UD 03.CONTROL Y GESTIÓN DE VERSIONES

PARTE 1 DE 4: HERRAMIENTAS CASE Y CONTROL DE VERSIONES

v2.0 10.11.20

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

Entornos de desarrollo (ED)

Sergio Badal
Carlos Espinosa
c.espinosamoreno@edu.gva.es

UD 03.CONTROL Y GESTIÓN DE VERSIONES

- **Pasos a seguir**

  1) Lee la documentación (PDF)
  2) Instala el software necesario (sigue los pasos)
  3) Realiza los TESTS todas las veces que quieras
  4) Acude al FORO DE LA UNIDAD
     - Para cualquier duda sobre esta unidad
  5) Acude al FORO DEL MÓDULO
     - Para cualquier duda sobre el módulo

UD 03.CONTROL Y GESTIÓN DE VERSIONES

- **¿Qué veremos en esta UNIDAD?**
  - **SEMANA 1**
    - PARTE 1 DE 4: **HERRAMIENTAS CASE Y CONTROL DE VERSIONES**
      - Herramientas CASE
      - Sistemas de control de versiones (VCS)
      - Tipos de VCS
    - PARTE 2 DE 4: **CONTROL DE VERSIONES MODO MONOUSUARIO (con GIT)**
  - **SEMANA 2**
    - PARTE 3 DE 4: **CONTROL DE VERSIONES MODO MULTIUSUARIO (con GIT)**
    - (INICIO) PRÁCTICA EVALUABLE: CONTROL DE VERSIONES
  - **SEMANA 3**
    - PARTE 4 DE 4: **CONTROL DE VERSIONES: INTEGRACIÓN DE GIT**
    - (ENTREGA) PRÁCTICA EVALUABLE: CONTROL DE VERSIONES

UD 03.CONTROL Y GESTIÓN DE VERSIONES

3.CONTROL Y GESTIÓN DE VERSIONES

3.1.1 HERRAMIENTAS CASE

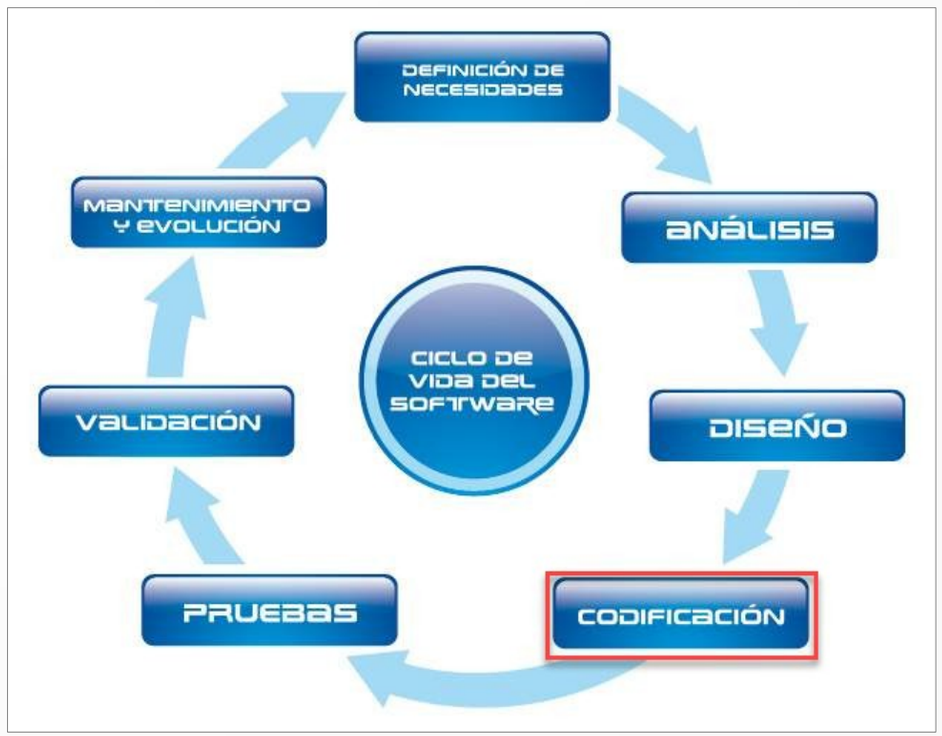
3.1.2 SISTEMAS CONTROL DE VERSIONES (VCS)

