escuela técnica superio
de ingeniería informátic

Gestión del código fuente
Source code management

Departamento de
Lenguajes y Sistemas Informáticos

Evolución y Gestión de la Configuración

Ejemplo de otro dominio

ME QUIERO MUDAR

Ejemplo de otro dominio

FABRICAR COCHES

¿Qué hay en estos escenarios que no siempre hay en los proyectos software?

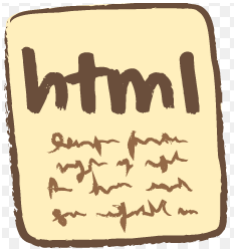
Índice

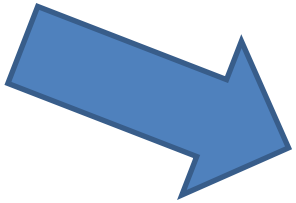
- Introducción
- Conceptos básicos
- Operaciones
- Gestión de ramas
- Uso de repositorios
- Principios
- Resumen
- Bibliografía

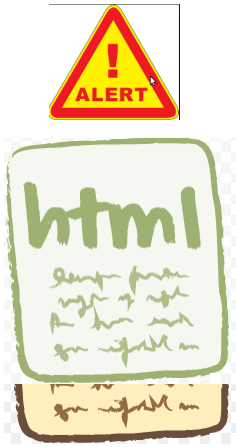
Escenarios

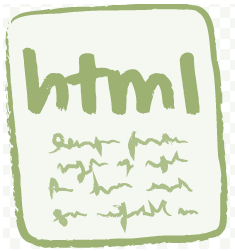
- ¿Podrías instalar el proyecto de IISSI/D&P en cualquier ordenador en un solo clic?
- ¿Sabrías la versión exacta del código que entregaste del proyecto en cada convocatoria?
- ¿En caso de que se detectara un *bug,* serías capaz de retomar exactamente la misma versión que entregaste, arreglar el bug y automáticamente desplegar los cambios?
- ¿Cómo gestionabas el trabajo en equipo? ¿Cómo "unías" las distintas partes de la aplicación?
- ¿Cómo gestionabas los entregables?

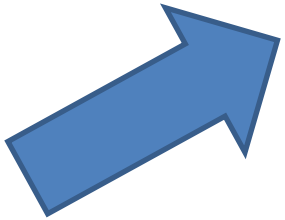
Escenario web









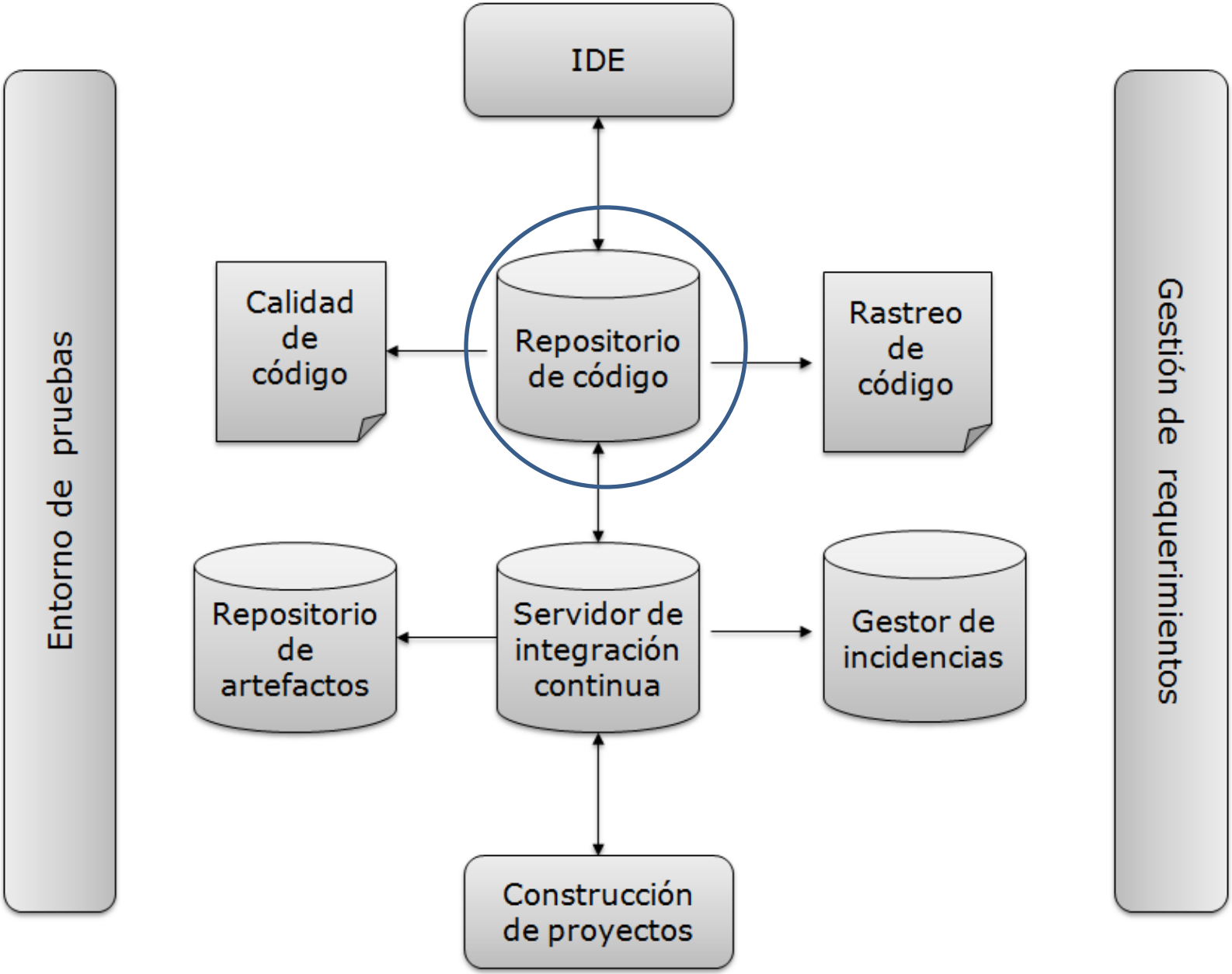


¿Cómo gestionar la colaboración?

El problema

¿Cómo gestionar la evolución y configuración del código fuente de nuestros proyectos?

Application Lifecycle Management



Índice

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

Conceptos básicos

Máquina del tiempo

Repositorio

Branch

Baseline

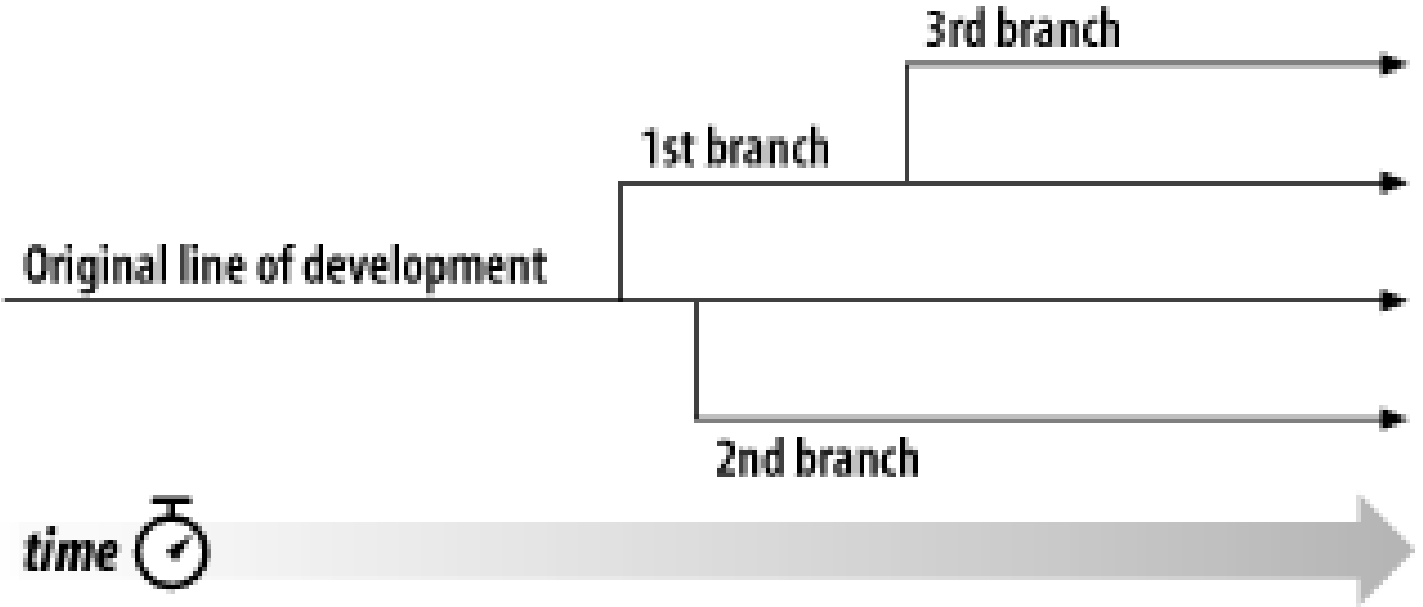
Sandbox

Repositorio de código

- Un repositorio de código es un sistema para guardar múltiples versiones de los ficheros de un proyecto de modo que cuándo se modifique un fichero se pueda acceder a las versiones anteriores
- Repositorio de código = Sistema de control de versiones = Control de código = sistema de control de revisiones = sistemas de gestión de código = “el repositorio” = “el svn”, “el git”, “el mercurial”…

