



# 1º DAM/DAW    Sistemas Informáticos

---

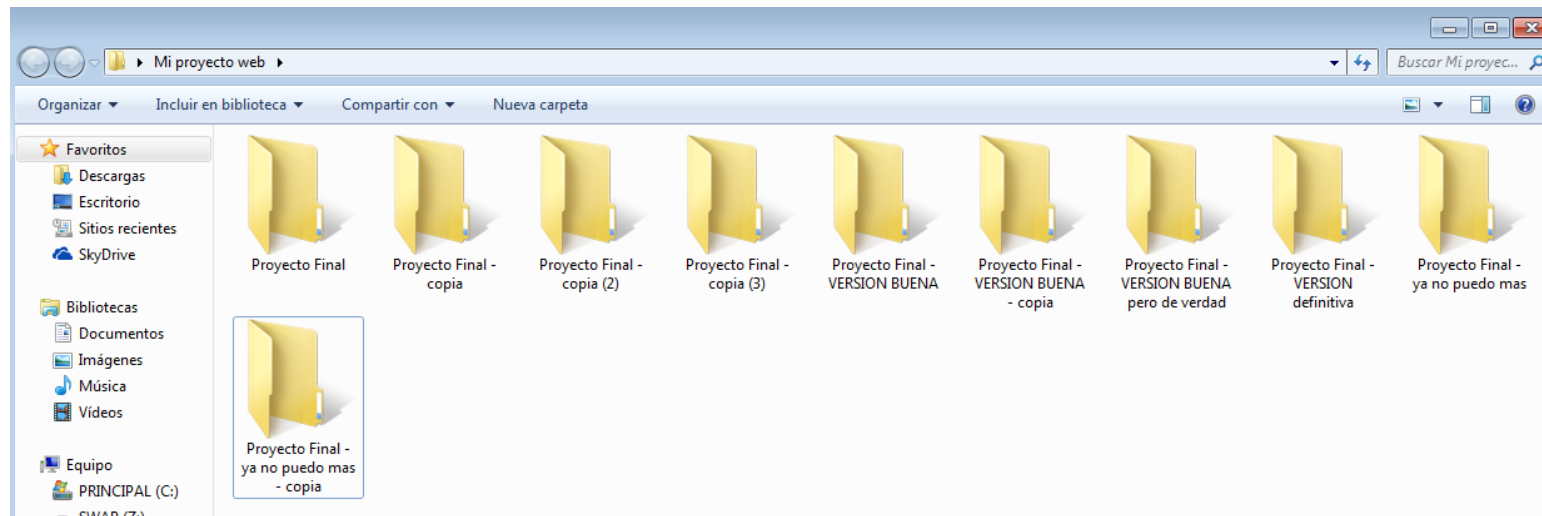
## U3. Sistemas de control de versiones - Git

### 1 - Sistemas de control de versiones - Git



## ¿Por qué debería importarte el control de versiones?

No debo ser el único que, a la hora de desarrollar un proyecto, una actividad o un simple documento, ha intentado guardar la evolución del mismo, creando una carpeta o una copia del documento cada vez que se hacía un cambio. De forma que, cuando lo hago parece que está todo clarísimo, pero cuando lo veo pasados unos días, no consigo aclararme... ¿te suena?...



## ¿Por qué debería importarte el control de versiones?

Si sólo hay una carpeta válida y trabajas tú de forma aislada, hasta puedes llegar a aclararte... Pero si son varias las carpetas válidas (versiones) y además trabajas en equipo, gestionar la información de este modo se complica exponencialmente.



## ¿Qué es un control de versiones?

- Un control de versiones consiste en **registrar todos los cambios realizados en un archivo o conjunto de archivos a lo largo del tiempo**, de modo que incluso se pueda recuperar versiones específicas más adelante.
- Una **versión** de un producto (o un fichero, o una carpeta, o un repositorio,...) es el **estado** en el que se encuentra el mismo **en un momento concreto**.
- Existen diferentes herramientas en el mercado para realizar el control de versiones.

