ceedcv
CENTRE ESPECÍFIC
D´EDUCACIÓ A DISTÀNCIA DE
LA COMUNITAT VALENCIANA

UD 03.CONTROL Y GESTIÓN DE VERSIONES

PARTE 1 DE 5: 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

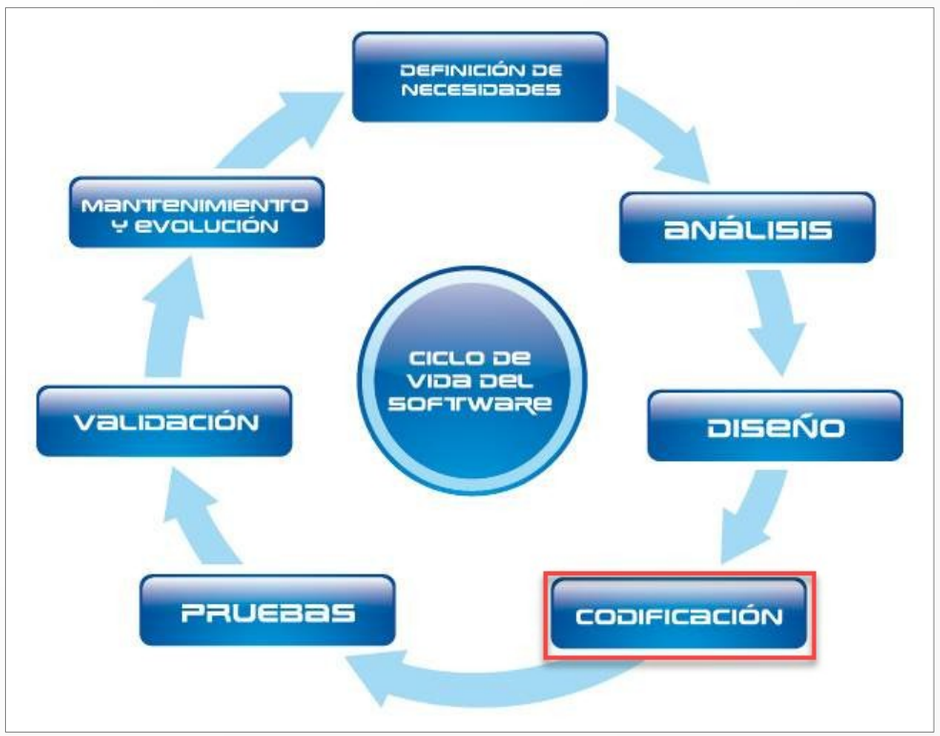
3.1.2 SISTEMAS CONTROL DE VERSIONES (VCS)

3.1.3 TIPOS DE VCS

3.1.4 INTRODUCCIÓN A GIT MONOUSUARIO

3.1.1 Herramientas CASE

- Hace años….
  - … un ~~programador~~ desarrollador era una persona que, **habitualmente en solitario**, desarrollaba aplicaciones con un número reducido de ficheros y una vida útil corta.
- Hoy en día …
  - … dado el expansivo uso de las TIC y gracias a Internet, el desarrollo de aplicaciones ha tenido un crecimiento exponencial y **la programación (fase de codificación) se realiza de manera colaborativa,** incluso entre equipos de **programadores** de todo el mundo.



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 definirlas 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

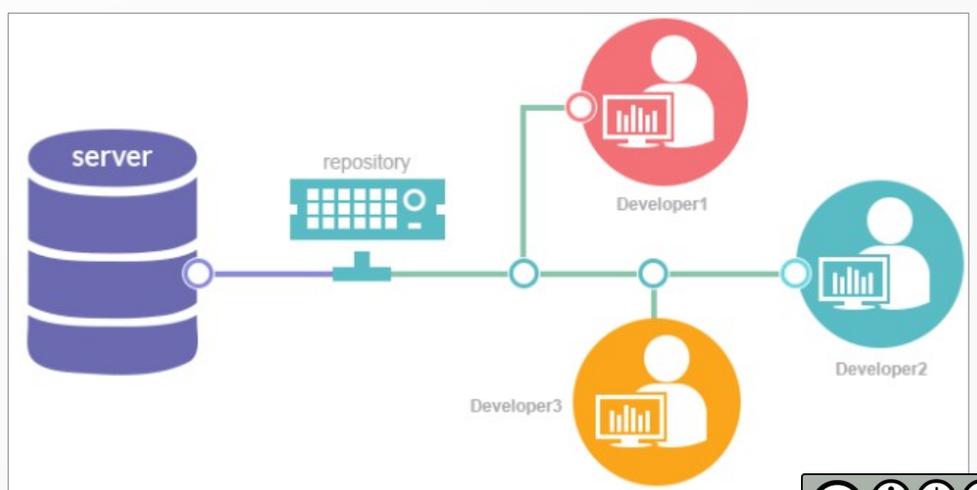
3.1.2 SISTEMAS CONTROL DE VERSIONES (VCS)

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.atareao.es/software/ofimatica/libreoffice-y-el-control-de-versiones-para-incorregibles/

3.1.2 SISTEMAS CONTROL DE VERSIONES (VCS)

- Funcionamiento:

  - Los ficheros de código fuente se guardan en un servidor
  - Los ficheros de este servidor no se modifican directamente
  - Cada desarrollador tiene su propia copia del código
  - Si un desarrollador quiere modificar un fichero, informa de ello previamente
  - Cuando acaba con la modificación, actualiza el fichero del servidor
  - Si dos desarrolladores modifican el mismo fichero, el sistema resuelve los conflictos automáticamente línea por línea, es decir, realiza automáticamente la mezcla entre las líneas que añade cada programador.
  - Si ambos programadores modifican las mismas líneas, el sistema avisa al último desarrollador para que resuelva el conflicto de manera manual.