3.1.3 TIPOS DE VCS

- *Un apunte.*

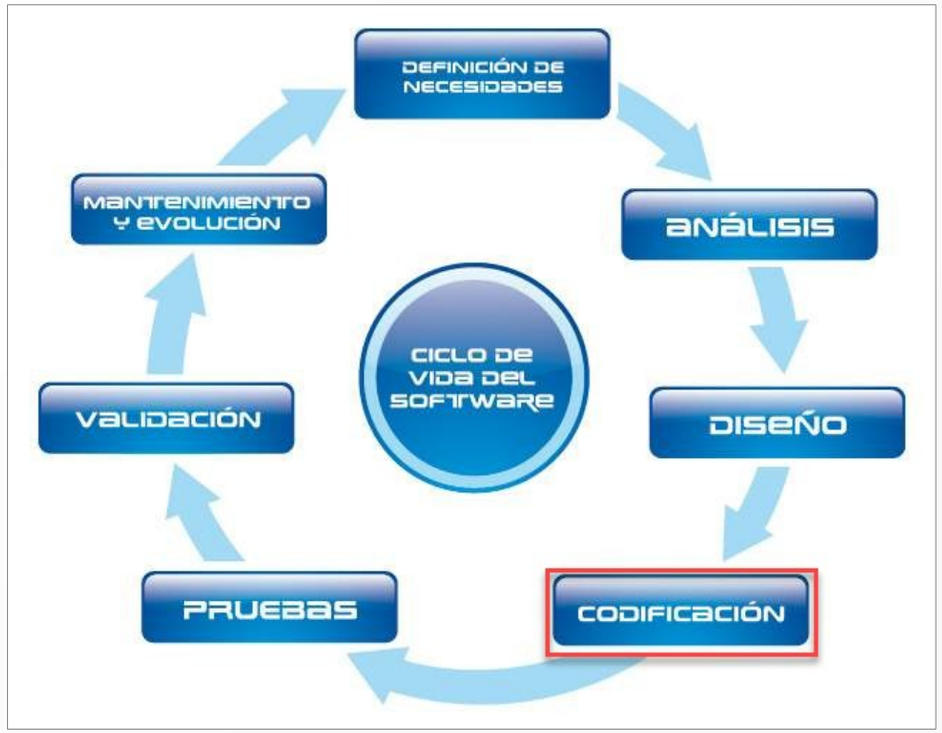
3.1.1 Herramientas CASE

- 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

- 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.



- **PREGUNTA …**
  - ¿Qué fases del ciclo de vida crees que se pueden realizar de manera colaborativa, es decir, por más de una persona al mismo tiempo?
  - Analiza los pros y contras de realizar cada una de ellas de manera colaborativa

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.**

***Debes ir acostumbrándote a los conceptos de CLIENTE y PRODUCTO si quieres comenzar en esto de la ISW.**

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**)

3.1.1 Herramientas CASE

RETO

Los más nuevos en esto de "la programación" solo piensan en los IDE cuando hablamos de herramientas CASE. ¿Conoces alguna herramienta CASE que no pueda agruparse en ninguna de las categorías anteriores? ¿Has usado alguna en concreto, esté o no en estas categorías?

Cuéntanos tu experiencia con "las CASE" en el FORO DE LA UNIDAD...

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

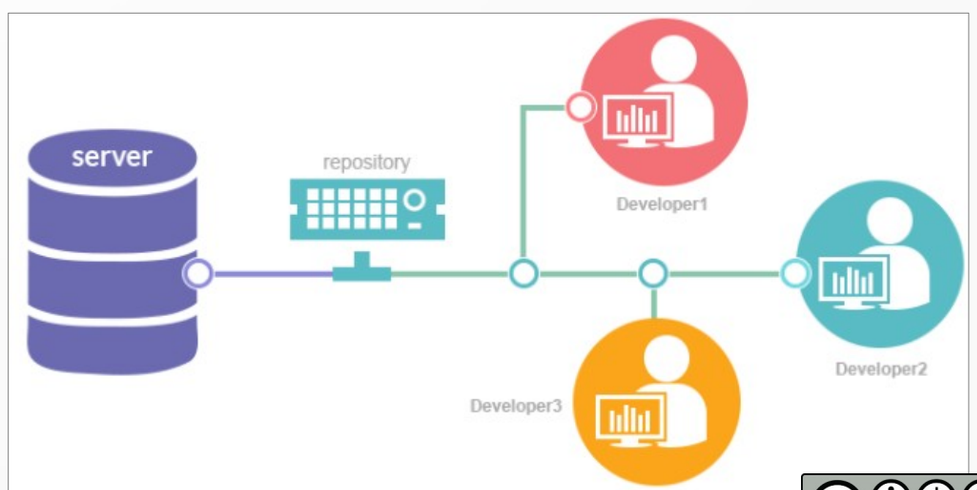
3.1.2 SISTEMAS CONTROL DE VERSIONES (VCS)

3.1.3 TIPOS DE VCS

- *Un apunte.*

3.1.1 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.

