U.T.4: Optimización y documentación.

[Fuente: Entornos de Desarrollo, Alicia Ramos, Ed.Garceta]
[Fuente: Análisis y diseño de Aplicaciones Informáticas de Gestión, Mario Piattini, Ed.RA-MA]

Optimización y documentación

- Control de versiones
- Documentación
- Refactorización

Optimización y documentación

- Herramientas de control de versiones:
  - Trazabilidad de los cambios sufridos por un proyecto software
  - Quién o quienes han ejecutado dichos cambios
- Herramientas de documentación y refactorización:
  - Qué hace cada clase o método
  - Código eficiente, legible y fácil de mantener

Control de versiones

- Capacidad de recordar todos los cambios que se realizan tanto en la estructura de directorios como en el contenido de los archivos.
- Herramientas de control de versiones:

Control de versiones

- Conceptos:
  - Repositorio
  - Revisión o versión
  - Etiquetar (tag)
  - Tronco (trunk)
  - Ramificar (branch)
  - Desplegar (Checkout)
  - Confirmar (commit)
  - Exportación
  - Importación
  - Actualizar
  - Fusión (merge)
  - Conflicto
    - Resolver conflicto

Control de versiones: Tipos

- Centralizados
  - Existe un repositorio central donde se archiva el código y la información asociada
  - Los distintos desarrolladores trabajan con copias de ese código.
  - La versión “oficial” de referencia es la que hay en ese repositorio.
  - Todos los programadores sincronizan sus versiones con esa.

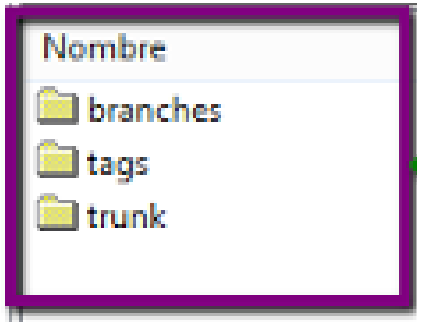
Control de versiones: Tipos

- Distribuidos
  - No existe un servidor central
  - Cada programador tiene su propio repositorio que puede sincronizar con el de cada uno de los demás.
  - En la práctica se suele definir un repositorio de referencia para albergar la versión “oficial” del proyecto.
  - Una de las plataformas más utilizada para alojar repositorios de git es GitHub.

Control de versiones con Subversion

- Herramienta multiplataforma de código abierto para el control de versiones.
- Utiliza una BD central llamada *Repositorio*
- El repositorio actúa como un servidor de ficheros con capacidad de recordar los cambios.
- Estructura de directorios recomendada
- Documentación Subversion

  https://subversion.apache.org/



Control de versiones con Subversion

- El ciclo de vida de subversion incluye las siguientes etapas:

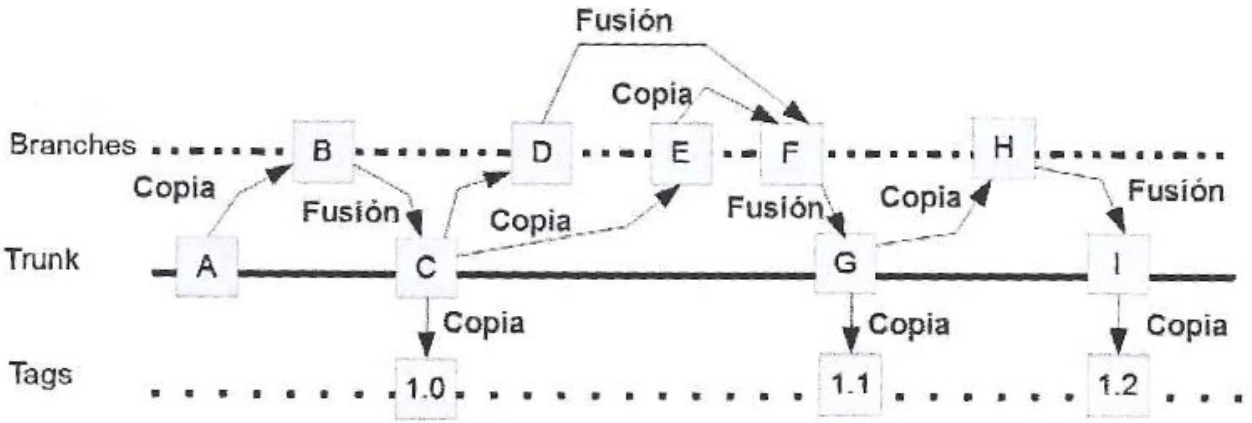
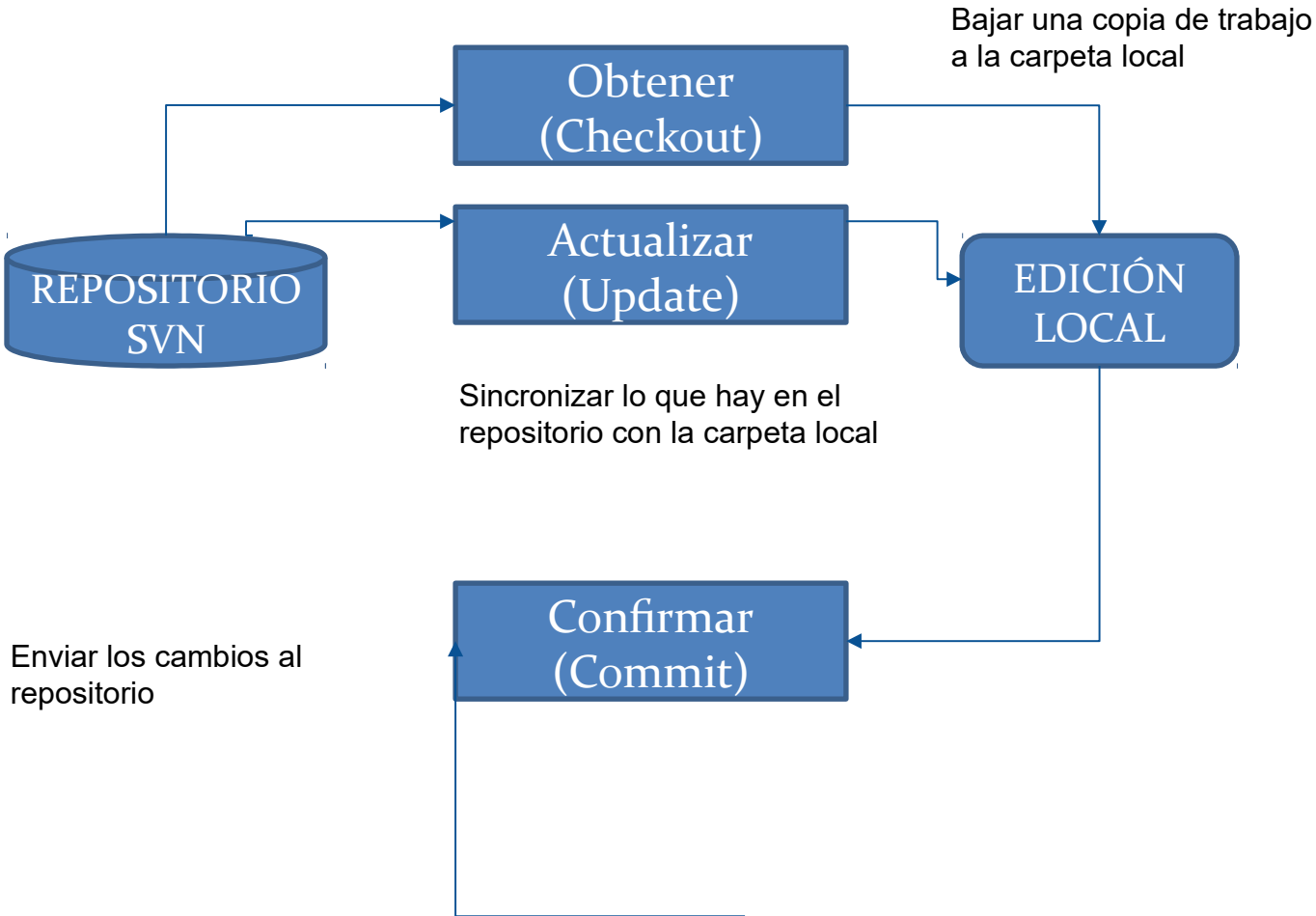


