

Aprendiendo  
con



**git**



GitLab

# ¿Qué es Git?

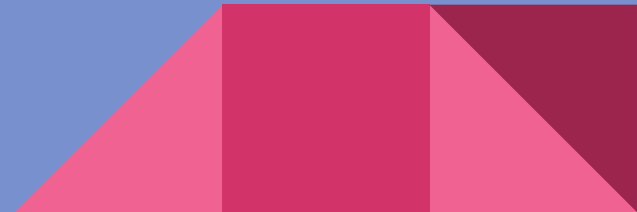
Git es un sistema de control de versiones distribuido, diseñado por Linus Torvalds. Está pensando en la eficiencia y la confiabilidad del mantenimiento de versiones de aplicaciones cuando estas tienen un gran número de archivos de código fuente. Git está optimizado para guardar todos estos cambios de forma atómica e incremental.



# ¿Qué es un sistema de control de versiones?

El SCV o VCS (por sus siglas en inglés) es un sistema que registra los cambios realizados sobre un archivo o conjunto de archivos a lo largo del tiempo, de modo que puedas llevar el historial del ciclo de vida de un proyecto, comparar cambios a lo largo del tiempo, ver quién los realizó o revertir el proyecto entero a un estado anterior.

Cualquier tipo de archivo que se encuentre en un ordenador puede ponerse bajo control de versiones.



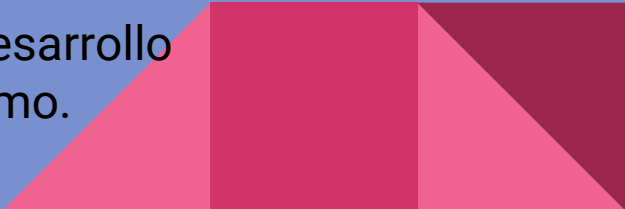
# Breve Historia de Git



Git fue creado en 2005 por el creador de Linux, Linus Torvald para tener un mejor sistema distribuido de control de versiones de los Repositorios que permita administrar múltiples ramas de desarrollo

- Control de versiones
- Distribuido
- Repositorios
- Ramas

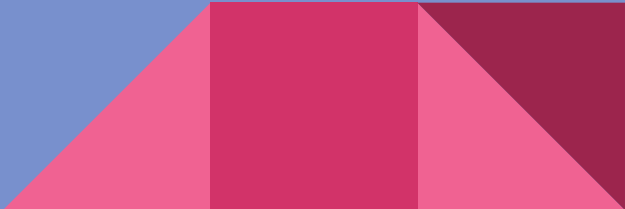
# Palabras clave asociadas a Git

- Control de versiones: Forma de almacenar los cambios atómicos de los archivos modificados en el tiempo.
  - Distribuido: Cada computador del equipo tiene una copia de toda la información (\*)
  - Repositorios: Estructura de datos que almacena metadatos de archivos y directorios específicos en vez de almacenar el archivo completo cada vez.
  - Ramas: Desviaciones del proyecto original para el desarrollo de características adicionales o variaciones del mismo.
- 

# Consideraciones

Se obtiene su mayor eficiencia con archivos de texto plano, ya que con archivos binarios no puede guardar solo los cambios, sino que debe volver a grabar el archivo completo ante cada modificación, por mínima que sea, lo que hace que incremente demasiado el tamaño del repositorio.

“Guardar archivos binarios en el repositorio de git es una mala práctica, únicamente deberían guardarse archivos pequeños (como logos) que no sufran casi modificaciones durante la vida del proyecto. Los binarios deben guardarse en un CDN”.



# Terminología



**git**

# Términos comúnmente usados en Git

**Tag** Marca única de un punto específico en el tiempo que identifica el estado del proyecto

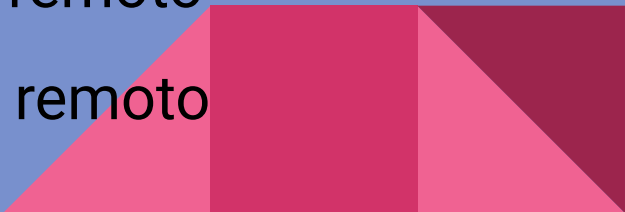
**Workspace** Directorio donde se almacenan los repositorios de manera local en la computadora

