



Sistemas de control de versiones

Raúl García García

Curso 2024/2025

Universidad San Pablo-CEU


Escuela Politécnica Superior

Campus de Montepríncipe

1 Git es un sistema de control de versiones

Guardar los ficheros en el disco basicamente.

Tiene una evolucion temporal, no tiene una version, tiene todas



¿Qué es control de versiones?

- » Gestión del desarrollo de cada elemento de un proyecto a lo largo del tiempo
- » Proporciona:
 - **Mecanismo de almacenaje** de cada **elemento** que deba gestionarse (archivos de código, imágenes, documentación...)
 - Posibilidad de **añadir, modificar, mover, borrar ...** esto es trabajar para el sistema de cont de ver
 - **Historial** de las acciones realizadas con cada elemento pudiendo volver a un estado anterior
 - Otros: generación de **informes** de cambios, informes de estado, **marcado** con nombre identificativo, etc.
- » Se trabaja sobre un **repositorio**, donde se almacena la información de todo el desarrollo
 - Útil para trabajar individualmente o en grupo
 - Alojado en un servidor local o remoto
 - Permite desarrollos colaborativos, incluso concurrentemente
 - Todo buen equipo profesional de desarrollo de software lo utiliza

18-ene.-25

Curso 2024/2025

página 2


Repositorio es un almacen

Cualquier fichero que sea relevante se guarda.

Si se cae la red, pues no se puede trabajar.

Esto se arregla si el repo no esta en un solo sitio. Cada uno tiene su copia. Lo malo es que como hay muchas copias, hay que unir las al final.

Los centralizados no tienen problemas de merge, porque solo hay una copia.



Tipos de VCS

» Centralizados → VCS En un solo sitio

- El repositorio se encuentra en una **localización única**
- Es necesario tener acceso a la ubicación del repositorio para poder trabajar
 - Esto implica generalmente **acceso a la red** (local o Internet)
- Ejemplos:
 - scscs, RCS, CVS, Subversion, ...*

» Distribuidos → DVCS

- No hay un repositorio único sino **múltiples copias (clones)**
- Para trabajar, un usuario debe obtener una **copia local** del repositorio, pero **no** necesita acceso a la red más que para publicar sus cambios
- Ejemplos:
 - Git (C), Mercurial, Bazaar (Python), fossil (C), BitKeeper, darcs, arch, monotone, ...*

18-ene.-25

Curso 2024/2025

página 3

3

Asumiendo que ya tienes el repositorio

sincroniza local con contenidos del repo

Nueva version con esos contenidos. Push para expandirlo a todos



Git

Ciclo básico de trabajo

0. **Crear** copia local → **checkout**
Se puede especificar una revisión o fecha particular

1. **Actualizar** la copia de trabajo → **update**
Permite recuperar las últimas modificaciones del repositorio e integrarlas en la copia de trabajo

2. Realizar **cambios** → **add, delete, copy, move**

3. **Examinar** cambios → **status, diff, revert**

4. **Fusionar** cambios → **merge, resolved**

5. **Enviar** cambios → **commit**
Requiere un mensaje 'log' que detalle las modificaciones realizadas

6. **Volver** al punto 1 (**update**)


18-ene.-25

Curso 2024/2025

página 4

Una copia local es una vista del repo en un momento determinado del tiempo

4




Vocabulario básico

- » **Copia de trabajo** (*working copy*)
 - Copia local de los ficheros de un repositorio (de una revisión específica)
- » **Check-out** (*co*) *sacar copia local*
 - Crea una copia de trabajo local desde el repositorio
 - Se puede especificar una revisión; si no se hace se toma la última
- » **Check-in** o **commit** (*ci*) *meter la copia*
 - Integra los cambios hechos a una copia local en el repositorio
- » **Import** *meter en repo vacío árbol con estructura*
 - Copia un árbol de directorios local (que no es en ese momento una copia de trabajo) en el repositorio por primera vez
- » **Actualizar** (*update*)
 - Integra los cambios que han sido hechos en el repositorio (por ejemplo por otras personas) en la copia de trabajo local
- » **Conflicto**
 - Ocurre cuando se realizan dos cambios al mismo documento, y el sistema es incapaz de reconciliar los mismos
- » **Resolver**
 - Intervención del usuario para atender un conflicto entre diferentes cambios al mismo documento