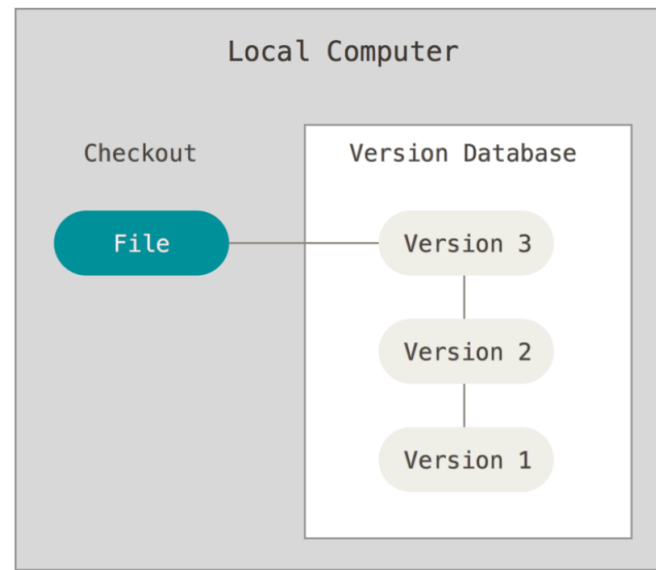


## ¿Qué es un sistema de control de versiones?

- A la **herramienta** que nos permite implementar un control de versiones, se le denomina **sistema de control de versiones (VCS Version Control System)** y nos puede llegar a ofrecer las siguientes ventajas:
  - **Rastreo de cambios** en los elementos de un proyecto.
  - **Control** sobre **diferentes versiones**.
  - **Gestión** del trabajo en **equipo**.
  - **Unificación del canal** en proyectos con **personal distribuido**.
  - Posibilidad de mantener **diferentes versiones activas**.
  - **Trazabilidad** de modificaciones y correcciones.

## Evolución de los sistemas de control de versiones

- Inicialmente se utilizaban **sistemas de control de versiones locales (LVCS)**:
  - Estos sistemas se basaban en una **base de datos local** en la que se llevaba el **registro de todos los cambios realizados en los archivos locales**.



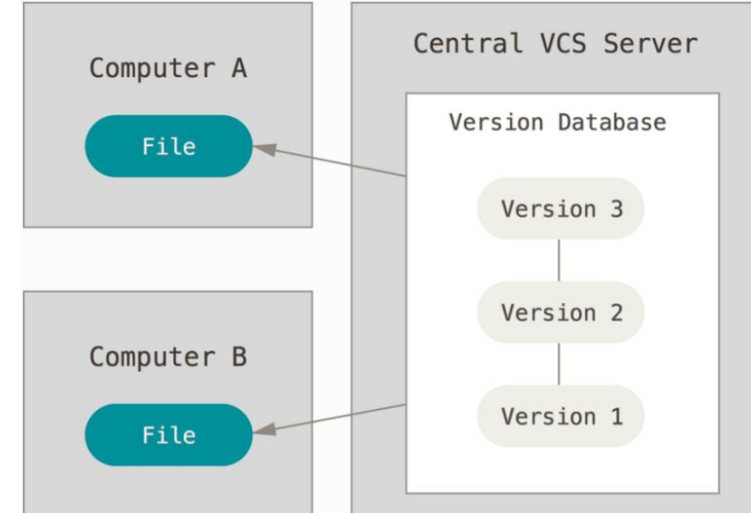
## Evolución de los sistemas de control de versiones

- Inicialmente se utilizaban **sistemas de control de versiones locales (LVCS)**:
  - El principal **inconveniente** de este tipo de sistema es que se basa en el trabajo **aislado** de un desarrollador en su máquina local. El escenario de los proyectos de ingeniería de software pronto obligó a tener que colaborar con otros desarrolladores, por lo que estos sistemas no resultaban prácticos.



## Evolución de los sistemas de control de versiones

- Posteriormente, **sistemas de control de versiones centralizados (CVCS):**
  - Para solucionar las limitaciones de los VCS locales, estos sistemas disponen de **un único servidor que contiene todos los archivos versionados y varios clientes, que descargan archivos desde ese lugar central.**





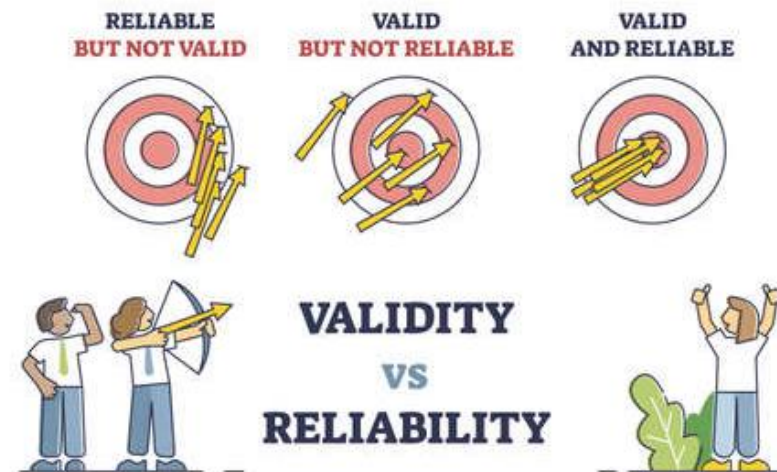
## Evolución de los sistemas de control de versiones