**Working Area** Zona donde están los archivos que han sido modificados (seguidos y no)

**Staging area** Archivos modificados / agregados que están marcados para aplicar los cambios en el siguiente commit



## Verbos comúnmente usados en Git

- Branch** Listar las ramas de desarrollo disponibles
  - Checkout** Cambiar entre ramas disponibles
    - Add** Agregar los cambios hechos al Staging Area
  - Commit** Agregar los cambios del Staging Area al Repositorio Local
    - Push** Enviar los cambios al repositorio remoto
    - Pull** Traer los cambios del repositorio remoto
- 

# Repositorios

Local  
repository

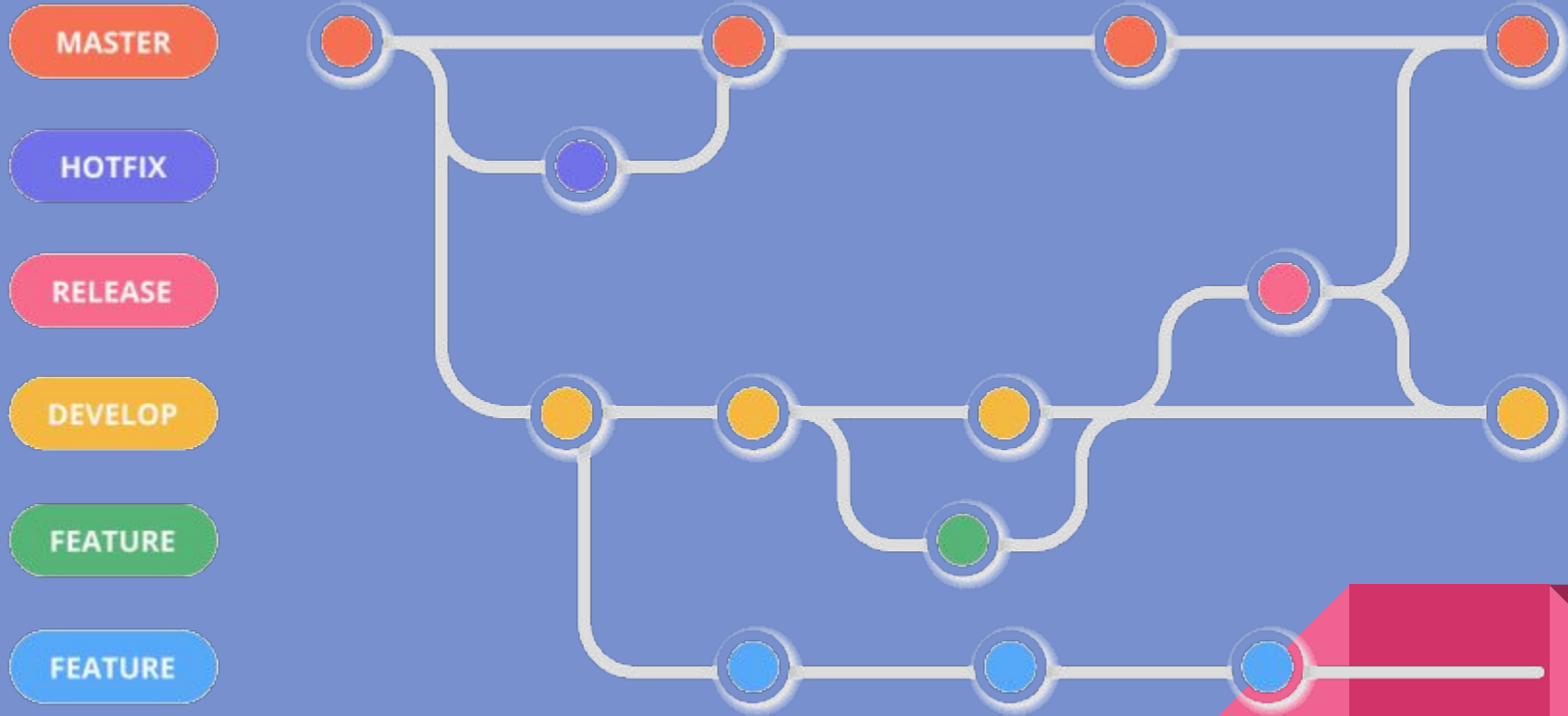
Copia local del repositorio en el computador

Remote  
repository

Repositorio alojado en un servidor remoto compartido (ej. GitLab)



# Ramas y Workflow



# Las 4 secciones principales de un proyecto de Git

## Working Directory

El directorio de trabajo (Working Directory)

Donde se hacen los cambios de cualquier Archivo

## Staging Area

El área de preparación (Staging Area)

