# Tema 10: Gestión de la configuración del software

GESTIÓN Y MANEJO DE PROYECTOS INFORMÁTICOS

MÁSTER EN INGENIERÍA INFORMÁTICA


# Índice

- Gestión de la configuración (GC).
- Cómo hacer una buena GC.
- Ejemplos de software de GC.
- Control de cambios.
- Control de versiones.

# Gestión de la configuración

DECSAI

- Conjunto de procesos que permiten asegurar la calidad de todo producto obtenido durante cualquiera de las etapas de desarrollo.
- Gestiona:
  - Control de cambios.
  - Control de versiones.
- Proporciona una imagen detallada del sofware a implementar en cada etapa del proceso.

# Terminología

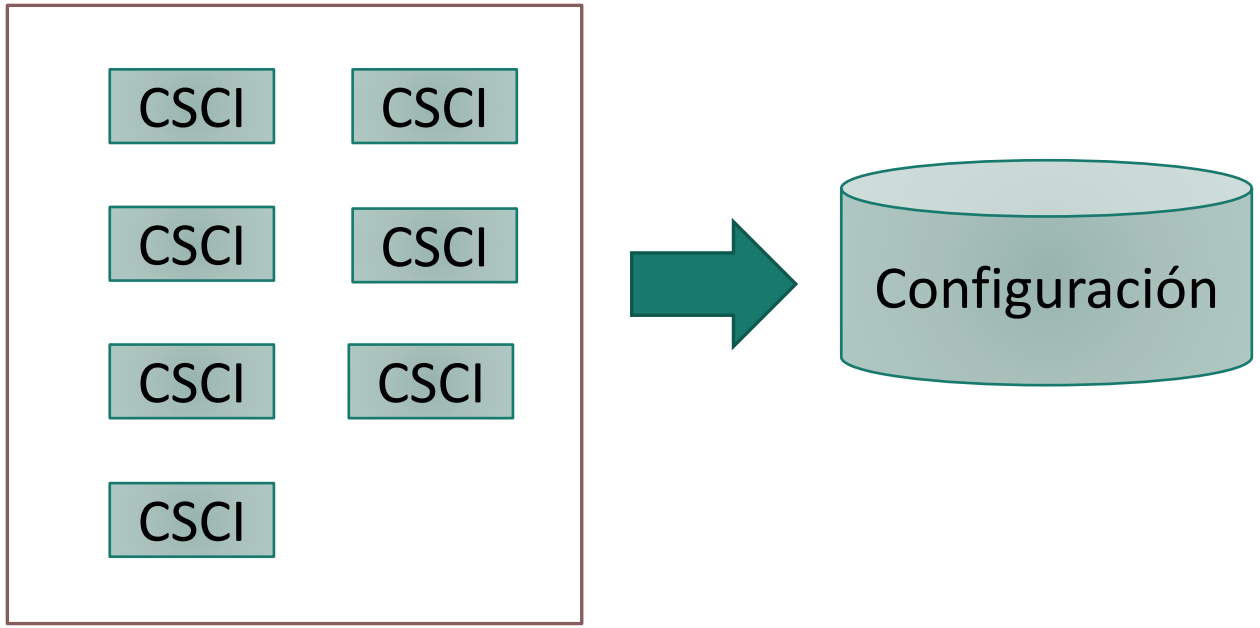
**Computer Sofware configuration item (CSCI)**

- Conjunto de elementos software que son tratados como una única entidad dentro del proceso de configuración del software.

# Terminología

**Configuración**

Colección de versiones concretas de ítems CSCI.



# Gestión de la configuración

- Según Métrica V3, la gestión de la configuración incluye:
  - Ejecutables.
  - Código fuente.
  - Modelos de datos.
  - Modelos de procesos.
  - Especificaciones de requisitos.
  - Pruebas.

# Gestión de la configuración

- Para cada elemento almacenado debe indicarse, como mínimo:
  - Nombre.
  - Versión.
  - Estado.
  - Localización.

# Gestión de la configuración

- Es importante que el proceso de gestión de configuración maneje de forma adecuada las siguientes tareas:
  - Elaboración de código fuente por varios desarrolladores simultáneamente.
  - Seguimiento del estado de cada una de las fases de desarrollo de software y sus cambios.
  - Integración de las diferentes partes del software en un todo.

# Cómo hacer una buena GC

- Objetivos:
  - Crear una base de datos donde se identifiquen todos los activos de la organización.
  - Tener acceso a información sobre las configuraciones para apoyar a los procesos de desarrollo del sofware.
  - Definir una base sólida para la gestión de incidentes, problemas y cambios.