- Posteriormente, **sistemas de control de versiones centralizados (CVCS):**
  - Este sistema se ha utilizado durante muchos años. Ejemplos comerciales: CVS, Subversion, Perforce, etc.



## Evolución de los sistemas de control de versiones

- Posteriormente, **sistemas de control de versiones centralizados (CVCS)**:
  - El principal **inconveniente** es que todo depende de la **fiabilidad del servidor centralizado**. Si éste cae, nadie podrá guardar cambios en los archivos y si no se logra restablecer el servicio mediante una copia de seguridad, se habrá perdido la información del proyecto.

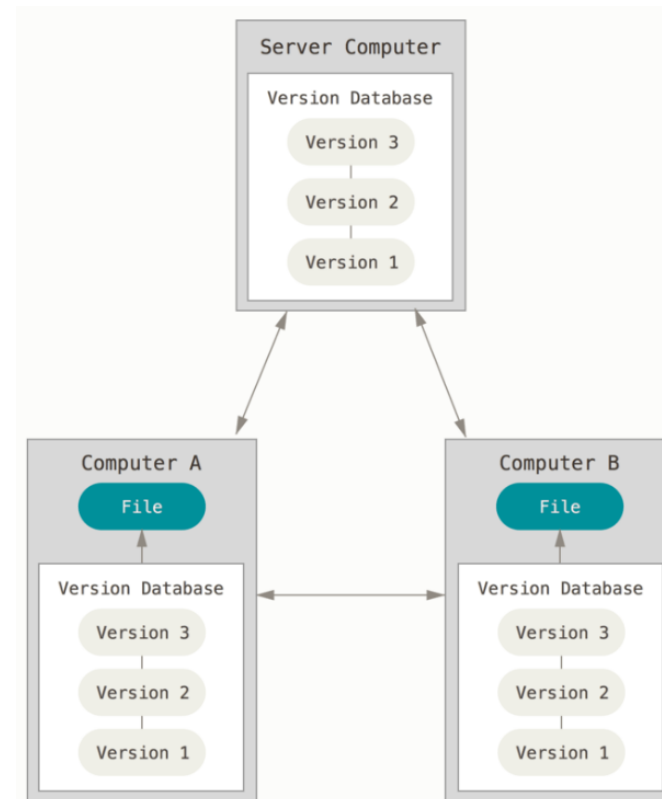


## Evolución de los sistemas de control de versiones

- Por último, **sistemas de control de versiones distribuidos (DVCS)**:
  - Para solucionar las limitaciones de los VCS centralizados, en los sistemas distribuidos los clientes no solo descargan la última copia de los archivos, sino que **se replica completamente el repositorio, con todo el historial**.
  - De esta manera, si un servidor deja de funcionar y estos sistemas estaban colaborando a través de él, cualquiera de los repositorios disponibles en los clientes puede ser copiado de nuevo al servidor, con el fin de restaurarlo. **Cada clon es realmente una copia completa de todos los datos**.

## Evolución de los sistemas de control de versiones

- Por último, **sistemas de control de versiones distribuidos (DVCS):**
  - **Git** utiliza este sistema.



## Git - Breve historia

- **Git** es, hoy en día, el sistema de control de versiones moderno **más utilizado** a nivel mundial.
- Es un proyecto de **código abierto**, maduro y con un mantenimiento activo.
- Fue desarrollado originalmente en 2005 por **Linus Torvalds** y su equipo, el famoso creador del kernel del sistema operativo Linux.



## Git - Breve historia

- El **kernel** de Linux es un **proyecto de desarrollo** de software de código abierto con un **alcance** bastante **amplio**.
- Durante la mayor parte del mantenimiento del kernel de Linux (1991-2002), los **cambios** en el software se realizaban **a través de parches y archivos**.
- En el **2002**, el **proyecto** del kernel de Linux **empezó a usar un DVCS** propietario llamado **BitKeeper**.