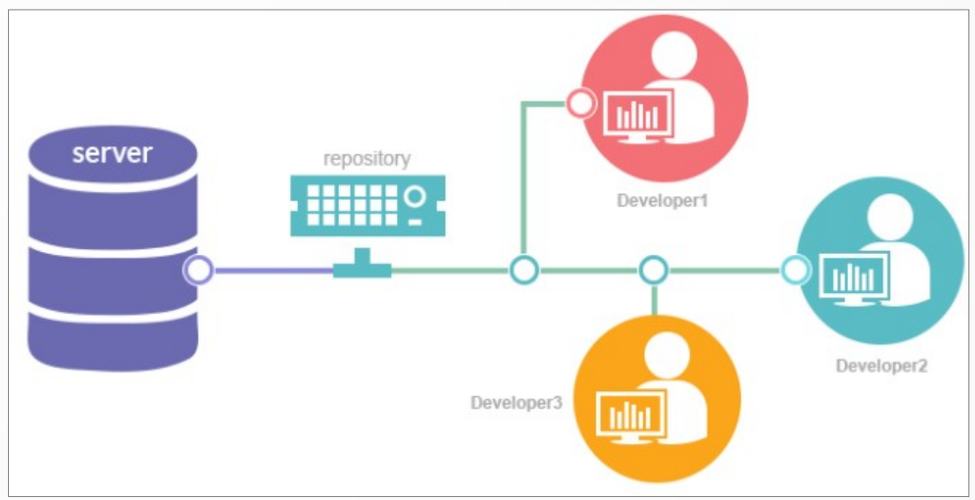
3.1.2 SISTEMAS CONTROL DE VERSIONES (VCS)

- **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)

- Vocabulario básico común a todo VCS

Repositorio *(repository)*	• base de datos donde se guardan los ficheros
Servidor *(server)*	• ordenador donde se guarda el repositorio
Cliente *(client)*	• ordenadores que están conectados al repositorio
Copia de trabajo o de directorio *(working copy or workings setk)*	• directorio local (en clientes) en el que se hacen los cambios
Tronco o principal *(trunk or main)*	• línea principal de código en el repositorio
Cabecera *(head)*	• última versión en el repositorio
Revisión *(revision)*	• versión en la que se encuentra un fichero
Rama *(branch)*	• copia separada de un fichero o una carpeta para uso privado
Conflicto *(conflict)*	• situación que ocurre cuando se intentan aplicar dos cambios contradictorios de un mismo fichero sobre el repositorio
Mensaje de registro *(checkin message)*	• mensaje corto que indica qué se ha cambiado en el fichero
Histórico de cambios *(changelog or history)*	• 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	• Añadir un fichero al repositorio por primera vez para permitir al sistema hacer su seguimiento
get latest/check out	• Traer desde el repositorio la última versión del fichero
check out for edit	• Traer desde el repositorio la última versión del fichero en modo "editable". En muchos sistemas el check out ya es editable.
check in/commit	• 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
merge	• Mezclar los cambios de un fichero a otro para actualizarlo.
resolve	• Resolver los problemas encontrados al hacer check in. Una vez solucionados, se hace el check in de nuevo.
diff /delta	• Comparar dos ficheros para encontrar las diferencias
revert	• Revertir la última versión: se descartan los cambios locales y se recarga la última versión que existe en el repositorio
update/syc	• Actualizar todos los ficheros con la última versión del repositorio
locking	• Bloquear un fichero, tomando su control para que nadie más pueda editarlo hasta desbloquearlo
breaking the lock	• Forzar el desbloqueo de un fichero

Workflow de Git

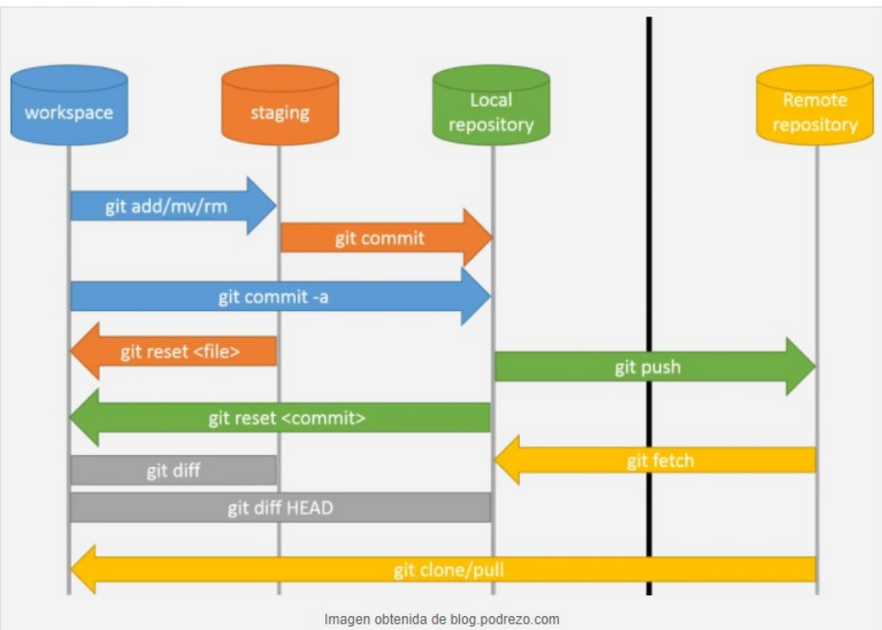


Imagen obtenida de blog.podrezo.com

UD 03.CONTROL Y GESTIÓN DE VERSIONES

3.CONTROL Y GESTIÓN DE VERSIONES

3.1.1 HERRAMIENTAS CASE

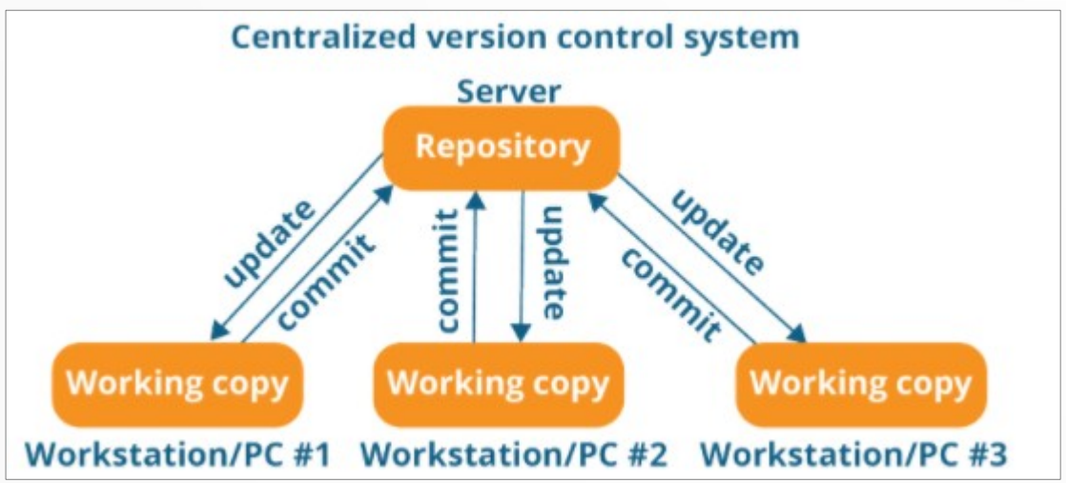
3.1.2 SISTEMAS CONTROL DE VERSIONES (VCS)

