



# UD 03.CONTROL Y GESTIÓN DE VERSIONES

## PARTE 1 DE 4: HERRAMIENTAS CASE Y CONTROL DE VERSIONES

### Entornos de desarrollo (ED)

Raúl Palao  
Sergio Badal

# UD 03.CONTROL Y GESTIÓN DE VERSIONES

## 3.CONTROL Y GESTIÓN DE VERSIONES

### 3.1.1 HERRAMIENTAS CASE

3.1.2 SISTEMAS CONTROL DE VERSIONES (VCS)

3.1.3 TIPOS DE VCS

3.1.4 INTRODUCCIÓN A GIT  
MONOUSUARIO



# 3.1.1 Herramientas CASE



- La vida del “**objeto programado**” (del **PRODUCTO**) es ahora larga ya que se desea amortizar la inversión realizada (el software es caro).
- Por este motivo, los ~~programadores~~ desarrolladores fueron diseñando **herramientas que les ayudaran en cada una de las fases del ciclo de vida.**
- Este tipo de herramientas se conocen como herramientas CASE:

## Computer-Aided Software Engineering

- Podríamos definir las como:
  - El software que ayuda a crear software.



# 3.1.1 Herramientas CASE



## Computer-Aided Software Engineering (CASE)

- De este modo, existen CASE para cada una de las fases del ciclo de vida del software pudiendo aplicarse algunas de ellas en una o más fases.
  - La fase de codificación es la que tiene más herramientas CASE
  - Lo habitual es tenerlas todas ellas integradas en un IDE
- Como hemos visto en unidades anteriores, estos IDE son aplicaciones modulares que, con un editor de código como elemento central, incluyen pluggins adicionales pero ...

**¡Hay MUCHÍSIMA VIDA MÁS ALLÁ DE LOS IDE!**

# 3.1.1 Herramientas CASE



## Computer-Aided Software Engineering (CASE)

1. entornos de desarrollo (como **Visual Studio Code**)
2. compiladores (como **javac**)
3. depuradores (como **Eclipse**)
4. refactorizadores (como **NetBeans**)
5. analizadores de rendimiento (como **JetProfiler**)
6. herramientas de análisis y diseño (como **DIA**)
7. herramientas de despliegue e instalación (como **Bamboo**)
8. editores de código (como **Notepad++**)
9. generadores de código (como el de **Ruby On Rails**)
10. generadores de documentación (como **javadoc**)
11. gestores de proyectos (como **Microsoft Projects**)
12. gestores de incidencias (como **Mantis**)
13. control de versiones (como **GIT**)

# UD 03.CONTROL Y GESTIÓN DE VERSIONES

## 3.CONTROL Y GESTIÓN DE VERSIONES

3.1.1 HERRAMIENTAS CASE

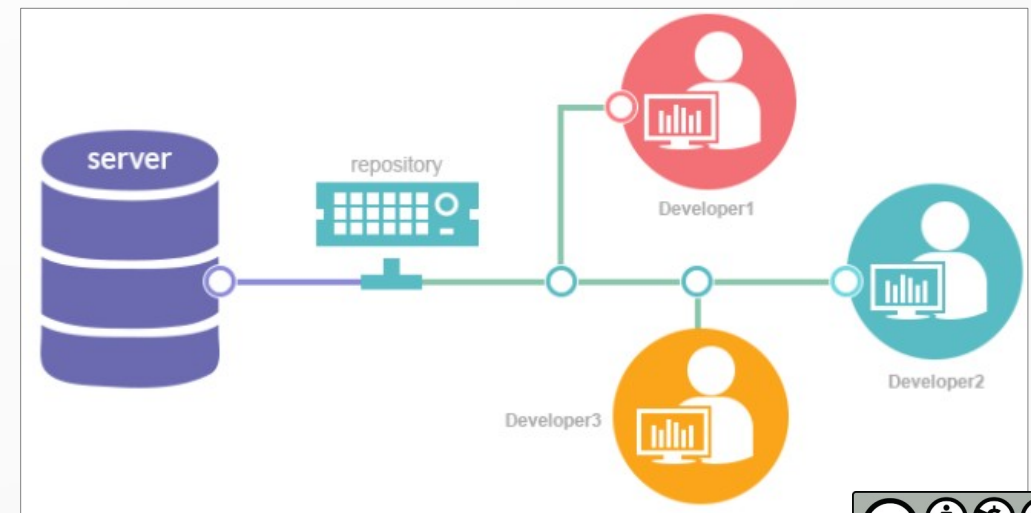
### **3.1.2 SISTEMAS CONTROL DE VERSIONES (VCS)**

3.1.3 TIPOS DE VCS

*3.1.4 INTRODUCCIÓN A GIT  
MONOUSUARIO*

# 3.1.2 SISTEMAS CONTROL DE VERSIONES (VCS)

- Un sistema de control de versiones, VCS (Version Control System) es una **herramienta CASE** que se emplea en la fase de integración y/o mantenimiento y **puede venir o no incluida en los IDE**.
- Durante la codificación de un proyecto, muchos de los archivos asociados a él irán evolucionando: se añadirán y modificarán.
  - En el caso de un **programador solitario**:
    - En su día a día hace cambios de su código frecuentemente (corrige código, añade nuevas características o lo modifica). Cada vez que se da un nuevo cambio, la antigua versión del fichero se sobrescribe.
    - Pero si este código genera error, es interesante poder volver a la versión anterior para comprender qué ha ocurrido.
  - Si esto se amplía a un escenario de trabajo colaborativo
    - Al estar varios **programadores** trabajando sobre un mismo fichero compartido en red, la situación se complica.





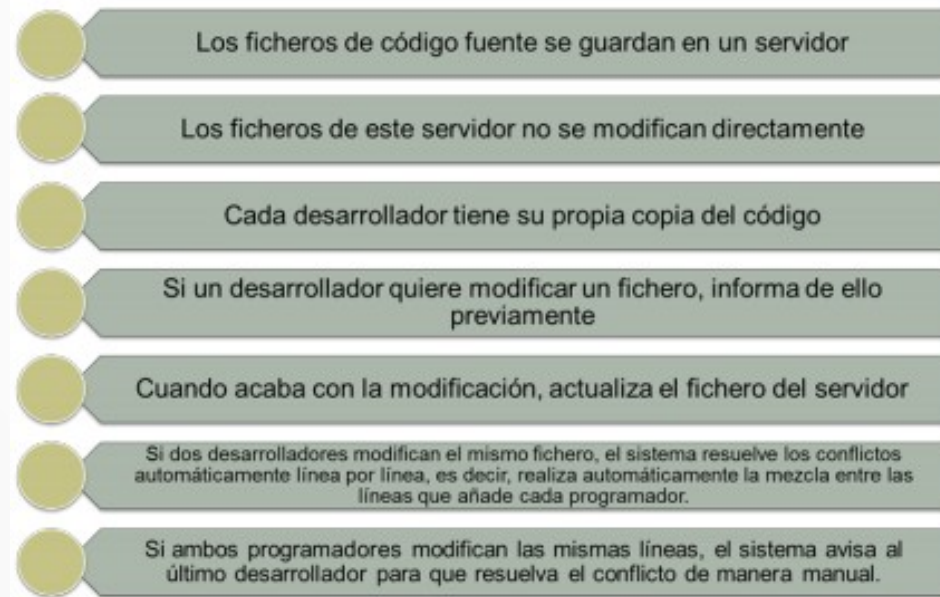
## 3.1.2 SISTEMAS CONTROL DE VERSIONES (VCS)

