- Introducción
- Conceptos básicos
- Operaciones
- Gestión de ramas
- Uso de repositorios
- Principios
- Resumen
- Bibliografía

Problema a resolver

Definir el "*usage model*" o modo de uso de la/s herramienta/s:

¿Cómo se gestionan las ramas? ¿qué permisos se dan a los usuarios? ¿cómo se hacen los *merge*? ¿con qué frecuencia? ¿por qué motivos se crean las ramas? ¿cuándo se eliminan?, etc, etc, etc…

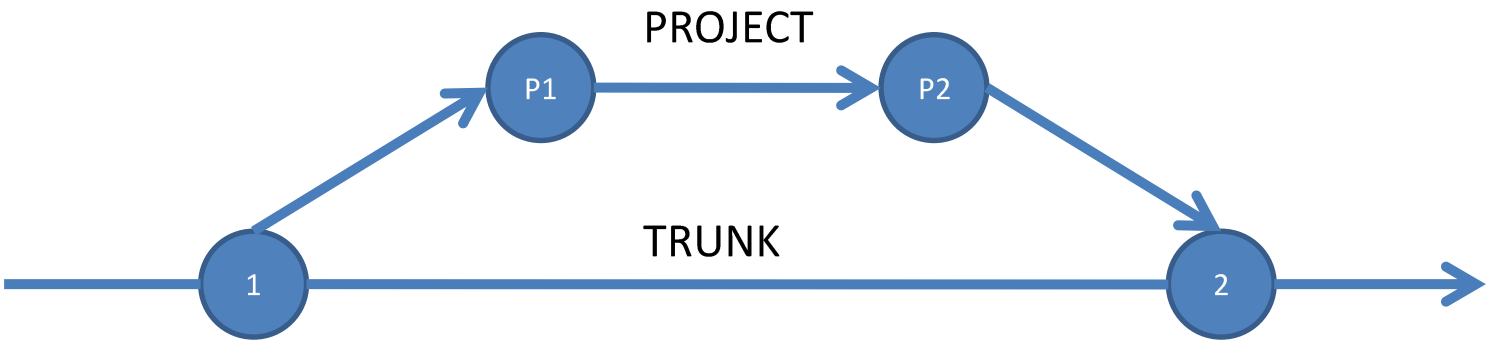
¿Por qué crear ramas?

Motivos para la creación de ramas [Humble & Farley]

- **Físicos**: Se crean ramas según la distribución "física" del proyecto, es decir, de su arquitectura.
- **Funcionales**: Según aspectos funcionales como pueden ser características (*features*), corrección de defectos (*bugfixing*)
- **Entorno**: por ejemplo, distintas versiones del sistema operativo / plataforma
- **Organizacionales**: Para organizar el trabajo en equipos, tareas, subproyectos, ….
- **Procedimentales**: Para pasar por distintas etapas de los proyectos.