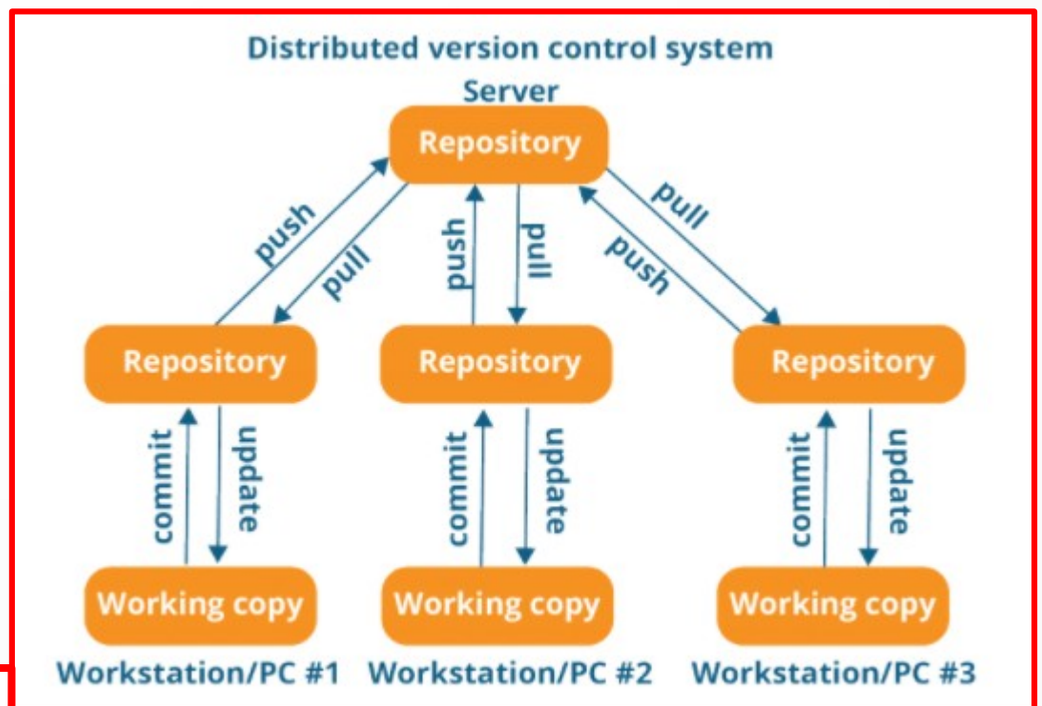
3.1.3 TIPOS DE VCS

3.1.4 INTRODUCCIÓN A GIT MONOUSUARIO

3.1.3 TIPOS DE VCS

- Centralizado (SVN) vs distribuído (GIT)



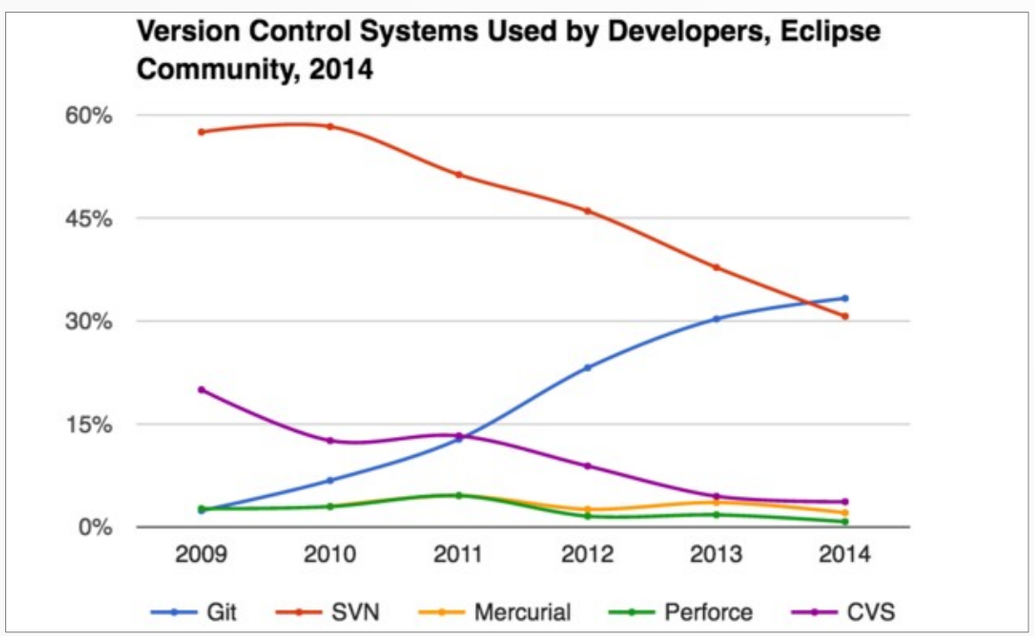


En un sistema distribuido tenemos terminología específica que se añade a la ya vista:

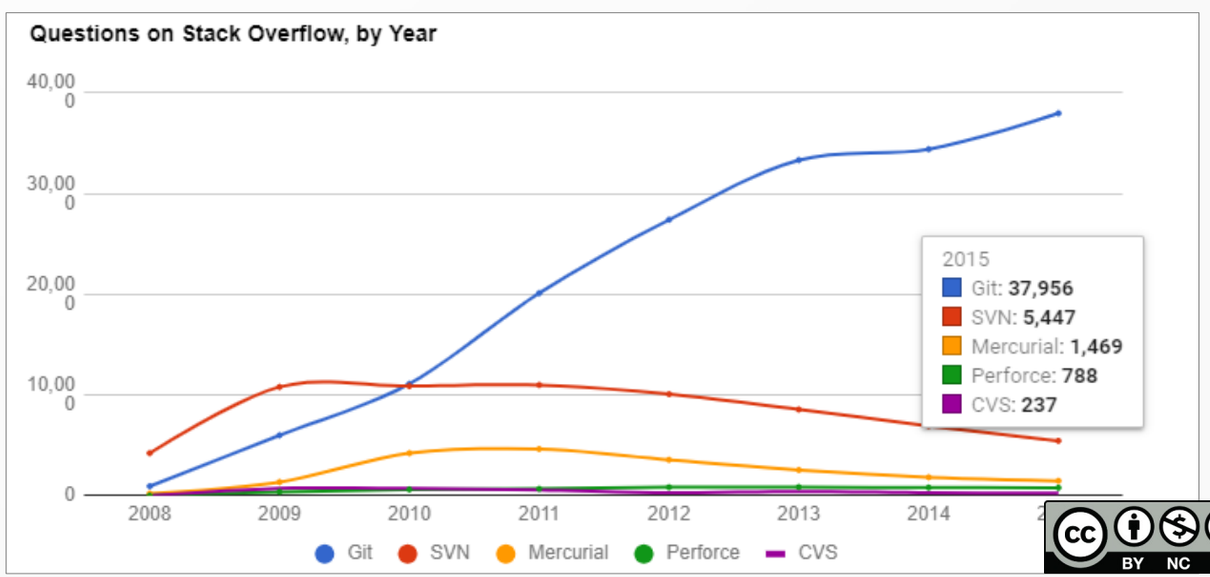
push	• (empujar) Envía un cambio desde un repositorio a otro.
pull	• (extraer) Coge los cambios desde un repositorio
clone	• (clonar) Trae una copia exacta del proyecto desde un repositorio a otro

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.

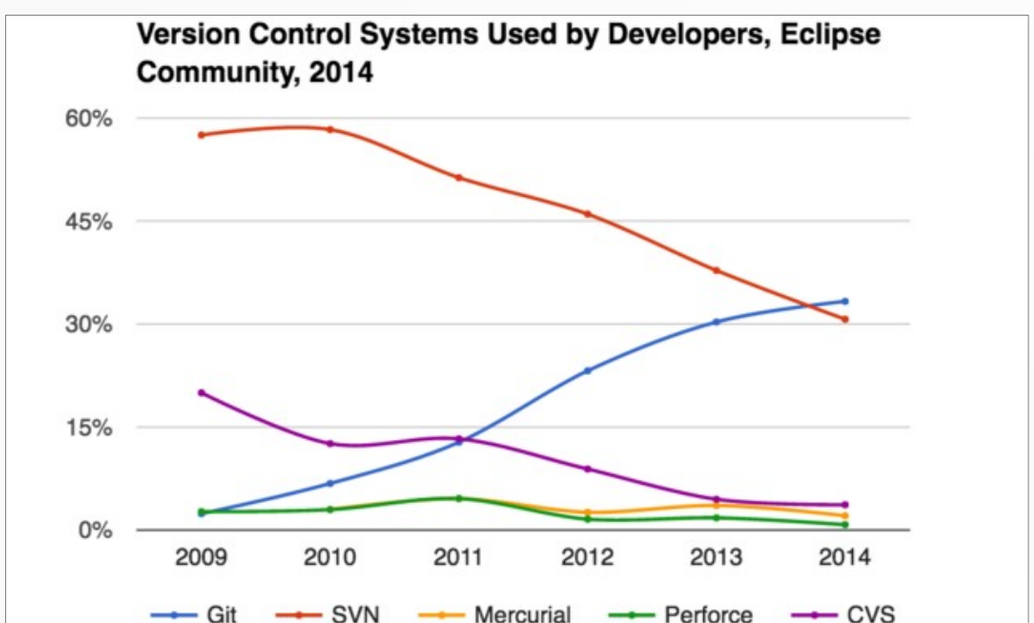


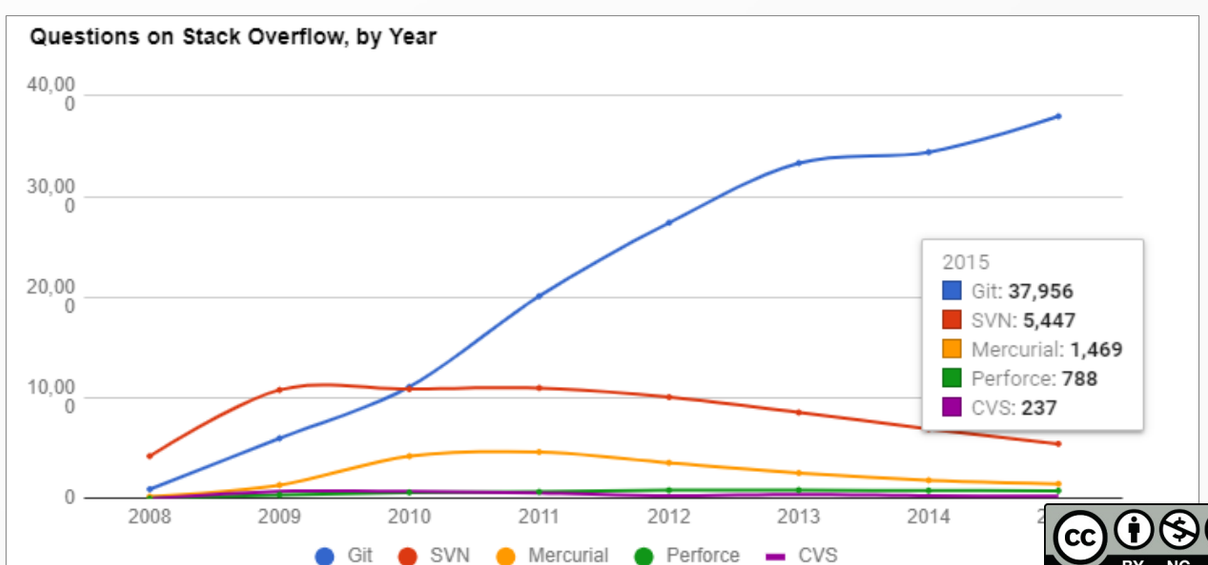
- Ejemplos más conocidos de código abierto (open source):
  - **Concurrent Versions System (CVS):** De código abierto, centralizado sin nueva versión desde 2008.
  - **Subversion (SVN)**: De código abierto y modelo distribuido.
    - FUE uno de los estándares de facto del modelo distribuido durante años.
  - **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)