# Cómo hacer una buena GC

- Beneficios:
  - Tener información precisa sobre los elementos de configuración y documentación.
  - Ayuda en la gestión financiera y en la previsión de gastos.
    - Costes de mantenimiento.
    - Costes de licencia software.
    - Renovación de contratos de mantenimiento.

# Cómo hacer una buena GC

- Beneficios:
  - Apoyo a los planes de emergencia.
    - El CMDB y la biblioteca de software facilita la restauración de servicios en caso de emergencia.
  - Permite llevar a cabo evaluaciones de impacto y planificar cambios correctos y seguros.
  - Evita el riesgo de interrupción de servicio después de un cambio.

# Ejemplos de software de GC

- Algunos de los sistemas de gestión de la configuración más comunes son:
  - Git.
  - Rational ClearCase.
  - Fossil.
  - AccuRev.

# Control de cambios

- Debe de realizarse de forma adecuada para evitar:
  - Cambios no aprobados.
  - Cambios no registrados y, por tanto, no conocidos.
  - Problemas de comunicación.

# Control de cambios

- Pasos para llevar a cabo un cambio:
  - Identificar el cambio.
  - Controlar el cambio.
  - Garantizar que el cambio se llevará a cabo de la forma adecuada.
  - Reportar los cambios a los interesados.

# Control de cambios

- ¿Quién debe estar involucrado en el cambio?

¡¡Todo el mundo!!

- Existen empresas que tiene personas dedicadas a este fin.

# Control de cambios

- Fundamentalmente, existen las siguientes fuentes cambio:
  - Nuevas condiciones en el negocio.
  - Cambios en las necesidades del cliente.
  - Restricciones de presupuesto (por ejemplo, mala gestión de riesgos).

# Control de cambios

- Partes implicadas en un proceso de gestión de cambios:
  - **Componentes**: Herramientas para la gestión de los CSCI en un proceso de GC.
  - **Procesos**: Procedimientos y tareas que permiten gestionar el cambio.
  - **Herramientas de automatización** de la construcción del. software.

# Control de cambios

- Y por supuesto …

Las personas

# Control de cambios

- ¿Cómo sé que un cambio se ha llevado con propiedad?
  - ¿Se ha realizado el cambio específico? ¿Se han incorporado modificaciones adicionales?
  - ¿Se ha realizado una revisión técnica formal para evaluar el cambio?
  - ¿Se han aplicado los estándares asumidos por el equipo de desarrollo?
  - ¿Se ha realizado un informe detallado del cambio?

# Control de cambios

- ¿Cómo asegurar la calidad del software durante el proceso de desarrollo?
  - Realización de documento de gestión de calidad.
  - Pasos:
    - Redacción del plan.
    - Aceptación por parte de todos los miembros implicados.
    - Aceptación del plan.

# Control de cambios

- ¿Qué elementos tiene que tener un plan de calidad del software?
  - Sección de documentación que rige el desarrollo, verificación, validación, uso y mantenimiento del software.
  - Estándares y convenios.
  - Técnicas de revisión e inspecciones a realizar en el proceso.

# Control de cambios

- ¿Qué elementos tiene que tener un plan de calidad del software?
  - Gestión de la configuración del software.
  - Sección de control de código.
  - Medios de comunicación durante el desarrollo.
  - Conservación y mantenimiento de la documentación.
  - Metodología de pruebas.

# Control de versiones

DECSAI

- Gestión de los cambios que se realizan sobre un determinado producto.

- Versión de un software: Estado del producto en un determinado momento de su desarrollo.

# Control de versiones

DECSAI

- Un sistema de control de versiones debe tener:
  - Mecanismo de almacenamiento de ficheros.
  - Modificación de los ficheros que almacena.
  - Registro histórico de los cambios realizados.
  - MUY DESEABLE: Poder volver a versiones anteriores del software.

# Control de versiones

- Dos tipos de control de versiones:
  - **Exclusiva**: Se indica que elemento se está modificando y nadie más lo modifica.
  - **Colaborativa**: Varios usuarios pueden modificar un mismo elemento a la vez.

# Control de versiones

DECSAI

- El control de versiones colaborativo debe tener:
  - **Mecanismos de combinación de elementos.**
  - **Resolución de conflictos.**

# Control de versiones

- **Flujo de trabajo.**
  - Indica como se relaciona entre sí el equipo que está trabajando sobre el proyecto software.
  - **Centralizado**: No se tiene acceso a las copias de los compañeros.
  - **Gestor de integraciones**: Escribes en tu repositorio pero tienes acceso de lectura a los demás.