## Git - Breve historia

- En el **2005**, la **relación** entre la comunidad que desarrollaba el kernel de Linux y la compañía que desarrollaba BitKeeper **se vino abajo**.
- La herramienta **dejó de ser ofrecida de manera gratuita**.



## Git - Breve historia

- **Esto impulsó** a la comunidad de desarrollo de Linux y en particular a Linus Torvalds, a **desarrollar su propia herramienta** basada en algunas de las lecciones que aprendieron mientras usaban BitKeeper.
- Algunos de los **objetivos** del nuevo sistema fueron los siguientes:
  - Velocidad.
  - Diseño sencillo.
  - Gran soporte para desarrollo no lineal (miles de ramas paralelas).
  - Completamente distribuido.
  - Capacidad de manejar grandes proyectos (como el kernel de Linux).



## Git - Breve historia

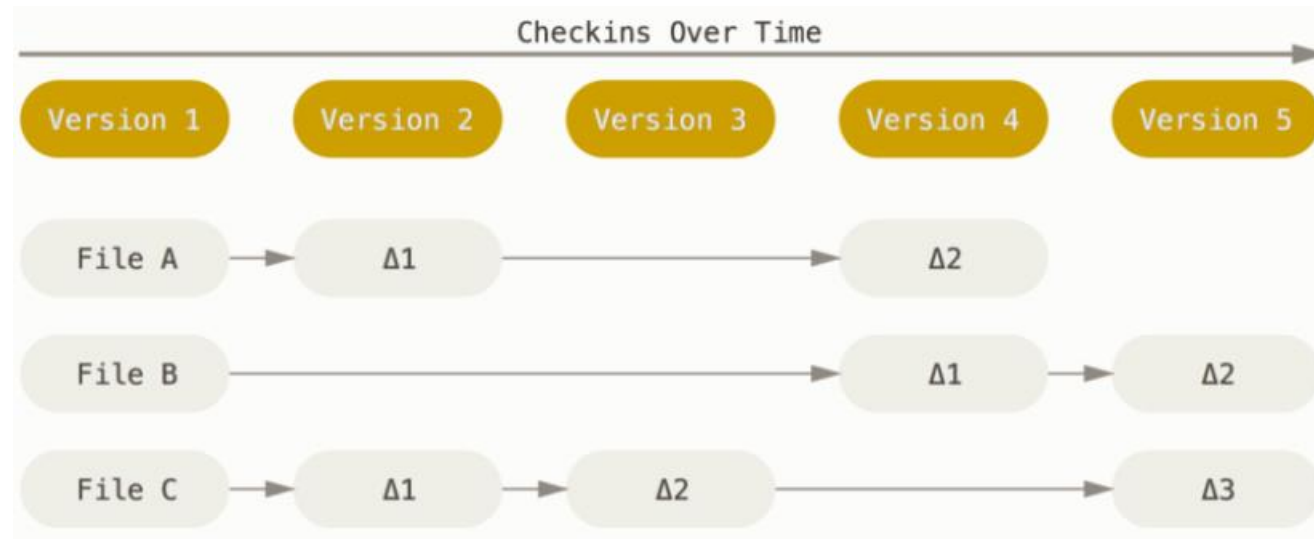
- Desde su **lanzamiento en el 2005**, Git ha ido evolucionando y madurando.
- Aporta un **sistema de ramificación** (branching) para **desarrollo no lineal**.
- Un **elevado número de proyectos de software dependen de Git** para el control de versiones, incluidos proyectos comerciales, sean o no de código abierto.
- Este sistema **se integra** y funciona **en una amplia variedad de SO's** (sistemas operativos) e **IDE's** (entornos de desarrollo integrados).



