



ES UNA HERRAMIENTA DE CONTROL DE VERSIONES QUE PERMITE ALOJAR LOS DIFERENTES PROYECTOS EN LOS QUE ESTÉN TRABAJANDO, PERMITIENDO GESTIONAR DE FORMA MUY SIMPLE Y GRÁFICA LAS DIFERENTES VERSIONES O MODIFICACIONES QUE SE VAYAN REALIZANDO EN CADA UNO DE SUS PROYECTOS.

- Fue desarrollada por Linux Torvalds creador del núcleo Linux.

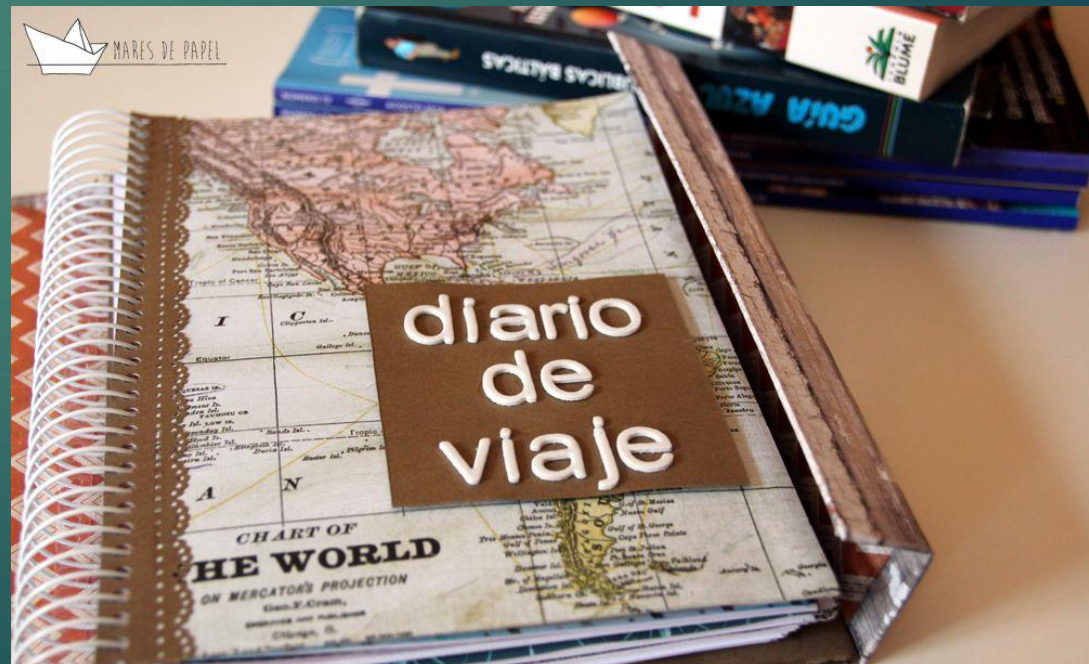


# Definición, clasificación y funcionamiento:

- ▶ Se llama control de versiones a la gestión de los diversos cambios que se realizan sobre los elementos de algún producto o una configuración del mismo. Una versión, revisión o edición de un producto, es el estado en el que se encuentra dicho producto en un momento dado de su desarrollo o modificación. Aunque un sistema de control de versiones puede realizarse de forma manual, es muy aconsejable disponer de herramientas que faciliten esta gestión dando lugar a los llamados sistemas de control de versiones o SVC (del inglés System Version Control). Estos sistemas facilitan la administración de las distintas versiones de cada producto desarrollado, así como las posibles especializaciones realizadas (por ejemplo, para algún cliente específico).
- ▶ ***Algunos Ejemplos de este tipo de herramientas son entre otros:***
- ▶ CVS, Subversion, SourceSafe, ClearCase, Darcs, Bazaar , Plastic SCM, **Git**, Mercurial, Perforce.