# Control de versiones

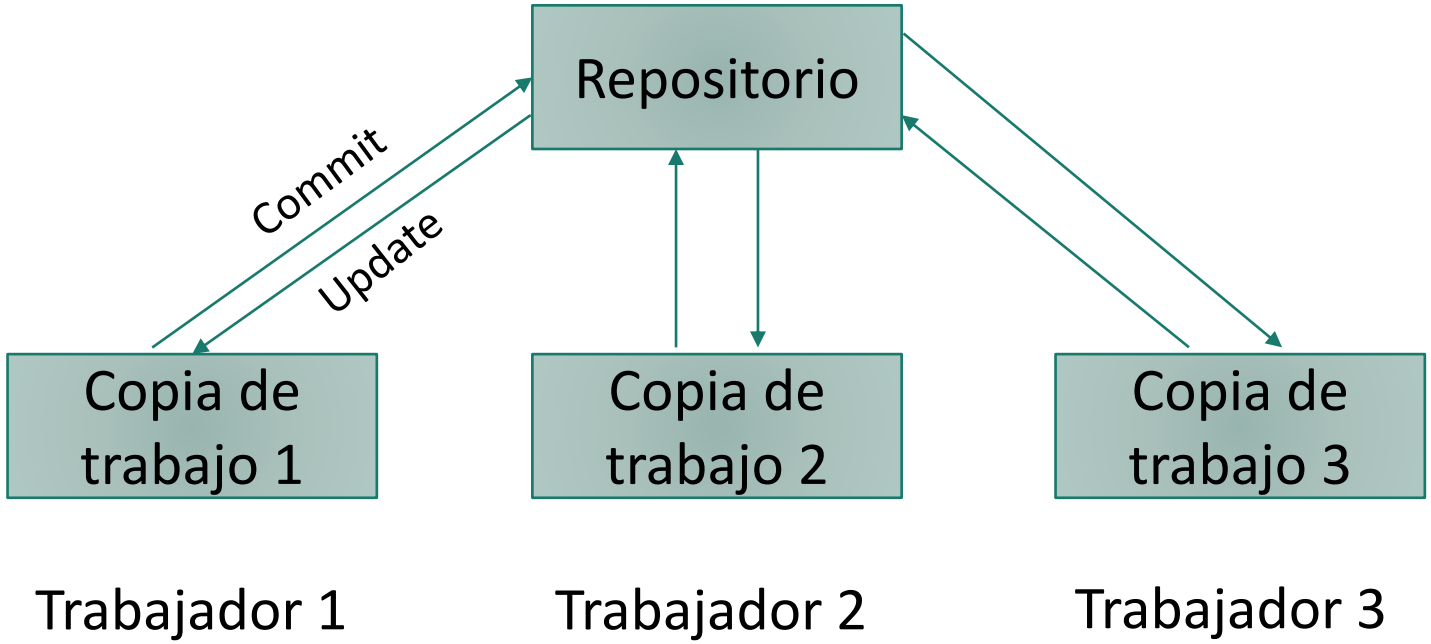
- **Flujo de trabajo con dictador y tenientes.**
  - Cada teniente tiene parte del repositorio.
  - El dictador integra todos los subrepositorios de los tenientes.
  - Es útil para equipos de trabajo muy grandes ya que permite delegar.

# Control de versiones

- Centralizado
  - Útil para equipos pequeños.
  - Tiene un servidor central para almacenar todos los archivos.
  - Se utiliza un servidor para acceder.