## Git - Fundamentos

### Copias de instantáneas, no diferencias.

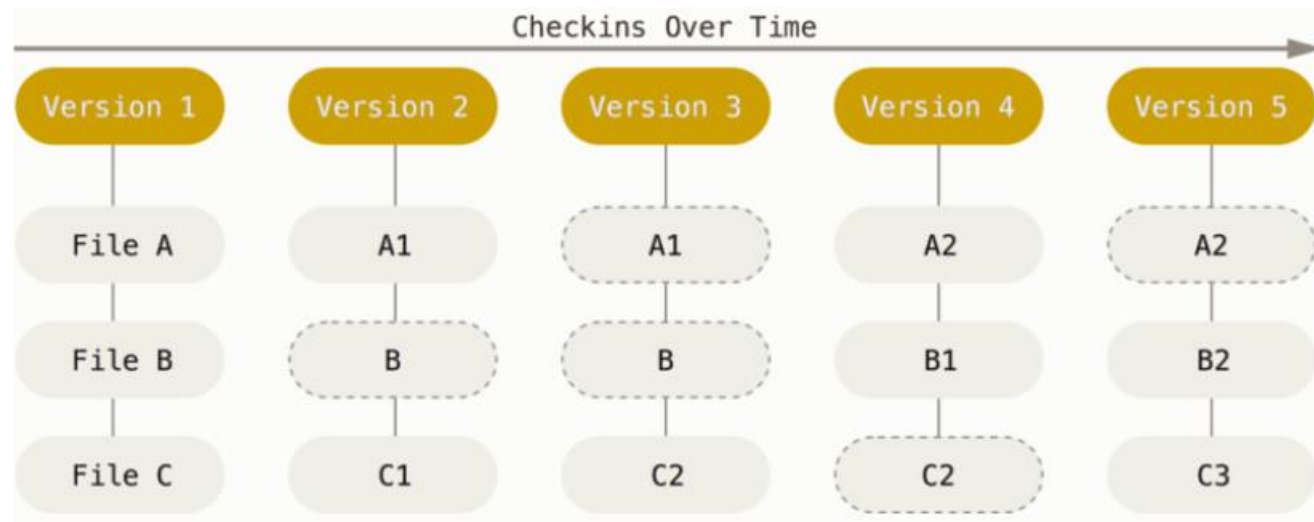
- La principal diferencia entre Git y cualquier otro VCS es la forma en la que maneja sus datos. La mayor parte de los VCS tratan la información que almacenan como un conjunto inicial de archivos, más las modificaciones hechas a cada uno de ellos a través del tiempo.



## Git - Fundamentos

### Copias de instantáneas, no diferencias.

- Git maneja sus datos como un conjunto de copias instantáneas de un sistema de archivos miniatura. Cada vez que confirmamos un cambio, o guardamos el estado de un proyecto en Git, se toma una “foto” del aspecto de todos los archivos en ese momento.



## Git - Fundamentos

**Una gran parte de las operaciones son locales.**

- La mayoría de las operaciones en Git sólo necesitan archivos y recursos locales para funcionar. De modo que **evitamos el coste temporal adicional del retardo de la red** en muchas de las operaciones.
- Dado que **tenemos toda la historia del proyecto en las instantáneas del disco local**, la mayoría de las **operaciones son inmediatas**.
- Podemos hacer prácticamente todo, **aunque no dispongamos de conexión**.



## Git - Fundamentos

### Integridad.

- Cualquier cambio, en un proyecto que use Git, es identificado y verificado mediante una **suma de comprobación (checksum)**.
- Una suma de comprobación sirve para verificar si hay algún cambio en la información, mediante discrepancias entre los valores obtenidos al hacer una comprobación previa y otra posterior.
- **Es imposible cambiar el contenido de cualquier archivo o directorio de un proyecto sin que Git lo sepa.** No se puede perder información durante una transmisión, o sufrir corrupción de archivos, sin que Git sea capaz de detectarlo.