- Este concepto no es exclusivo del desarrollo de software, todos estamos acostumbrados a diferentes ediciones de libros, revisiones de coches, televisores, etc.; sin embargo ha tomado mucha más relevancia en la era digital.
- Programas como **Microsoft Word** y su primo **LibreWriter** nos permiten gestionar diferentes versiones del mismo documento, compararlas, etc.
  - <https://www.atarea.es/software/ofimatica/libreoffice-y-el-control-de-versiones-para-incorregibles/>



# 3.1.2 SISTEMAS CONTROL DE VERSIONES (VCS)

- Funcionamiento:



# 3.1.2 SISTEMAS CONTROL DE VERSIONES (VCS)

- **Un VCS debería usarse siempre.**
- Las situaciones en que podríamos haber necesitado usarlo son:
  - Si tras hacer un cambio en el código surge un error y queremos volver a la versión anterior
  - Si queremos recuperar código perdido y nuestra copia es demasiado antigua
  - Si queremos estar al día de la última versión del desarrollo si trabajamos en equipo
  - Si queremos mantener varias versiones del mismo producto
  - Si tenemos dos versiones del código y queremos ver las diferencias
  - Si queremos realizar pruebas de errores en la aplicación
  - Si queremos probar código que ha hecho otro programador
  - Si queremos se hizo un cambio y queremos saber qué, cuándo y dónde
  - Si queremos experimentar con nuevas características sin interferir en el código



# 3.1.2 SISTEMAS CONTROL DE VERSIONES (VCS)

- Preocupante realidad:
  - Cuando se usa en entornos individuales suele usarse como una mera copia de seguridad de nuestros ficheros
  - **¡Es justamente a la inversa como debemos usarlo!**
  - Debemos incluir los datos de nuestro VCS en nuestra copia de seguridad.

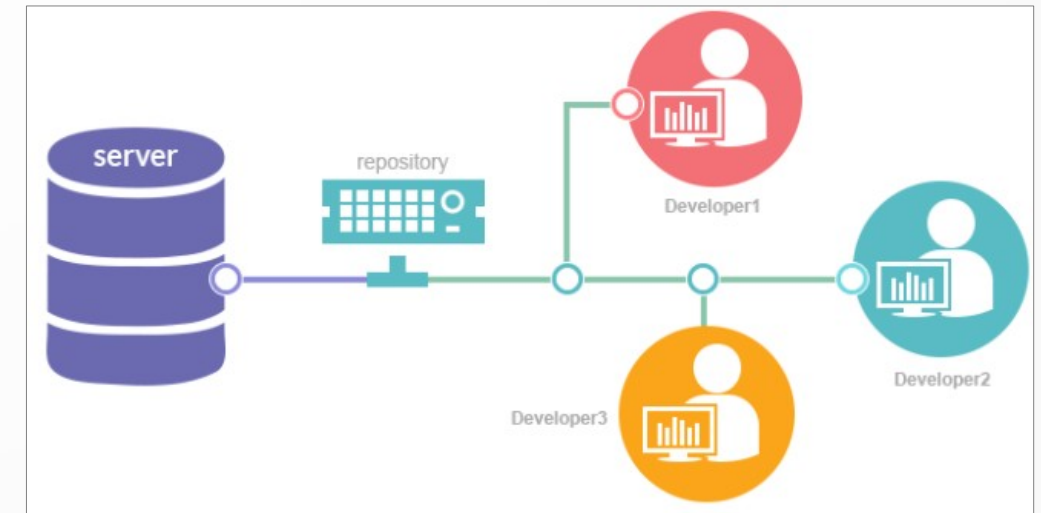




# 3.1.2 SISTEMAS CONTROL DE VERSIONES (VCS)

- Vocabulario básico común a todo VCS

Repositorio ( <i>repository</i> )	• base de datos donde se guardan los ficheros
Servidor ( <i>server</i> )	• ordenador donde se guarda el repositorio
Cliente ( <i>client</i> )	• ordenadores que están conectados al repositorio
Copia de trabajo o de directorio ( <i>working copy or workings setk</i> )	• directorio local (en clientes) en el que se hacen los cambios
Tronco o principal ( <i>trunk or main</i> )	• línea principal de código en el repositorio
Cabecera ( <i>head</i> )	• última versión en el repositorio
Revisión ( <i>revision</i> )	• versión en la que se encuentra un fichero
Rama ( <i>branch</i> )	• copia separada de un fichero o una carpeta para uso privado
Conflicto ( <i>conflict</i> )	• situación que ocurre cuando se intentan aplicar dos cambios contradictorios de un mismo fichero sobre el repositorio
Mensaje de registro ( <i>checkin message</i> )	• mensaje corto que indica qué se ha cambiado en el fichero
Histórico de cambios ( <i>changelog or history</i> )	• listado de todos los cambios realizados en el fichero desde su creación