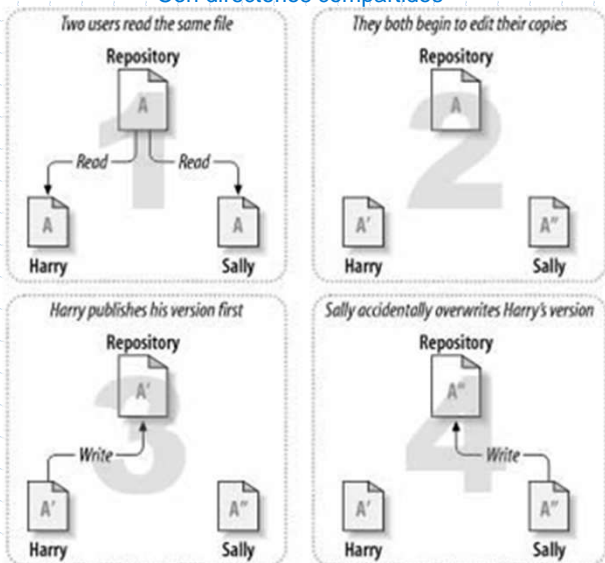
18-ene.-25 Curso 2024/2025 página 5

5



El problema de compartir ficheros

Con directorios compartidos



Harry entrega antes

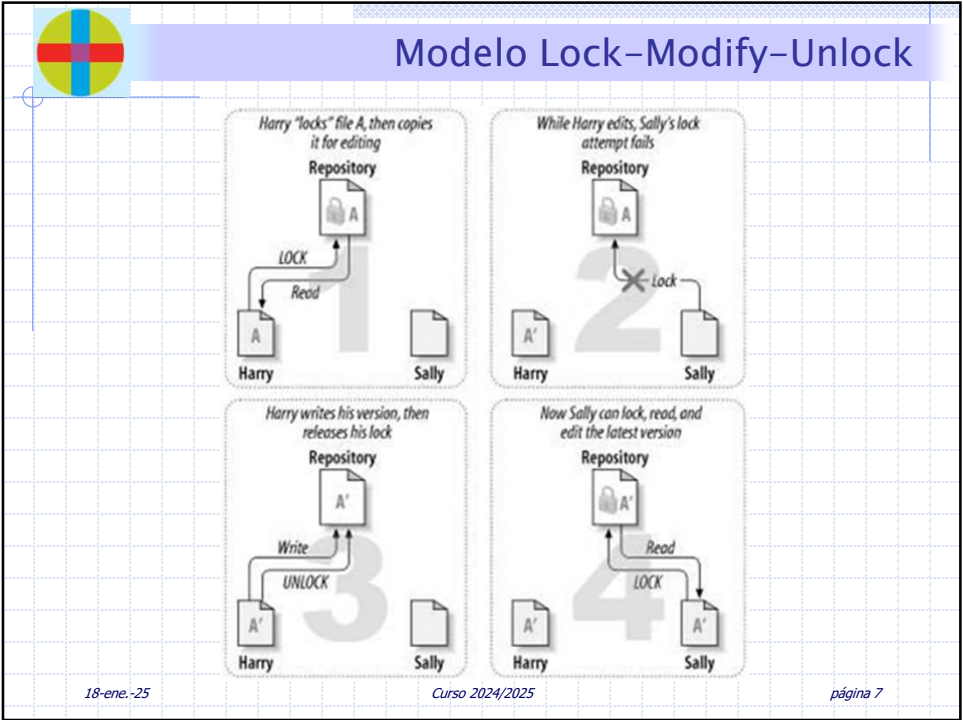
Sally despues, pero se pierde la de Harry