## Git - Fundamentos

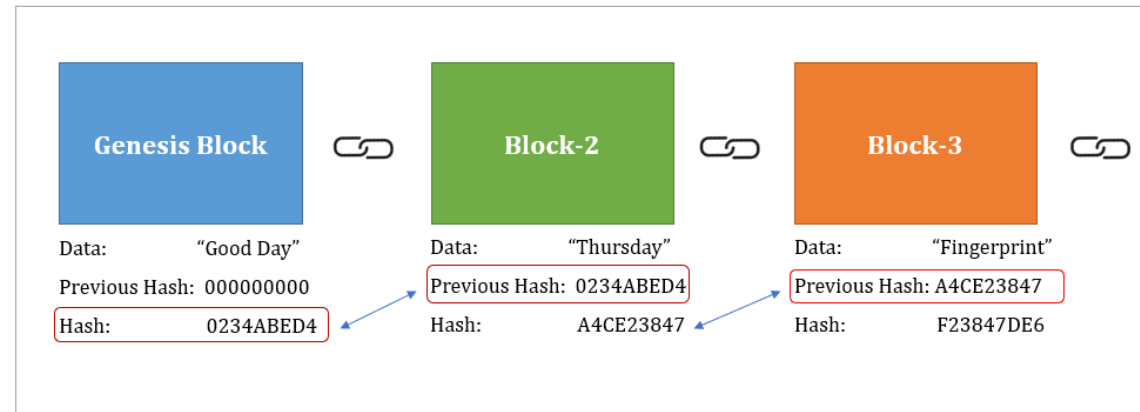
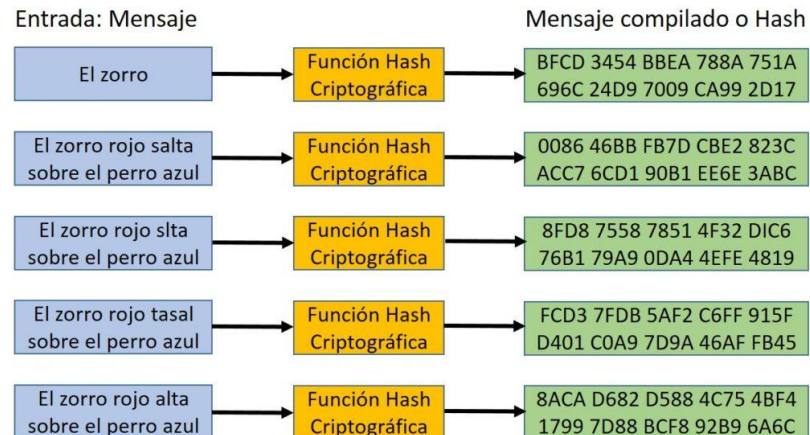
### Integridad.

- El mecanismo que usa Git para generar esta suma de comprobación se conoce como **función resumen (hash function) SHA-1**.
- Se trata de una **cadena de 40 caracteres hexadecimales** (0-9 y a-f), y se calcula basándose en los contenidos del archivo y/o estructura del directorio en Git.
- Ejemplo: 24b9da6552252987aa493b52f8696cd6d3b00373
- Estos valores hash aparecen “por todos lados” en Git, porque son usados con mucha frecuencia. De hecho, **internamente Git guarda los archivos por el valor hash de sus contenidos**, no por el nombre de los archivos.

## Git - Fundamentos

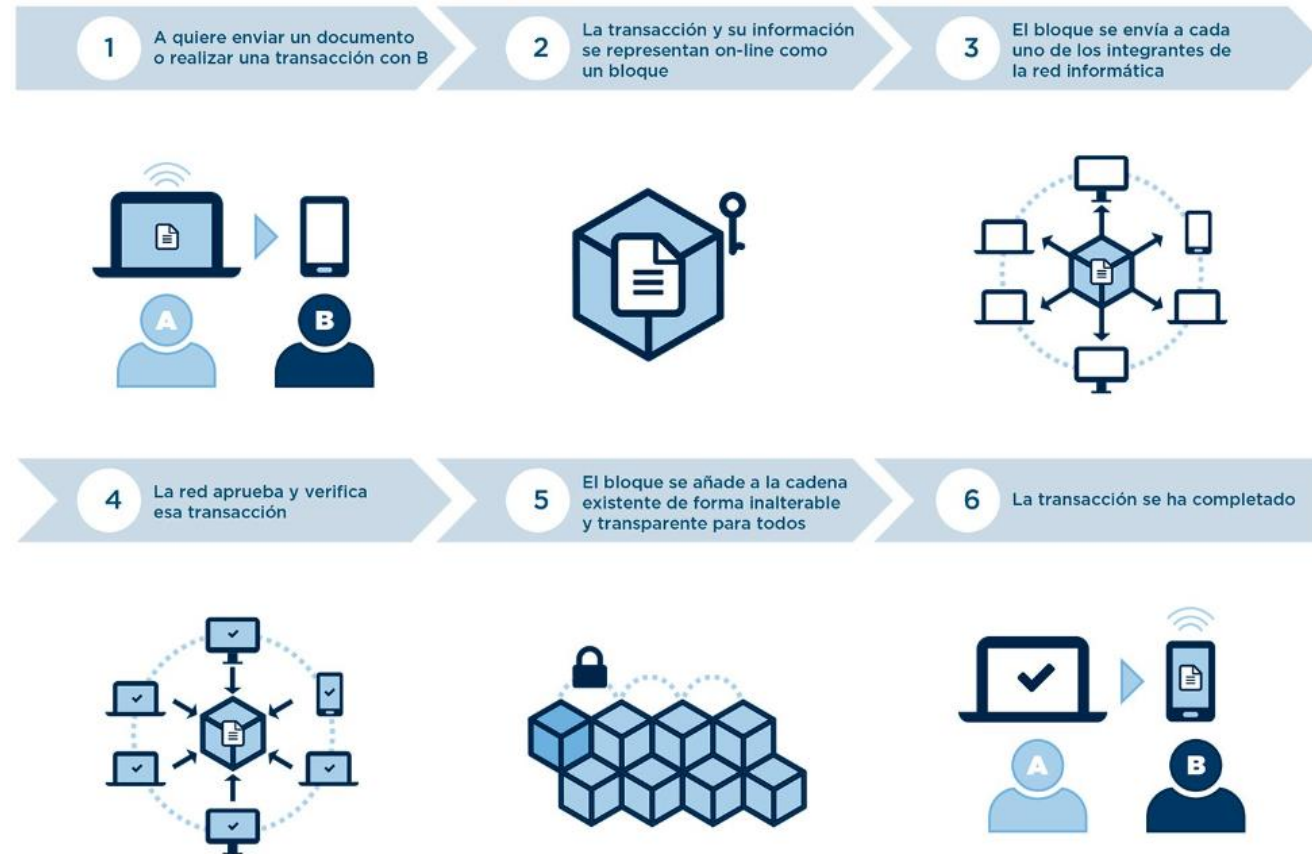
### Integridad.