Donde se indican los archivos a seguir y preparan los cambios a ser almacenados

## Local Repository

El directorio de Git (Git Directory, Repository) en la PC local

Donde se guardan sólo los segmentos cambiados de los archivos indicados

## Remote Repository

Servidor remoto (Gitlab)

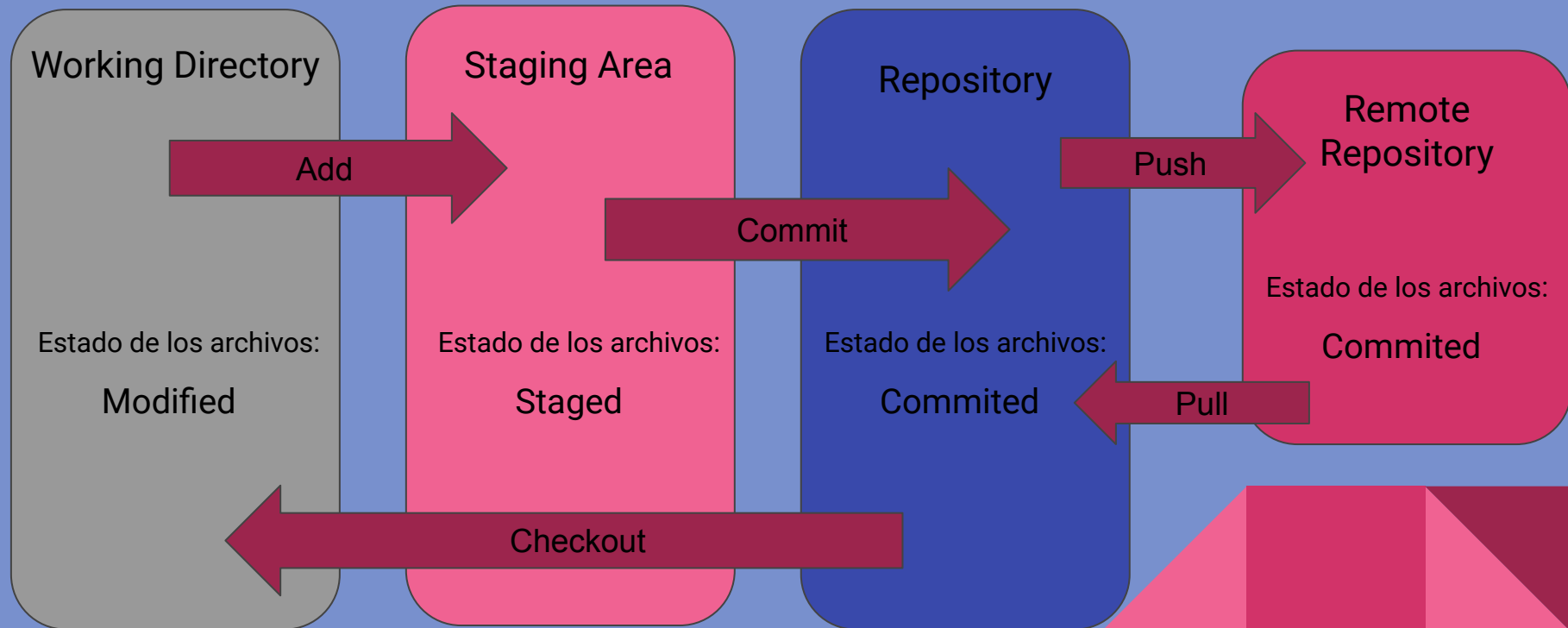
# Ciclo básico de trabajo en Git

- Se modifica una serie de archivos en el directorio de trabajo.
- Se preparan los archivos añadiéndoles al área de preparación o staging. 'git add'
- Se confirman los cambios: las instantáneas de los archivos que están en el área de staging se almacenan de forma permanente en el directorio de Git. 'git commit'