# Git es como un diario de viaje:

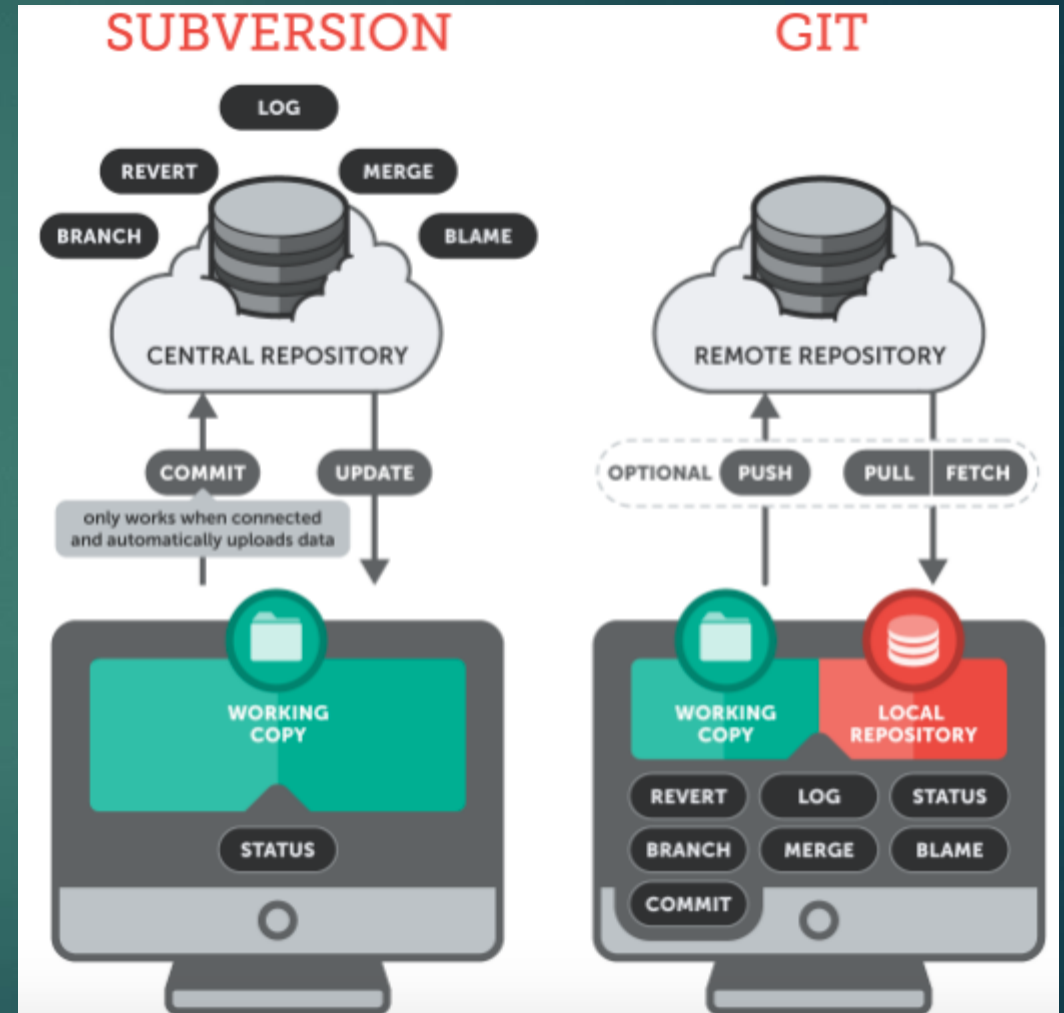
- ▶ Una Bitácora de un Capitán que va dejando todo los registro de las rutas de sus viajes.
- ▶ En este caso Git les va a permitir ir registrando cada modificación que vayan realizando a su proyecto.
- ▶ Ese registro se va a llevar a cabo con los commit.





# Ventajas de Git:

- ▶ Podemos trabajar a nivel local sin tener que comprometer el repositorio principal a cada cambio que realicemos.
- ▶ Gracias a este sistema, los desarrolladores del proyecto podrán trabajar de forma independiente hasta el momento en el que tengan que poner en común con el resto del equipo su código, controlando los cambios en las versiones y mejorando el seguimiento al desarrollo.