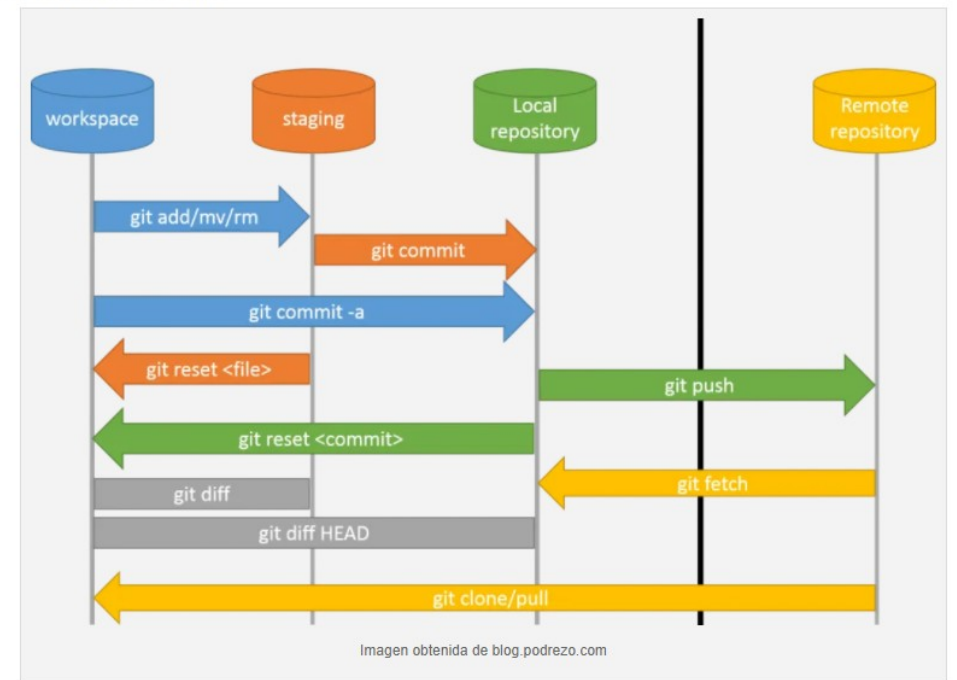


# 3.1.2 SISTEMAS CONTROL DE VERSIONES (VCS)

- Acciones comunes a todo VCS

add	<ul style="list-style-type: none"><li>Añadir un fichero al repositorio por primera vez para permitir al sistema hacer su seguimiento</li></ul>
get latest/check out	<ul style="list-style-type: none"><li>Traer desde el repositorio la última versión del fichero</li></ul>
check out for edit	<ul style="list-style-type: none"><li>Traer desde el repositorio la última versión del fichero en modo "editable". En muchos sistemas el check out ya es editable.</li></ul>
check in/commit	<ul style="list-style-type: none"><li>Subir un fichero modificado al repositorio. Se le asigna un número de versión y los demás programadores pueden hacer un check out o un sync para obtener la última versión</li></ul>
merge	<ul style="list-style-type: none"><li>Mezclar los cambios de un fichero a otro para actualizarlo.</li></ul>
resolve	<ul style="list-style-type: none"><li>Resolver los problemas encontrados al hacer check in. Una vez solucionados, se hace el check in de nuevo.</li></ul>
diff /delta	<ul style="list-style-type: none"><li>Comparar dos ficheros para encontrar las diferencias</li></ul>
revert	<ul style="list-style-type: none"><li>Revertir la última versión: se descartan los cambios locales y se recarga la última versión que existe en el repositorio</li></ul>
update/syc	<ul style="list-style-type: none"><li>Actualizar todos los ficheros con la última versión del repositorio</li></ul>
locking	<ul style="list-style-type: none"><li>Bloquear un fichero, tomando su control para que nadie más pueda editarlo hasta desbloquearlo</li></ul>
breaking the lock	<ul style="list-style-type: none"><li>Forzar el desbloqueo de un fichero</li></ul>

Workflow de Git



# UD 03.CONTROL Y GESTIÓN DE VERSIONES

## 3.CONTROL Y GESTIÓN DE VERSIONES

3.1.1 HERRAMIENTAS CASE

3.1.2 SISTEMAS CONTROL DE VERSIONES (VCS)

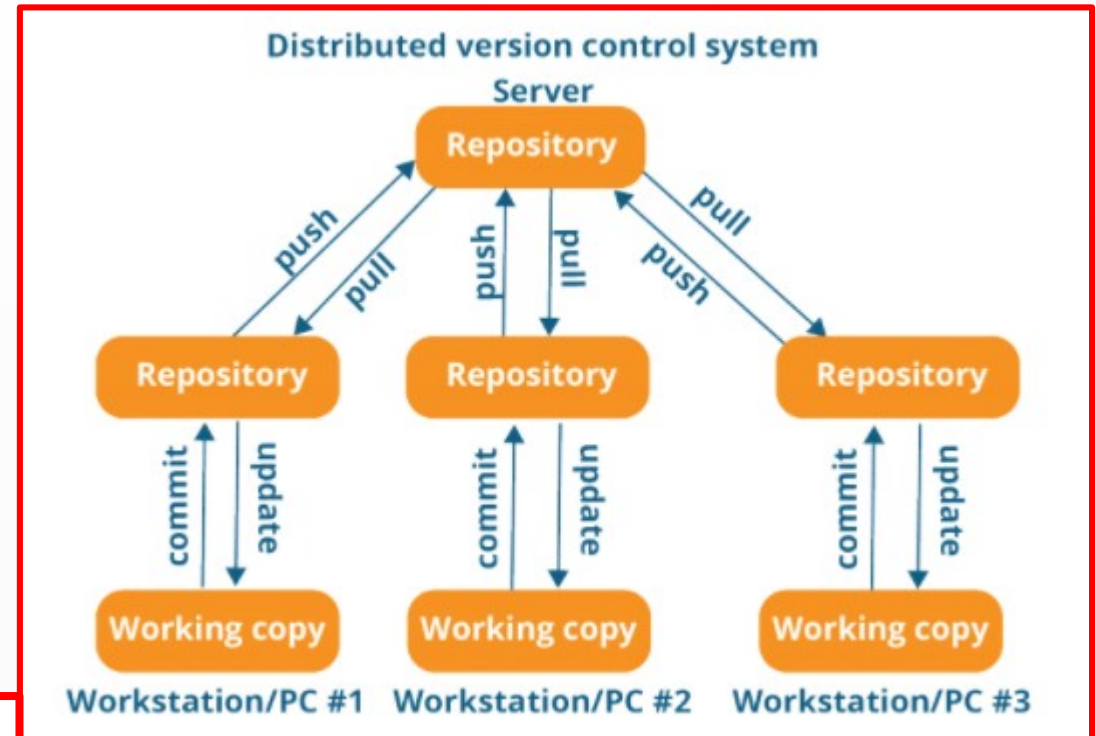
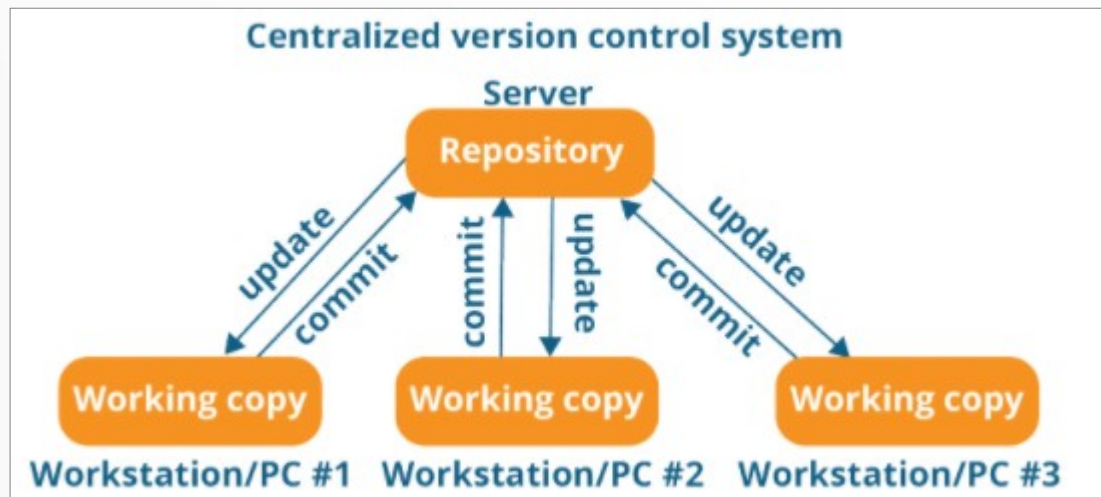
### 3.1.3 TIPOS DE VCS

3.1.4 INTRODUCCIÓN A GIT  
MONOUSUARIO



# 3.1.3 TIPOS DE VCS

- Centralizado (SVN) vs distribuído (GIT)



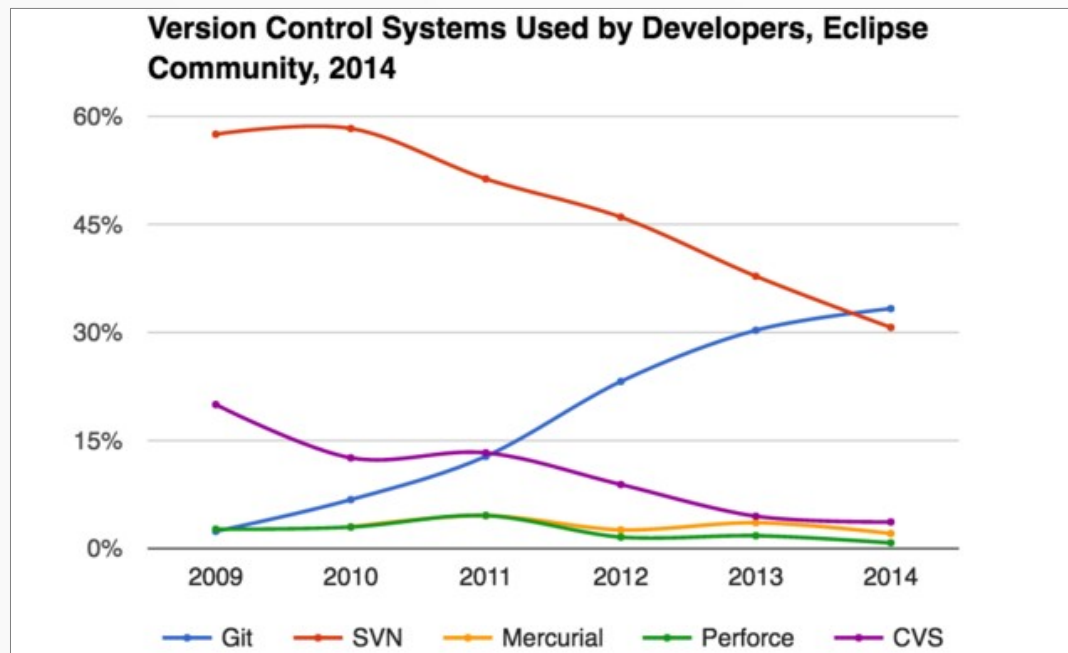
En un sistema distribuido tenemos terminología específica que se añade a la ya vista:

push	• (empujar) Envía un cambio desde un repositorio a otro.
pull	• (extraer) Coge los cambios desde un repositorio
clone	• (clonar) Trae una copia exacta del proyecto desde un repositorio a otro

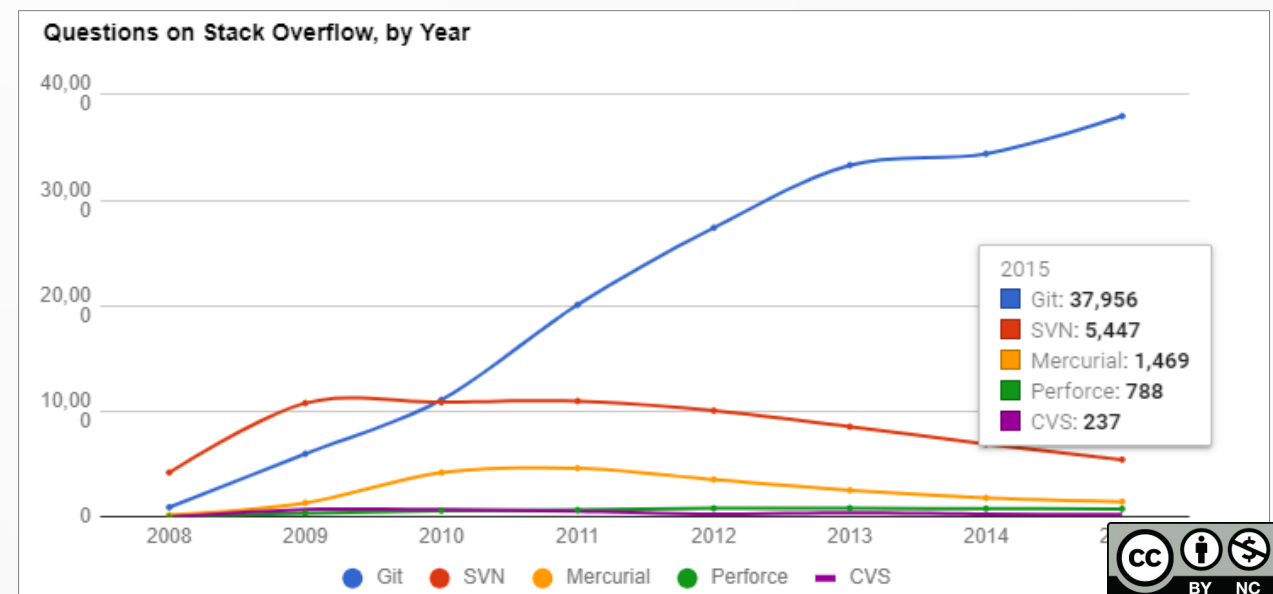


# 3.1.3 TIPOS DE VCS

- Ejemplos más conocidos de código propietario:
  - **Visual SourceSafe y Visual Studio Team Foundation Server:**
    - De Microsoft, **código propietario** y centralizado.
  - **BitKeeper:**
    - Usado durante años para el control del Kernel de Linux.
    - **Código propietario** y distribuido.



- Ejemplos más conocidos de código abierto (open source):
  - **Concurrent Versions System (CVS):** De código abierto, centralizado sin nueva versión desde 2008.
  - **Subversion (SVN):** De código abierto y modelo distribuido.
    - **FUE uno de los estándares de facto del modelo distribuido** durante años.
  - **Mercurial:** De código abierto y modelo distribuido.
    - Surgió como alternativa a BitKeeper.
    - **Es un estándar de facto en modelos distribuidos.**
  - **GIT:** De código abierto y modelo distribuido.
    - En este módulo vamos a trabajar con GIT.
    - **Creado por el finlandés Linus Torvalds (fundador de LINUX)**