Branch

- **Branch**: una línea de desarrollo independiente que puede compartir parte de historia común con otras Concepto de branch hija, branch padre

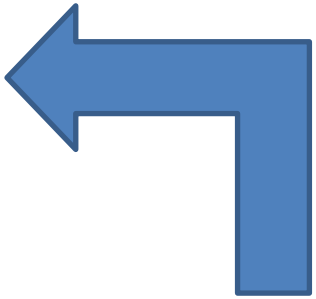


Baseline

- **Baseline:** Los cambios en el software requieren del paso por una serie de estados Identificar los estados importantes de los artefactos de software es lo que se llama crear un “baseline” del producto.
- También se suele conocer como “tagging” “labeling” “snapshoting”
- Las “baseline” deben ser **inmutables**.
- Estados frecuentes:
  - Desarrollo -> Pruebas -> Pre-producción -> Producción.

Sandbox

- **Sandbox/workspace/working copy**



checkout

Pero…¿Sólo guardamos el código fuente? ¿qué más?

Índice

Introducción

Conceptos básicos

Operaciones

Gestión de ramas

Uso de repositorios

Principios

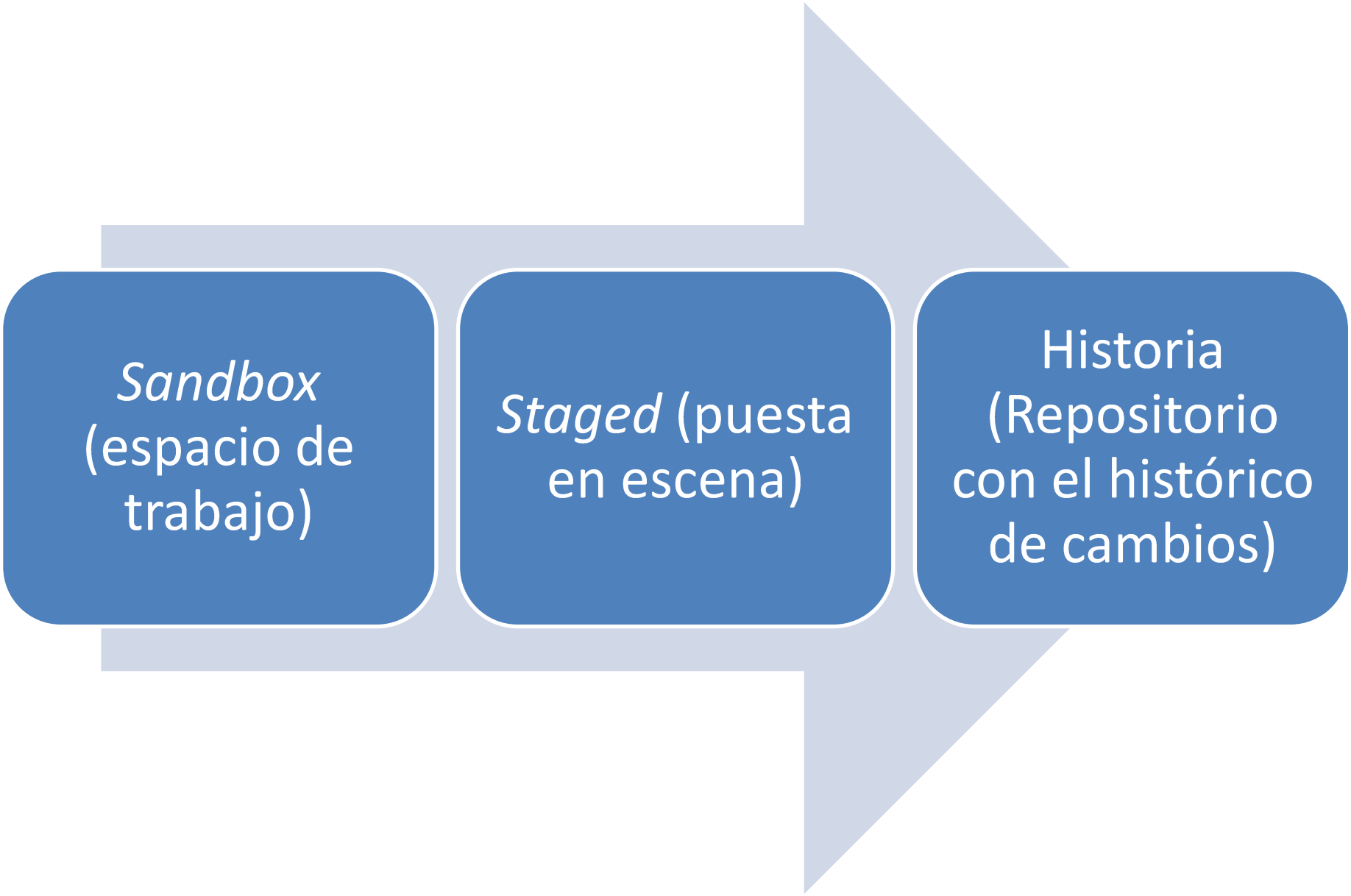
Resumen

Bibliografía

Operaciones

¿Cómo hacer las operaciones básicas en una “máquina del tiempo”?

Lugares de un *configuration item*



Operaciones frecuentes

- **Checkout**: tomar los artefactos del repositorio y llevarlo al área de trabajo (*sandbox*) para realizar modificaciones

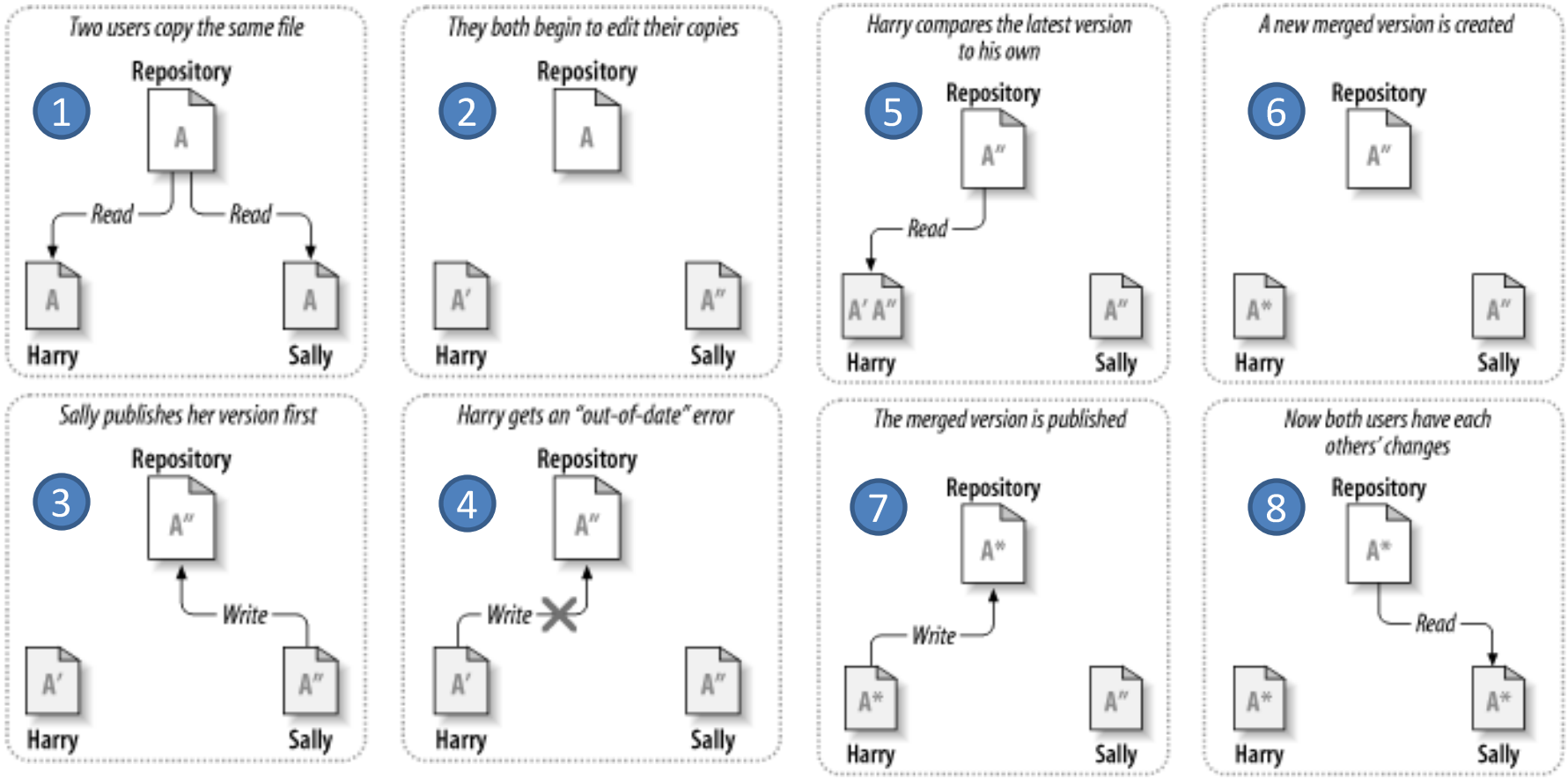
checkout

OJO: puede haber “juguetes” nuevos

Pregunta:
¿tengo que meter los ficheros de configuración de mi sandbox en el área de trabajo?