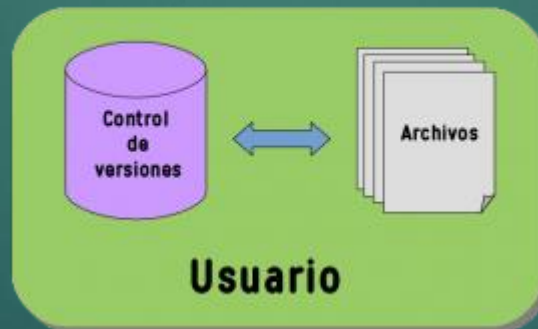
# Terminología:

- ▶ Repositorio ("repository") El repositorio es el lugar en el que se almacenan los datos actualizados e históricos de cambios.

# PUEDEN SER :

1. **Locales:** Los cambios son guardados localmente y no se comparten con nadie. Esta arquitectura es la antecesora de las dos siguientes. El modelo local utiliza una copia de la base de control de versiones y una copia de los archivos del proyecto. Este tipo es el más sencillo y no es recomendable cuando se trabaja en equipo ya que todos tienen que acceder a los mismos archivos.

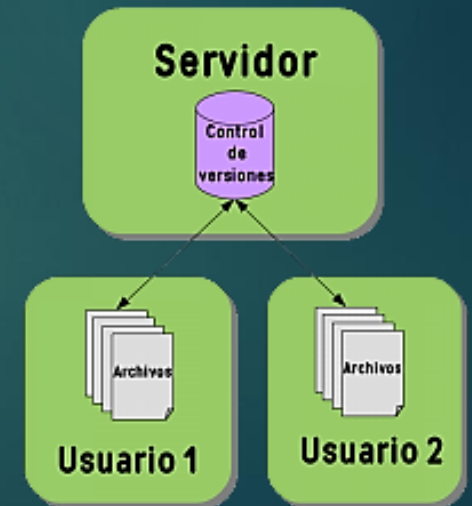
**Modelo local**



# Centralizados:

- ▶ Existe un repositorio centralizado de todo el código, del cual es responsable un único usuario (o conjunto de ellos). Se facilitan las tareas administrativas a cambio de reducir flexibilidad, pues todas las decisiones fuertes (como crear una nueva rama) necesitan la aprobación del responsable.
- ▶ Se realiza en un servidor que se encargará de recibir y dar los cambios realizados en el archivo a cada uno de los usuarios.

Modelo centralizado

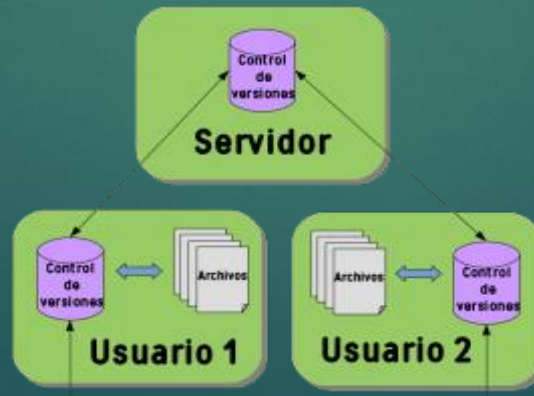




# Distribuidos:

- ▶ Cada usuario tiene su propio repositorio. Los distintos repositorios pueden intercambiar y mezclar revisiones entre ellos. Es frecuente el uso de un repositorio, que está normalmente disponible, que sirve de punto de sincronización de los distintos repositorios locales.
- ▶ El modelo distribuido es el más utilizado, en este caso cada usuario tiene un control de versiones propio que a su vez son manejadas por el servidor.

**Modelo distribuido**




# Repaso de la instalación:

- ▶ **Instalar GIT en diferentes sistemas operativos.**
- ▶ Linux (Debian): `sudo apt install git`
- ▶ Linux (Centos): `sudo yum install git`
- ▶ MacOS: `brew install git` (O descargar instalador).
- ▶ Windows: Descargar Instalador.
- ▶ ***“En el proceso de instalación en Windows, es importante seleccionar la opción de Git Bash”***

Git


git-scm.com


 **git** --distributed-even-if-your-workflow-isnt


Search entire site...