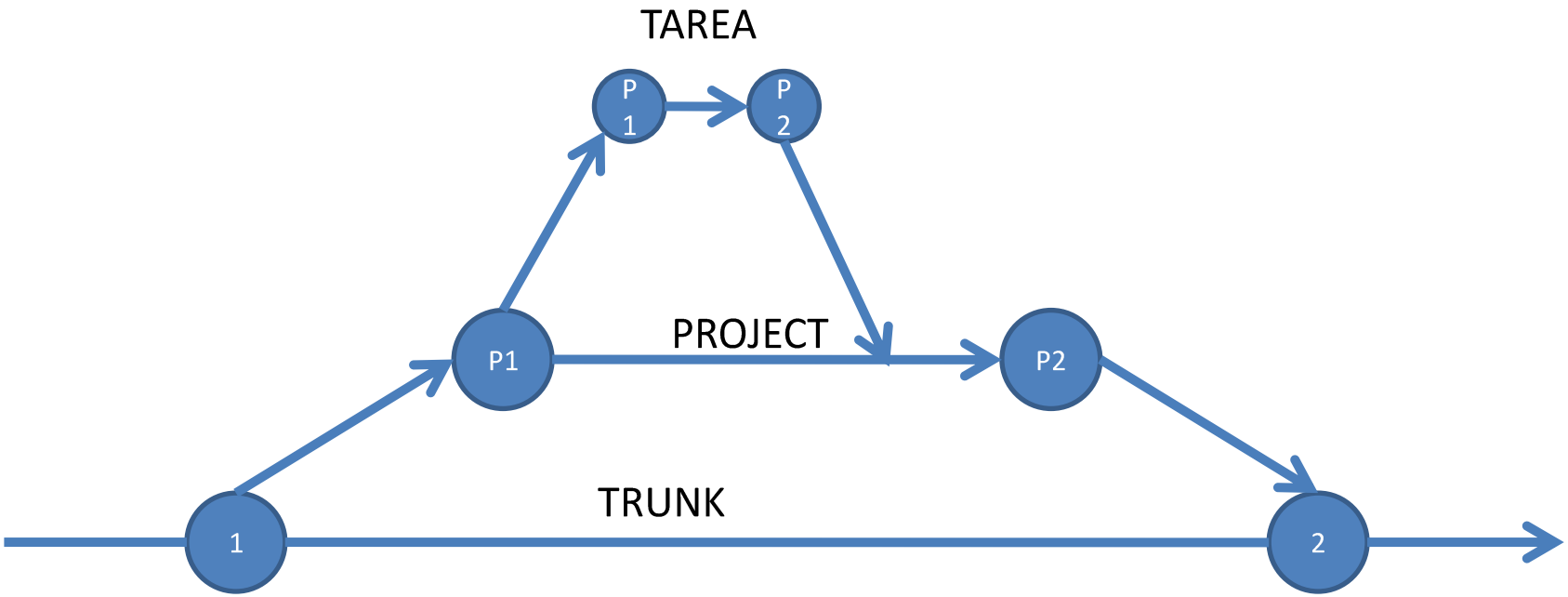
Gestión de variantes/branching

- Tipos de *branch:*
  - *Main branch: trunk, root, mainline, master*
    - *Es el padre de todos los demás branch*
  - *Project release branch:*