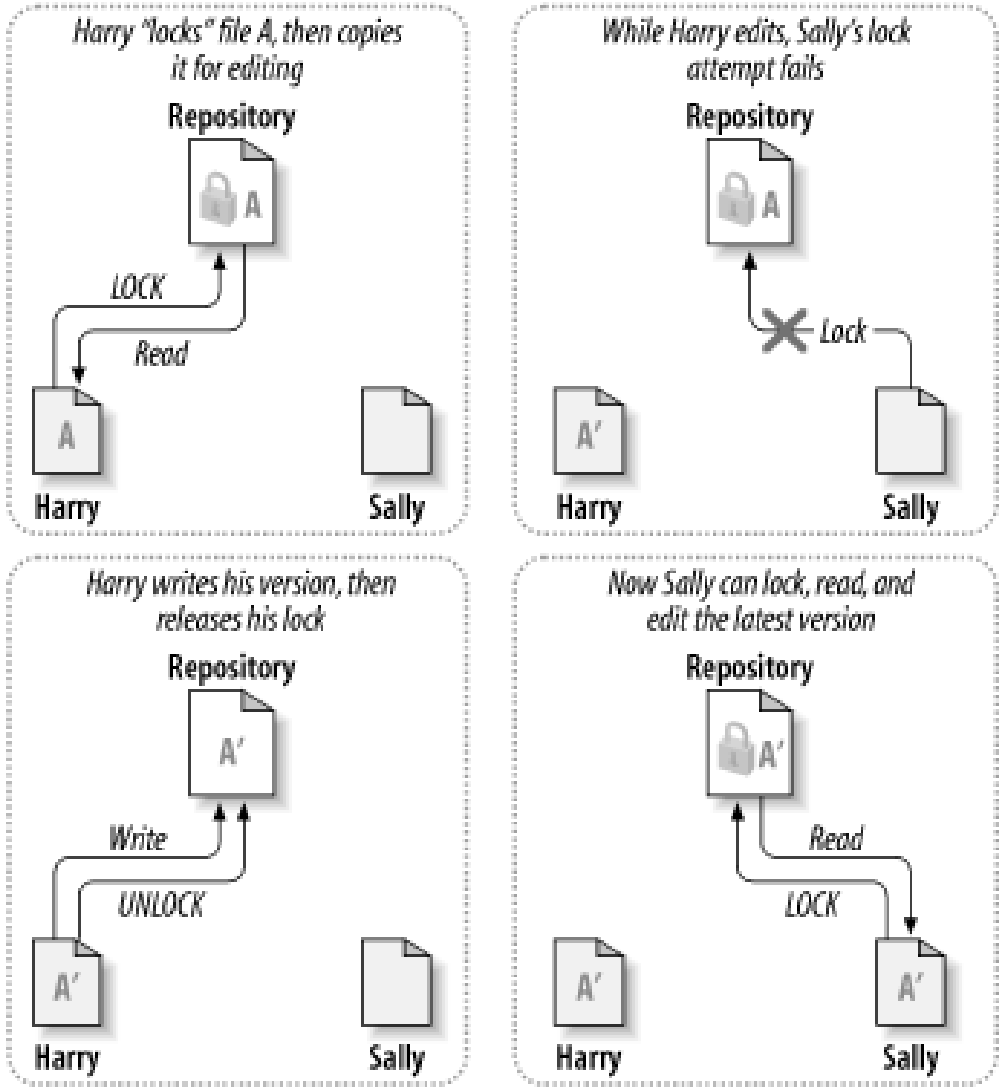
Operaciones frecuentes

- **Checkout no reservado**



Operaciones frecuentes

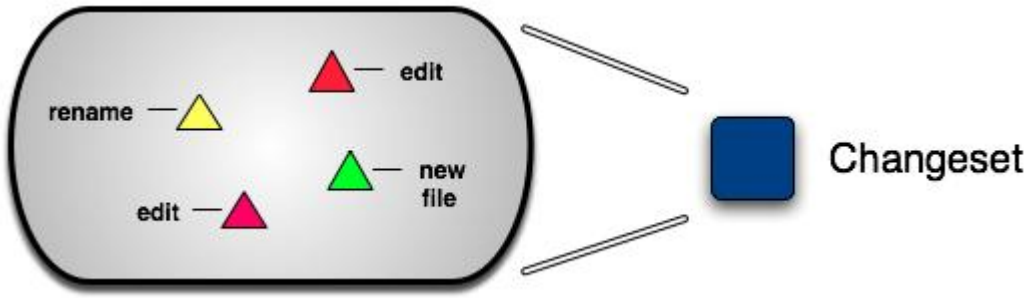
- **Checkout reservado (pesimista)** : cuándo se hace, no puede haber más de una persona al mismo tiempo modificando el código



¿Cuándo conviene hacerlo?

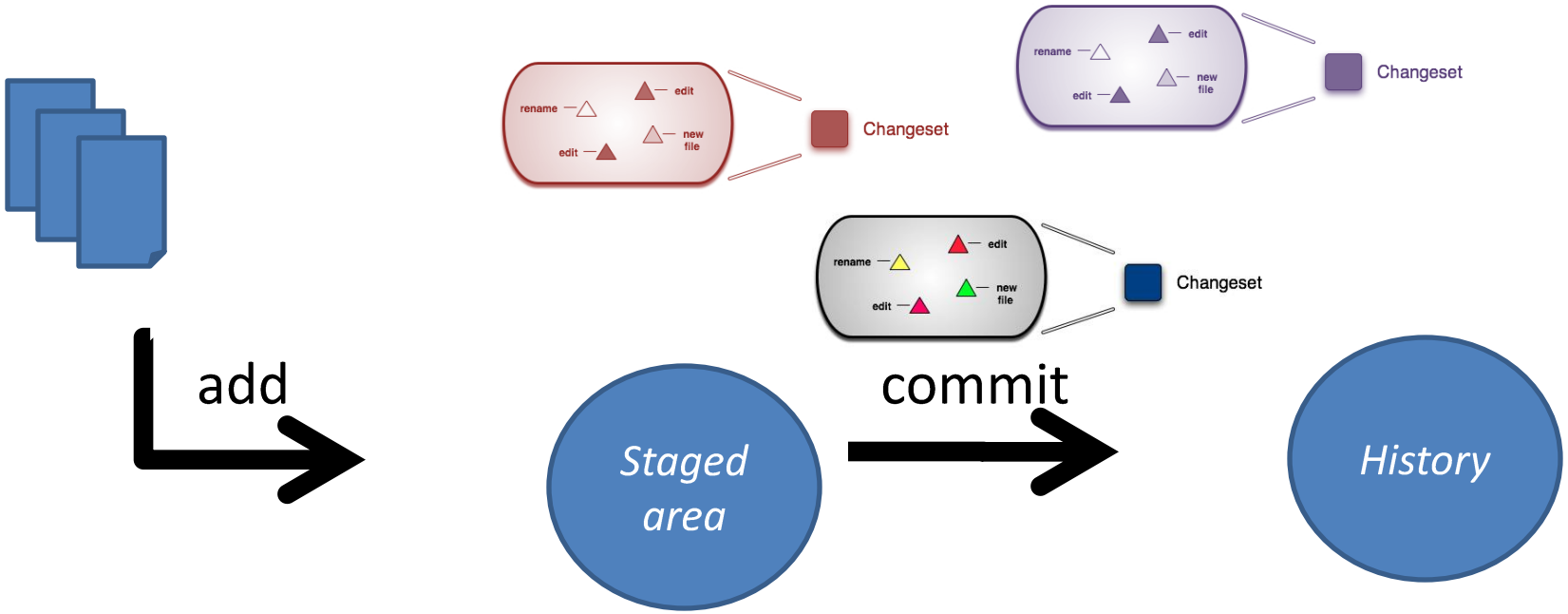
Operaciones frecuentes

- **Changeset**: conjunto de cambios que deben ser tratados como un conjunto indivisible (en svn se conoce como “revision”, en git como “commit”).