# Control de versiones

## Centralizado



# Control de versiones

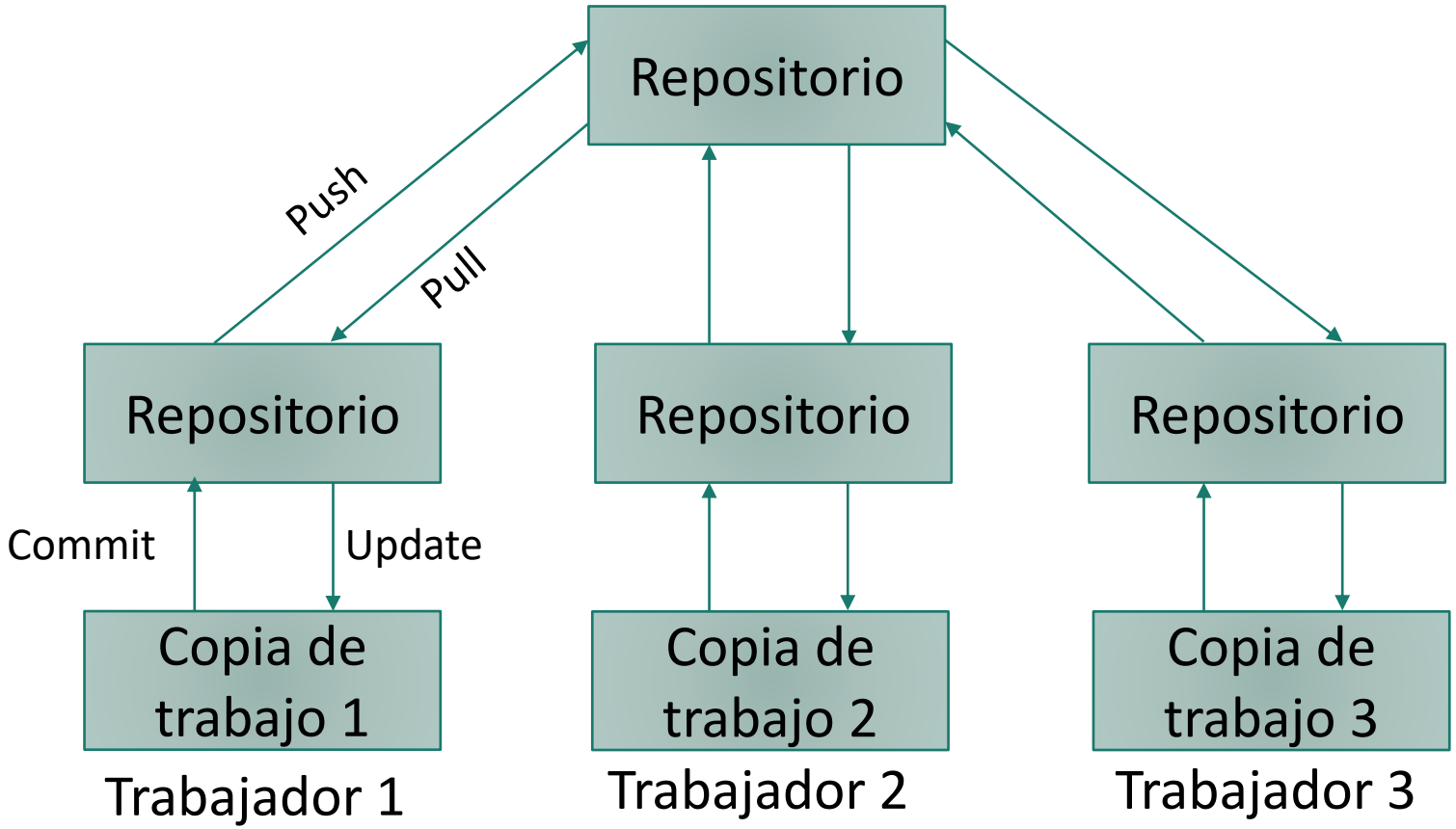
- Centralizado
  - Inconvenientes:
    - No está disponible localmente → hay que estar conectado a Internet.
    - Si el servidor se rompe se pierden todos los datos.

# Control de versiones

- Distribuido
  - Cada miembro del equipo tiene una copia local del repositorio.
  - No se requiere un servidor central para almacenar todos los archivos.

# Control de versiones

## Distribuido



# Control de versiones

- Distribuido
  - Beneficios:
    - Todas las operaciones son muy rápidas → no hay que acceder a un servidor remoto.
    - Los cambios commit no modifican el repositorio principal.
    - Los trabajadores pueden compartir cambios para obtener feedback antes de hacer push.

# GIT

DECSAI

- Originalmente para ser usado en bajo nivel por otros programas.
- Diseñado por Linus Torvalds.
- Contempla el desarrollo no lineal:
  - Gestión de diferentes ramas.
  - Mezclado de diferentes versiones.

