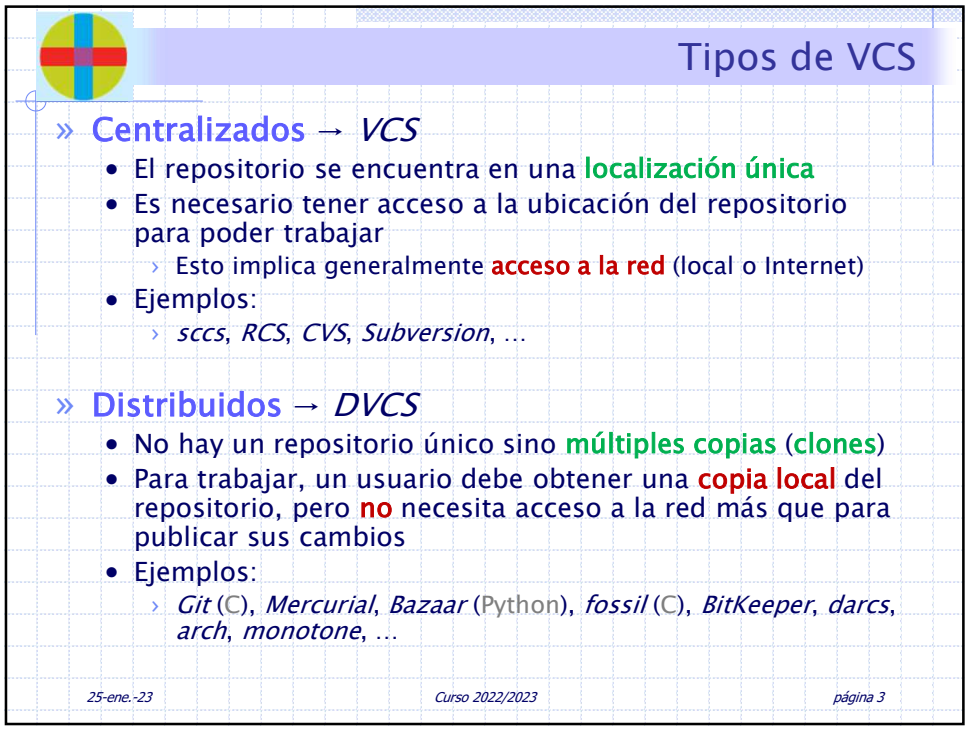
1

¿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** …
  - **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

2


Tipos de VCS
» Centralizados → VCS
• 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:
› sccs, 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, …
25-ene.-23 Curso 2022/2023 página 3

3


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)
25-ene.-23 Curso 2022/2023 página 4