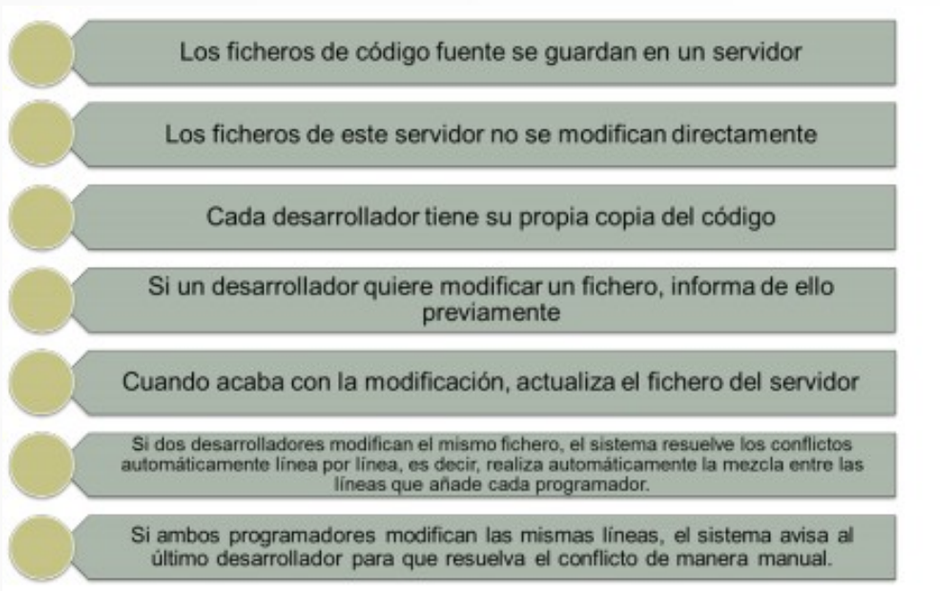


3.1.1 SISTEMAS CONTROL DE VERSIONES (VCS)

- 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/

3.1.1 SISTEMAS CONTROL DE VERSIONES (VCS)

- Funcionamiento:



3.1.1 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

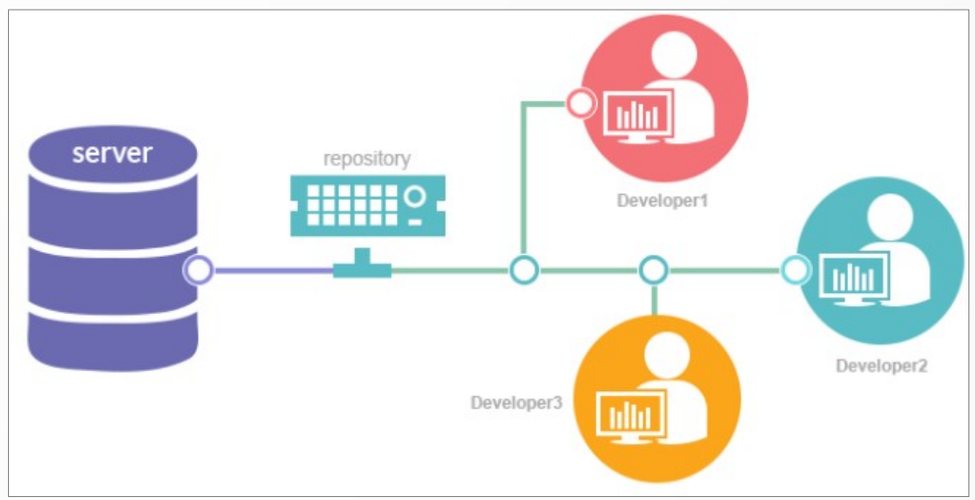
3.1.1 SISTEMAS CONTROL DE VERSIONES (VCS)

- 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.