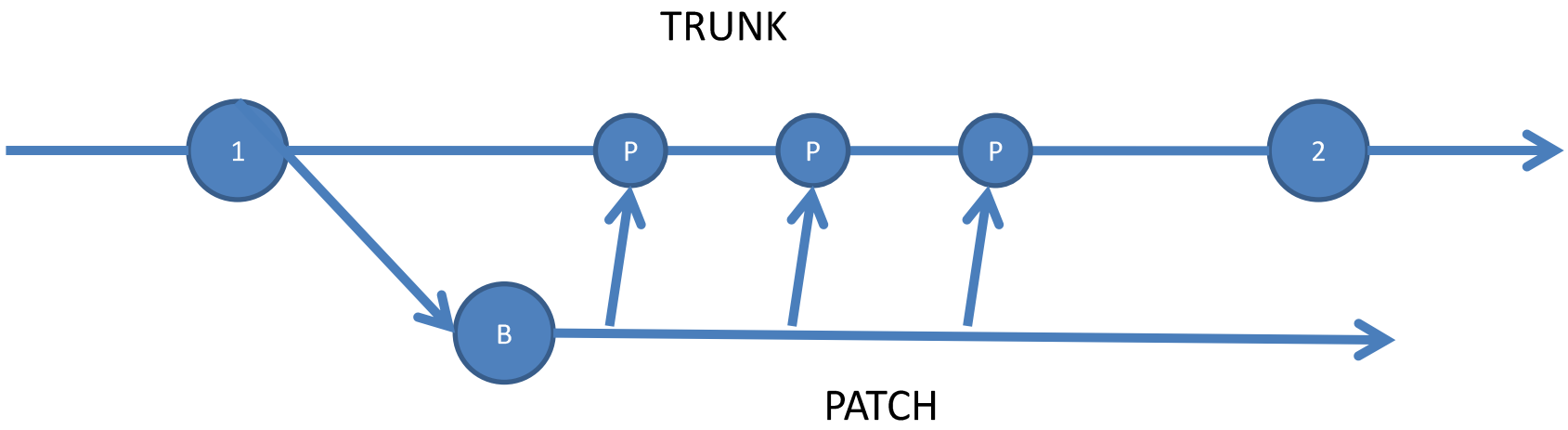
Gestión de variantes/branching

- Tipos de *branches:*
  - *De gestión de tarea:*



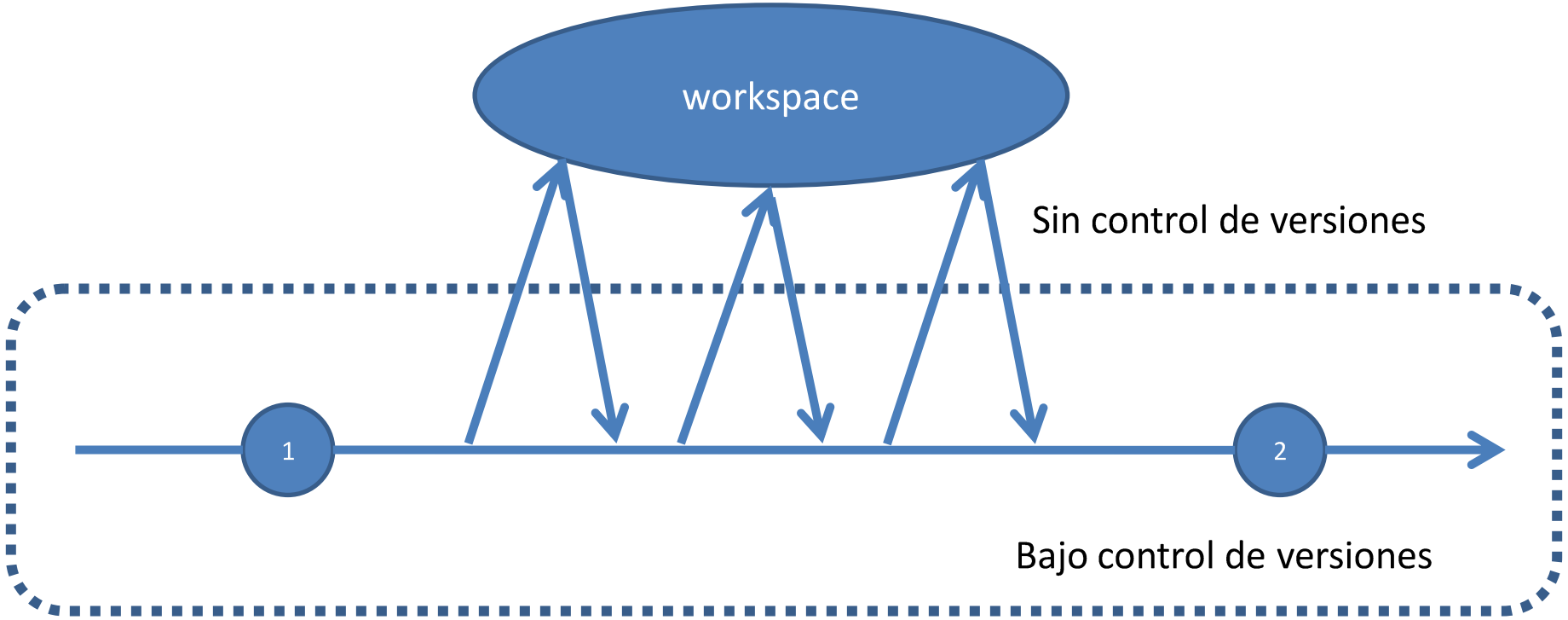
Gestión de variantes/branching

- Tipos de *branches:*
  - *De patch: los cambios producidos deben propagarse a todas las branch que haya*



