

MÓDULO: ENTORNOS DE DESARROLLO

1º DAM- DISEÑO APLICACIONES MULTIPLATAFORMA

Control de versiones.
GIT y GitHub.



Control de versiones.

Concepto

El **control de versiones** es una herramienta software que registra los cambios realizados sobre un archivo o conjunto de archivos a lo largo del tiempo, de modo que puedas recuperar versiones específicas más adelante.

Así por ejemplo, un diseñador gráfico quiere mantener cada versión de su imagen y tener acceso a ellas viendo las modificaciones realizadas en cada versión. Incluso un escritor o poeta puede necesitar hacer uso de un sistema de control de versiones. Y por supuesto para el desarrollador de software, que es el que realmente ideó la creación de esta herramienta.

Versión de un fichero → Estado del fichero en un momento concreto de su vida.

Probablemente una forma de realizar un control de versiones “*doméstico*”, sería guardar cada versión con un nombre diferente, incluso añadiendo fecha y hora:

- Trabajo.txt
- Trabajo1.txt
- Trabajo2 13-05-2019
- Trabajo2 14-05-2019 10:30
- Trabajo2 14-05-2019 14:40

De esta forma, hemos conseguido una amalgama de ficheros que pueden llevarnos a confusión y error por la posible pérdida de alguno por sobreescritura.

Control de versiones. Concepto

Control de versiones. GIT y GitHub

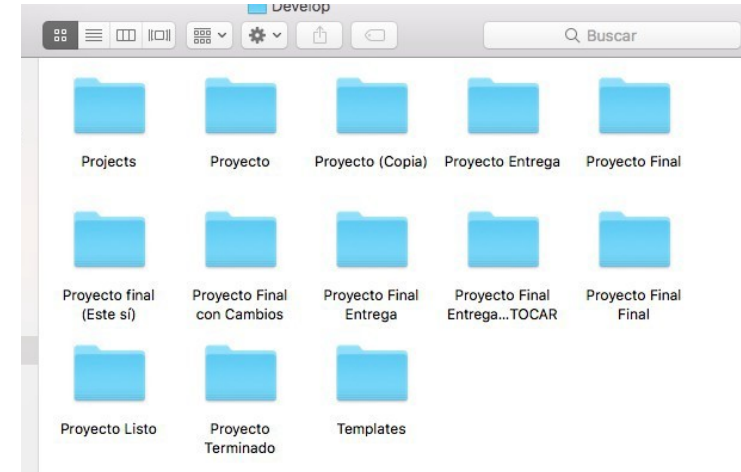
El control de versiones es una de las tareas fundamentales para la administración de un proyecto de desarrollo de software en general.

Surge de la necesidad de mantener y llevar control del código que vamos programando, conservando sus distintos estados.

Es absolutamente necesario para el trabajo en equipo, pero resulta útil incluso a desarrolladores independientes.

Aunque trabajemos solos, sabemos la necesidad de gestionar los cambio entre distintas versiones de un mismo código. Todos los programadores, se han visto en la necesidad de tener dos o más copias de un mismo archivo, para no perder su estado anterior cuando vamos a introducir diversos cambios.

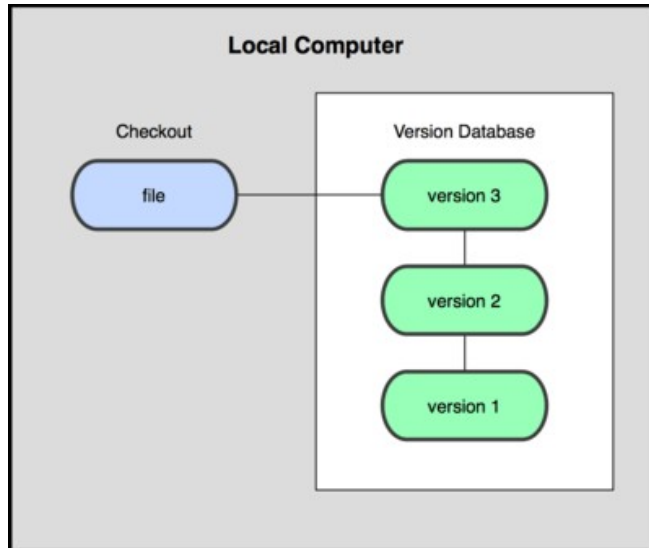
Para ir solucionando esto en principio se usó la copia con diferentes nombres, pero aunque esta acción nos sirva para salir del paso, no es lo más cómodo ni mucho menos lo más práctico.



Control de versiones locales.

- Sistemas de control de versiones locales.