3.1.1 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



3.1.1 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

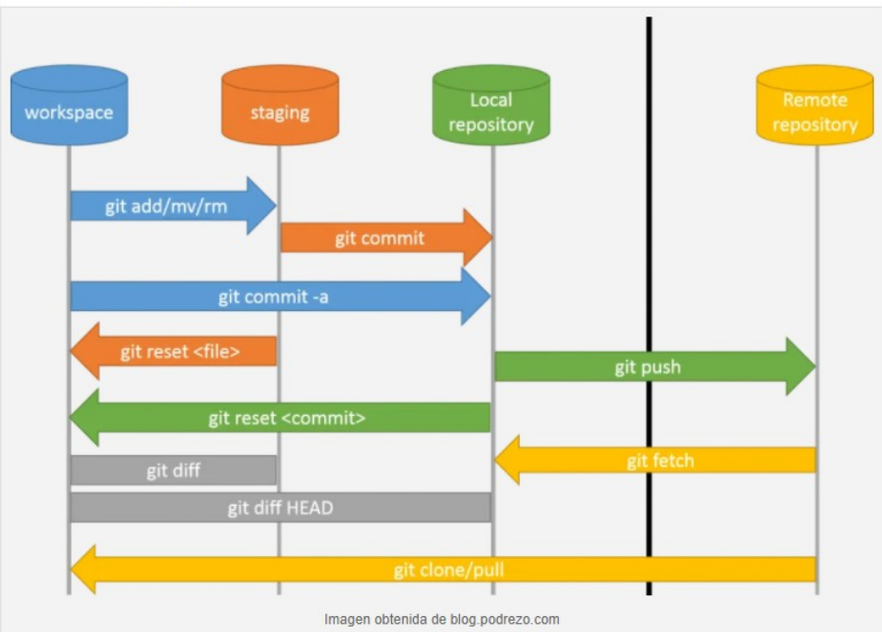


Imagen obtenida de blog.podrezo.com

UD 03.CONTROL Y GESTIÓN DE VERSIONES

3.CONTROL Y GESTIÓN DE VERSIONES

3.1.1 HERRAMIENTAS CASE

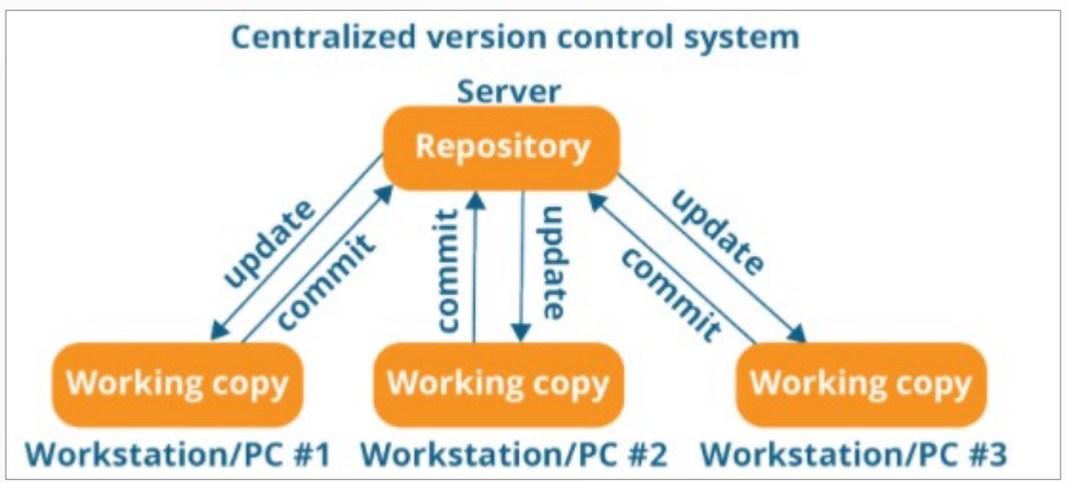
3.1.2 SISTEMAS CONTROL DE VERSIONES (VCS)