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.**

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
```

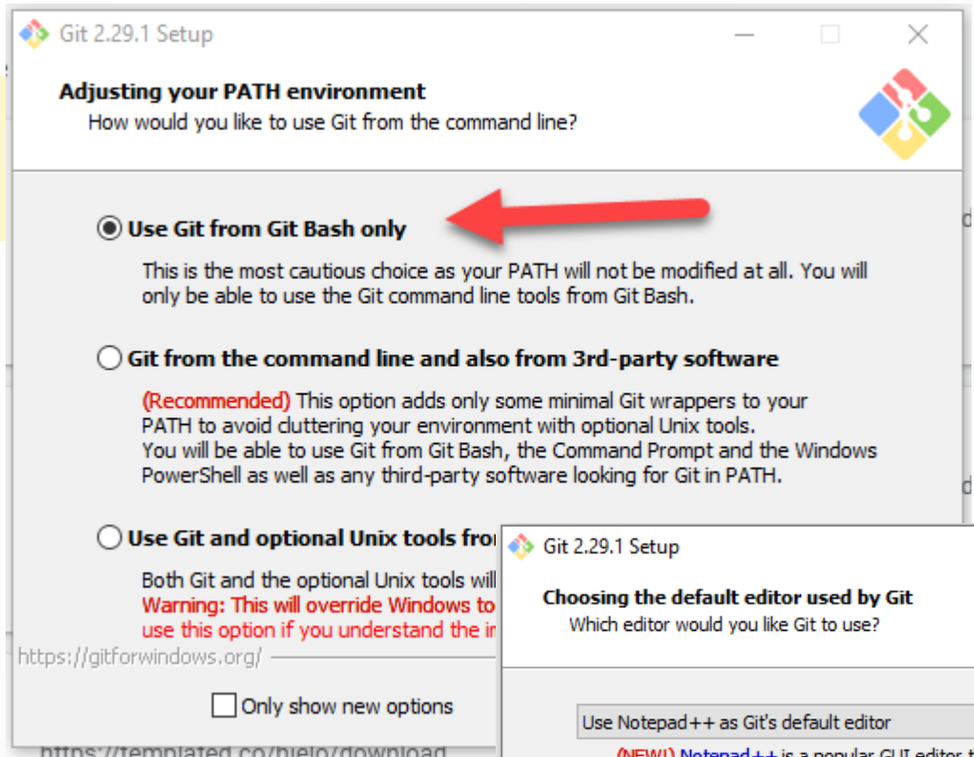
En Windows o MacOSX se utiliza un instalador visual, es decir, que es también muy sencillo.

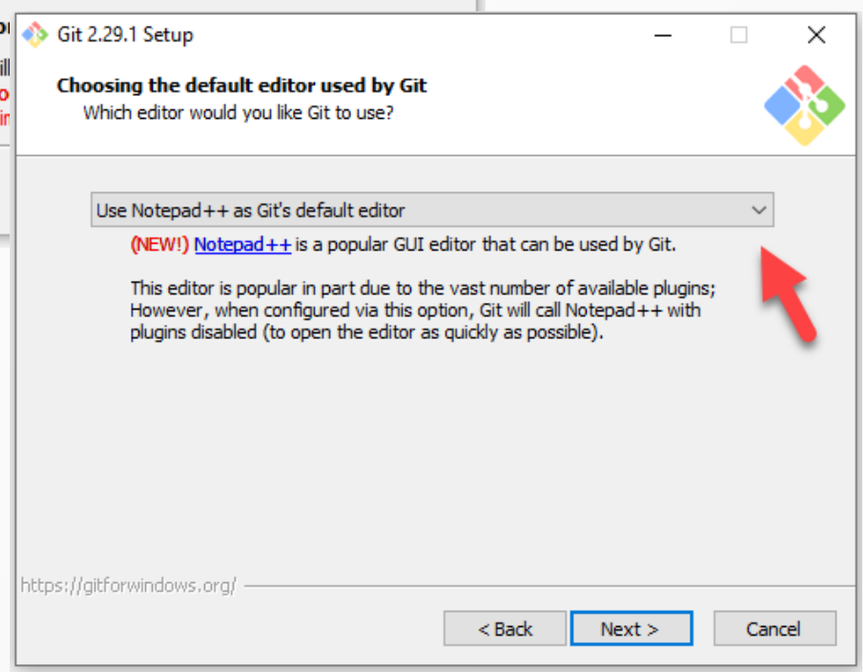
3.1.4 GIT MONOUSUARIO

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**:



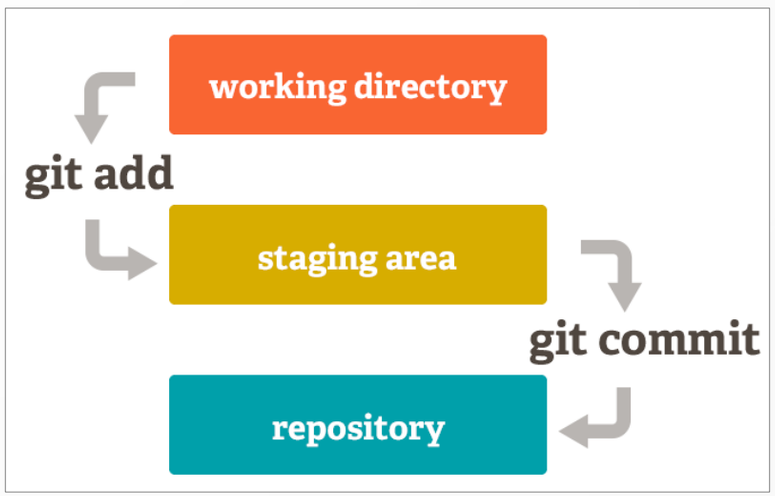


3.1.4 GIT MONOUSUARIO

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) Identifícate [**git config --global user.name XXX**] y

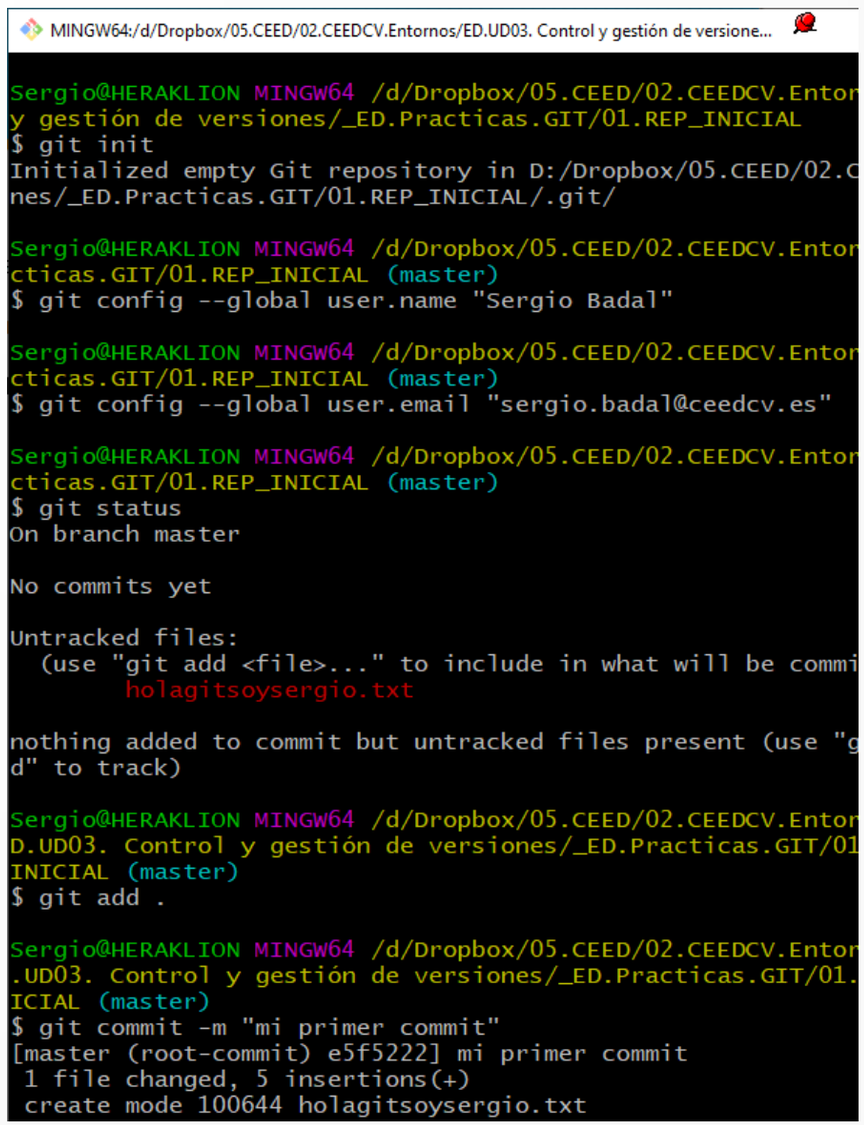
      [git config --global user.email XXX]

   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"**]