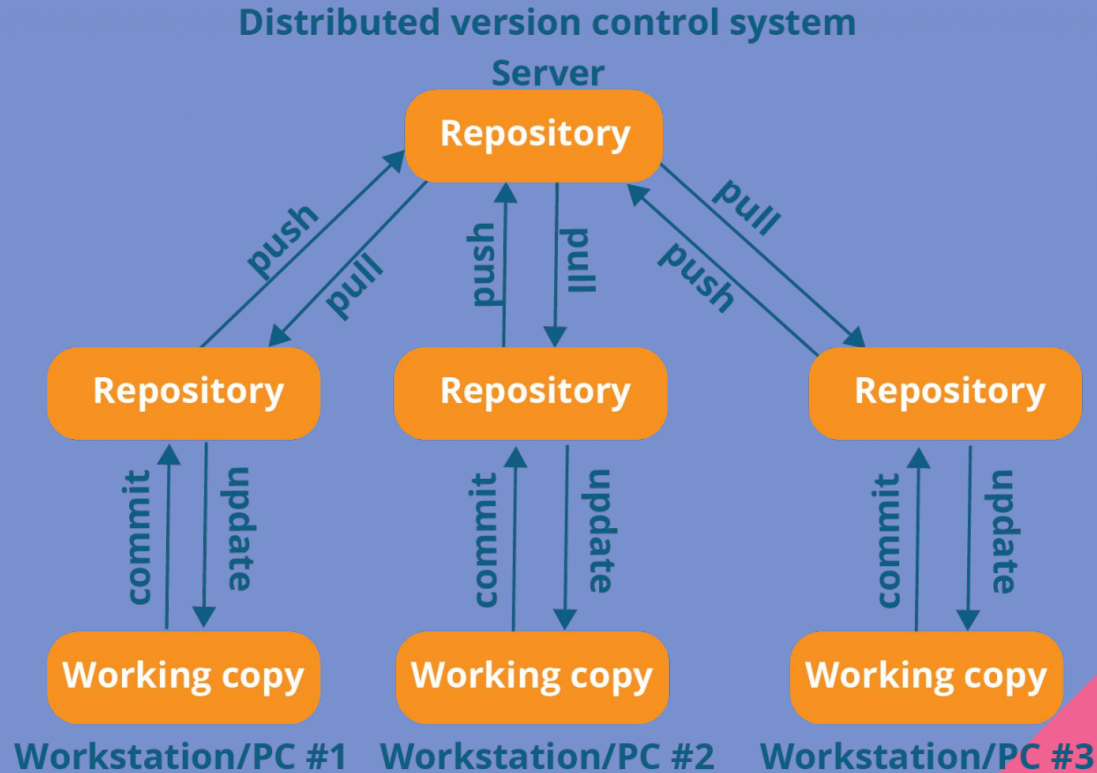
# Estados de un archivo

- **Committed**: Si una versión concreta de un archivo está en el directorio de Git, se considera confirmada.
- **Staged**: Si ha sufrido cambios desde que se obtuvo del repositorio, pero ha sido añadida al área de preparación, está preparada.
- **Modified**: Y si ha sufrido cambios desde que se obtuvo del repositorio, pero no se ha preparado, está modificada.