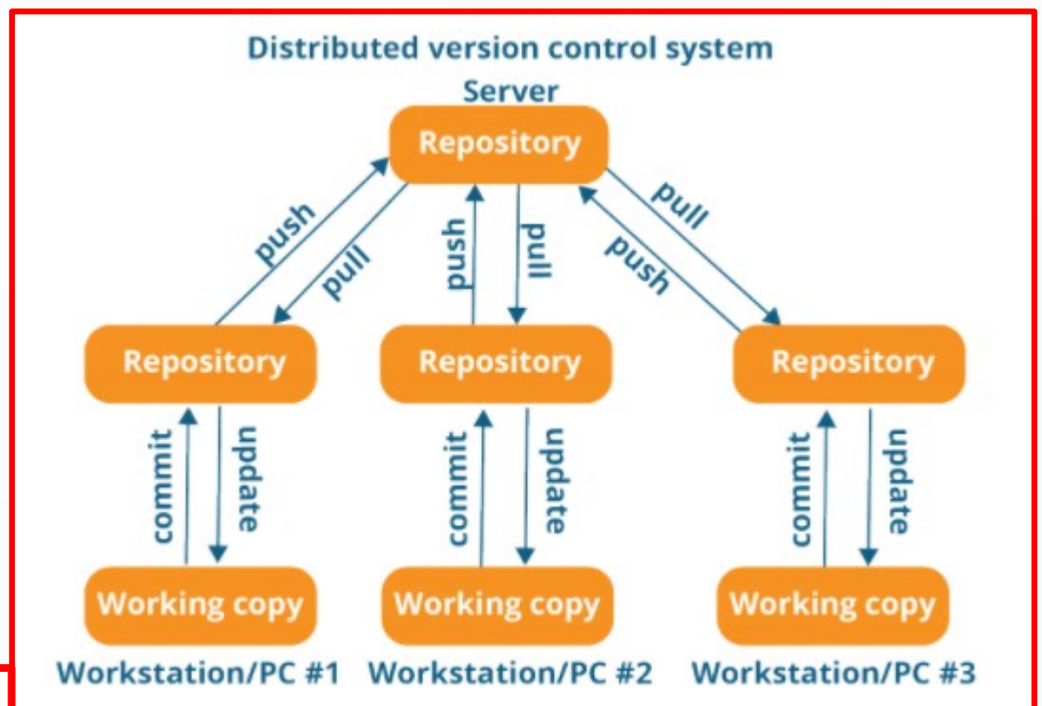
3.1.3 TIPOS DE VCS

- *Un apunte.*

3.1.3 TIPOS DE VCS

- Centralizado (SVN) vs distribuído (GIT)



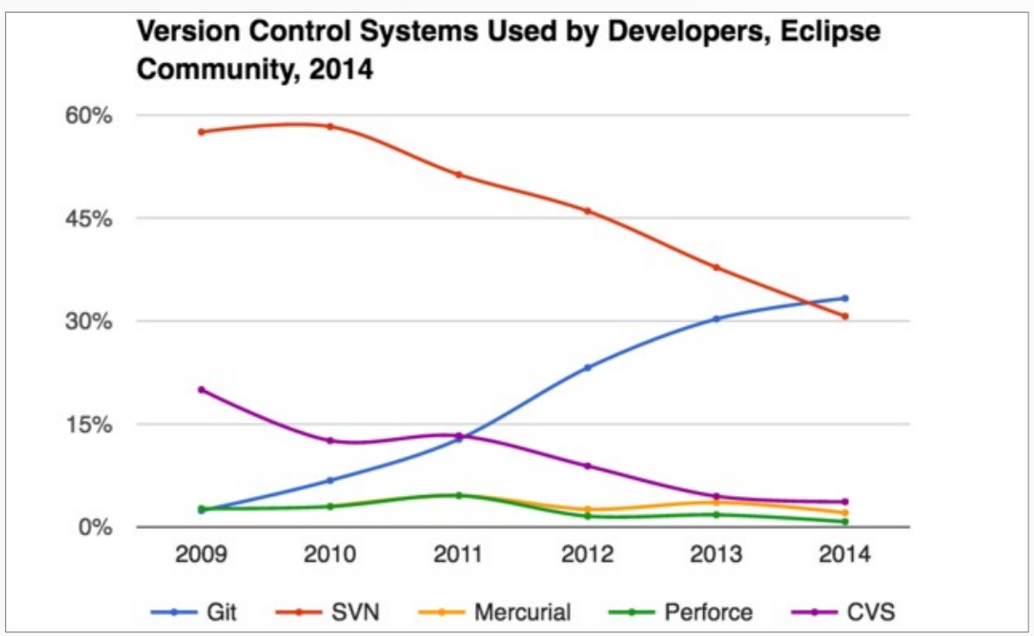


En un sistema distribuido tenemos terminología específica que se añade a la ya vista:

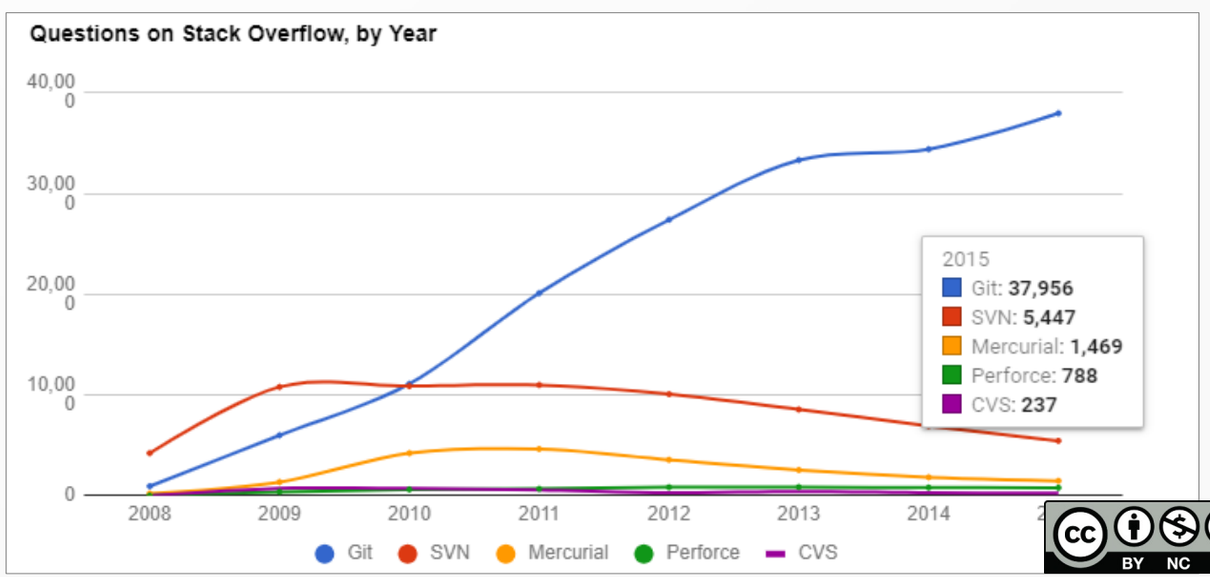
push	• (empujar) Envía un cambio desde un repositorio a otro.
pull	• (extraer) Coge los cambios desde un repositorio
clone	• (clonar) Trae una copia exacta del proyecto desde un repositorio a otro

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.