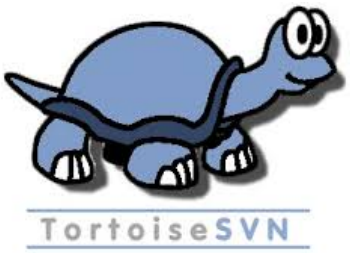
Figura 4.1. Ciclo de vida de subversión.

Operaciones básicas de control de versiones



Control de versiones: software

- Servidor Subversion para Windows
  - Visual SVN Server
  - Cliente Tortoise SVN



- Subversion en Eclipse
- Subversion en Netbeans

Control de versiones: Visual Subversion (SVN) Server

- Servidor de Subversión que se instala y administra fácilmente en entorno Windows.
- Se instala como un servidor web
  - Incluye un servidor Apache
- *Visual SVN Server* nos permite una configuración más rápida y amigable que otros servidores de control de versiones.

http://www.visualsvn.com/server/download/

Control de versiones: Visual SVN Server

- Algunos tutoriales para su instalación y configuración:

http://jppm1850.blogspot.com.es/2013/10/instalacion-y-configuracion-de-visual.html

https://www.visualsvn.com/server/getting-started/

http://tortoisesvn.net/docs/nightly/TortoiseSVN_es/tsvn-qs-basics.html

Control de versiones: Práctica 1

Desde el navegador de repositorios de Tortoise, selecciona la carpeta *trunk,* sube una carpeta y dos archivos cualesquiera, y añade un mensaje a las subidas.

Observa las revisiones que se han generado. Accede a cada una de ellas desde el botón *HEAD*, muestra el registro de revisiones, selecciona cada una de las revisiones y observa el mensaje añadido para esa revisión y los archivos afectados.

Visualiza el gráfico de revisiones.

Control de versiones: Git y GitHub

- ***Git*** es un sistema de control de versiones distribuido.
  - Tendremos una copia del repositorio completo en local.
  - Opcional y regularmente, los cambios los enviaremos a repositorios remotos, como puede ser GitHub.

- ***GitHub*** es un servicio para alojamiento de repositorios de software gestionados por el sistema de control de versiones Git.

Control de versiones: Git y GitHub

- Github es un proyecto comercial, a diferencia de la herramienta Git que es un proyecto de código abierto.

- Sin embargo, en Github es gratuito alojar proyectos Open Source.

Control de versiones: Git y GitHub

- Git es un sistema de control de versiones distribuido por lo tanto no necesita Github u otro sitio de alojamiento del código.

- Simplemente con tener Git instalado en el ordenador, tengo un sistema de control de versiones completo.

- Pero usar Github nos permite muchas facilidades, sobre todo a la hora de compartir código fuente.

Control de versiones: Git

- Instalar Git en el equipo local (Windows):
  - Como comando en la línea de comandos de Windows
  - *Git Bash*: es la propia línea de comandos de Git para Windows.
- Comprobar la instalación:

git version

Control de versiones: Git, primeros pasos

- Configurar dos parámetros básicos que nos identifican como usuario:
  - Nuestro correo electrónico y nuestro nombre:

    git config --global user.name "tu nombre"

    git config --global user.email "tu correo"
  - Las opciones que configures como --global se almacenarán en un archivo de la carpeta home del usuario llamado