# Operaciones Locales y Remotas



# Sistema Distribuido





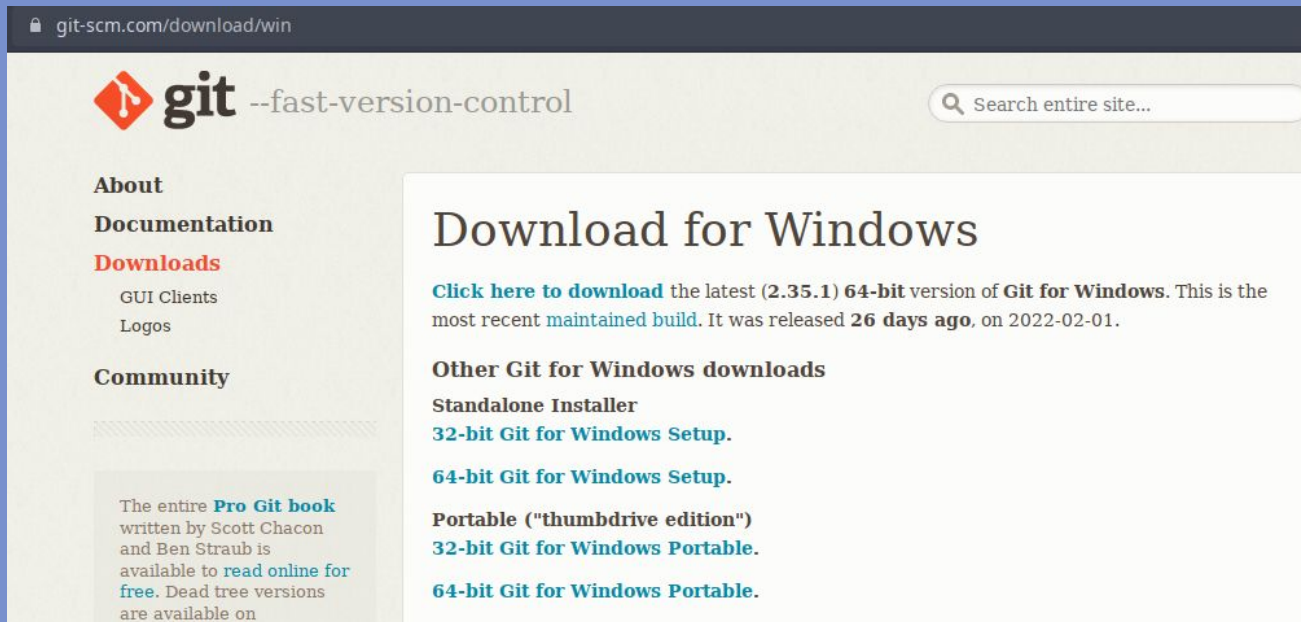
# Configuración



**git**


# Instalando Git en Windows

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



The screenshot shows the Git website's Windows download page. The browser address bar displays "git-scm.com/download/win". The page header features the Git logo and the tagline "--fast-version-control", along with a search bar. The left sidebar contains navigation links for "About", "Documentation", "Downloads", and "Community". The main content area is titled "Download for Windows" and provides instructions on how to download the latest 64-bit version of Git for Windows, including a link to the download page and information about the release date. Below this, there are sections for "Other Git for Windows downloads" listing standalone installers and portable versions in both 32-bit and 64-bit.

git-scm.com/download/win

 **git** --fast-version-control

Search entire site...

**About**

**Documentation**

**Downloads**

- GUI Clients
- Logos

**Community**

The entire **Pro Git book** written by Scott Chacon and Ben Straub is available to [read online for free](#). Dead tree versions are available on

## Download for Windows

[Click here to download](#) the latest (2.35.1) **64-bit** version of **Git for Windows**. This is the most recent [maintained build](#). It was released **26 days ago**, on 2022-02-01.

### Other Git for Windows downloads

**Standalone Installer**

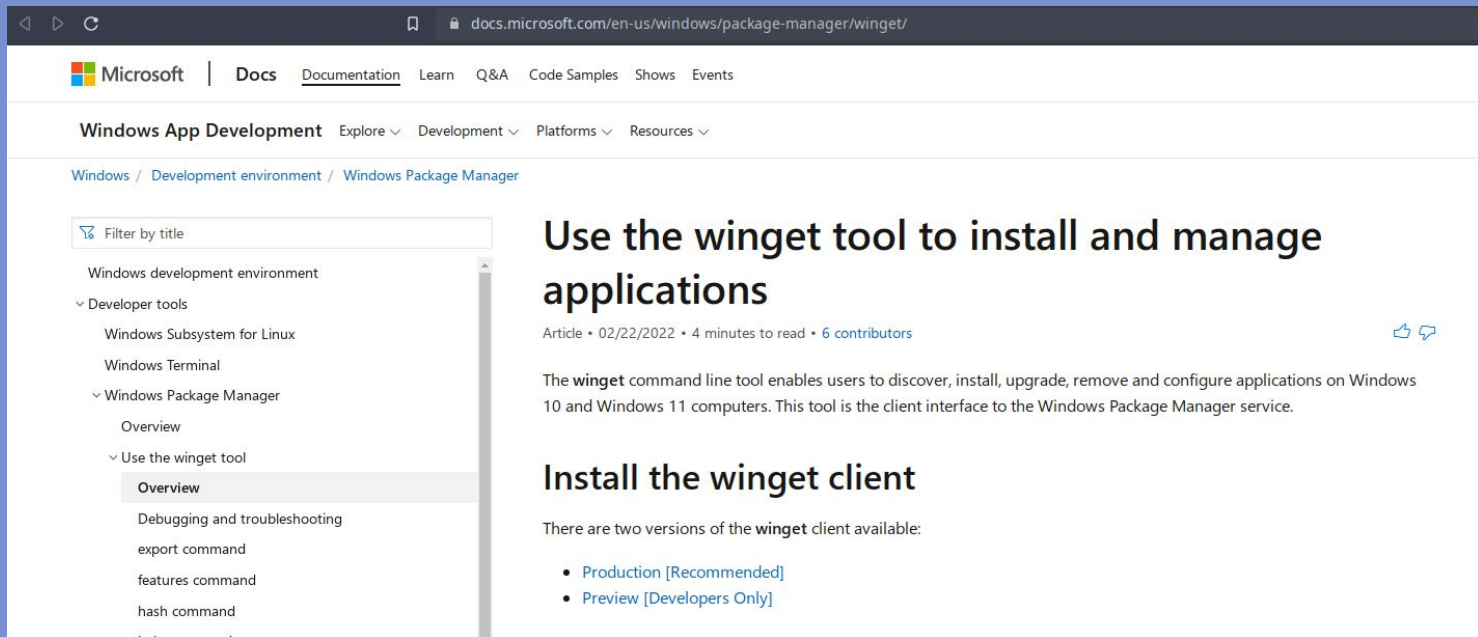
- [32-bit Git for Windows Setup.](#)
- [64-bit Git for Windows Setup.](#)