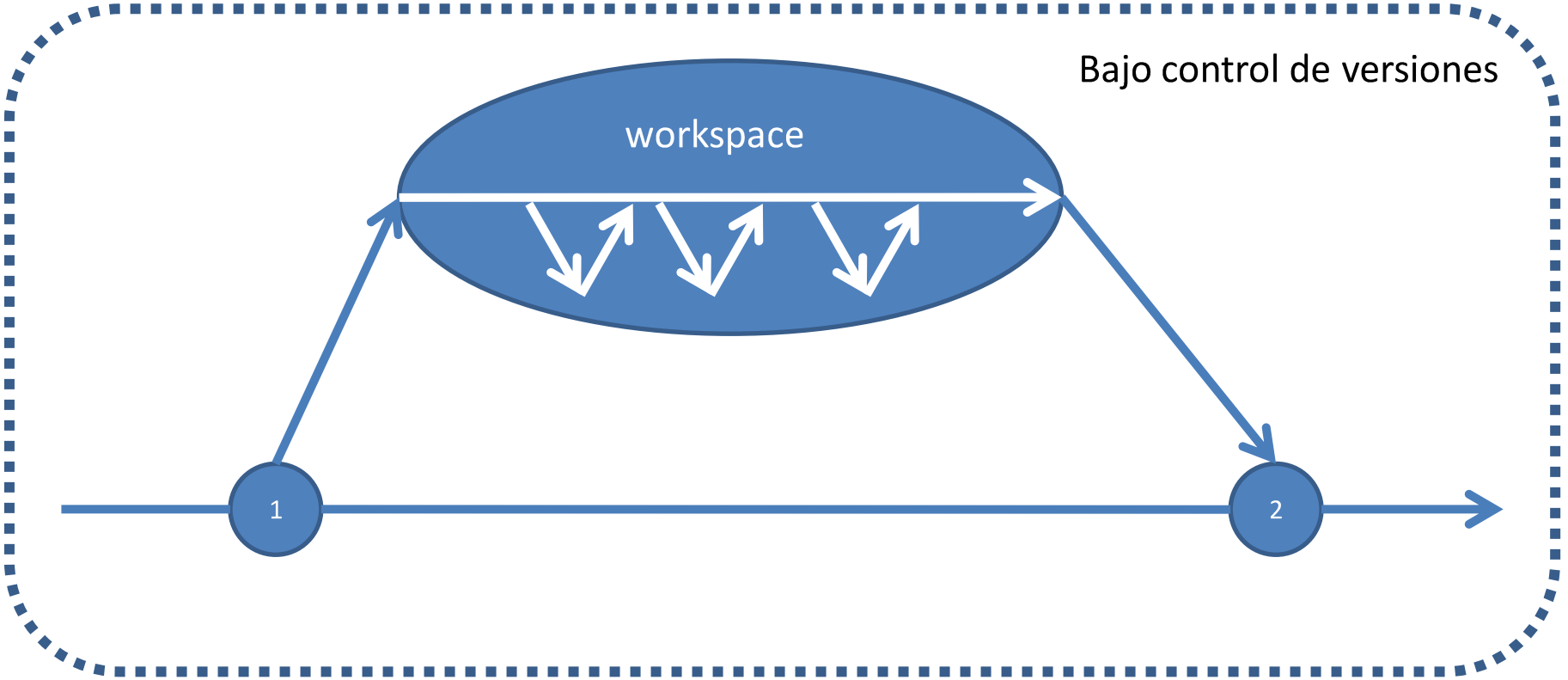
Branching y workspaces

- Workspace sin branch

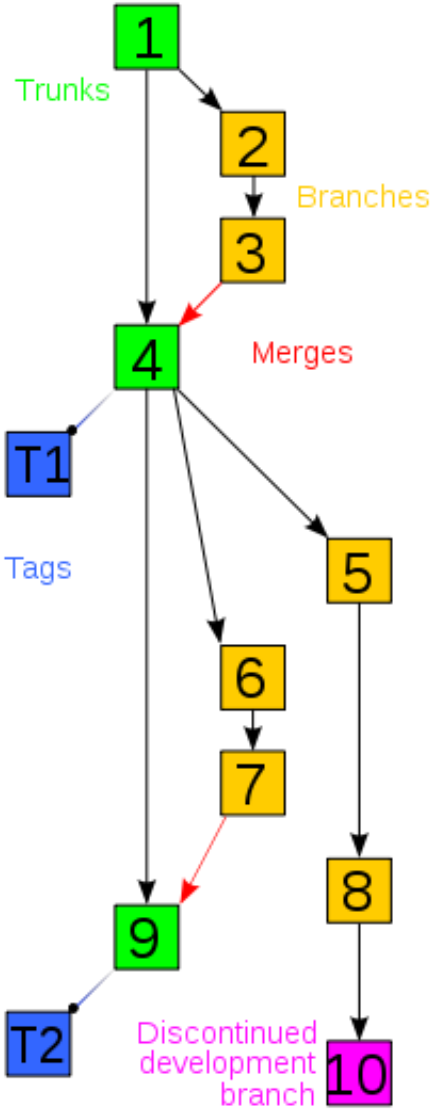


Branching y workspaces

- Workspace con branch



Resumen



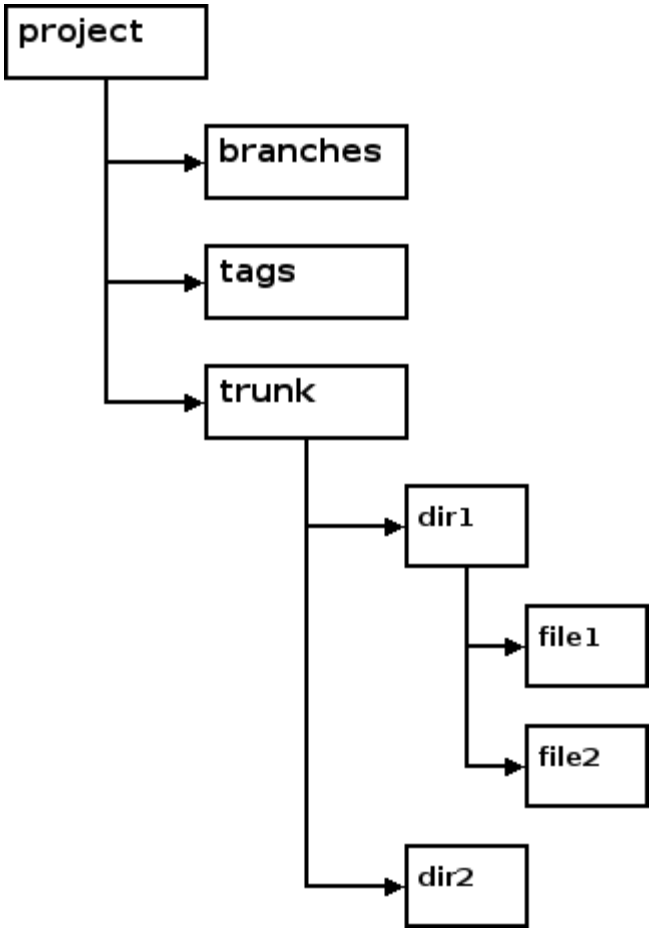
La clave: ¿cómo se organiza todo esto?

¡¡OBSERVANDO SE APRENDE!!

http://en.wikipedia org/wiki/File:Revision_controlled_project_visualization-2010-24-02.svg

Índice

Disposición habitual del repositorio



- Trunk = root, main branch, mainline.
- Baseline = tagging, labelling

El proceso habitual

Procedimiento de uso habitual de un sistema de control de versiones tipo SVN:

- Descarga de ficheros inicial (*Checkout)*
- Ciclo de trabajo habitual:
  - Modificación de los ficheros
  - Actualización de ficheros en local (*Update*)
  - Resolución de conflictos (si los hay)
  - Actualización de ficheros en repositorio (*Commit)*