18-ene.-25 Curso 2024/2025 página 6

6

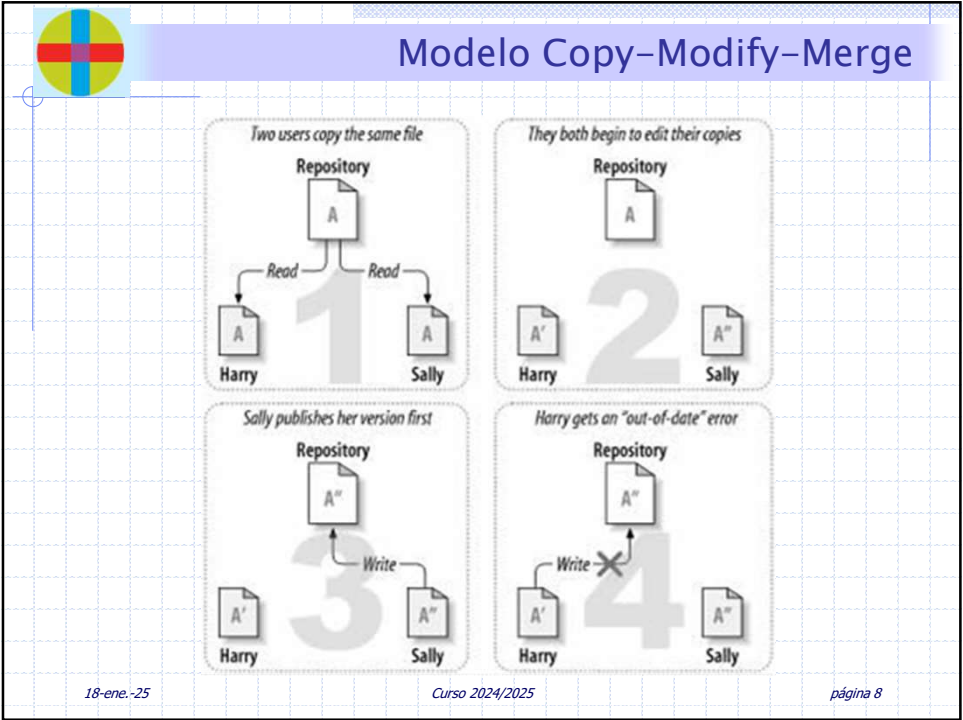
Esperar a que acabe el otro, para poder acceder