**Portable ("thumbdrive edition")**

- [32-bit Git for Windows Portable.](#)
- [64-bit Git for Windows Portable.](#)

# Instalando Git en Windows

<https://docs.microsoft.com/en-us/windows/package-manager/winget/>



The screenshot shows a web browser displaying the Microsoft Docs page for 'Use the winget tool to install and manage applications'. The page includes a navigation menu with 'Docs', 'Documentation', 'Learn', 'Q&A', 'Code Samples', 'Shows', and 'Events'. The main content area features a search bar, a left-hand navigation pane with a tree view, and the main article text. The article title is 'Use the winget tool to install and manage applications', dated 02/22/2022, with a 4-minute read time and 6 contributors. The article summary states that the winget command line tool enables users to discover, install, upgrade, remove, and configure applications on Windows 10 and Windows 11 computers. Below the summary, the section 'Install the winget client' is visible, with the text 'There are two versions of the winget client available:' followed by a bulleted list of 'Production [Recommended]' and 'Preview [Developers Only]'.

Microsoft | Docs Documentation Learn Q&A Code Samples Shows Events

Windows App Development Explore Development Platforms Resources

Windows / Development environment / Windows Package Manager

Filter by title

- Windows development environment
- Developer tools
  - Windows Subsystem for Linux
  - Windows Terminal
  - Windows Package Manager
    - Overview
    - Use the winget tool
      - Overview**
      - Debugging and troubleshooting
      - export command
      - features command
      - hash command
      - help command

## Use the winget tool to install and manage applications

Article • 02/22/2022 • 4 minutes to read • 6 contributors

The **winget** command line tool enables users to discover, install, upgrade, remove and configure applications on Windows 10 and Windows 11 computers. This tool is the client interface to the Windows Package Manager service.

### Install the winget client

There are two versions of the **winget** client available:

- [Production \[Recommended\]](#)
- [Preview \[Developers Only\]](#)

```
winget install --id Git.Git -e --source winget
```

# Instalando Git en Linux

**Debian** `sudo apt update && sudo apt install git`

**Fedora** `sudo dnf -y update && sudo dnf -y install git`

**Arch** `sudo pacman -Sy git`



# Comandos Básicos

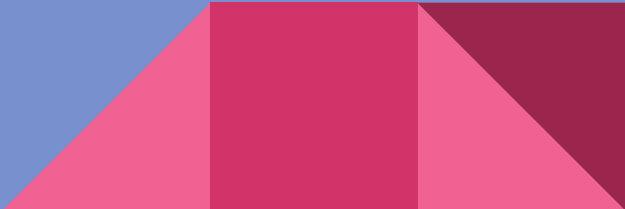


**git**

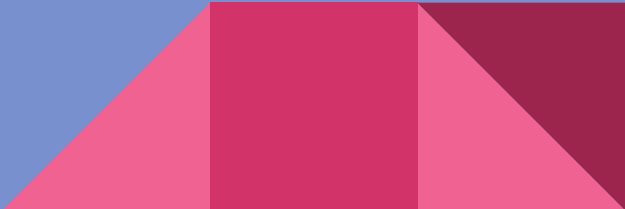
# Crear repositorios y commits

- git init: inicializa un repositorio de GIT en la carpeta donde se ejecute el comando.
- git add: añade los archivos especificados al área de preparación (staging).
- git commit -m "commit description": confirma los archivos que se encuentran en el área de preparación y los agrega al repositorio.
- git commit -am "commit description": añade al staging area y hace un commit mediante un solo comando. (No funciona con archivos nuevos)

# Crear repositorios y commits

- git status: ofrece una descripción del estado de los archivos (untracked, ready to commit, nothing to commit).
  - git rm (. -r, filename) (--cached): remueve los archivos del index.
  - git config --global user.email <tu@email.com>: configura un email.
  - git config --global user.name <Nombre como se verá en los commits>: configura un nombre.
  - git config --list: lista las configuraciones.
- 