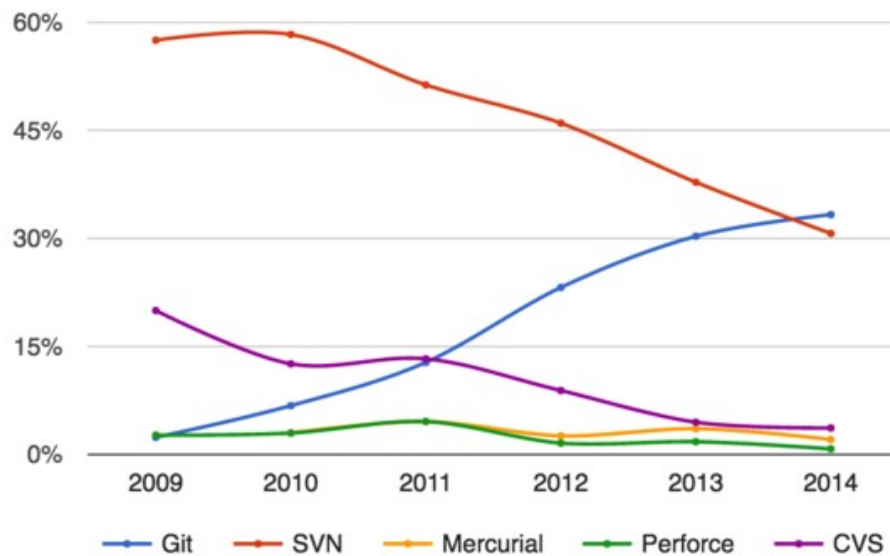


# 3.1.3 TIPOS DE VCS

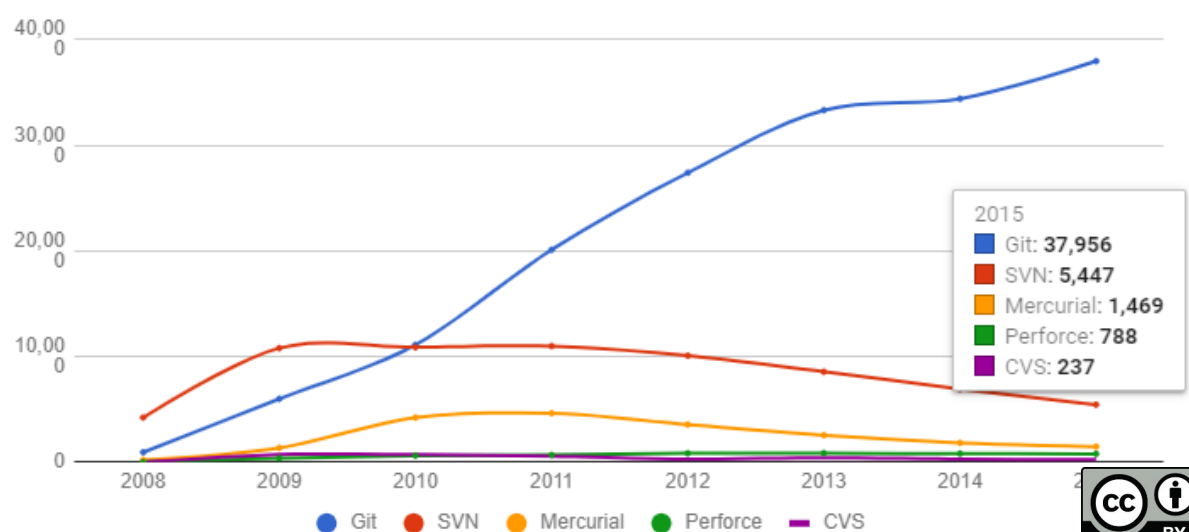


- Ejemplos más conocidos de código abierto (open source):
  - **Concurrent Versions System (CVS):** De código abierto, centralizado sin nueva versión desde 2008.
  - **Subversion (SVN):** De código abierto y modelo distribuido.
    - FUE uno de los estándares de facto del modelo distribuido durante años.
  - **Mercurial:** De código abierto y modelo distribuido.
    - Surgió como alternativa a BitKeeper.
    - Es un estándar de facto en modelos distribuidos.
  - **GIT:** De código abierto y modelo distribuido.
    - En este módulo vamos a trabajar con GIT.
    - Creado por el finlandés Linus Torvalds (fundador de LINUX)

Version Control Systems Used by Developers, Eclipse Community, 2014



Questions on Stack Overflow, by Year



# UD 03.CONTROL Y GESTIÓN DE VERSIONES

## **3.CONTROL Y GESTIÓN DE VERSIONES**

3.1.1 HERRAMIENTAS CASE

3.1.2 SISTEMAS CONTROL DE VERSIONES (VCS)

3.1.3 TIPOS DE VCS

**3.1.4 INTRODUCCIÓN A GIT  
MONOUSUARIO**

# 3.1.4 GIT MONOUSUARIO

- Las siguientes indicaciones son las recomendadas para poder seguir sin problemas este pequeño tutorial **si eres nuevo en GIT**.
- **Si ya has trabajado alguna vez con GIT** y lo tienes instalado en cualquier otra versión, sistema operativo o con cualquier otra configuración **no es necesario que desinstales nada**.



git





# 3.1.4 GIT MONOUSUARIO

## Primero

- Instalaremos git

## Instalación

La instalación de **Git** en Linux dependerá de la plataforma, pero suele ser un proceso trivial ya que viene empaquetado en todas las distribuciones.



```
# En Ubuntu/debian  
$ sudo apt-get install git-core  
  
# En Archlinux  
$ sudo pacman -S git
```

En Windows o MacOSX se utiliza un instalador visual, es decir, que es también muy sencillo.

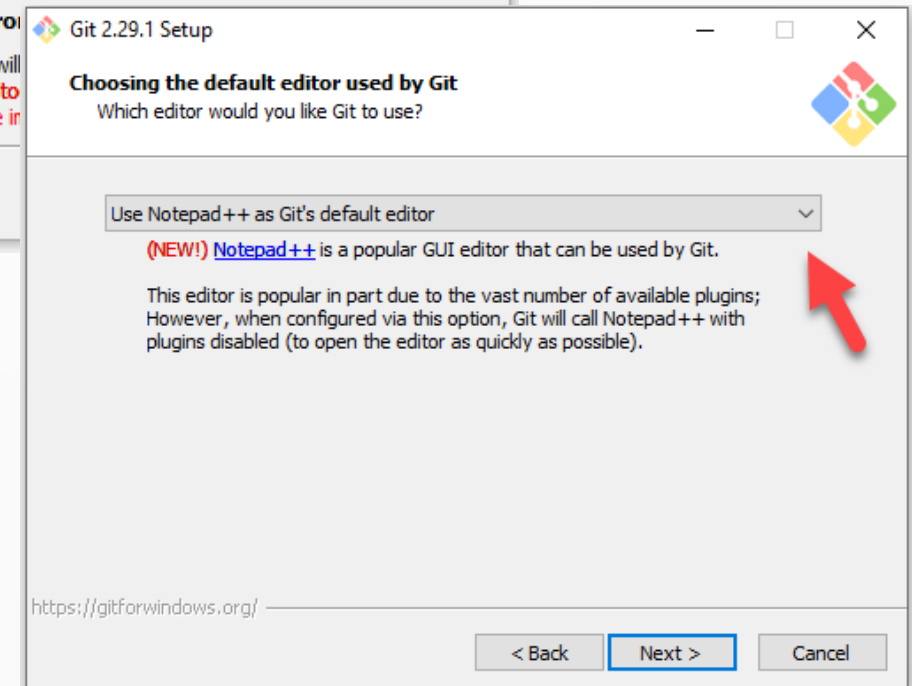
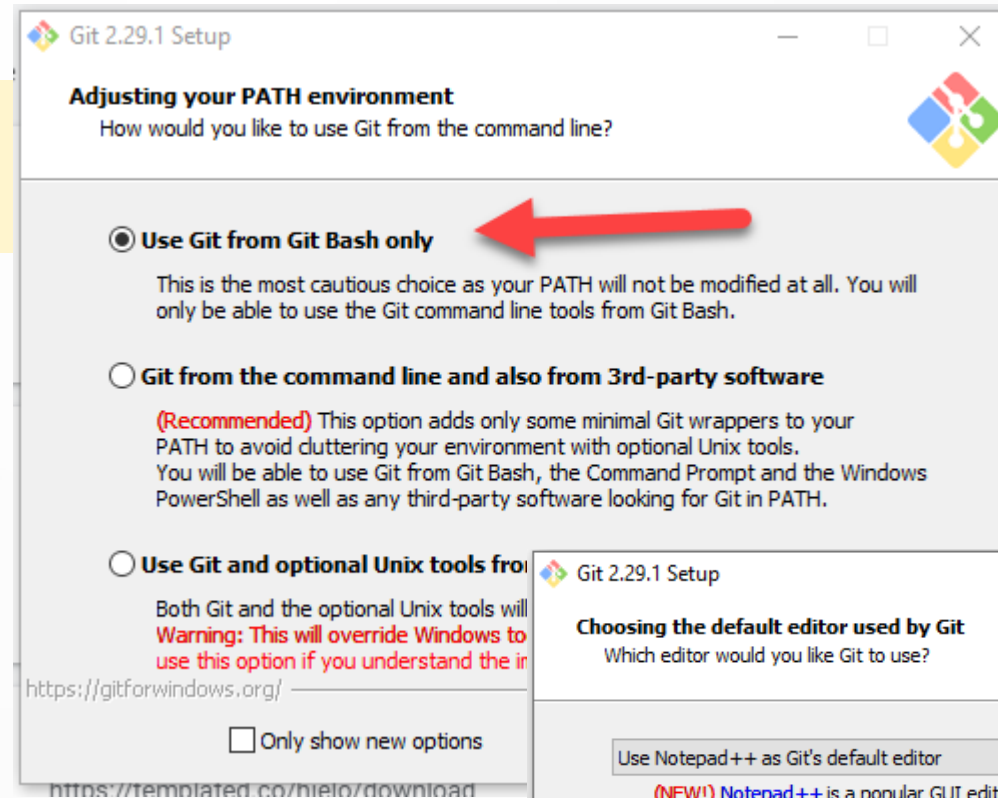
# 3.1.4 GIT MONOUSUARIO