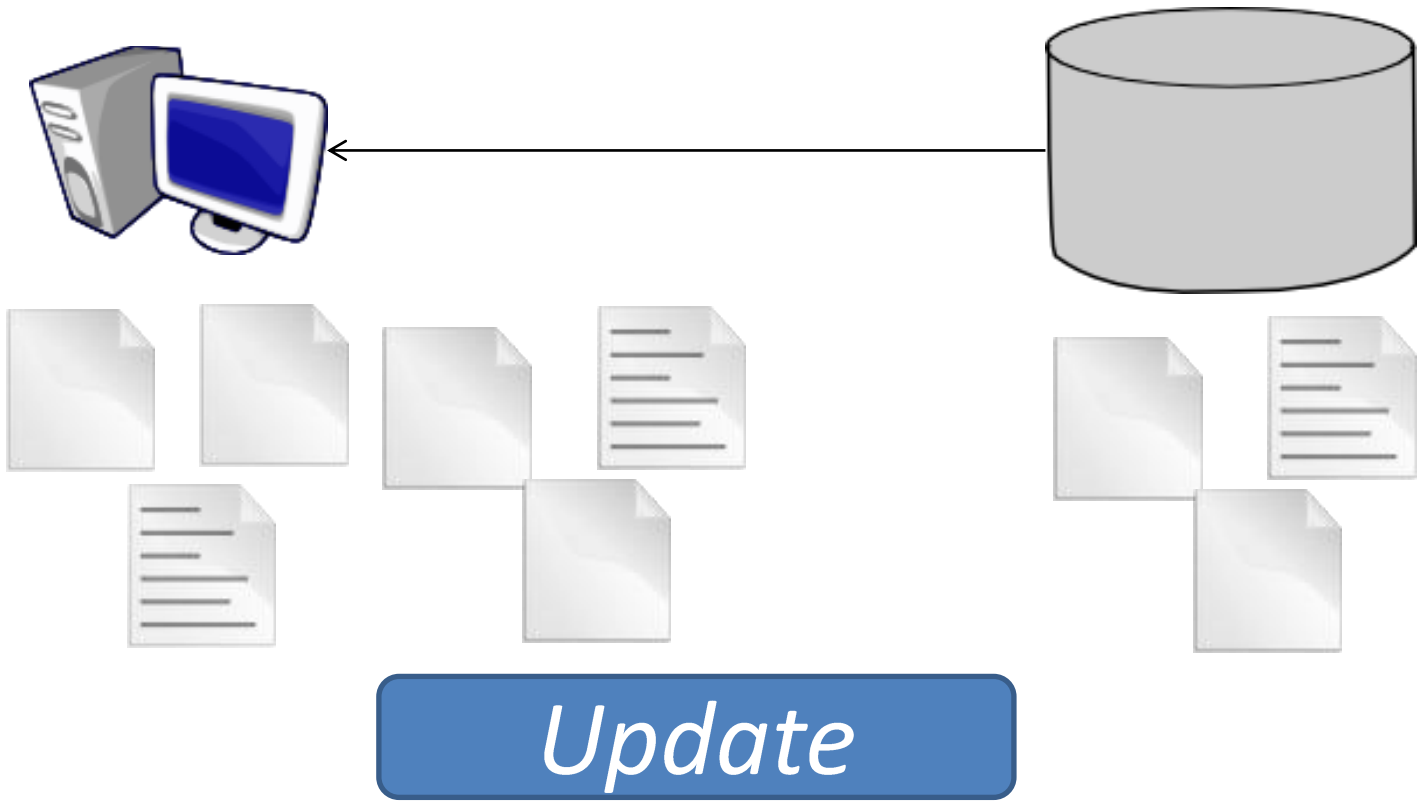
Descargando los ficheros

- El primer paso es bajarse los ficheros del repositorio
- El *checkout* sólo se hace la primera vez que se usan esos ficheros