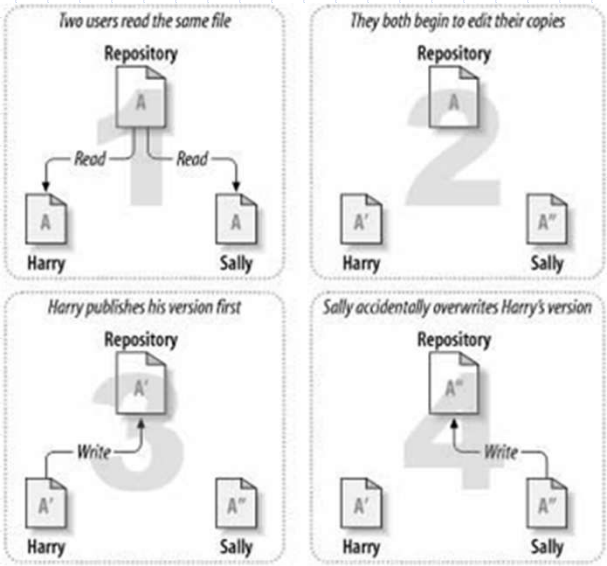
4

Vocabulario básico

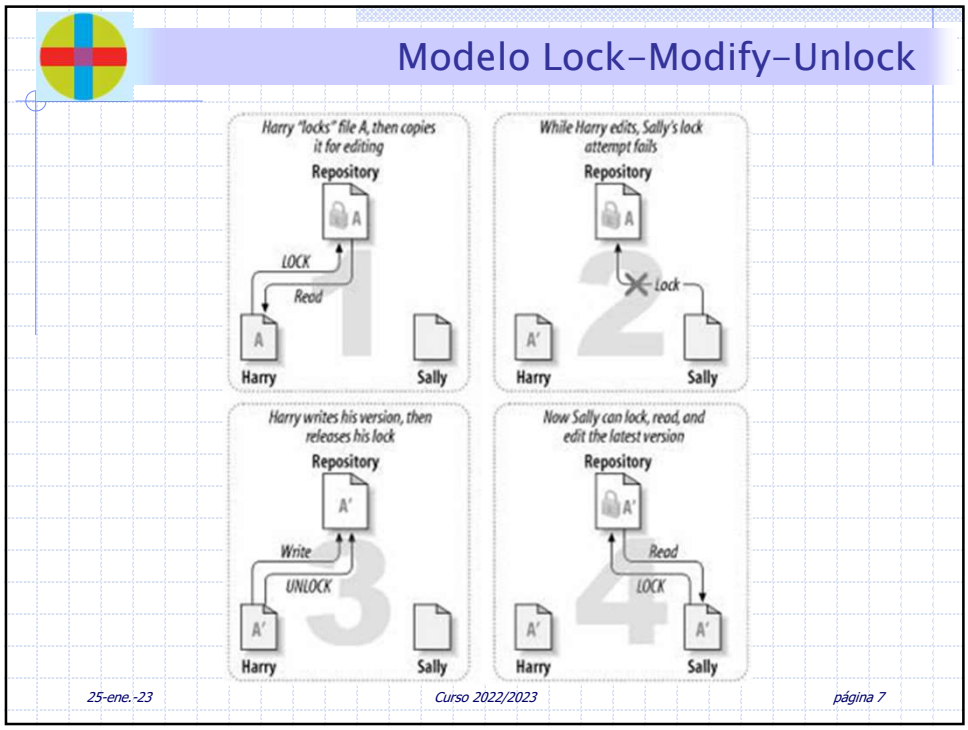
- Copia de trabajo (*working copy*)
  - Copia local de los ficheros de un repositorio (de una revisión específica)
- *Check-out* (*co*)
  - Crea una copia de trabajo local desde el repositorio
  - Se puede especificar una revisión; si no se hace se toma la última
- *Check-in* o *commit* (*ci*)
  - Integra los cambios hechos a una copia local en el repositorio
- *Import*
  - 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