## Primero

- Instalaremos git

### • Paso 1: Instalar GIT

- Instala previamente **Notepad++**, si no lo has hecho ya, para tu sistema operativo y procesador (32/64):
  - <https://notepad-plus-plus.org/downloads/>
- Descarga la última versión de Git para tu sistema operativo:
  - <https://git-scm.com/downloads>
- Si instalas en Windows, te recomendamos que **dejes todas las opciones por defecto salvo estas que verás a la derecha:**

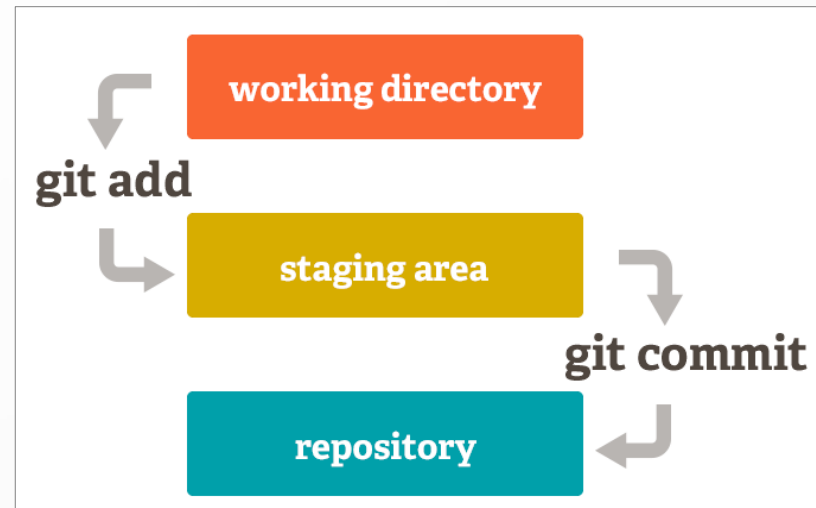


# 3.1.4 GIT MONOUSUARIO

## Segundo

- Crearemos un repositorio de prueba **01\_REP\_INICIAL** (de un solo archivo) en nuestro PC, realizando las gestiones básicas sobre él.

- 1) Crearemos una nueva carpeta.
- 2) Le diremos que esa carpeta es el WORKSPACE de nuestro repositorio con **[git init]**
- 3) Nos identificaremos **[git config --global]**
- 4) Copiaremos un único archivo en el WORKSPACE.
- 5) Lo añadiremos al almacenamiento temporal (STAGING) con la orden **[git add]**.
- 6) Lo grabaremos en el repositorio (LOCAL REPOSITORY) con la orden **[git commit]**



# 3.1.4 GIT MONOUSUARIO

- **Paso 2.1: Crear un repositorio de prueba con un único archivo**

- 1) Crea una carpeta llamada ED.Practicas.GIT y, dentro de ella, crea otra una carpeta llamada **01\_REP\_INICIAL**
- 2) Crea un archivo llamado **holasoyXXX.txt** con el texto “Hola, soy XXX”, y grábalo en esa carpeta.
- 3) Haz botón derecho sobre la carpeta y pulsa **Git Bash Here** para abrir la consola de BASH. Estaremos emulando una consola Linux.
- 4) Luego, desde la consola:
  - 1) Crea el Repositorio [**git init**]
  - 2) Identifcate [**git config --global user.name XXX**] y  
**[git config --global user.name XXX]**
  - 3) Mira el estado del repositorio [**git status**]
  - 4) Añade al staging el archivo [**git add .**]
  - 5) Grábalo en el repositorio con:  
**[git commit -m “mi primer commit”]**

```
MINGW64:/d/Dropbox/05.CEED/02.CEEDCV.Entornos/ED.UD03. Control y gestión de versione...
Sergio@HERAKLION MINGW64 /d/Dropbox/05.CEED/02.CEEDCV.Entor
y gestión de versiones/_ED.Practicas.GIT/01.REP_INICIAL
$ git init
Initialized empty Git repository in D:/Dropbox/05.CEED/02.C
nes/_ED.Practicas.GIT/01.REP_INICIAL/.git/

Sergio@HERAKLION MINGW64 /d/Dropbox/05.CEED/02.CEEDCV.Entor
cticas.GIT/01.REP_INICIAL (master)
$ git config --global user.name "Sergio Badal"

Sergio@HERAKLION MINGW64 /d/Dropbox/05.CEED/02.CEEDCV.Entor
cticas.GIT/01.REP_INICIAL (master)
$ git config --global user.email "sergio.badal@ceedcv.es"

Sergio@HERAKLION MINGW64 /d/Dropbox/05.CEED/02.CEEDCV.Entor
cticas.GIT/01.REP_INICIAL (master)
$ git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be commi
    holagitsoysergio.txt

nothing added to commit but untracked files present (use "g
d" to track)

Sergio@HERAKLION MINGW64 /d/Dropbox/05.CEED/02.CEEDCV.Entor
D.UD03. Control y gestión de versiones/_ED.Practicas.GIT/01
INICIAL (master)
$ git add .

Sergio@HERAKLION MINGW64 /d/Dropbox/05.CEED/02.CEEDCV.Entor
.UD03. Control y gestión de versiones/_ED.Practicas.GIT/01.
ICIAL (master)
$ git commit -m "mi primer commit"
[master (root-commit) e5f5222] mi primer commit
1 file changed, 5 insertions(+)
create mode 100644 holagitsoysergio.txt
```



# 3.1.4 GIT MONOUSUARIO

- **Paso 2.2: Nuevos cambios**

- 1) Prueba ahora a hacer cambios en el archivo de prueba, añadiendo otra frase diferente. Por ejemplo “¿Qué tal?” en una nueva línea. Al cambiarlo, debes saber que SOLO se modifica en el WORKSPACE (en la carpeta de tu PC).
- 2) Luego, vuelve a ver el estado del repositorio con `[git status]`.
- 3) Te saldrá en rojo como modificado. Pásalo al STAGING con `[git add .]`
- 4) Finalmente, grábalo en el REPOSITORIO con `[git commit -m “mi segundo commit”]`

```
MINGW64:/d/Dropbox/05.CEED/02.CEEDCV.Entornos/ED.UD03. Control y gestión de versiones/_ED.Practicas.GIT/01.REP_INICIAL
Sergio@HERAKLION MINGW64 /d/Dropbox/05.CEED/02.CEEDCV.Entornos/ED.UD03. Control y gestión de versiones/_ED.Practicas.GIT/01.REP_INICIAL (master)
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   holagitsoysergio.txt

no changes added to commit (use "git add" and/or "git commit -a")

Sergio@HERAKLION MINGW64 /d/Dropbox/05.CEED/02.CEEDCV.Entornos/ED.UD03. Control y gestión de versiones/_ED.Practicas.GIT/01.REP_INICIAL (master)
$ git add .

Sergio@HERAKLION MINGW64 /d/Dropbox/05.CEED/02.CEEDCV.Entornos/ED.UD03. Control y gestión de versiones/_ED.Practicas.GIT/01.REP_INICIAL (master)
$ git commit -m "mi segundo commit"
[master 12f8c24] mi segundo commit
1 file changed, 1 insertion(+), 3 deletions(-)
```

# 3.1.4 GIT MONOUSUARIO

- **Paso 2.3: Consultar los cambios en el repositorio (ver log)**

1) Prueba la orden de la derecha:

`[git log]` para ver el log

2) Ten en cuenta que el HEAD apunta al último **commit**

```
MINGW64:/d/Dropbox/05.CEED/02.CEEDCV.Entornos/ED.UD03. Control y gestión de versiones/_ED.Practicas.GIT/01.REP_
Sergio@HERAKLION MINGW64 /d/Dropbox/05.CEED/02.CEEDCV.Entornos/ED.UD03. Control y gestión de versiones/_ED.Practicas.GIT/01.REP_INICIAL (master)
$ ls -la
total 13
drwxr-xr-x 1 Sergio 197121  0 oct. 29 14:18 ./
drwxr-xr-x 1 Sergio 197121  0 oct. 29 17:19 ../
drwxr-xr-x 1 Sergio 197121  0 oct. 29 18:51 .git/
-rw-r--r-- 1 Sergio 197121 63 oct. 29 18:51 holagitsoysergio.txt

Sergio@HERAKLION MINGW64 /d/Dropbox/05.CEED/02.CEEDCV.Entornos/ED.UD03. Control y gestión de versiones/_ED.Practicas.GIT/01.REP_INICIAL (master)
$ git log
commit 403dff3d9450185d96da46716f789fd7c46e3b42 (HEAD -> master)
Author: Sergio Badal <sergio.badal@ceedcv.es>
Date:   Thu Oct 29 18:51:50 2020 +0100

    mi segundo commit

commit 2abd05d375dea55a4e1643d8db7a9318bb946de4
Author: Sergio Badal <sergio.badal@ceedcv.es>
Date:   Thu Oct 29 18:51:34 2020 +0100

    mi primer commit
```

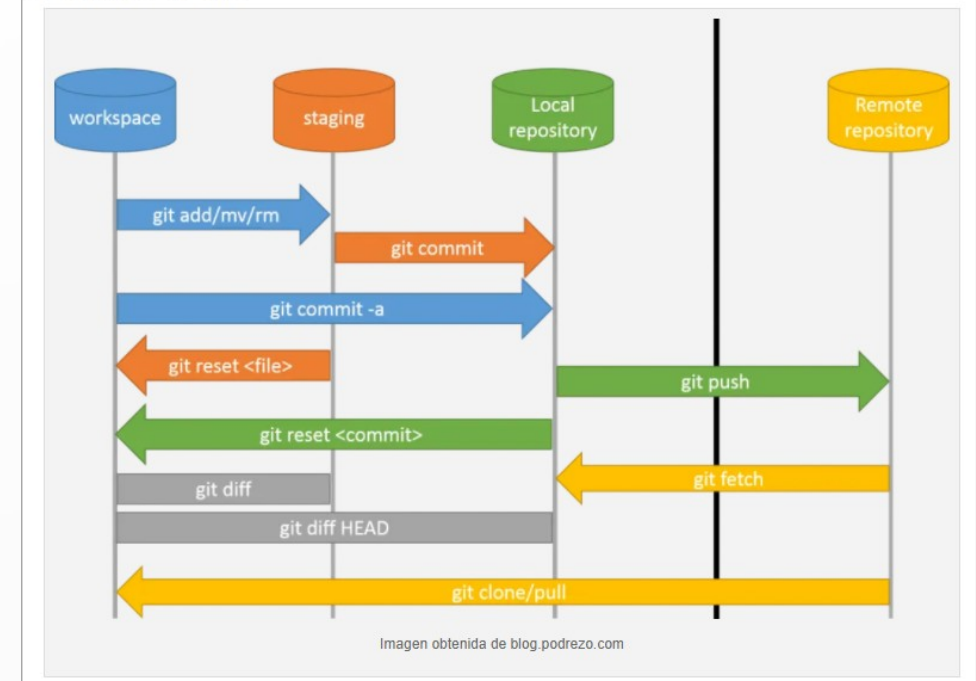
# 3.1.4 GIT MONOUSUARIO

## Tercero

- Crearemos una cuenta en GitHub (si no la tenemos creada ya), subiremos nuestro repositorio e interactuaremos con la cuenta.

- 1) Ahora crearemos una cuenta en GitHub (en la nube) [<https://github.com/join>]
- 2) Crearemos un repositorio llamado **01\_REP\_INICIAL** (desde la web)
- 3) Enlazaremos el repositorio local con esa cuenta/repositorio [**git remote add**]
- 4) Subiremos nuestro archivo a la nube [**git push**]
- 5) Haremos un cambio en la nube (desde la web)
- 6) Lo descargaremos a nuestro repositorio [**git pull**]
- 7) Haremos un cambio en local (notepad++)
- 8) Lo subiremos a la nube [**git push**]

## Workflow de Git



# 3.1.4 GIT MONOUSUARIO

## Tercero

- Crearemos una cuenta en GitHub (si no la tenemos creada ya), subiremos nuestro repositorio e interactuaremos con la cuenta.

**Primero crea una cuenta en GitHub**, preferentemente con tu correo de GVA: <https://github.com/join>





# 3.1.4 GIT MONOUSUARIO

Tercero

- Crearemos una cuenta en GitHub (si no la tenemos creada ya), subiremos nuestro repositorio e interactuaremos con la cuenta.

**Desde hace unos años y por seguridad, GitHub no usa contraseñas sino TOKENS**, sigue los siguientes pasos para obtenerlo:

- 1) Ve al icono de tu cuenta de usuario (arriba a la derecha) y despliega SETTINGS
- 2) A la izquierda, selecciona > DEVELOPER SETTINGS > PERSONAL ACCESS TOKENS
- 3) TOKENS (CLASSIC) > GENERATE NEW TOKEN (CLASSIC)
- 4) PONLE UN NOMBRE > SELECCIONA TODAS LAS OPCIONES > GENERATE TOKEN
- 5) COPIA EL TOKEN Y GUÁRDALO (ACTUARÁ COMO CONTRASEÑA).



Make sure to copy your token now as you will not be able to see it again.

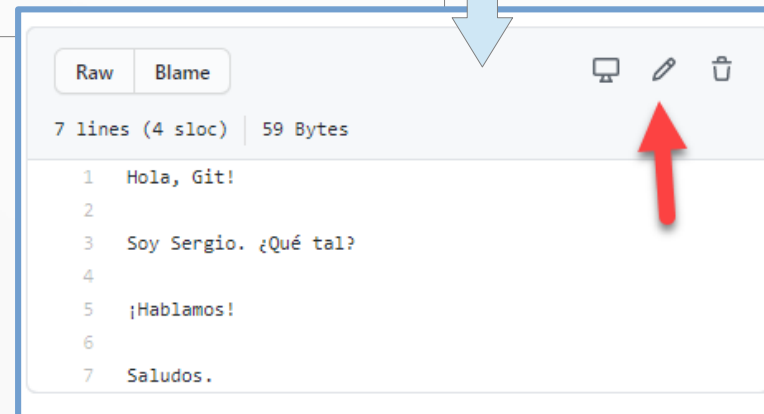
ghp\_ywPZmhxuhTkZhP8QRT6hUIYMQ3FWpI3TjagG