checkout

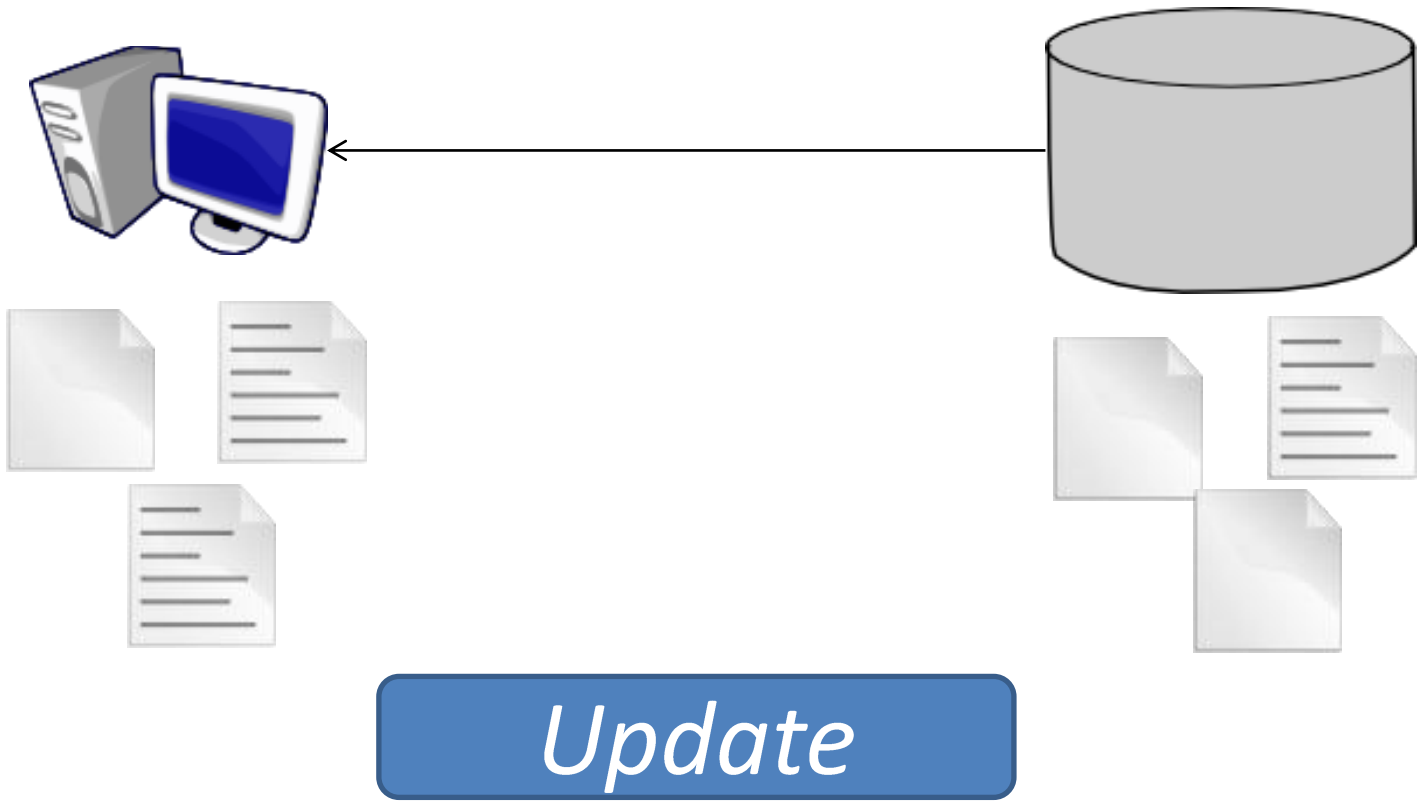
Actualizando los ficheros

- Modifica los ficheros en local
- Sincronizar los ficheros con los del repositorio



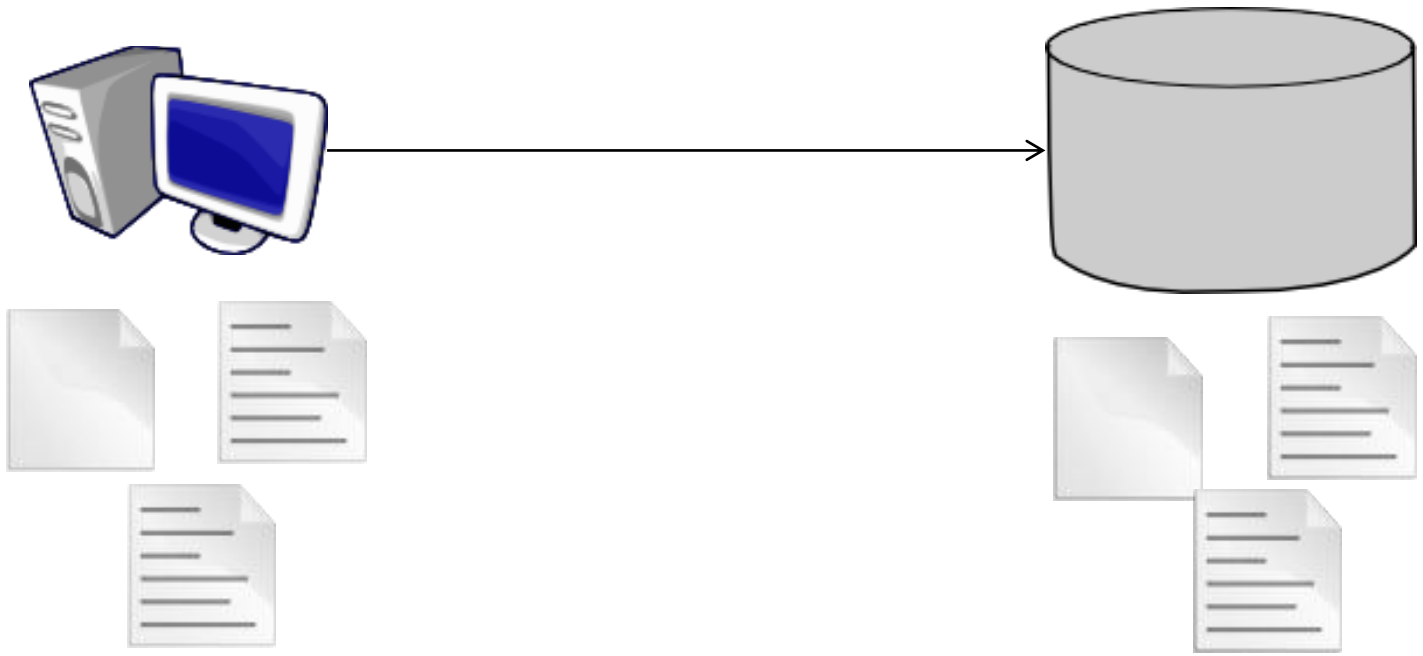
Actualizando los ficheros

- Modifica los ficheros en local
- Sincronizar los ficheros con los del repositorio