5

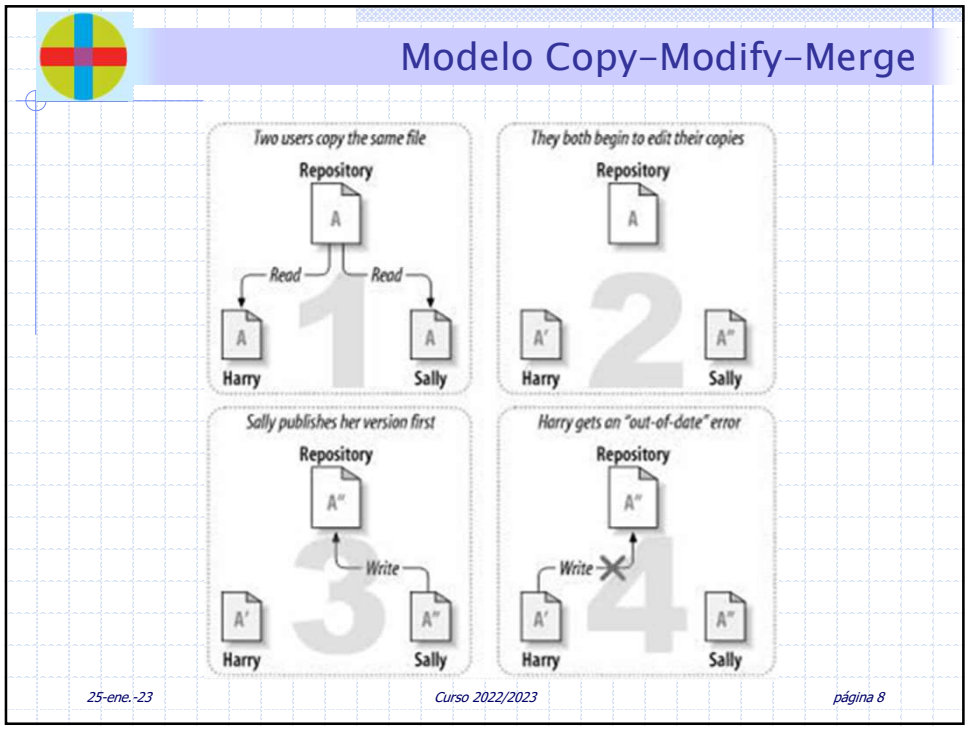
El problema de compartir ficheros



6

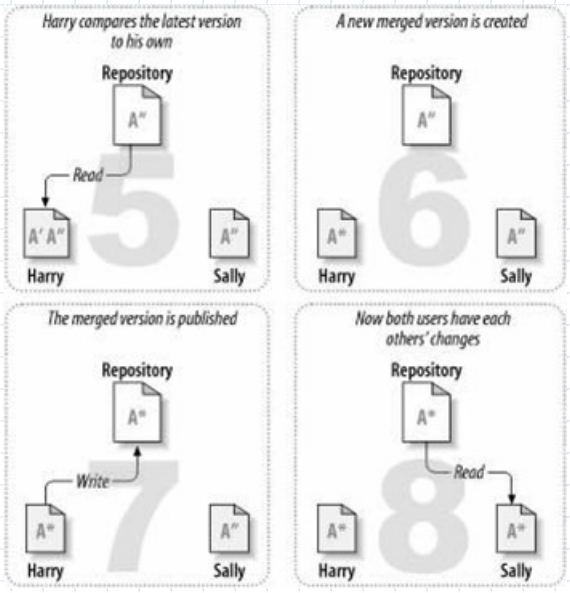

Modelo Lock-Modify-Unlock
Harry "locks" file A, then copies it for editing
Repository
A
LOCK
Read
A
Harry
Sally
1
While Harry edits, Sally's lock attempt fails
Repository
A
Lock
A'
Harry
Sally
2
Harry writes his version, then releases his lock
Repository
A'
Write
UNLOCK
A'
Harry
Sally
3
Now Sally can lock, read, and edit the latest version
Repository
A'
Read
LOCK
A'
A'
Harry
Sally
4
25-ene.-23
Curso 2022/2023
página 7

7


Modelo Copy-Modify-Merge
Two users copy the same file
Repository
A
Read
Read
A
A
Harry
Sally
1
They both begin to edit their copies
Repository
A
A'
A''
Harry
Sally
2
Sally publishes her version first
Repository
A''
Write
A'
A''
Harry
Sally
3
Harry gets an "out-of-date" error
Repository
A''
Write
A'
A''
Harry
Sally
4
25-ene.-23
Curso 2022/2023
página 8

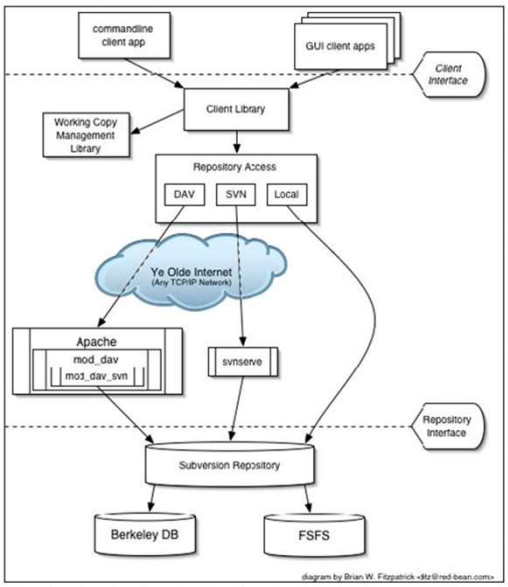
8

Modelo Copy-Modify-Merge



9

Esquema de VCS centralizado (subversion)



10

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

¿Preguntas?