    .gitconfig
  - Información sobre configuración de git:

    git config –list
  - Información sobre el historial de cambios en el repositorio:

    git log -n

Control de versiones: Git, primeros pasos

- Un *repositorio* de git no es más que un directorio de nuestro ordenador que está bajo el control de git.
- Creamos un nuevo repositorio en local:
  - Inicializar el repositorio Git sobre la carpeta del proyecto:

    git init → ***.git***
  - Añadir archivos al repositorio (staging area).

    git add index.html

    git status
  - Confirmar los cambios en el repositorio para ser monitorizados por Git.

    git commit –m «comentario del commit»

Control de versiones: Git, primeros pasos

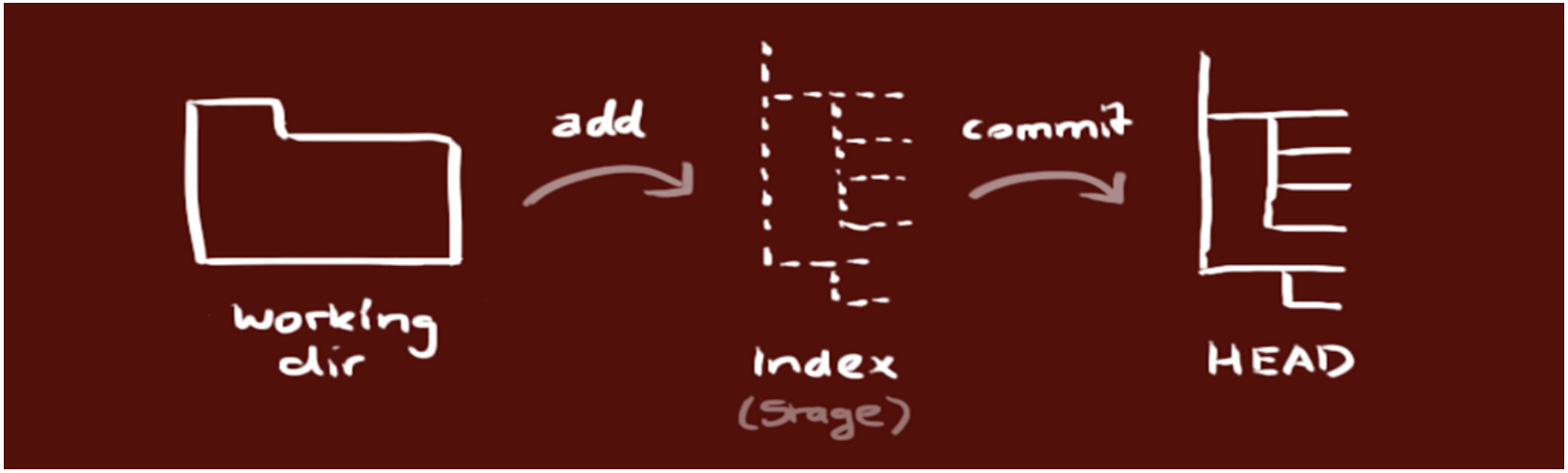
- Un repositorio también puede iniciarse copiando (*clonando*) otro ya existente.
  - Clonamos un proyecto de GitHub en nuestro equipo y trabajamos sobre él.

    git clone https://github.com/oslugr/repo-ejemplo.git

    git clone C:\Users\Carmen\Documents\proy1
  - Git usa su propio protocolo para el acceso remoto pero también soporta otros como ssh, http, https…
  - Con *git clone* no es necesario crear un directorio para el proyecto clonado.