# 3.1.4 GIT MONOUSUARIO

- **Paso 3.1: Crear un repositorio en la nube e interactuar con los dos**

- 1) Entra en [<https://github.com/join>] y crea una cuenta
- 2) Desde la web, crea un repositorio llamado **01\_REP\_INICIAL**
- 3) Ya en la consola, en local, enlaza el local con esa cuenta/repositorio  
`[git remote add origin https://github.com/usuario/01_REP_INICIAL]`
- 4) Sube el repositorio a la nube `[git push origin master]`
- 5) Haz un cambio en la nube (desde la web de gitHub)
- 6) Descárgalo a tu repositorio en local `[git pull origin master]`



```
MINGW64:/d/Dropbox/05.CEED/02.CEEDCV.Entornos/ED.UD03. Control y gestión de version...
Sergio@HERAKLION MINGW64 /d/Dropbox/05.CEED/02.CEEDCV.Entornos/ED.UD03. Control y gestión de versiones/_ED.Practicas.GIT/01.REP_INICIAL (master)
$ git remote add origin https://github.com/enyzing/01_REP_INICIAL
Sergio@HERAKLION MINGW64 /d/Dropbox/05.CEED/02.CEEDCV.Entornos/ED.UD03. Control y gestión de versiones/_ED.Practicas.GIT/01.REP_INICIAL (master)
$ git push origin master
Enumerating objects: 6, done.
Counting objects: 100% (6/6), done.
Delta compression using up to 4 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (6/6), 531 bytes | 177.00 KiB/s, done.
Total 6 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/enyzing/01_REP_INICIAL
 * [new branch]      master -> master

Sergio@HERAKLION MINGW64 /d/Dropbox/05.CEED/02.CEEDCV.Entornos/ED.UD03. Control y gestión de versiones/_ED.Practicas.GIT/01.REP_INICIAL (master)
$ git pull origin master
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), 687 bytes | 2.00 KiB/s, done.
From https://github.com/enyzing/01_REP_INICIAL
 * branch            master       -> FETCH_HEAD
   fa65c99..f592926  master       -> origin/master
Updating fa65c99..f592926
Fast-forward
 holagitsoysergio.txt | 2 ++
 1 file changed, 2 insertions(+)
```

# 3.1.4 GIT MONOUSUARIO

- **Paso 3.2: Haz un cambio más en local**

- 1) Haz un cambio en local con (notepad++)
- 2) Añádelo al STAGING [**git add .**]
- 3) Añádelo al REPOSITORIO  
con [**git commit -m "mi tercer commit"**]
- 4) Súbelo a la nube [**git push origin master**]

```
Sergio@HERAKLION MINGW64 /d/Dropbox/05.CEED/02.CEEDCV.Entornos/UD03. Control y gestión de versiones/_ED.Practicas.GIT/01.REP_INICIAL (master)
$ git add .

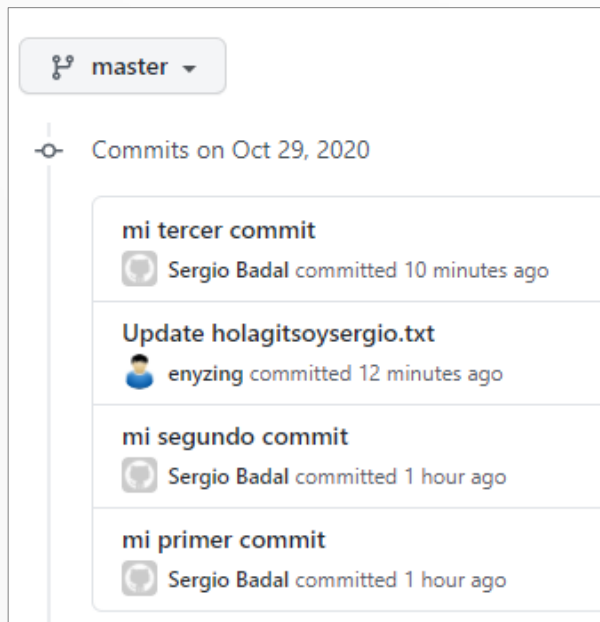
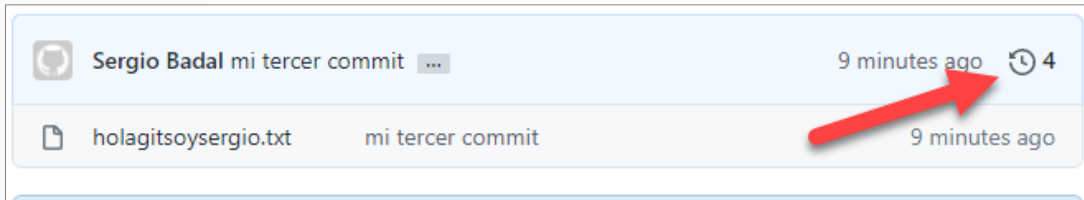
Sergio@HERAKLION MINGW64 /d/Dropbox/05.CEED/02.CEEDCV.Entornos/UD03. Control y gestión de versiones/_ED.Practicas.GIT/01.REP_INICIAL (master)
$ git commit -m "mi tercer commit"
[master 38d3d72] mi tercer commit
1 file changed, 2 insertions(+)

Sergio@HERAKLION MINGW64 /d/Dropbox/05.CEED/02.CEEDCV.Entornos/UD03. Control y gestión de versiones/_ED.Practicas.GIT/01.REP_INICIAL (master)
$ git push origin master
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 4 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 318 bytes | 159.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/enyzing/01_REP_INICIAL
f592926..38d3d72 master -> master
```

# 3.1.4 GIT MONOUSUARIO

- **Paso 3.3: Consultar los cambios en el repositorio (log)**

- 1) Prueba las órdenes de la derecha [**git log**] para ver el log
- 2) Recuerda que [**clear**] limpia la consola.
- 3) Ten en cuenta que el HEAD apunta al último commit
- 4) Ve también a github y comprueba el log en la nube



```
MINGW64:/d/Dropbox/05.CEED/02.CEEDCV.Entornos/ED.UD03. Control y gestión de versione...
Sergio@HERAKLION MINGW64 /d/Dropbox/05.CEED/02.CEEDCV.Entornos/ED
.UD03. Control y gestión de versiones/_ED.Practicas.GIT/01.REP_IN
ICIAL (master)
$ ls -la
total 13
drwxr-xr-x 1 Sergio 197121  0 oct. 29 14:18 ./
drwxr-xr-x 1 Sergio 197121  0 oct. 29 12:42 ../
drwxr-xr-x 1 Sergio 197121  0 oct. 29 14:19 .git/
-rw-r--r-- 1 Sergio 197121 73 oct. 29 14:18 holagitsoysergio.txt

Sergio@HERAKLION MINGW64 /d/Dropbox/05.CEED/02.CEEDCV.Entornos/ED
.UD03. Control y gestión de versiones/_ED.Practicas.GIT/01.REP_IN
ICIAL (master)
$ git log
commit 38d3d726261c83d13ff5287c5e83cb533b3da150 (HEAD -> master,
origin/master)
Author: Sergio Badal <sergio.badal@ceedcv.es>
Date: Thu Oct 29 14:19:39 2020 +0100

    mi tercer commit

commit f5929264d2ae8466ddcaafffa5a9316daed1ea66
Author: enyzing <sergio.badal@gmail.com>
Date: Thu Oct 29 14:17:13 2020 +0100

    Update holagitsoysergio.txt

commit fa65c9922f2304db311354d350b617c6ad5bc6ab
Author: Sergio Badal <sergio.badal@ceedcv.es>
Date: Thu Oct 29 13:31:46 2020 +0100

    mi segundo commit

commit e5f52224373cf57003bdd4addc5c2224be2f039e
Author: Sergio Badal <sergio.badal@ceedcv.es>
Date: Thu Oct 29 13:06:27 2020 +0100

    mi primer commit
```