Git is a [free and open source](#) distributed version control system designed to handle everything from small to very large projects with speed and efficiency.


Git is [easy to learn](#) and has a [tiny footprint with lightning fast performance](#). It outclasses SCM tools like Subversion, CVS, Perforce, and ClearCase with features like [cheap local branching](#), convenient [staging areas](#), and [multiple workflows](#).




**About**  
The advantages of Git compared to other source control systems.


**Documentation**  
Command reference pages, Pro Git book content, videos and other material.


**Downloads**  
GUI clients and binary releases for all major platforms.


**Community**  
Get involved! Bug reporting, mailing list, chat, development and more.

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

Latest source Release  
**2.35.1**  
[Release Notes \(2022-01-29\)](#)  
Download for Windows







 [Windows GUIs](#) [Tarballs](#)

 [Mac Build](#) [Source Code](#)

Companies & Projects Using Git

GoogleFACEBOOKMicrosoft

  **NETFLIX**

# ¿Cómo verificar si ya está instalado GIT?

- ▶ En Linux o MacOS: abrir la terminal y ejecutar el comando git
- ▶ Windows: buscar o ejecutar el programa Git Bash.
- ▶ Para conocer que versión de GIT está instalada.
- ▶ `git --version`

MINGW64:/c/Users/Paloma

Paloma@DESKTOP-CEETV4P MINGW64 ~ (master)

\$ git --version

git version 2.35.1.windows.1

Paloma@DESKTOP-CEETV4P MINGW64 ~ (master)

\$




# Ejecutar desde Git Bash:

- ▶ Comenzamos configurando Git en nuestra ventana de comando Git bash. Digitamos el siguiente comando y luego damos **Enter**.



# Comandos para la configuración de GIT.

- ▶ **Git config:** es un comando que permite configurar todos los aspectos de cómo funcionará Git de manera local, global o system.
- ▶ **git config** mantiene su valor entre actualizaciones. Por lo tanto, se debe configurar solo una vez. Todos los archivos de configuración tienen la misma sintaxis, pero un alcance diferente. Esto ofrece mucha flexibilidad.
- ▶ Existe 3 comandos para el almacenamiento:
- ▶ **Local.**
- ▶ **Global.**
- ▶ **System.**
- ▶ Además, es importante recordar que cada nivel anula los valores del nivel anterior.
- ▶ Prioridad:
- ▶ **Local > Global > System**

- 
- ▶ **Local:** las configuraciones locales están disponibles solo para el repositorio actual. Puede hacer que git lea y escriba desde la computadora que se está utilizando solo localmente .

### **git config --local**

- ▶ **Global:** las configuraciones globales están disponibles para los usuarios actuales para todos los proyectos.

### **git config --global**

- ▶ **System:** ésta configuración están disponibles para cada usuario en el sistema y se requiere que tengas permisos de administración.

### **git config --system**

## En nuestra primera actividad de Git colocaremos nuestras firmas con nombre y apellido y correo con configuración local

```
MINGW64:/c/Users/Paloma
Paloma@DESKTOP-CEETV4P MINGW64 ~ (master)
$ git config --local
usage: git config [<options>]

Config file location
  --global      use global config file
  --system      use system config file
  --local       use repository config file
  --worktree    use per-worktree config file
  -f, --file <file> use given config file
  --blob <blob-id> read config from given blob object

Action
  --get          get value: name [value-pattern]
  --get-all     get all values: key [value-pattern]
  --get-regexp   get values for regexp: name-regex [value-pattern]
  --get-urlmatch get value specific for the URL: section[.var] URL
  --replace-all replace all matching variables: name value [value-pattern]

  --add          add a new variable: name value
  --unset        remove a variable: name [value-pattern]
  --unset-all   remove all matches: name [value-pattern]
  --rename-section rename section: old-name new-name
  --remove-section remove a section: name
  -l, --list     list all
  --fixed-value  use string equality when comparing values to 'value-pattern'

  -e, --edit     open an editor
  --get-color    find the color configured: slot [default]
  --get-colorbool find the color setting: slot [stdout-is-tty]

Type
  -t, --type <> value is given this type
  --bool        value is "true" or "false"
  --int         value is decimal number
  --bool-or-int value is --bool or --int
  --bool-or-str value is --bool or string
  --path        value is a path (file or directory name)
  --expiry-date value is an expiry date

Other
  -z, --null     terminate values with NUL byte
  --name-only    show variable names only
  --includes     respect include directives on lookup
  --show-origin  show origin of config (file, standard input, blob, command line)
  --show-scope   show scope of config (worktree, local, global, system, command)
  --default <value> with --get, use default value when missing entry

Paloma@DESKTOP-CEETV4P MINGW64 ~ (master)
$ |
```