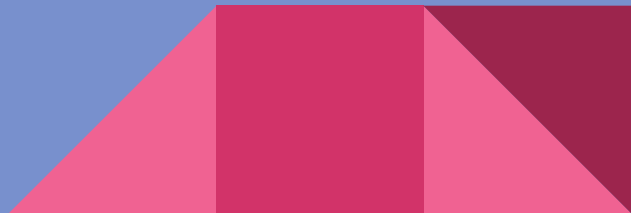
# Analizar cambios en los archivos de un proyecto Git

- *git log*: lista de manera descendente los commits realizados.
  - *git show filename*: permite ver la historia de los cambios en un archivo.
  - *git diff <commit1> <commit2>*: compara diferencias entre en cambios confirmados.
- 



# Analizar cambios en los archivos de un proyecto Git

- *git log --stat*: además de listar los commits, muestra la cantidad de bytes añadidos y eliminados en cada uno de los archivos modificados.
- *git log --all --graph --decorate --oneline*: muestra de manera comprimida toda la historia del repositorio de manera gráfica y embellecida.

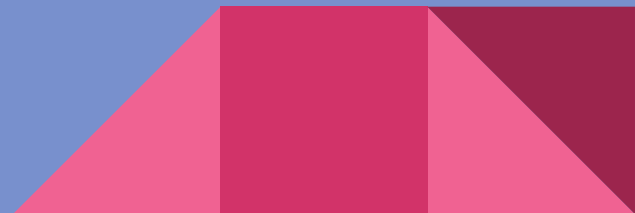


# Volver en el tiempo con branches y checkout

- `git reset <commit> --soft/hard`: regresa al commit especificado, eliminando todos los cambios que se hicieron después de ese commit.
- `git checkout <commit/branch> <filename>`: permite regresar al estado en el cual se realizó un commit o branch especificado, pero no elimina lo que está en el staging area.

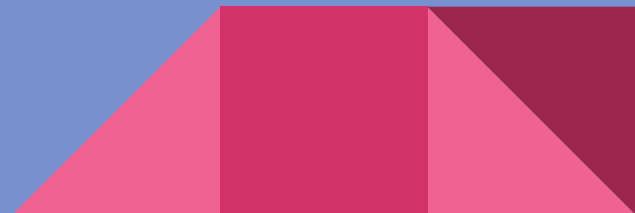
# Git Rm

Este comando nos ayuda a eliminar archivos de Git sin eliminar su historial del sistema de versiones. Esto quiere decir que si necesitamos recuperar el archivo solo debemos “viajar en el tiempo” y recuperar el último commit antes de borrar el archivo en cuestión.



# Git RM

- `git rm --cached <archivo/s>`: Elimina los archivos del área de Staging y del próximo commit pero los mantiene en nuestro disco duro.
- `git rm --force <archivo/s>`: Elimina los archivos de Git y del disco duro. Git siempre guarda todo, por lo que podemos acceder al registro de la existencia de los archivos, de modo que podremos recuperarlos si es necesario (pero debemos usar comandos más avanzados).

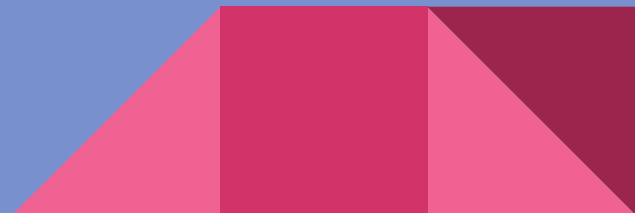


# Git Reset

## Git reset

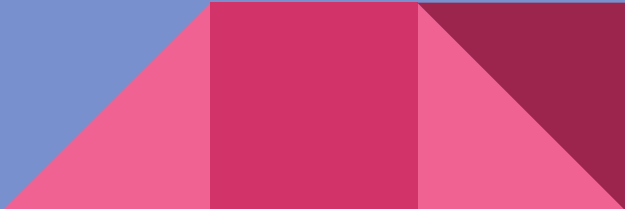
Con git reset volvemos al pasado sin la posibilidad de volver al futuro. Borramos la historia y la debemos sobrescribir.

- `git reset --soft`: Vuelve el branch al estado del commit especificado, manteniendo los archivos en el directorio de trabajo y lo que haya en staging considerando todo como nuevos cambios. Así podemos aplicar las últimas actualizaciones a un nuevo commit.