- Como curiosidad, las técnicas de criptografía mediante funciones hash, junto con el concepto de cadena de bloques (blockchain) como estructura de información distribuida, han propiciado, entre otras aplicaciones, la aparición y utilización de las criptomonedas.



## Git - Fundamentos

### Integridad. Blockchain





## Git - Fundamentos

### Git sólo añade información.

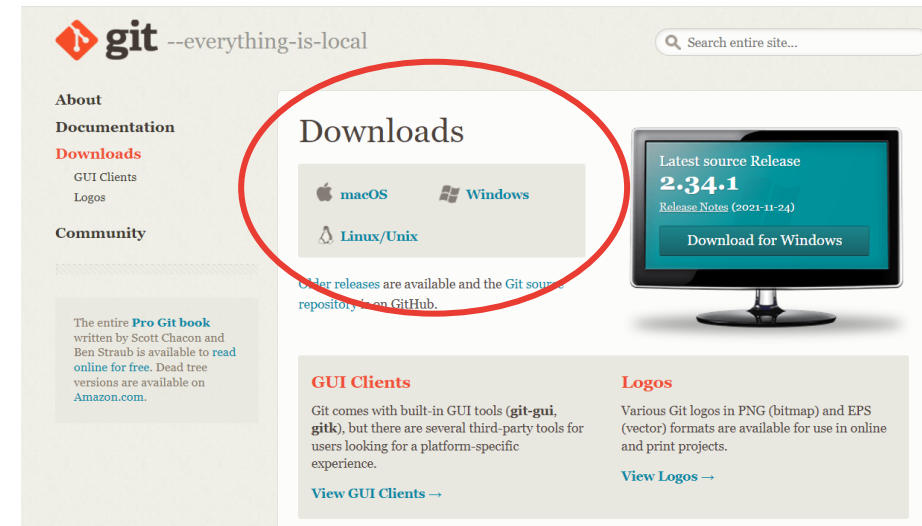
- Es poco habitual que Git haga algo que no se pueda reparar, o que de algún modo borre información.
- Se podrían perder cambios que no se han confirmado todavía, pero después de confirmar o consolidar una copia instantánea en Git es difícil perder algo.
- **Esto hace que usar Git sea una experiencia fiable.**



## Git. Instalación

- Podemos **descargar Git** a través de la página oficial e instalarlo:

<https://git-scm.com/downloads>



- En este enlace tenemos un detalle para la instalación en diferentes plataformas:

<https://git-scm.com/book/es/v2/Inicio---Sobre-el-Control-de-Versiones-Instalaci%C3%B3n-de-Git>

## Git Bash. Consola

- Git permite interactuar con el usuario de varias formas. Una de ellas es mediante la **consola de Git**, es decir, a través de **línea de comandos**.
- Una vez instalado Git, su consola permite usar algunos comandos típicos de Linux y una serie de **comandos específicos para Git**.



```
MINGW64; c:/Users/pmart
pmart@LAPTOP-LDPE3MBP MINGW64 ~
$ 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
pull.rebase=false
credential.helper=manager-core
credential.https://dev.azure.com.usehttppath=true
init.defaultbranch=master
pmart@LAPTOP-LDPE3MBP MINGW64 ~
$ |
```