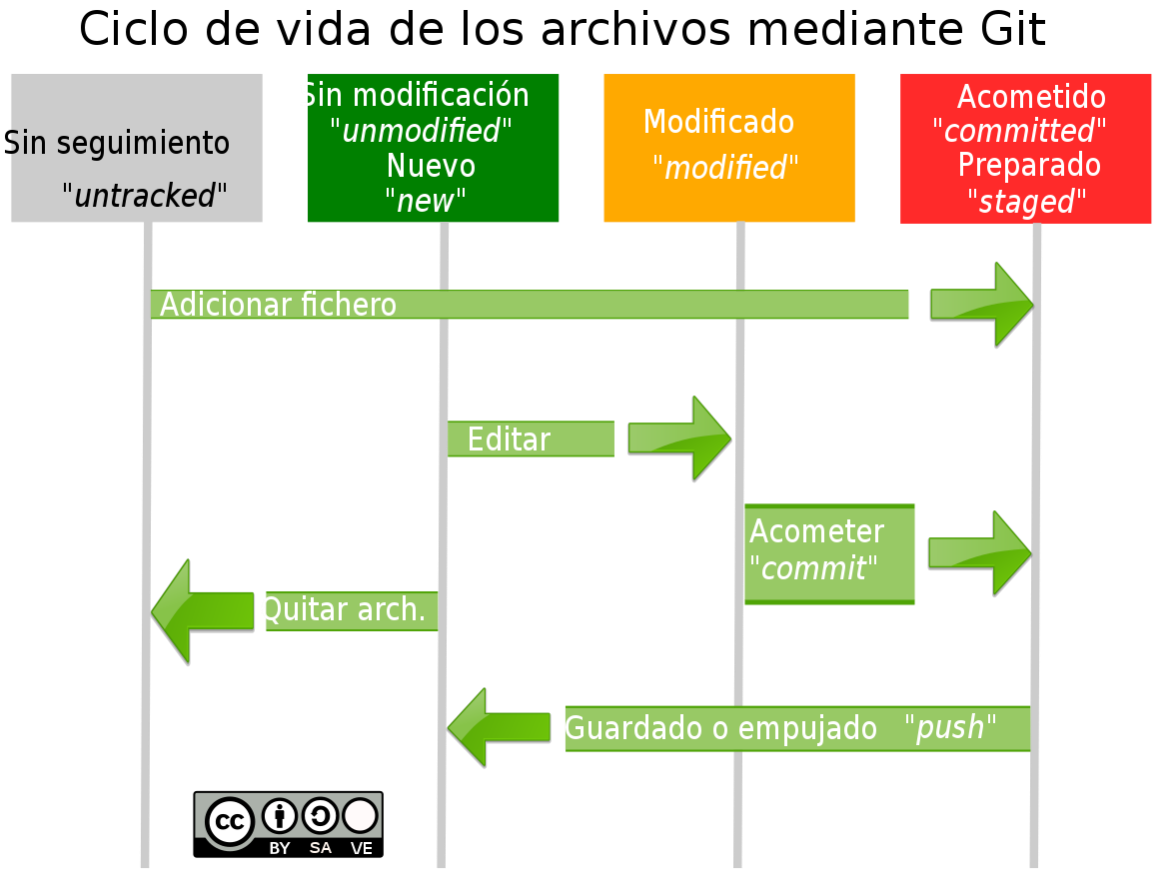
# GIT

DECSAI

## Características

- Cada programador recibe una copia del historial de desarrollo.
- Los cambios realizados se propagan entre los diferentes repositorios locales.
- Las diferentes ramas creadas pueden fusionarse.
- Los almacenes de información pueden publicarse por protocolos HTTP y FTP.

# GIT




Ciclo de vida de los archivos mediante GIT.

# GIT


Git workflow

# GIT

DECSAI

## Terminología

- **Línea base**: Versión aprobada de un documento o fichero fuente.
- **Abrir rama**: Un determinado módulo se duplica y a partir de entonces hay 2 versiones.
- **Desplegar** (check out): Una copia de trabajo local se obtiene del repositorio.
- **Publicar** (check in): Los cambios se integran en el repositorio.

DECSAI

## Terminología

- **Conflicto:**
  - 2 usuarios, u1 y u2 despliegan un archivo A.
  - u1 hace cambios en las líneas del archivo A.
  - u2 no actualiza y hace cambios en esas mismas líneas.
  - u2 envía los cambios al archivo A.

¿Qué código es el bueno? ¿El de u1 o el de u2?