# Para configurar el usuario que va a escribir en la bitácora.

- ▶ Esto mostrará datos de la identidad con la que hemos creado el usuario así como otros datos de la máquina con la que estamos trabajando. Con git config podremos configurar git para registrar diferente identidades, por si usamos un ordenador para diferentes desarrolladores o si nos interesa registrar los cambios bajo diferentes nombres. Esto lo podemos realizar mediante los comandos:



**git config --local user.name** "Nombre Apellido"

**git config --local user.email** "tuemail@ejemplo.com" (para configurar el usuario que va a escribir en la "Bitácora" desde una máquina).

```
Paloma@DESKTOP-CEETV4P MINGW64 ~ (master)
$ git config --local user.name natalia lucero

Paloma@DESKTOP-CEETV4P MINGW64 ~ (master)
$ git config --local user.email natty lucero@gamil.com

Paloma@DESKTOP-CEETV4P MINGW64 ~ (master)
$ |
```

Para traer todas las actualizaciones que hemos realizado

**git config --list**

**git config --l (atajo)**

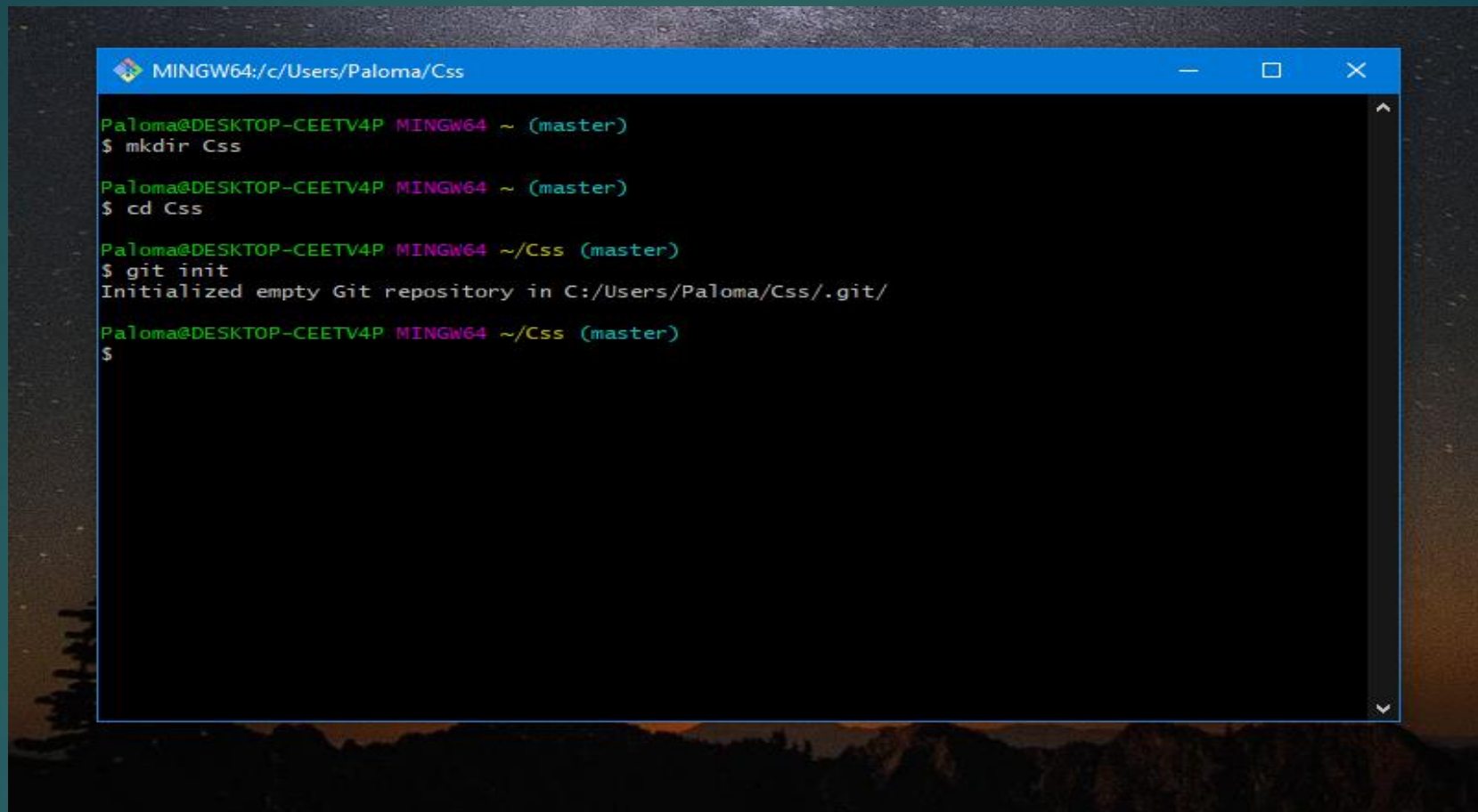
**Escribir Clear para limpiar nuestra terminal o salir con q**

```
Paloma@DESKTOP-CEETV4P MINGW64 ~ (master)
$ git config --list
diff.astextplain.textconv=astextplain
filter.lfs.clean=git-lfs clean -- %f
filter.lfs.smudge=git-lfs smudge -- %f
filter.lfs.process=git-lfs filter-process
filter.lfs.required=true
http.sslbackend=openssl
http.sslcainfo=C:/Program Files/Git/mingw64/ssl/certs/ca-bundle.crt
core.autocrlf=true
core.fscache=true
core.symlinks=false