## Git Bash. Consola - Configuración inicial

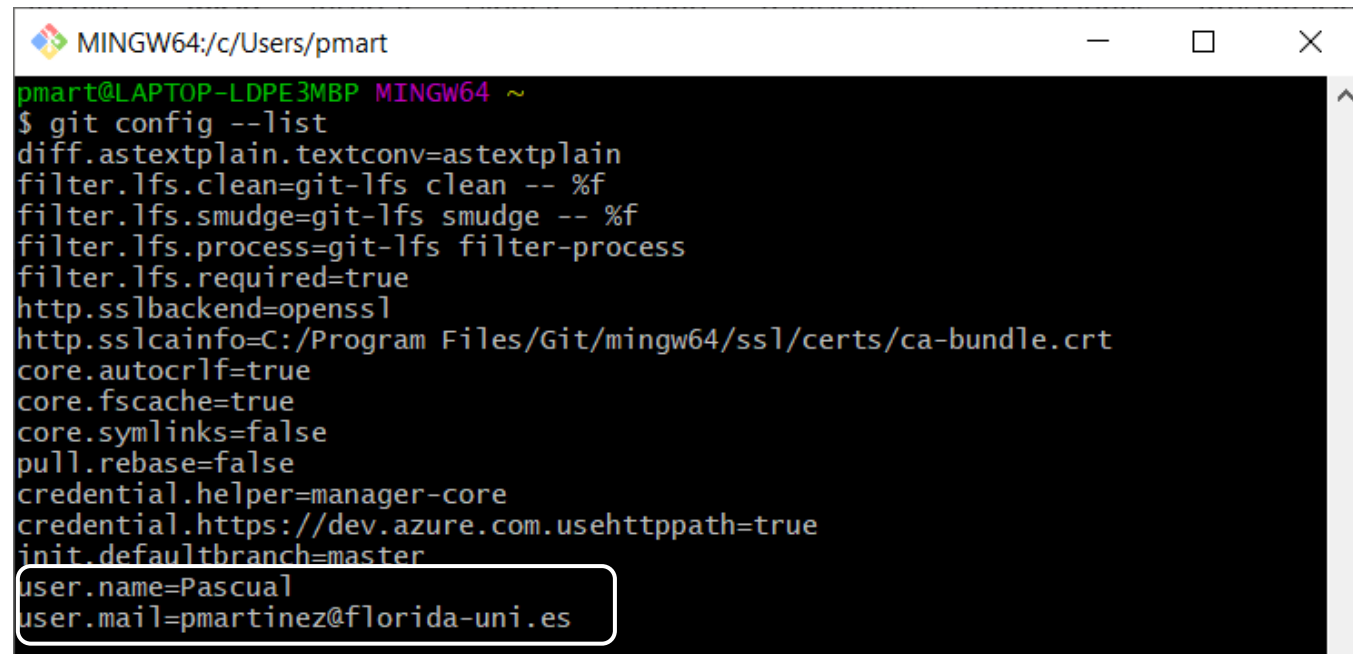
- Lo primero que se recomienda hacer es una configuración inicial de identidad, donde **podremos indicar nuestro nombre y nuestro correo**. Esta información se usará para firmar o identificar los cambios que se vayan haciendo en un repositorio.

```
$ git config --global user.name "John Doe"  
$ git config --global user.email johndoe@example.com
```

## Git Bash. Consola - Configuración inicial

- Podrás confirmar la configuración de tu identidad, mediante el comando que te muestra la lista de variables configuradas.

```
$ git config --list
```



```
MINGW64:/c/Users/pmart
pmart@LAPTOP-LDPE3MBP MINGW64 ~
$ 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
pull.rebase=false
credential.helper=manager-core
credential.https://dev.azure.com.usehttppath=true
init.defaultbranch=master
user.name=Pascual
user.mail=pmartinez@florida-uni.es
```

## Git Bash. Consola - Ayuda

- Podrás obtener ayuda, mediante el comando genérico:

```
$ git help
```

- O mediante comandos más específicos:

```
$ git help <verb>  
$ git <verb> --help  
$ man git-<verb>
```

- Por ejemplo, si queremos ver la página del manual para el comando config:

```
$ git help config
```