DECSAI

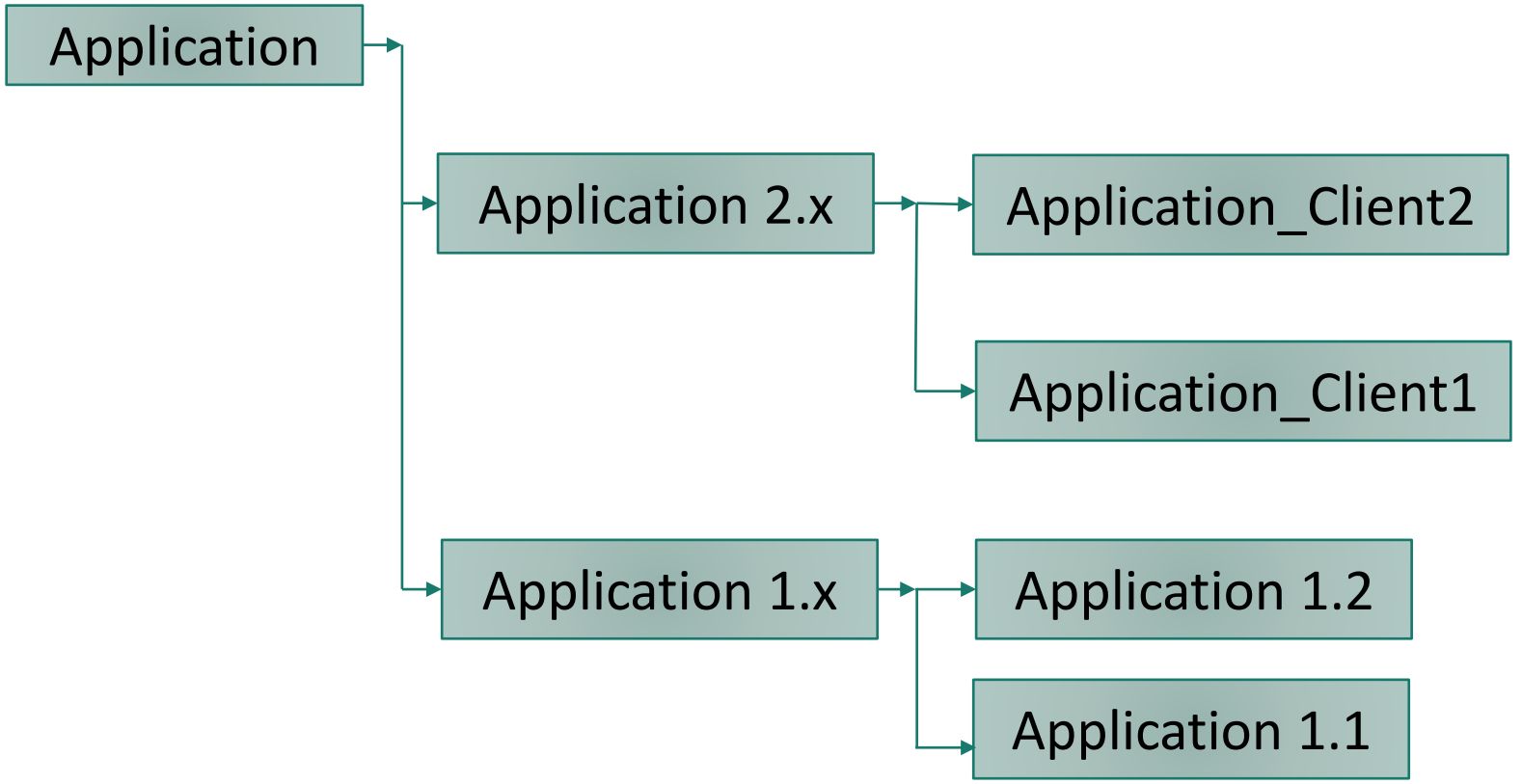
## Terminología

- **Resolver un conflicto:**
  - El usuario interviene manualmente y resuelve el conflicto.
- **Lista de cambios**:
  - Conjunto de cambios hecho en un commit.

# AccuRev

- Control de versiones centralizado con modelo cliente-servidor.
- Utiliza TCP/IP para las comunicaciones.
- Se utiliza un modelo de arquitectura de flujo (stream) que forma una estructura jerárquica.
- Mediante herencia los streams padres pasan propiedades a los hijos.

# AccuRev


Application
Application 2.x
Application_Client2
Application_Client1
Application 1.x
Application 1.2
Application 1.1

