre"**
  - **git config --global user.email "tuemail@dominio.com"**
- Para crear un repositorio en una carpeta local:
  - **git init**
- Para clonar un repositorio:
  - **git clone <url-del-repositorio>**

•git config --global user.name "Tu Nombre"

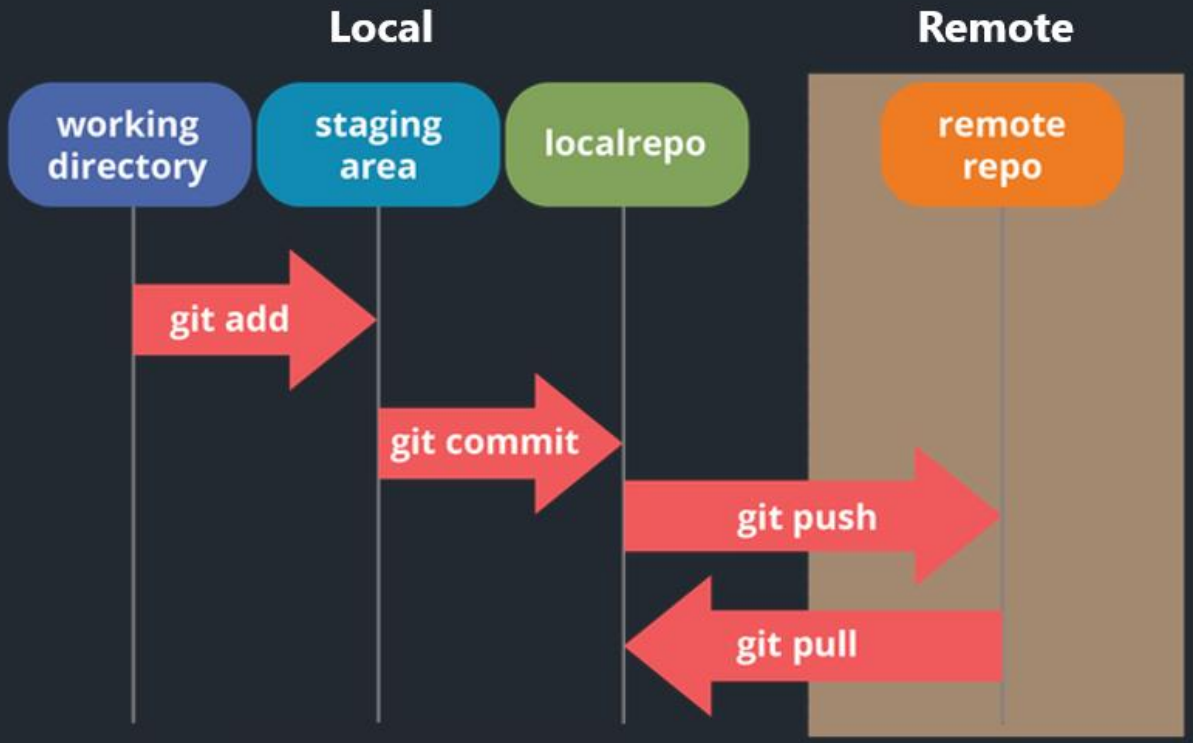
Comandos de Git

- Ver el estado actual de los archivos en el repositorio
  - **git status**
- Agregar archivos al staging área
  - **git add <archivo>**
  - **git add .** (Agrega todos los archivos al staging area)
- Realizar un commit (Guardar cambios en el repositorio con una descripción)
  - **git commit -m "Mensaje descriptivo"**

Comandos de Git

- Obtener los cambios de un repositorio remoto
  - **git fetch**
- Traer y fusionar cambios remotos (Pull)
  - **git pull**
- Enviar cambios al repositorio remoto (Push)
  - **git push**

Ejemplo de flujo



Cheat sheet

GIT BASICS

`git init <directory>`	Create empty Git repo in specified directory. Run with no arguments to initialize the current directory as a git repository.
`git clone <repo>`	Clone repo located at `<repo>` onto local machine. Original repo can be located on the local filesystem or on a remote machine via HTTP or SSH.
`git config user.name <name>`	Define author name to be used for all commits in current repo. Devs commonly use `--global` flag to set config options for current user.
`git add <directory>`	Stage all changes in `<directory>` for the next commit. Replace `<directory>` with a `<file>` to change a specific file.
`git commit -m "<message>"`	Commit the staged snapshot, but instead of launching a text editor, use `<message>` as the commit message.
`git status`	List which files are staged, unstaged, and untracked.
`git log`	Display the entire commit history using the default format. For customization see additional options.
`git diff`	Show unstaged changes between your index and working directory.

UNDOING CHANGES

`git revert <commit>`	Create new commit that undoes all of the changes made in `<commit>`, then apply it to the current branch.
`git reset <file>`	Remove `<file>` from the staging area, but leave the working directory unchanged. This unstages a file without overwriting any changes.
`git clean -n`	Shows which files would be removed from working directory. Use the `-f` flag in place of the `-n` flag to execute the clean.

REWRITING GIT HISTORY

`git commit --amend`	Replace the last commit with the staged changes and last commit combined. Use with nothing staged to edit the last commit's message.
`git rebase <base>`	Rebase the current branch onto `<base>`. `<base>` can be a commit ID, branch name, a tag, or a relative reference to HEAD.
`git reflog`	Show a log of changes to the local repository's HEAD. Add `--relative-date` flag to show date info or `--all` to show all refs.

GIT BRANCHES

`git branch`	List all of the branches in your repo. Add a `<branch>` argument to create a new branch with the name `<branch>`.
`git checkout -b <branch>`	Create and check out a new branch named `<branch>`. Drop the `-b` flag to checkout an existing branch.
`git merge <branch>`	Merge `<branch>` into the current branch.

REMOTE REPOSITORIES

`git remote add <name> <url>`	Create a new connection to a remote repo. After adding a remote, you can use `<name>` as a shortcut for `<url>` in other commands.
`git fetch <remote> <branch>`	Fetches a specific `<branch>`, from the repo. Leave off `<branch>` to fetch all remote refs.
`git pull <remote>`	Fetch the specified remote's copy of current branch and immediately merge it into the local copy.
`git push <remote> <branch>`	Push the branch to `<remote>`, along with necessary commits and objects. Creates named branch in the remote repo if it doesn't exist.

Enlaces de interés

- Learn git visually: https://learngitbranching.js.org/?locale=es_ES
- Complete cheat sheet: https://www.datacamp.com/cheat-sheet/git-cheat-sheet
- Github: https://github.com/