Operaciones frecuentes

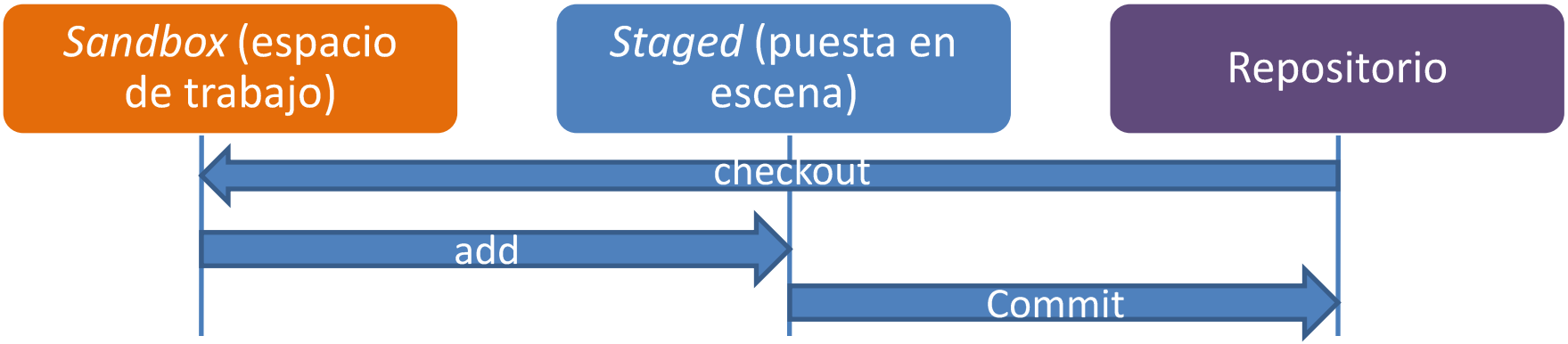
- **Add**:Añadir un elemento a la "máquina del tiempo" de modo que su estado y versiones sean controlados por al misma
- **Commit**: Guardar en la máquina del tiempo un conjunto de "changesets" Pueden ser atómicos o no.



Operaciones frecuentes

¿Qué hay en un commit?

- Datos modificados + metadatos
- Un fichero *rastreado* puede estar en tres estados (en el caso de GIT):
  - Confirmado (*committed*)
  - Modificado (*modified*)
  - Preparado (*staged*)

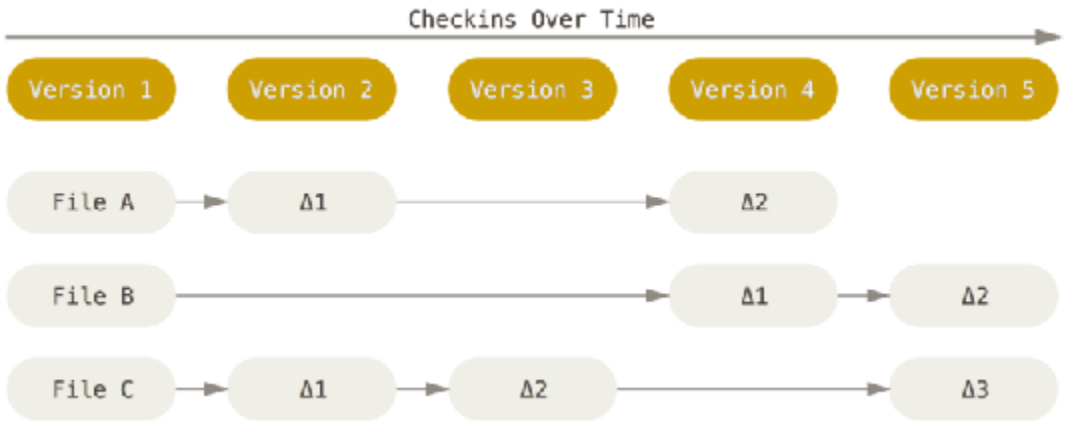


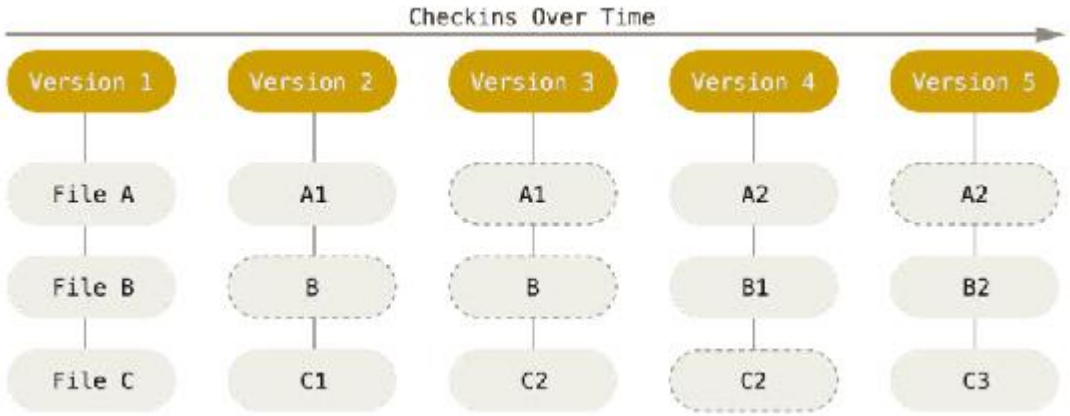
Operaciones básicas

- **Crear ramas/branches**

- **Copybranches vs deltas:** Algunos de los gestores de código requieren que una rama tenga un duplicado de todos los artefactos (copybranch). Otras herramientas en cambio, sólo usan una "subrama" que contiene sólo los artefactos que han cambiado A estas subramas se les suele llamar en este caso, "deltas".

Operaciones básicas



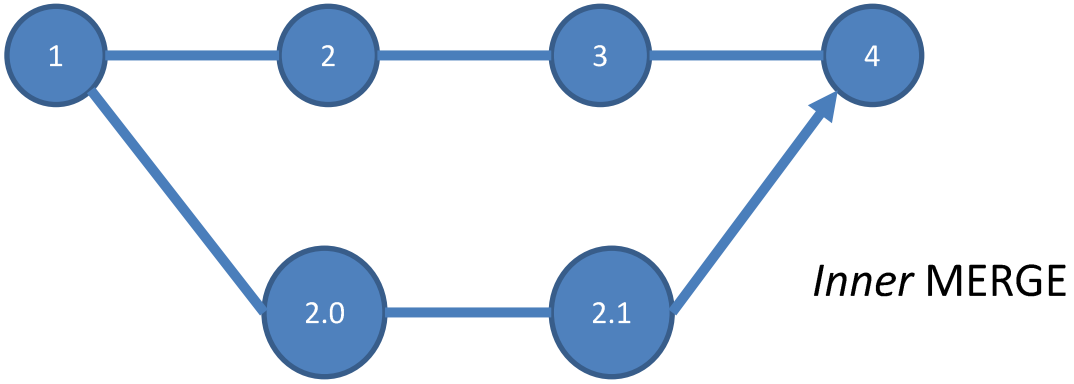


Cada objeto en git tiene un checksum asociado Hay tres tipos de objetos:

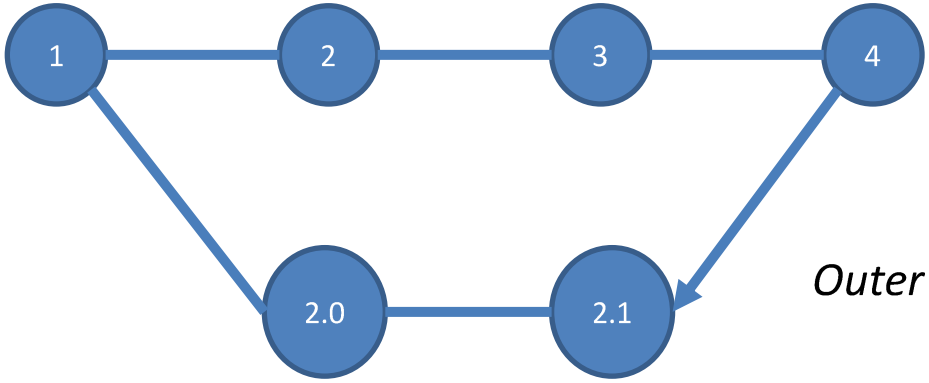
- *blobs -> checksum* de los ficheros que han cambiado
- *tree -> checksum* de la estructura de directorios de los *blobs*
- *commit -> checksum asociada* al conjunto de cambios de un *tree*

Operaciones frecuentes

- **Merging:** Es el proceso de mezclar dos o más ramas de artefactos



Inner MERGE



Outer MERGE

Operaciones frecuentes

- **Resolucion de conflictos**. Cuando se hace *merge* pueden aparecer conflictos Algunos pueden resolverse de manera automática, otros requieren intervención manual ***Resolver conflictos es duro***.
- ***Diff***. Hacer *diff* significa mirar la diferencia que hay entre artefactos de un repositorio A partir de un *diff* se puede generar un ***patch***, es decir, un conjunto de cambios a realizar a un conjunto de artefactos (ver http://es.wikipedia.org/wiki/Diff probar con http://www.quickdiff.com). Cuándo un conjunto de cambios se aplican a una serie de artefactos, se dice que se ha aplicado un *patch*.