3.1.4 GIT MONOUSUARIO

- **Paso 2.2: Nuevos cambios**

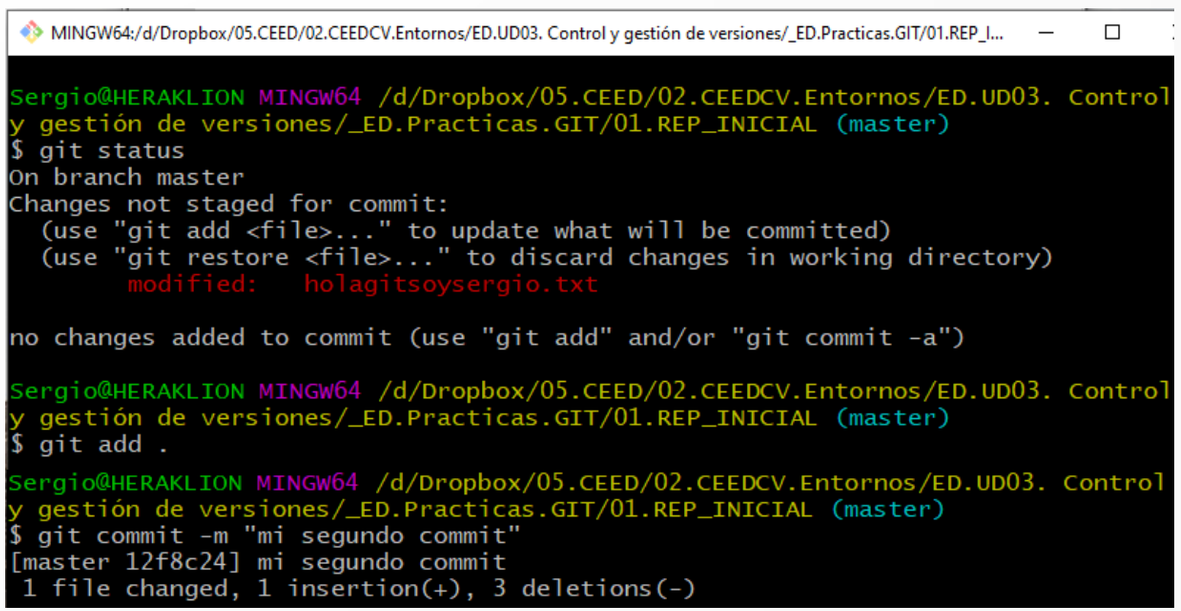
  1) Prueba ahora a hacer cambios en el archivo de prueba, añadiendo otra frase diferente. Por ejemplo "¿Qué tal?" en una nueva línea. Al cambiarlo, debes saber que SOLO se modifica en el WORKSPACE (en la carpeta de tu PC).

  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 al REPOSITORIO

     con [**git commit -m "mi segundo commit"**]



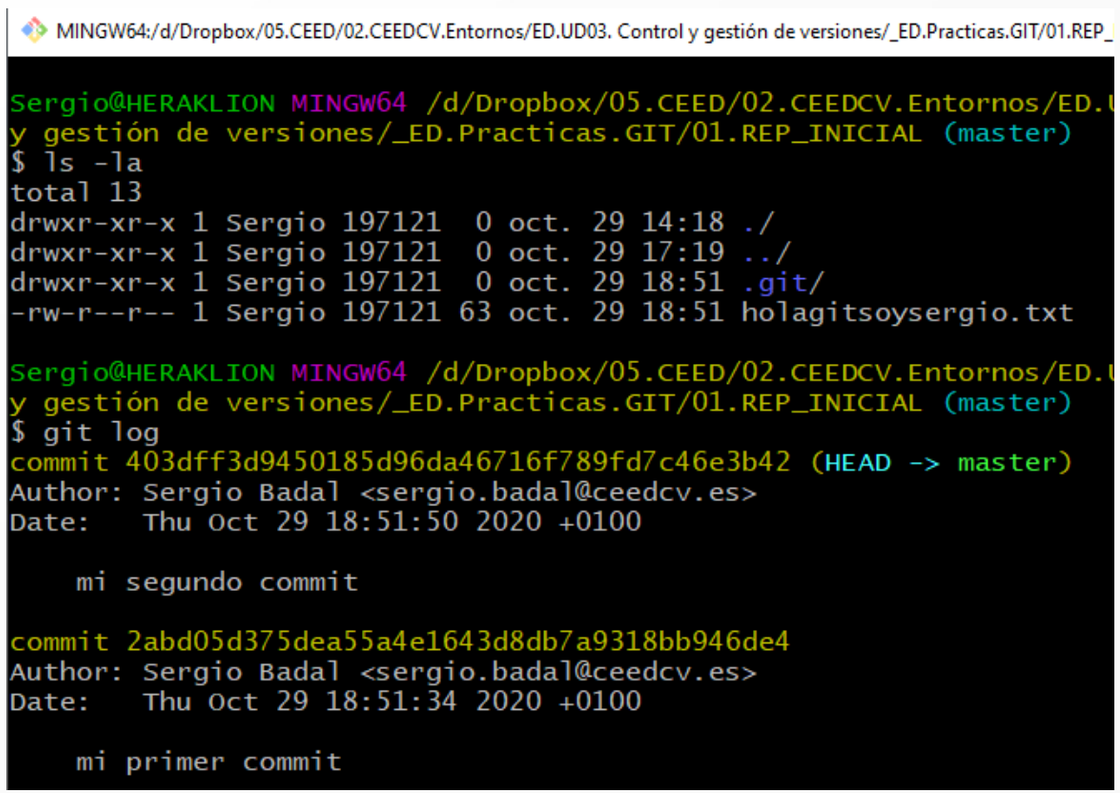
3.1.4 GIT MONOUSUARIO

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

  1) Prueba la órden de la derecha:

     [**git log**] para ver el log

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

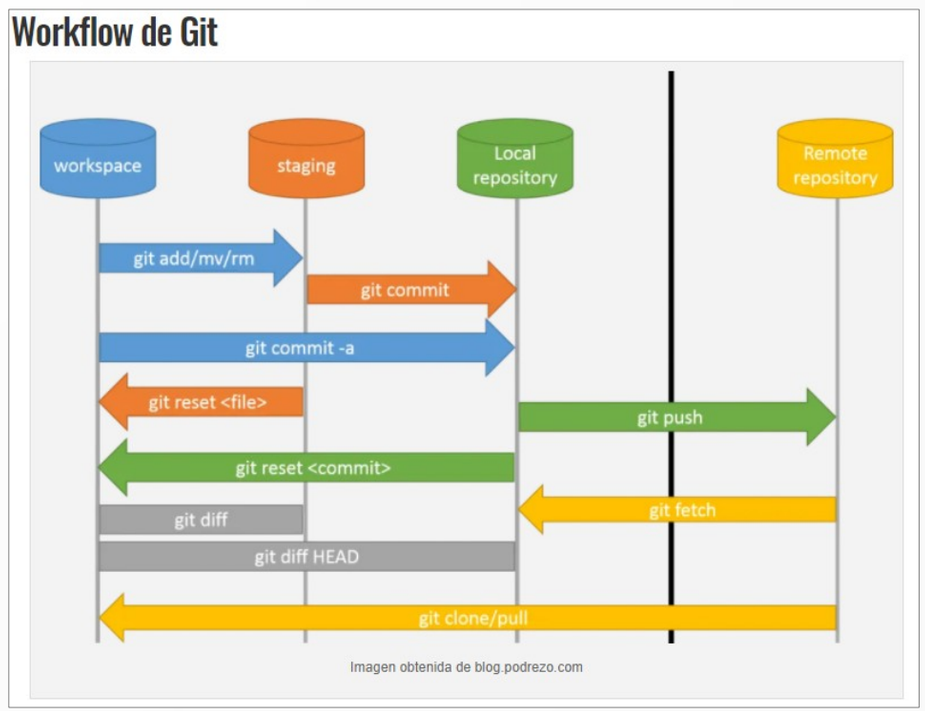