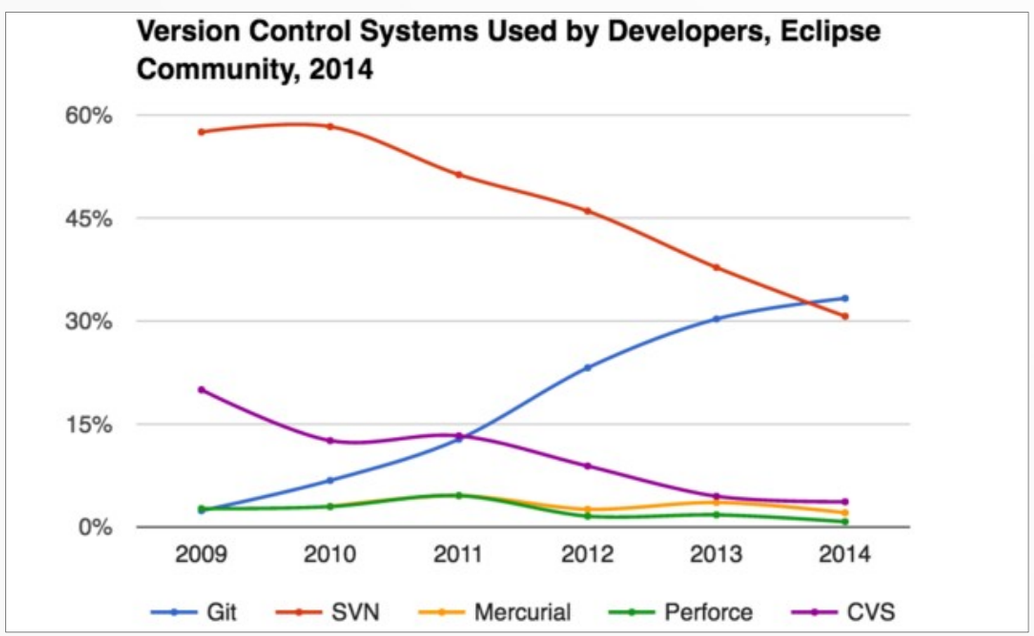


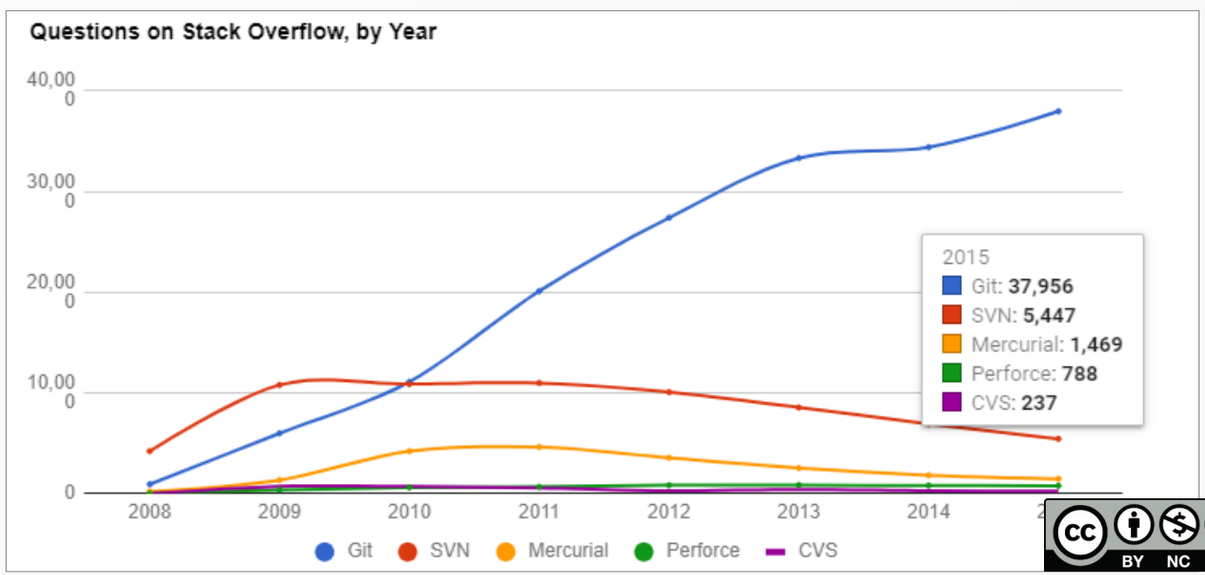
- Ejemplos más conocidos de código abierto (open source):
  - **Concurrent Versions System (CVS):** De código abierto, centralizado sin nueva versión desde 2008.
  - **Subversion (SVN)**: De código abierto y modelo distribuido.
    - FUE uno de los estándares de facto del modelo distribuido durante años.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