Para hacer frente a este problema, los programadores desarrollaron hace tiempo **VCSs** (Version Control System) **locales** que contenían una simple base de datos en la que se llevaba registro de todos los cambios realizados sobre los archivos.



- Una de las herramientas de control de versiones más popular fue un sistema llamado RCS (Revision Control System). Esta herramienta funciona guardando conjuntos de parches (es decir, las diferencias entre archivos) de una versión a otra en un formato especial en disco; puede entonces ver cómo era un archivo en cualquier momento sumando los distintos parches.

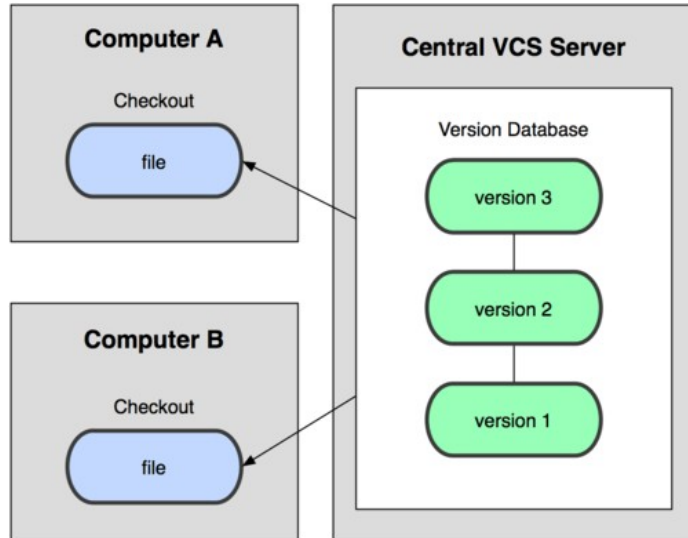
Aquí el control de versiones se llevaba a cabo en el computador de cada uno de los desarrolladores y **no existía una manera eficiente de compartir el código** entre ellos.

Control de versiones centralizados.

- Sistemas de control de versiones centralizados.

También conocidos por las siglas CVCSS (Centralized Version Control Systems)

Para facilitar la colaboración de múltiples desarrolladores en un solo proyecto, los VCSs evolucionaron y en vez de almacenar los cambios y versiones en el disco duro de los desarrolladores, estos se almacenaban en un **servidor central**.



- Estos sistemas, como **Subversion**, y **Perforce**, tienen un único servidor que contiene todos los archivos versionados, y varios clientes que descargan los archivos desde ese lugar central.

Durante muchos años éste ha sido el estándar para el control de versiones.

- **Ventajas y desventajas de los CVCSS frente a los VCS locales.**

- ✓ **Ventajas:**

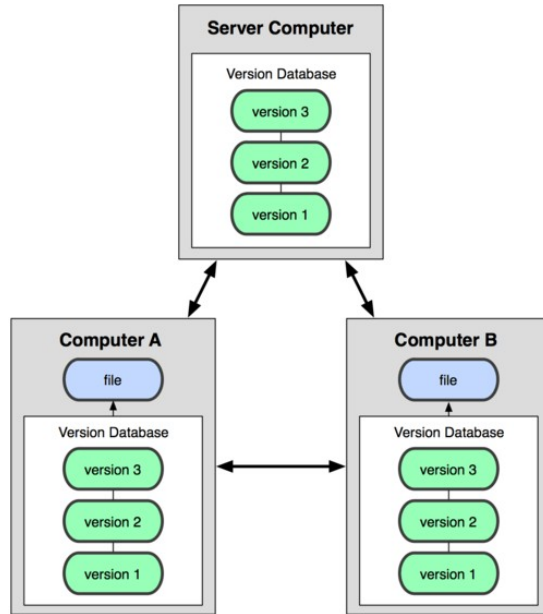
- ✓ Todo el mundo puede saber, hasta cierto punto, en qué están trabajando los otros colaboradores del proyecto.
- ✓ Los administradores tienen control detallado de qué hace cada colaborador del proyecto.
- ✓ Es más fácil de administrar que las BBDD locales de cada uno.

- ✓ **Desventajas:**

- ✓ Un fallo en el servidor centralizado. Si el servidor se cae durante un rato, entonces durante ese periodo de tiempo nadie puede colaborar o guardar cambios versionados de aquello en que están trabajando.
- ✓ Si el disco duro en el que se encuentra la base de datos central se corrompe, y no se han llevado copias de seguridad adecuadamente, se pierde todo: toda la historia del proyecto, salvo aquellas versiones que la gente pueda tener en sus máquinas locales.

- Sistemas de control de versiones distribuidos.

Los nuevos retos que se plantearon era conseguir evitar el problema de la pérdida de las versiones del proyecto centralizadas en un solo servidor además de dar la posibilidad de que varios usuarios trabajaran en un mismo archivo al mismo tiempo.