3.1.4 GIT MONOUSUARIO

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**]



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**

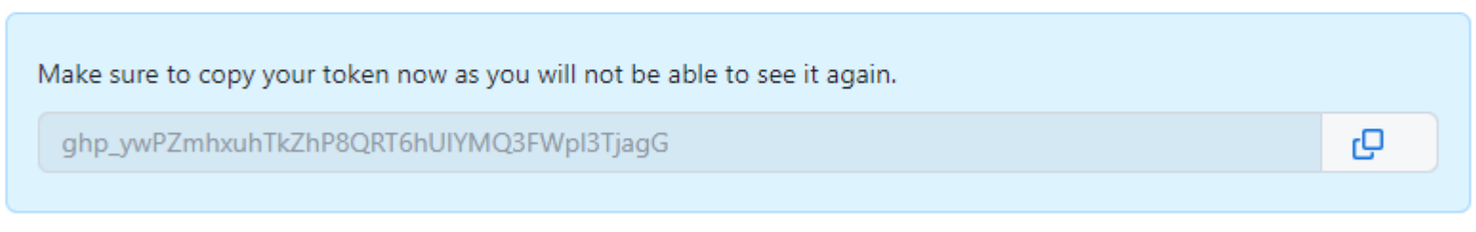
3.1.4 GIT MONOUSUARIO

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:

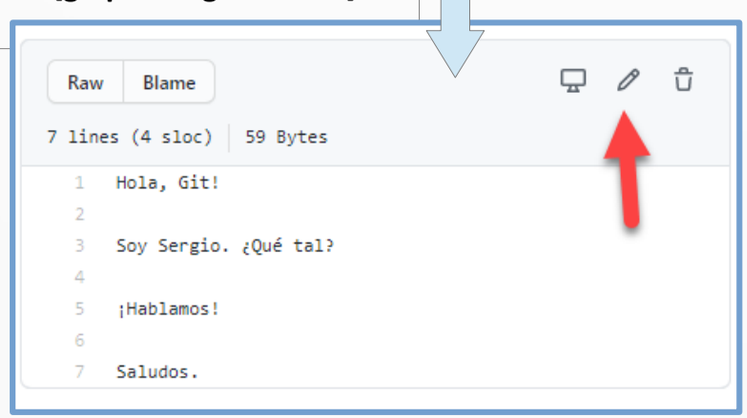
1) Ve al icono de tu cuenta de usuario (arriba a la derecha) y despliega SETTINGS
2) A la izquierda, selecciona > DEVELOPER SETTINGS > PERSONAL ACCESS TOKENS
3) TOKENS (CLASSIC) > GENERATE NEW TOKEN (CLASSIC)
4) PONLE UN NOMBRE > SELECCIONA TODAS LAS OPCIONES > GENERATE TOKEN
5) COPIA EL TOKEN Y GUÁRDALO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

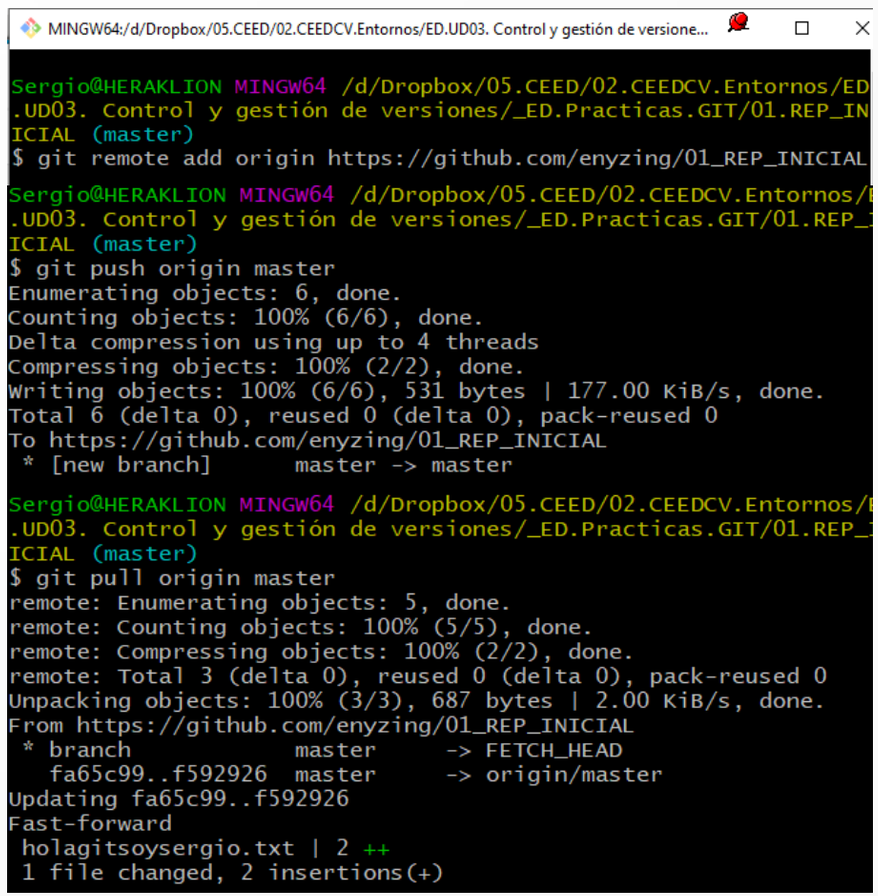