core.editor=nano.exe
pull.rebase=false
credential.helper=manager-core
credential.https://dev.azure.com.usehttppath=true
init.defaultbranch=master
user.name=Natalia Lucero
user.email=nattyLucero777@gmail.com
core.repositoryformatversion=0
core.filemode=false
core.bare=false
core.logallrefupdates=true
core.symlinks=false
core.ignorecase=true
gui.wmstate=normal
gui.geometry=893x435+997+208 175 196

Paloma@DESKTOP-CEETV4P MINGW64 ~ (master)
$
```

## ¿Cómo creas tu primer repositorio?

- ▶ El comando **ls** muestra todos los archivos en carpeta.
- ▶ El comando **mkdir** (make directory) crea una carpeta.
- ▶ Ejemplo: **mkdir** nombre-del-proyecto
- ▶ **cd** nombre-del-proyecto/ (para entrar a la carpeta existente).
- ▶ **git init** (una vez dentro de la carpeta, ejecutar este comando).



```
MINGW64: c:/Users/Paloma/Css

Paloma@DESKTOP-CEETV4P MINGW64 ~ (master)
$ mkdir Css

Paloma@DESKTOP-CEETV4P MINGW64 ~ (master)
$ cd Css

Paloma@DESKTOP-CEETV4P MINGW64 ~/Css (master)
$ git init
Initialized empty Git repository in C:/Users/Paloma/Css/.git/

Paloma@DESKTOP-CEETV4P MINGW64 ~/Css (master)
$
```

“Todos los archivos y carpetas que empiezan con . (punto) son archivos o carpetas ocultos”

# Actividad 1: Git: tarea para próxima clase.

- ▶ Crear un carpeta de Git con usuario local con sus nombre, apellido y el e-mail.
- ▶ Todas las carpetas que van a crear para trabajar en Python o Java, deben crearse desde el repositorio de Git.

Ej: **mkdir python**

**git init**