# Monotone

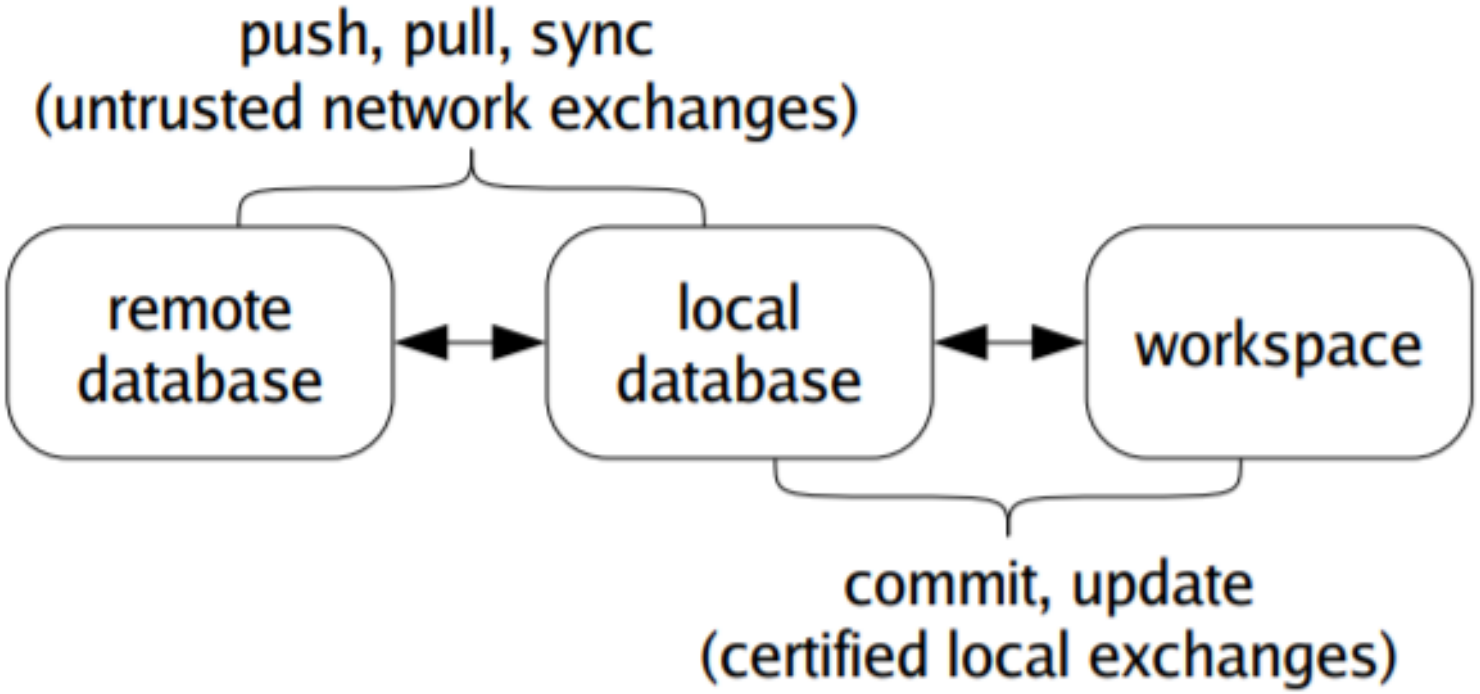
- Modelo distribuido de control de versiones.
- Firma de revisiones usando RSA.
- Permite importar proyectos de CVS.
- Protocolo propio: netsync.

# Monotone

- Se distinguen 4 elementos:
- **Keystore**: Almacenamiento de llaves privadas.
- **Workspace**: Área de trabajo.
- **Local database.**
- **Remote database**.