Conflictos *sintácticos* vs conflictos *semánticos*

- 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…

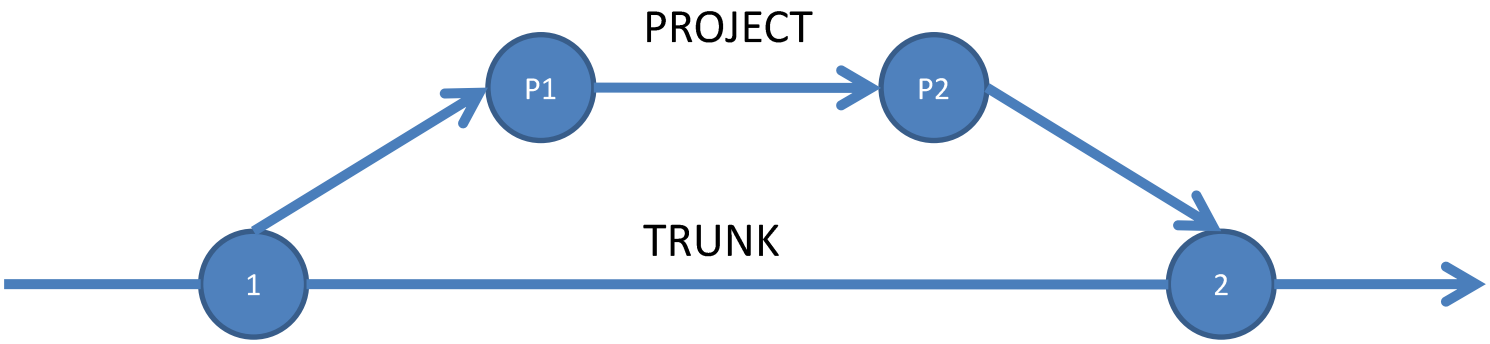
¿Por qué crear ramas?