3.1.4 GIT MONOUSUARIO

- **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**]





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_
ICIAL (master)
$ git add .

Sergio@HERAKLION MINGW64 /d/Dropbox/05.CEED/02.CEEDCV.Entornos/
.UD03. Control y gestión de versiones/_ED.Practicas.GIT/01.REP_
ICIAL (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_
ICIAL (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

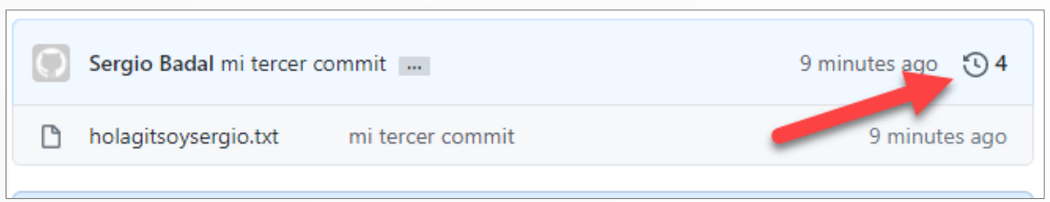
- ## Paso 3.3: Consultar los cambios en el repositorio (log)

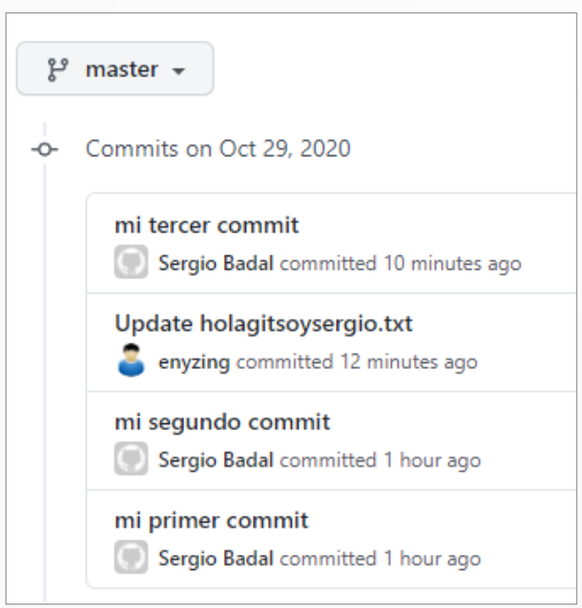
1) Prueba las órdenes de la derecha [**git log**] para ver el log

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
```