# Monotone

**Monotone**

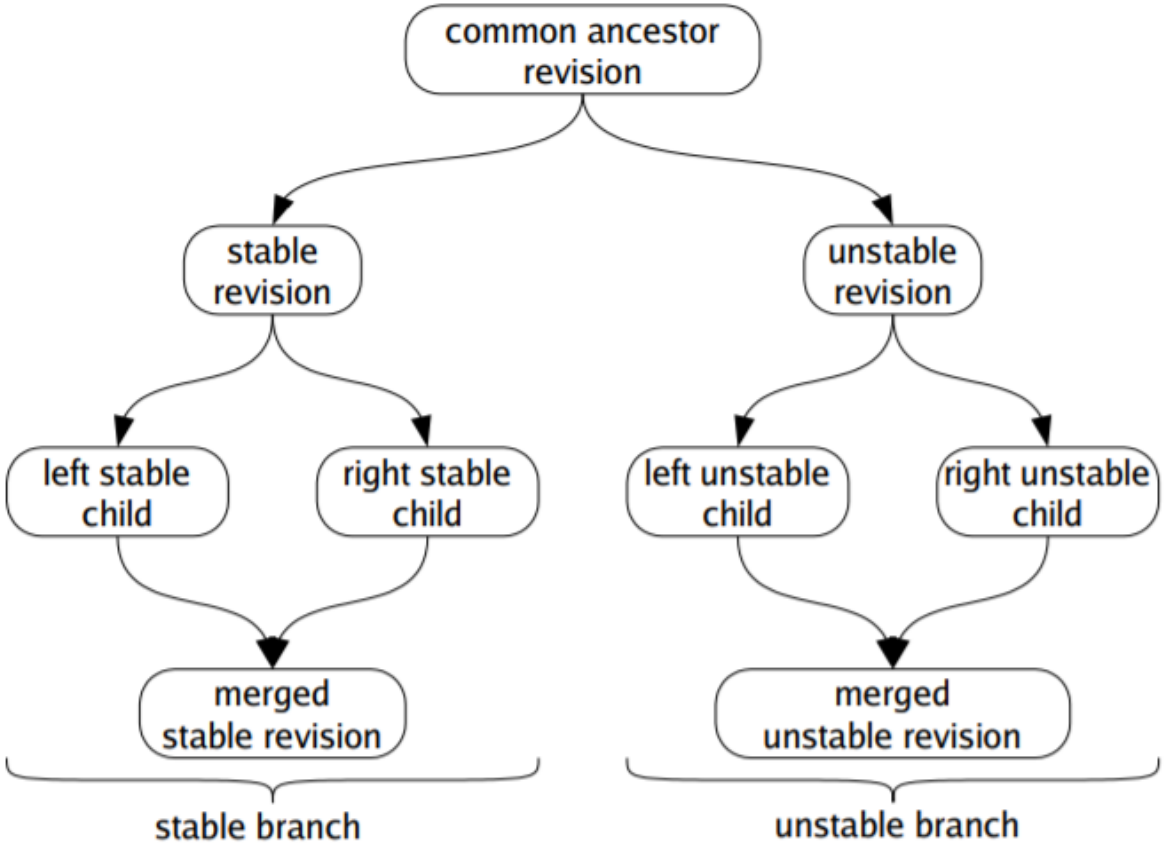



https://www.monotone.ca/monotone.pdf

# Monotone

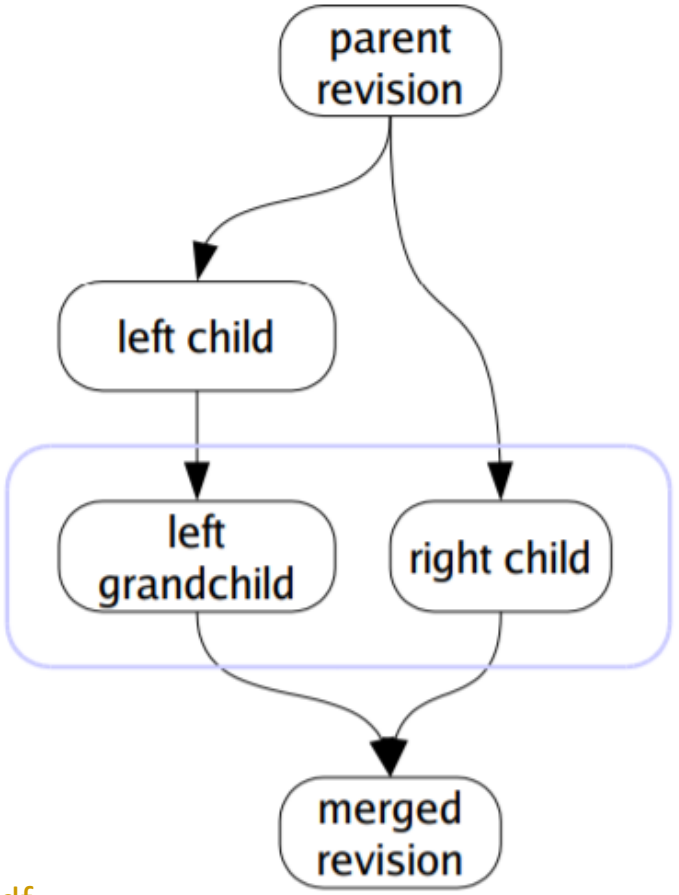
- **Push**: Copia datos de tu base de datos local al repositorio.
- **Pull**: Se copian los datos de la base de datos local al repositorio global.
- **Sync**: Copia datos en ambas direcciones. La base de datos local y la remota pasan a ser iguales.

# Monotone




https://www.monotone.ca/monotone.pdf

# Monotone




the heads of the branch (before the merge)

https://www.monotone.ca/monotone.pdf

# Monotone

DECSAI

## Resolución de conflictos

- **Contenido de fichero:**
  - Si las líneas no coinciden, combina.
  - Si coinciden pregunta.

- **Nombre duplicado.**
  - ¿Es el mismo fichero o el contenido difiere?
  - Mismo fichero: Se borra uno.
  - Distinto fichero: Se renombra.

# Monotone

DECSAI

## Resolución de conflictos

- **Colisión de espacios de trabajo.**
  - Alguien sube un fichero con el mismo nombre que uno que tienes en tu espacio de trabajo pero que no has subido.
  - Subes un fichero que tiene el mismo nombre que un directorio.
  - Alguien elimina un directorio que, en tu entorno de trabajo, contiene elementos con versión y sin versión. Los elementos que monotone no conoce, permanecen.

# Referencias

- Gestión de la configuración.
- GBAdvisors – Consejos para una buena gestión de la configuración
- Sistemas de control de versiones
- GIT
- Terminología del control de versiones

# Referencias


- Scoot Chacon; Ben Straub. ProGit. Apress.
- Brad Appleton; Stephen P. Berczuk. Software Configuration Management Patterns: Effective Teamwork, Practical Integration. Addison-Wesley Professional, 2002
- Roger Pressman. Ingeniería del Software. Un enfoque práctico. 5º edición.

# Ejercicio

Escoge dos sistemas de control de versiones y realiza un pequeño trabajo de una cara indicando cómo funcionan, cuáles son sus principales características y qué ventajas tiene uno frente al otro.