Actualizando el repositorio

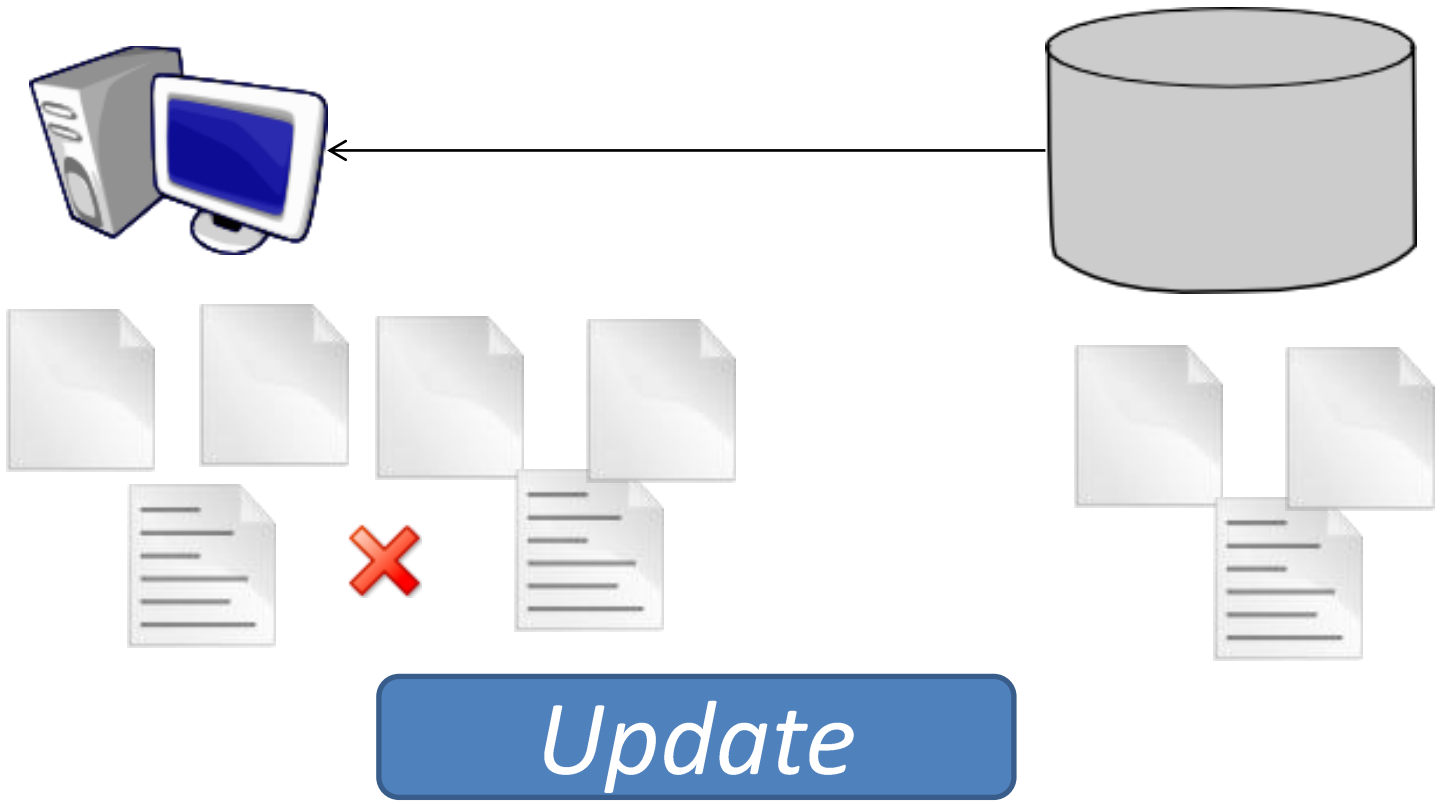
- Modifica los ficheros en el repositorio
- El sistema de control de versiones comprueba que las versiones que se suben estén actualizadas