- ***Un apunte.***

UN APUNTE

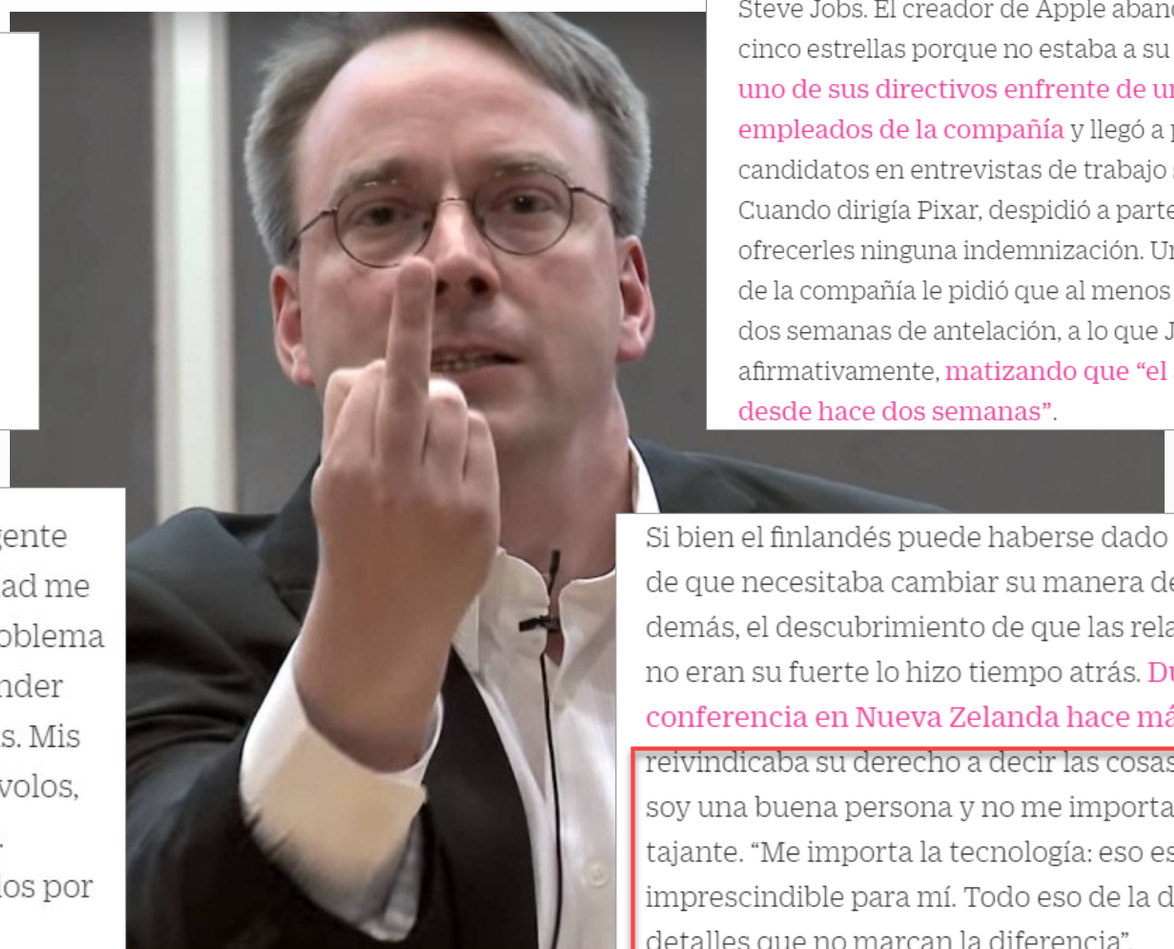
- La importancia de la gestión de las emociones

Linus Torvalds, el genio con carácter que quiere dejar de parecerse a Steve Jobs

Por El País Retina

https://retina.elpais.com/retina/2018/09/18/talento/1537283687_671772.html

"Recientemente, la gente de nuestra comunidad me advirtió sobre mi problema a la hora de comprender las emociones ajenas. Mis ataques han sido frívolos, no han sido profesionales y estaban fuera de lugar. Quiero pedir disculpas a quienes se vieron afectados por mi comportamiento", se sinceró a través de un comunicado.



La exigencia de Torvalds con quienes trabajan en su plataforma tiene su particular precedente en una de las figuras más mitificadas de la revolución tecnológica: Steve Jobs. El creador de Apple abandonó un hotel de cinco estrellas porque no estaba a su altura, despidió a uno de sus directivos enfrente de un auditorio lleno de empleados de la compañía y llegó a preguntar a candidatos en entrevistas de trabajo sobre su virginidad. Cuando dirigía Pixar, despidió a parte de la plantilla sin ofrecerles ninguna indemnización. Una antigua empleada de la compañía le pidió que al menos se les anunciara con dos semanas de antelación, a lo que Jobs contestó afirmativamente, matizando que "el aviso es retroactivo desde hace dos semanas".

Si bien el finlandés puede haberse dado cuenta estos días de que necesitaba cambiar su manera de actuar con los demás, el descubrimiento de que las relaciones sociales no eran su fuerte lo hizo tiempo atrás. Durante una conferencia en Nueva Zelanda hace más de tres años, reivindicaba su derecho a decir las cosas sin tapujos. "No soy una buena persona y no me importa", afirmaba tajante. "Me importa la tecnología: eso es lo imprescindible para mí. Todo eso de la diversidad son detalles que no marcan la diferencia".

UN APUNTE

- La gestión de las emociones es tan importante como la adquisición de destrezas y conocimientos.
- Sin una correcta gestión de emociones tu vida personal y profesional ***puede*** ser un fracaso.