En los **sistemas de control de versiones distribuidos**, DVCSs (Distributed Version Control Systems), los clientes no descargan la última versión de los archivos sino que **replican completamente el repositorio** (almacén).

Así, si un servidor cae, y estos sistemas estaban colaborando a través de él, cualquiera de los repositorios de los clientes puede copiarse en el servidor para restaurarlo. Cada vez que se descarga una versión, en realidad se hace una copia de seguridad completa de todos los datos.

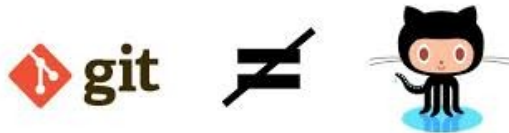
Muchos de estos sistemas tienen varios repositorios con los que trabajar, por lo que puedes colaborar con distintos grupos de gente simultáneamente dentro del mismo proyecto. Esto te permite establecer varios flujos de trabajo que no son posibles en sistemas centralizados

- **Git es un sistema de control de versiones distribuido.**

Es de código abierto y multiplataforma, por lo que se puede usar y crear repositorios locales en todos los sistemas operativos más comunes, Windows, Linux o Mac.


Para usar Git hay que instalarlo en la computadora.

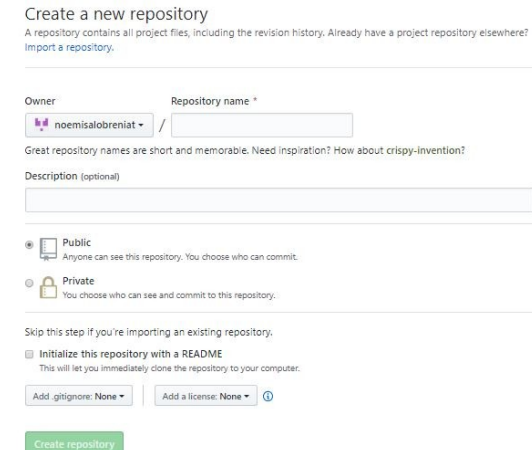
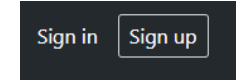
- **Github es un servicio para alojamiento de repositorios** de software gestionados por el sistema de control de versiones Git. Github es un sitio web pensado para hacer posible el compartir el código de una manera más fácil y al mismo tiempo darle popularidad a la herramienta de control de versiones en sí, que es Git.



Definición de repositorio en Github: espacio centralizado donde se almacena, organiza, mantiene y difunde información digital, habitualmente archivos informáticos, que pueden contener trabajos científicos, conjuntos de datos o software.


Crear una cuenta en GitHub.

- Acceder a github.com y pulsar *Sign up* para nuevo usuario.
- Rellenar el formulario.
- Pulsar 
- NO es necesario elegir *plan de uso* ni especificar *interés*.
- Crear el primer repositorio:
 - Nombre del repositorio
 - Descripción (opcional)



Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Owner:  noemisalobreniat / Repository name *

Great repository names are short and memorable. Need inspiration? How about crispy-invention?

Description (optional)

☒ Public
Anyone can see this repository. You choose who can commit.

☐ Private
You choose who can see and commit to this repository.

Skip this step if you're importing an existing repository.

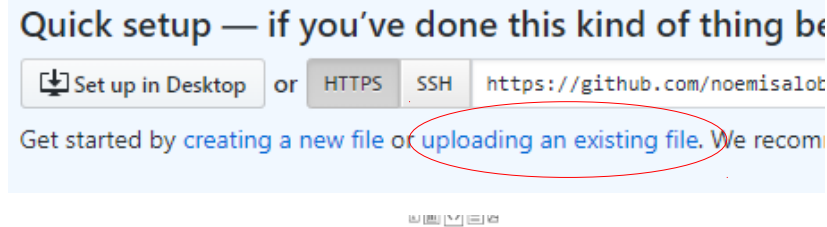
☐ Initialize this repository with a README
This will let you immediately clone the repository to your computer.

Add .gitignore: None Add a license: None ⓘ

NOTA: Desde 2019, Microsoft permite crear repositorios privados “sin pasar por caja”, con un máximo de tres colaboradores.

<https://www.muylinux.com/2019/01/08/github-free-repositorios-privados-gratuitos-ilimitados/>

Subir un archivo.



Drag files here to add them to your repository

Or [choose your files](#)

Eliminar un repositorio.

Dentro del repositorio a eliminar → Settings

Settings

Al final de la página → Delete this repository

Delete this repository

Información obtenida de:
<https://git-scm.com/>
<https://medium.com>
<https://desarrolloweb.com/>