Commit

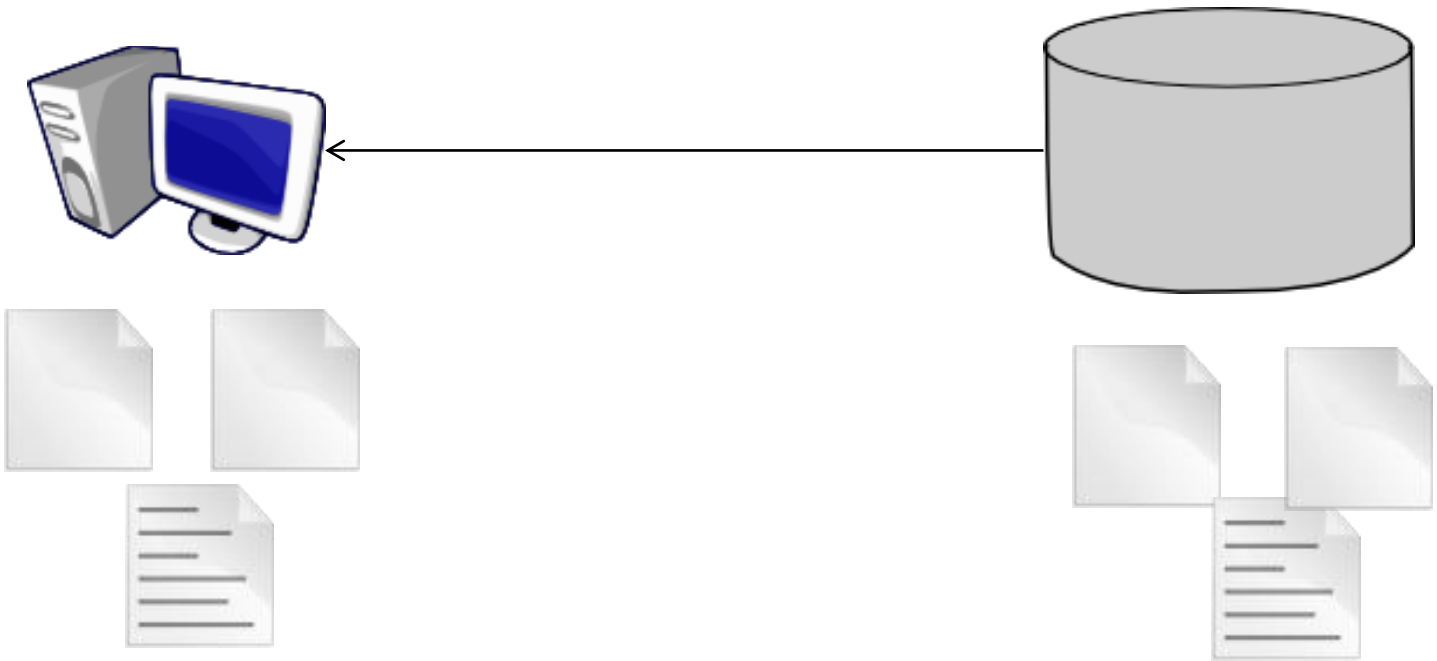
Actualizando los ficheros

- Modifica los ficheros en local
- Sincronizar los ficheros con los del repositorio



Actualizando los ficheros

- Modifica los ficheros en local
- Sincronizar los ficheros con los del repositorio



Update + resolución de conflictos

Sistemas Distribuidos

- ¿Qué es un sistema de control de versiones distribuido?
  - Es aquel en el que los usuarios mantienen un repositorio en local de modo que no existe un repositorio centralizado "*per se*"

¿Por qué un sistema distribuido?

Sistemas Distribuidos

- ¿Cuáles son las principales diferencias?
  - No hay un repositorio central (aunque por convención se suele usar uno)
  - Cada usuario tiene su repositorio en local
  - Los *commits* son locales
  - Aparecen dos nuevas operaciones: ***pushing*** (especie de *commit* pero en remoto), ***pulling*** (especie de update del repo remoto).
  - Concepto de ***rebasing***: permite cambiar "la máquina del tiempo" del repositorio local para empaquetar los cambios en un sólo *commit* con objeto de enviarlo al repositorio remoto

Índice

¿Cómo detecto que me falta *source code management*?

- No hay herramienta de gestión del código
  - "Every team should use one, no matter how small" [Humble]
- Las herramientas que se usan no son fiables
- Los usuarios no están entrenados por lo que ni las mejores herramientas los pueden ayudar
- No existe un proceso de *release* y *deployment*
- Gestión compleja de las ramas llevan a que los usuarios cometan errores
- No existe buena comunicación entre el equipo
- Demasiadas partes inestables en la estructura del código y complejidad

Source code management

Principios [Aiello]

- El código siempre está a salvo, *nunca* se pierde (efecto Y2K)
- El código sigue una línea temporal
- Gestionar las variantes del código debe ser fácil usando *branches*
- Si hay distintas *branches* estas deben poder fusionarse facilmente (*merge*)
- La gestión del código debe ser ágil y reproducible
- Debe permitir la trazabilidad y rastreo de los cambios
- Debe ayudar a mejorar la productividad y la calidad del código

Índice

Resumen

- ¿Qué hemos aprendido?
  - El cambio es inevitable
  - Si no se gestiona bien puede haber muchos problemas
  - Los conceptos básicos de gestión del código: branches, repo, tag, sandbox,…
  - Distintas estrategias de gestión de las ramas
  - Escenarios típicos de uso de un SCV
- ¿Qué veremos en las siguientes lecciones?
  - En práctica de laboratorio trabajaremos con gestores de versiones para poner en práctica distintos escenarios y también para gestionar las peticiones de cambio
  - Gestión de entregas, liberaciones, cambios…

Índice

Bibliografía