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

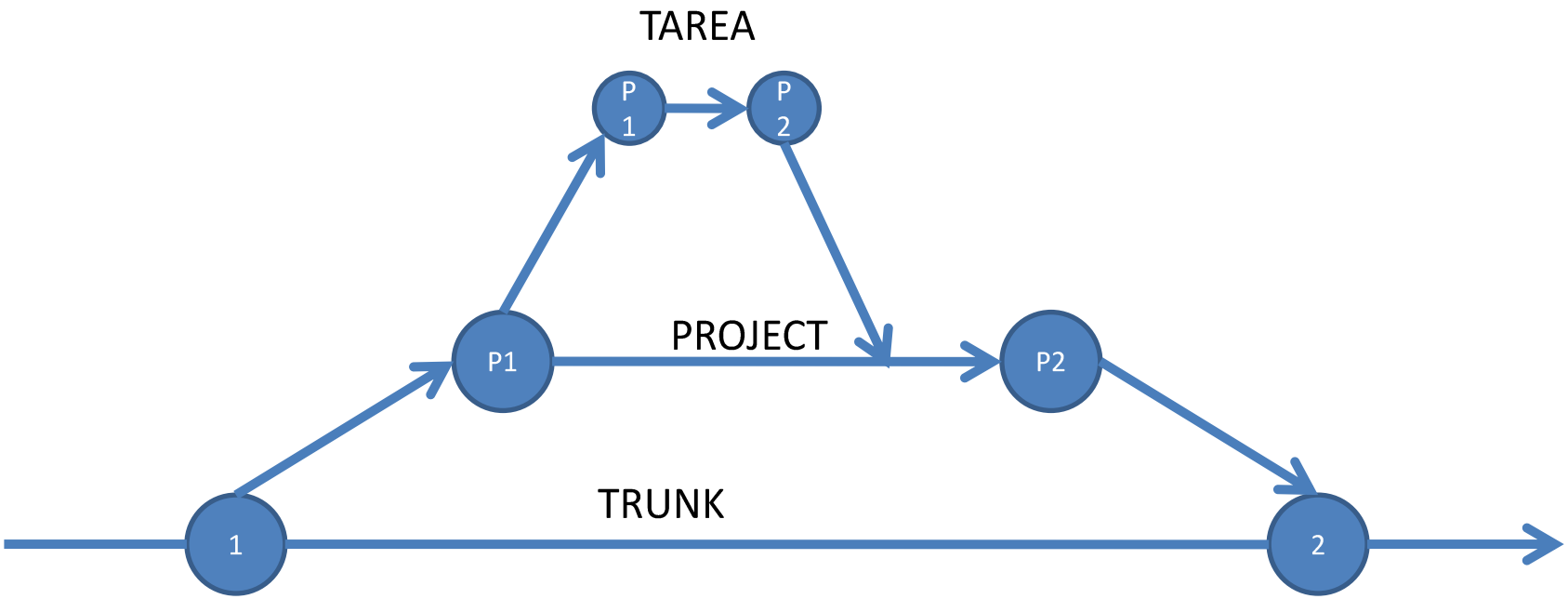
Gestión de variantes/branching

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



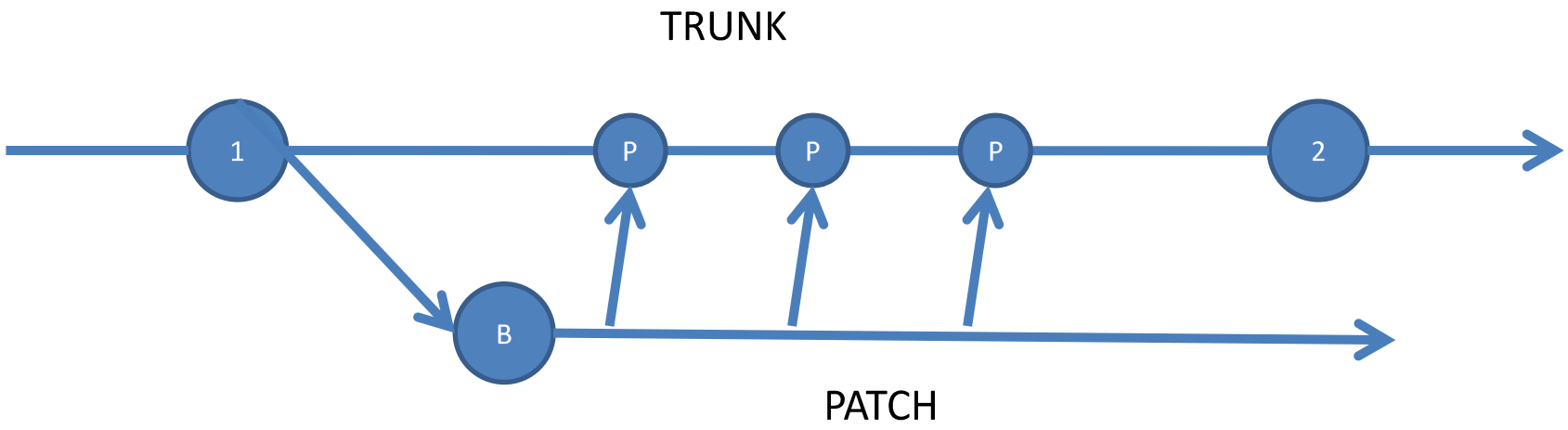
Gestión de variantes/branching

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



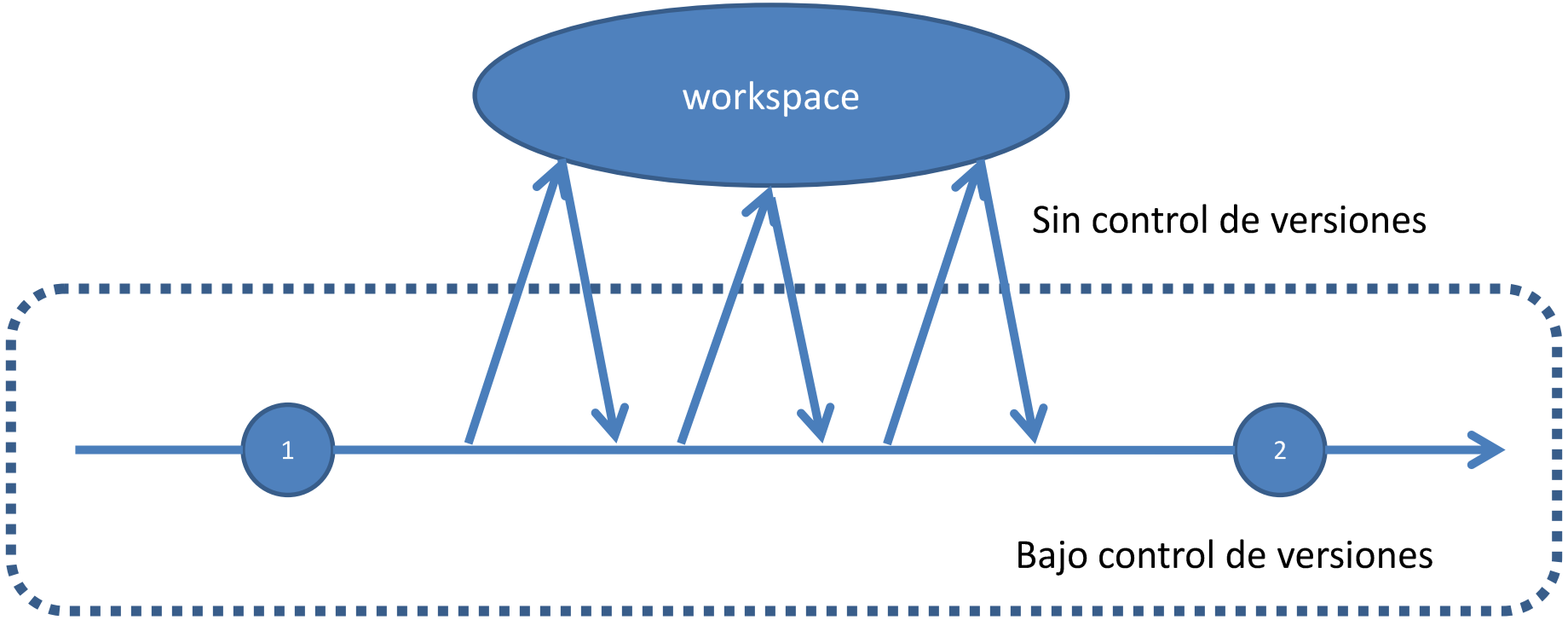
Gestión de variantes/branching

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



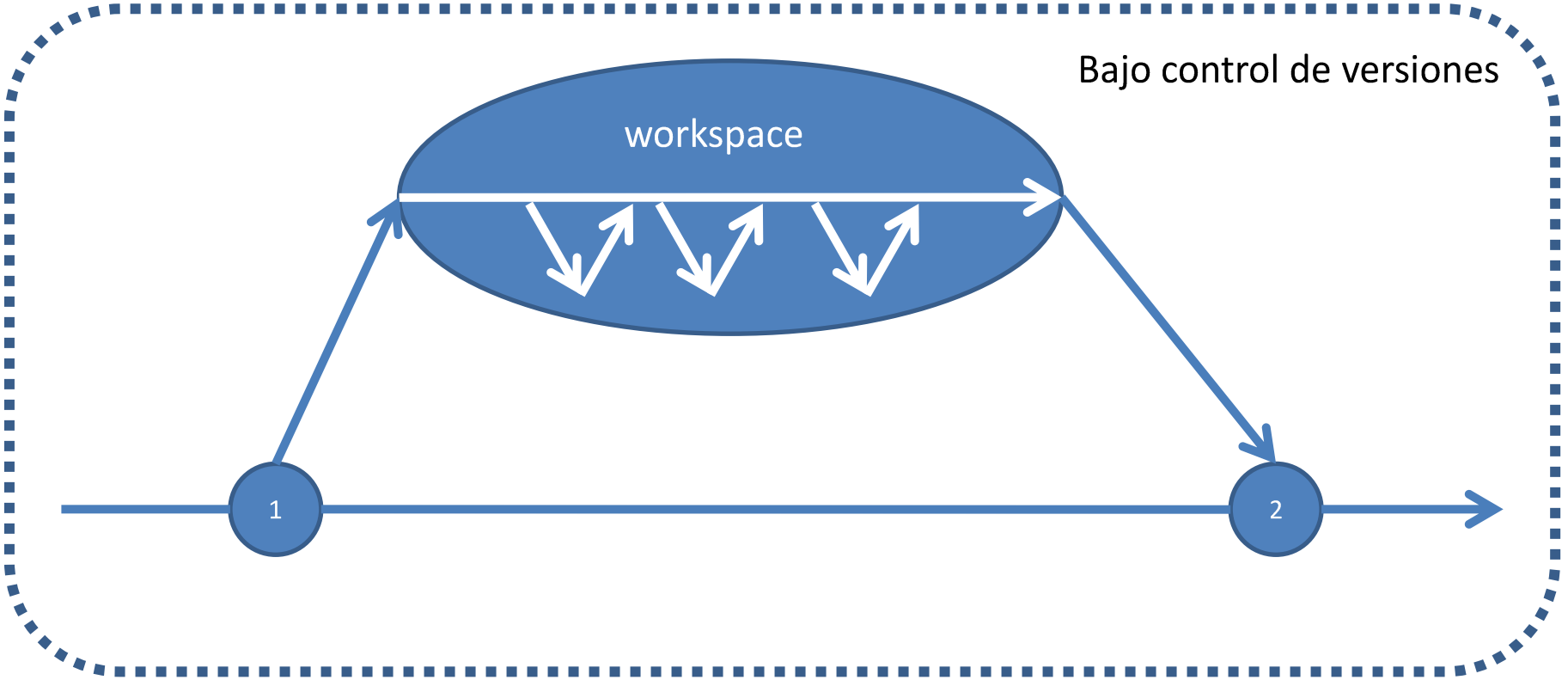
Branching y workspaces

- Workspace sin branch

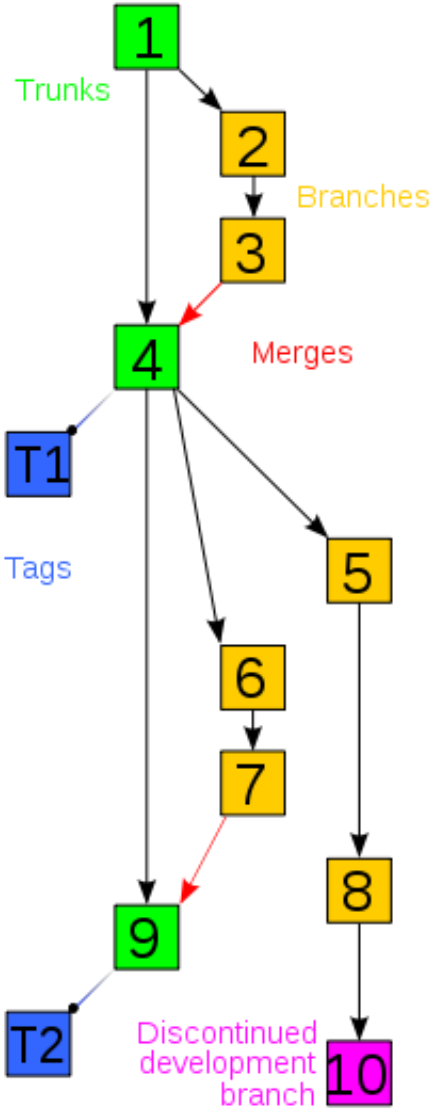


Branching y workspaces