Cuando lo abre ya Sally ya tiene los cambios



7

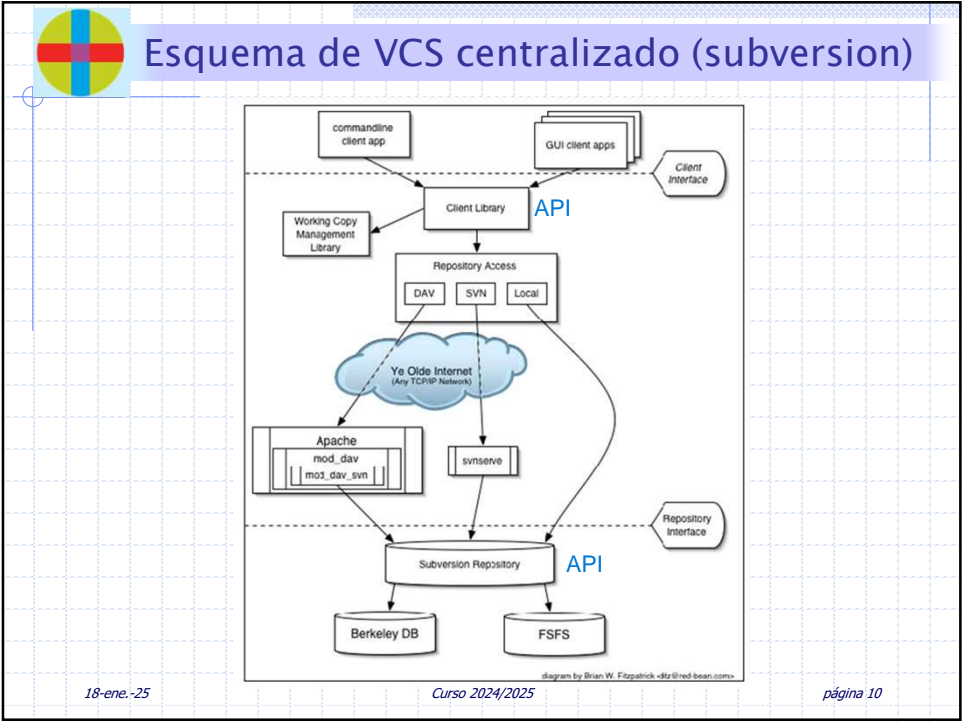
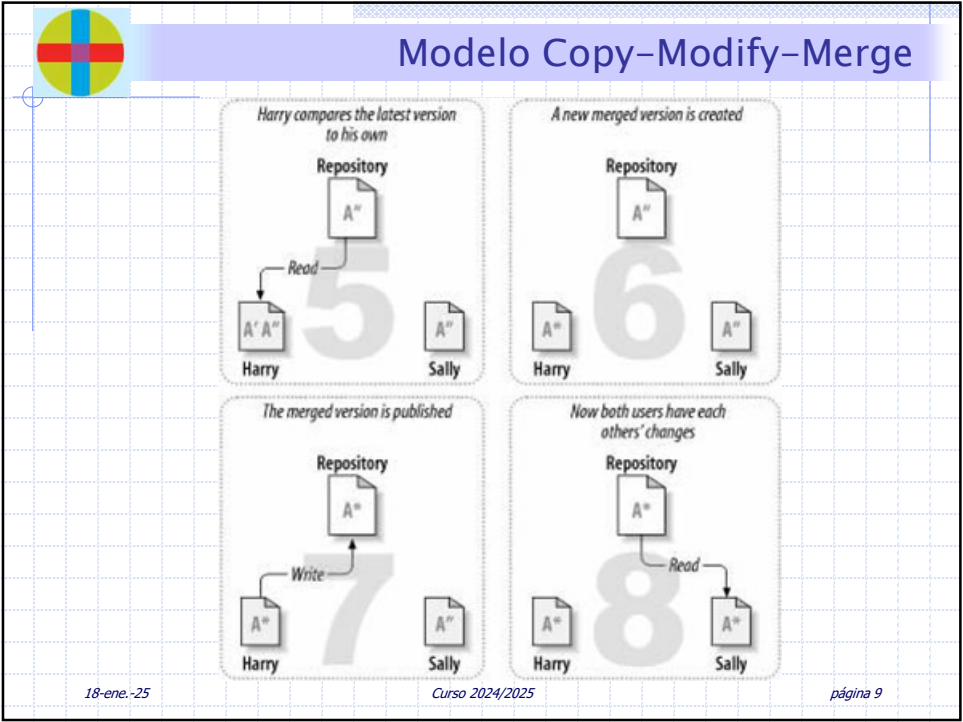
Deja que todo el mundo lea a la vez, pero si entrega alguien antes ya no te deja entregar




8

tienes que volver a coger el archivo nuevo, y fusionarlo con el tuyo.

Un merge





Consejos y buenas prácticas


- » No entregar código que no funciona o inacabado
 - Si hay que hacerlo, crear una **rama** y mezclarla cuando funcione
- » Añadir siempre un **mensaje en cada commit** explicando qué se entrega y por qué
- » Ejecutar **update con mucha frecuencia** y siempre al principio de una sesión de desarrollo
- » Ejecutar **commit con frecuencia**
 - Las entregas muy grandes y espaciadas aumentan la probabilidad de **conflictos**
- » Ante la duda, **resolver los conflictos** con el resto de los miembros del equipo
- » **No versionar ficheros autogenerados por herramientas**
- » Usar un repositorio por proyecto

18-ene.-25 Curso 2024/2025 página 11

11



¿Preguntas?



18-ene.-25 Curso 2024/2025 página 12

12