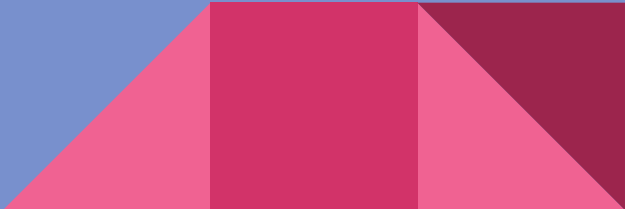
# Git Reset

## Git reset

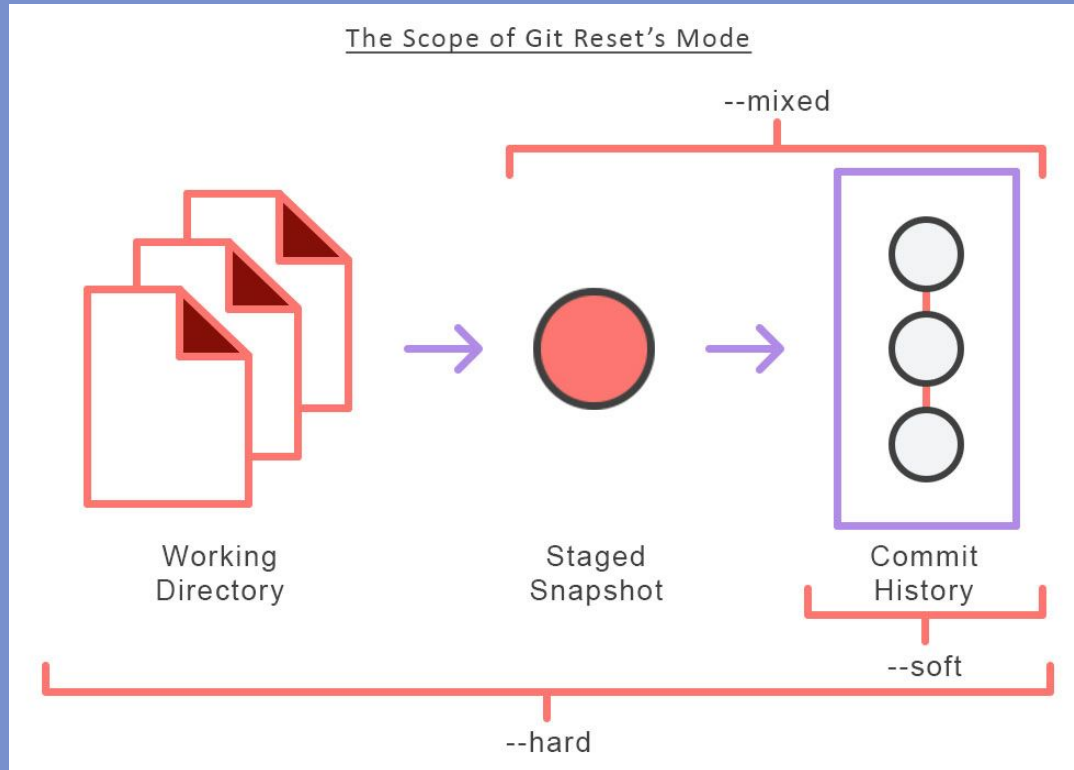
- `git reset --hard`: Borra absolutamente todo. Toda la información de los commits y del área de staging se borra del historial.
  - `git reset HEAD`: No borra los archivos ni sus modificaciones, solo los saca del área de staging, de forma que los últimos cambios de estos archivos no se envíen al último commit. Si se cambia de opinión se los puede incluir nuevamente con `git add`.
- 

# Git Reset

## Git reset

- `git reset --hard`: Borra absolutamente todo. Toda la información de los commits y del área de staging se borra del historial.
  - `git reset HEAD`: No borra los archivos ni sus modificaciones, solo los saca del área de staging, de forma que los últimos cambios de estos archivos no se envíen al último commit. Si se cambia de opinión se los puede incluir nuevamente con `git add`.
- 

# Git Reset





# Ramas o Branches

Al crear una nueva rama se copia el último commit en esta nueva rama. Todos los cambios hechos en esta rama no se reflejarán en la rama master hasta que hagamos un merge.

- *git branch <new branch>*: crea una nueva rama.
  - *git checkout <branch name>*: se mueve a la rama especificada.
  - *git merge <branch name>*: fusiona la rama actual con la rama especificada y crea un nuevo commit de esta fusión.
  - *git branch*: lista las ramas creadas.
- 