- 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

Introducción

Conceptos básicos

Operaciones

Gestión de ramas

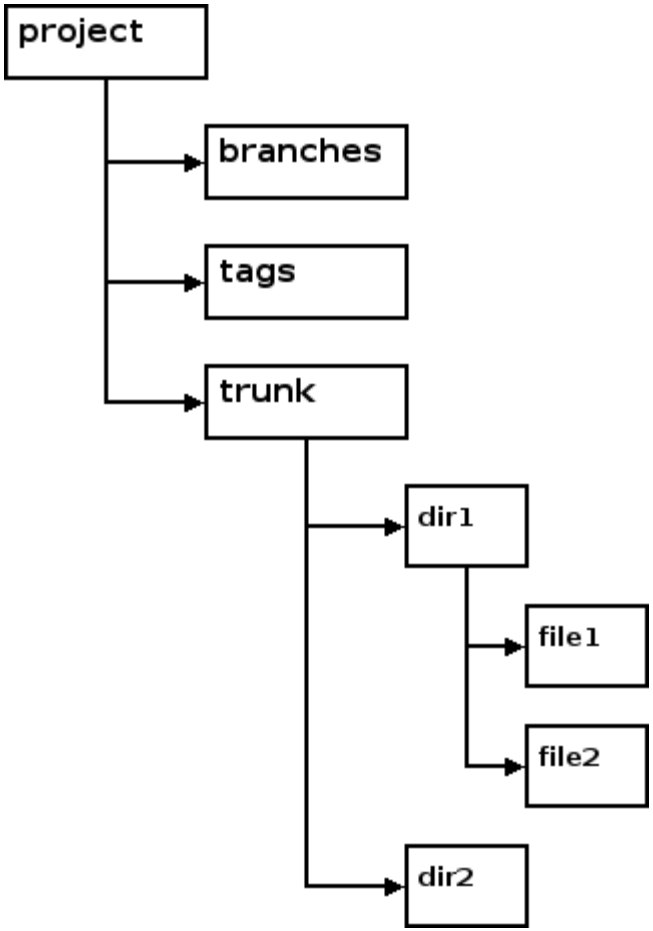
Uso de repositorios

Principios

Resumen

Bibliografía

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)*

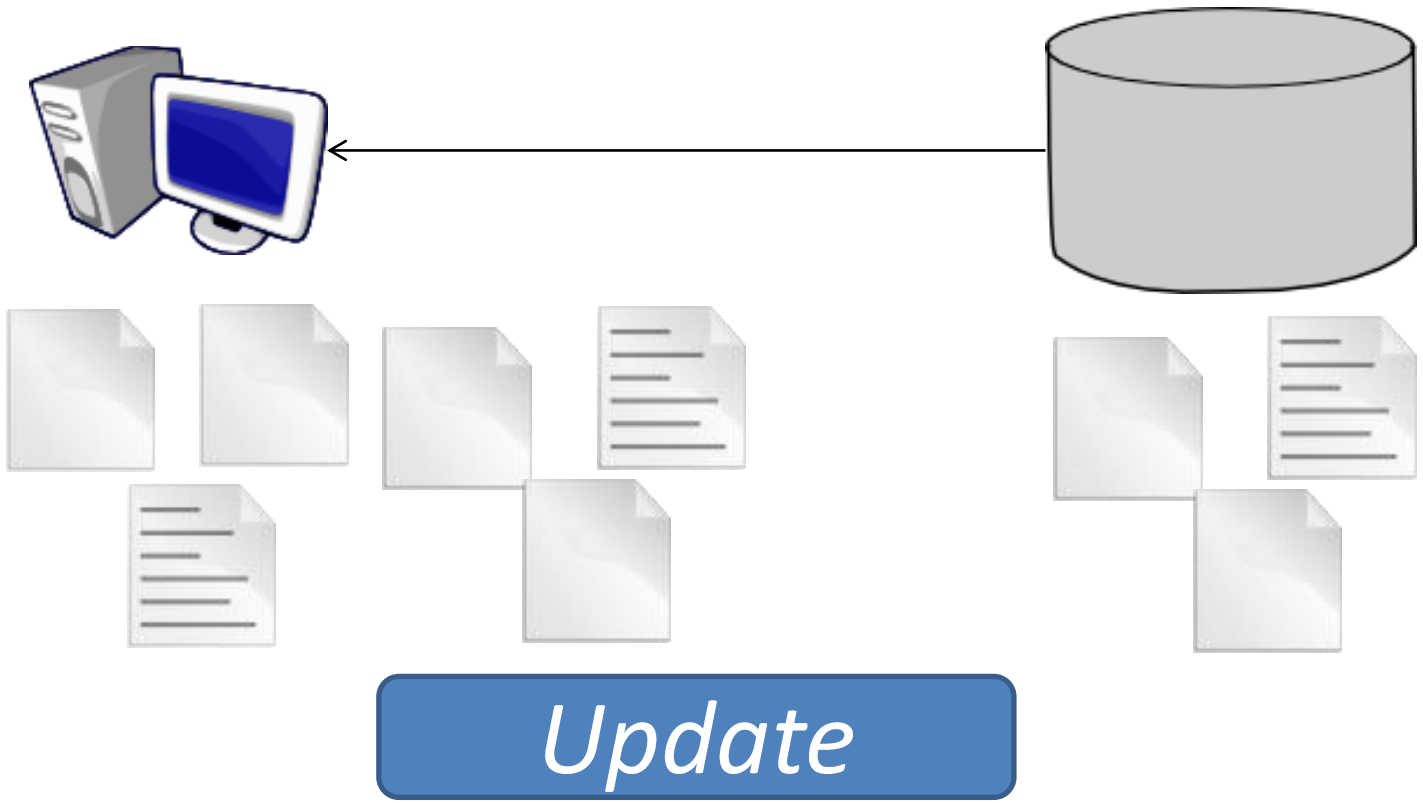
Descargando los ficheros

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

checkout

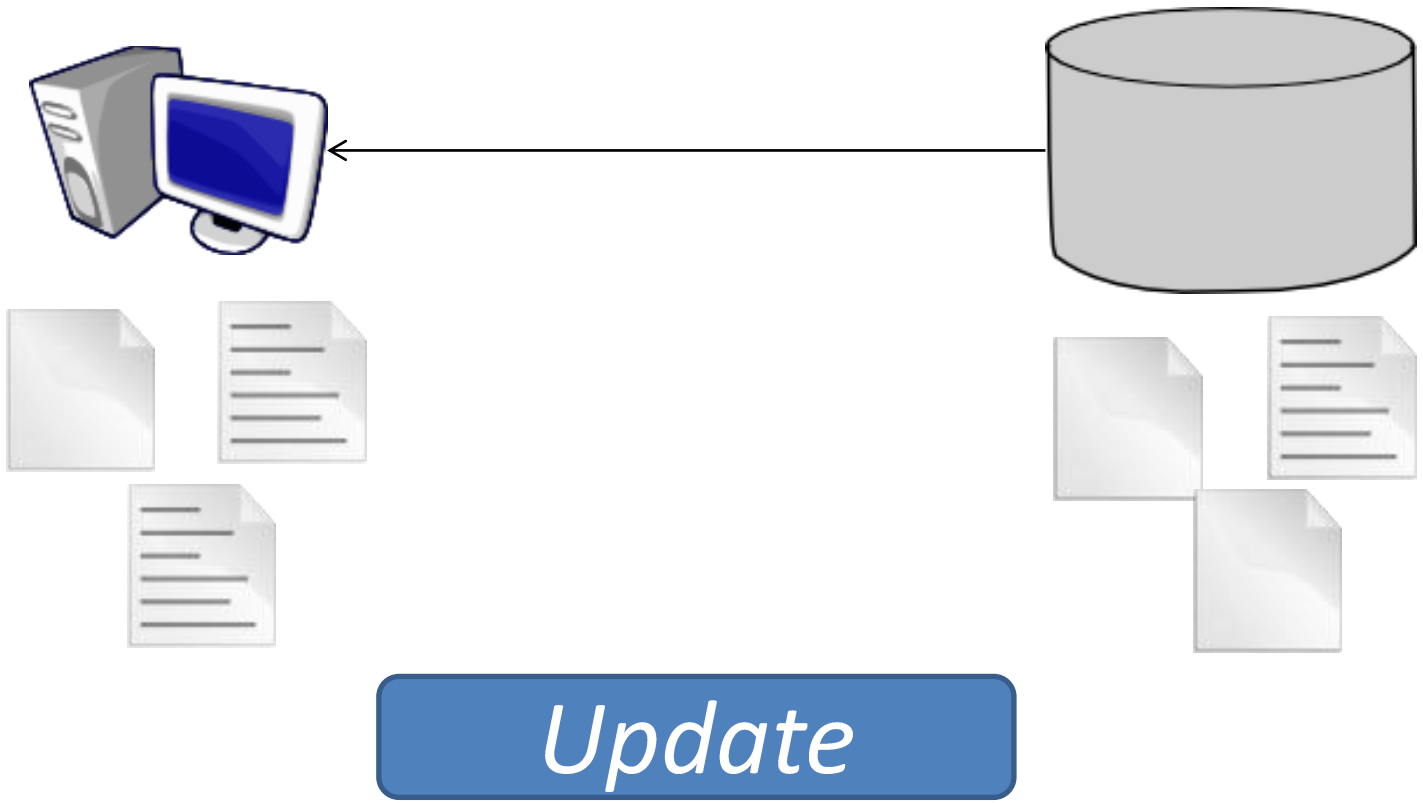
Actualizando los ficheros

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



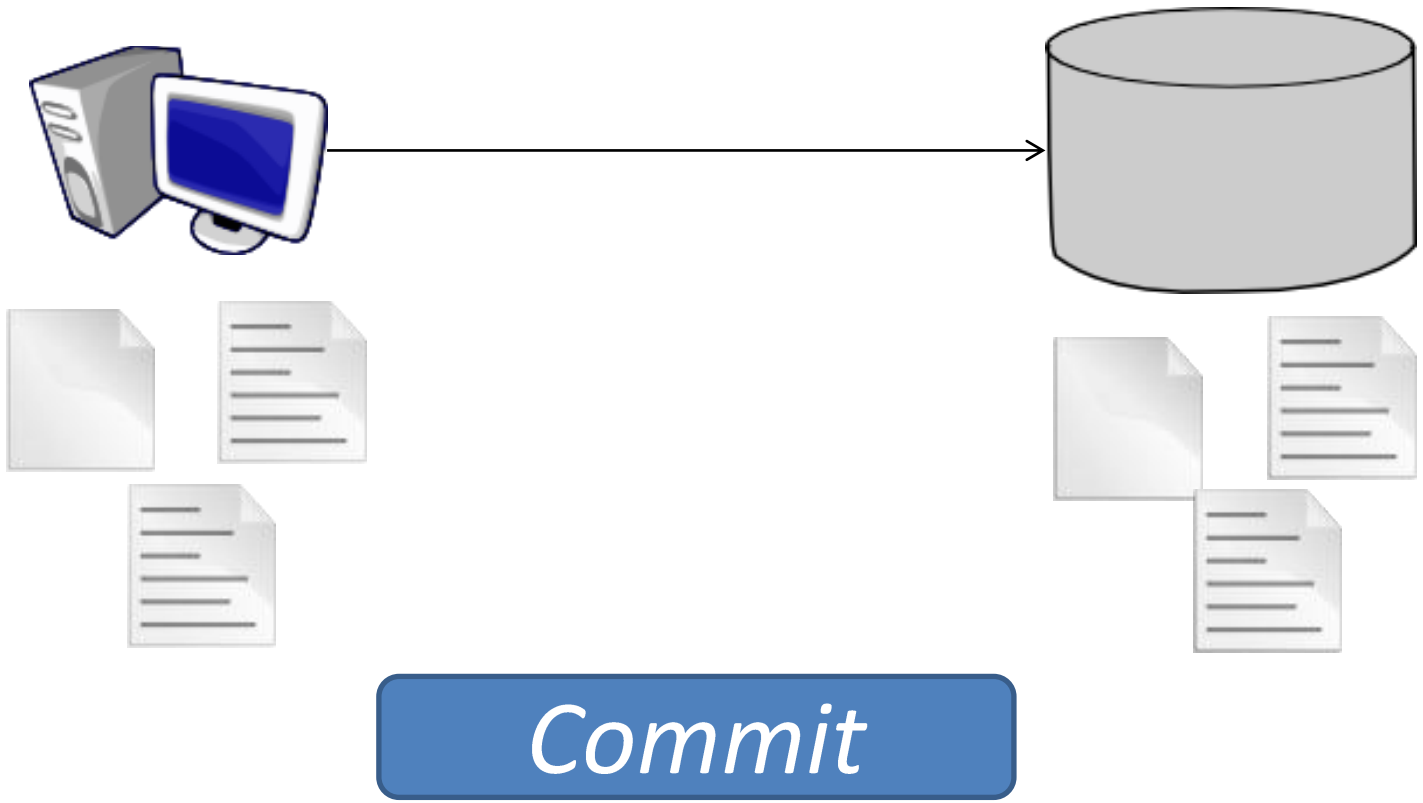
Actualizando los ficheros

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



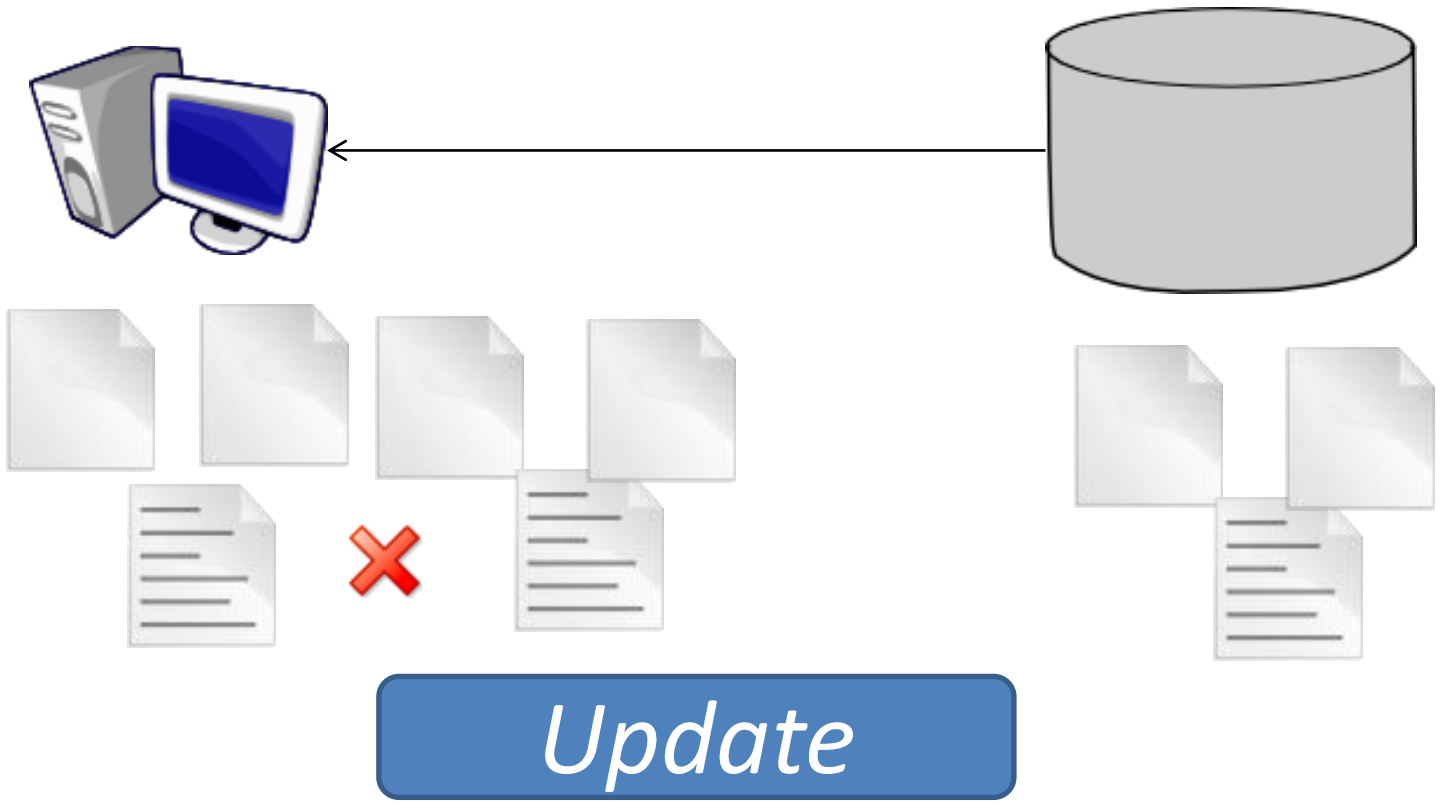
Actualizando el repositorio

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



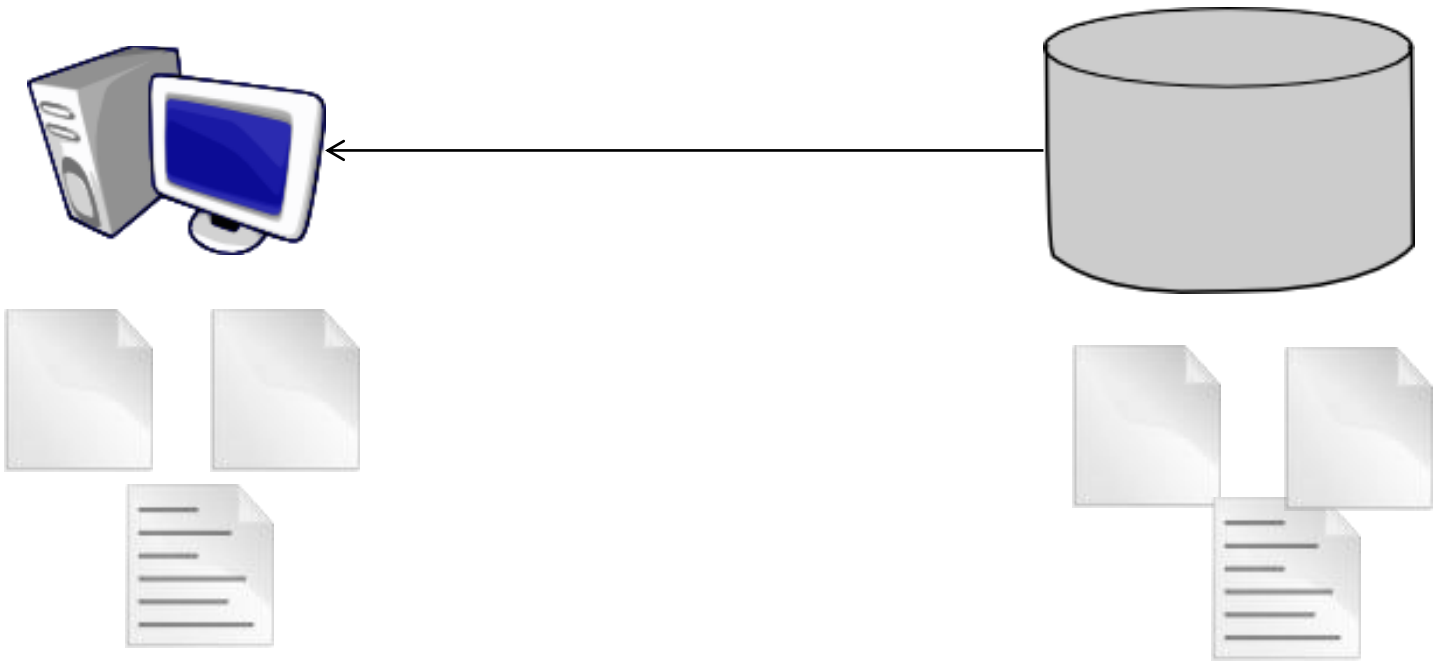
Actualizando los ficheros

- 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

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

¿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

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

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

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

Bibliografía