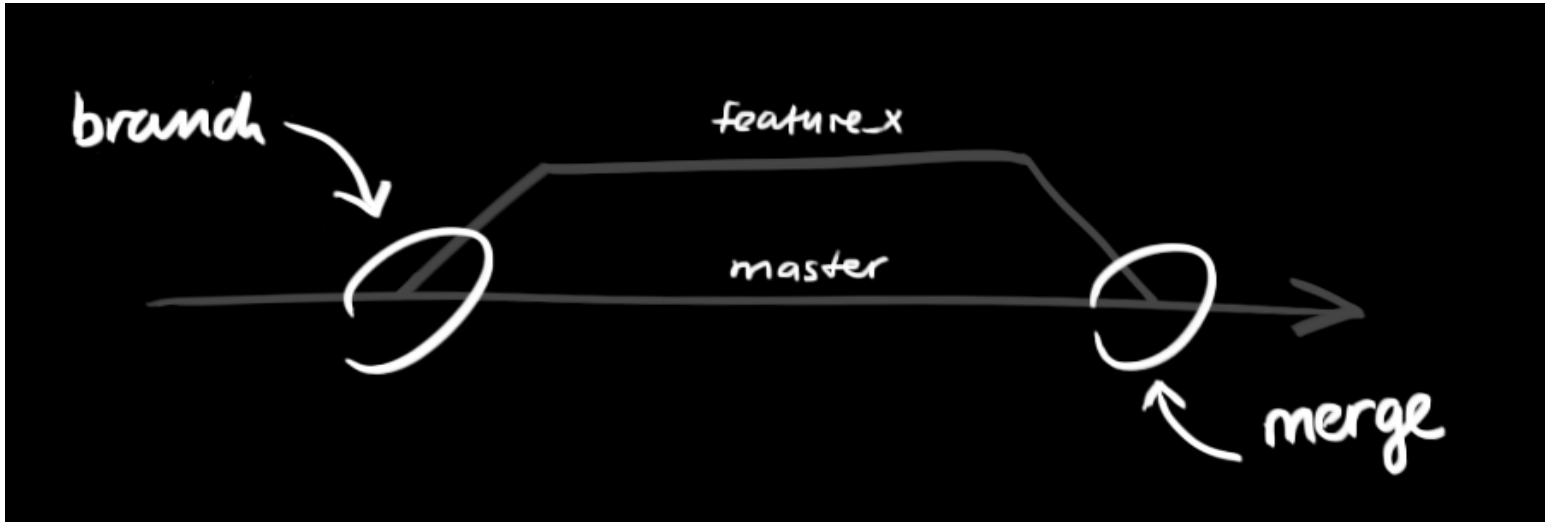
Control de versiones: Git, flujo de trabajo

- El repositorio local esta compuesto por tres “árboles“ administrados por git.
- El primero es tu Directorio de trabajo que contiene los archivos, el segundo es el Staging Area (Index) que actúa como una zona intermedia, y el último es el HEAD que apunta al último commit realizado.



Control de versiones: Git, las ramas

- Las Ramas son utilizadas para desarrollar funcionalidades aisladas unas de otras.
- La rama master es la rama "por defecto" cuando creas un repositorio.
- Crea nuevas ramas durante el desarrollo y fusiónalas a la rama principal cuando termines.



Control de versiones: Git, las ramas

- Crea una nueva rama llamada "feature_x" y cámbiate a ella usando:

  git checkout -b feature_x

- Vuelve a la rama principal:

  git checkout master

- Borra la rama:

  git branch -d feature_x

- Una rama nueva no estará disponible para los demás a menos que subas (push) la rama a tu repositorio remoto

  git push origin <branch>

Control de versiones: Git y GitHub

- Descargas
  - Sitio oficial de git https://git-scm.com/
  - Código fuente del proyecto git https://github.com/
- Documentación Git
  - https://git-scm.com/book/es/v1
  - https://git-scm.com/book/en/v2
  - https://progit2.s3.amazonaws.com/en/2016-03-22-f3531/progit-en.1084.pdf
- Documentación GitHub
  - https://conociendogithub.readthedocs.org/en/latest/data/introduccion/#que-es-github

Control de versiones: Git y GitHub

- Más documentación
  - http://www.desarrolloweb.com/articulos/introduccion-git-github.html
  - http://www.desarrolloweb.com/articulos/entiende-instala-configura-git.html
  - http://www.desarrolloweb.com/articulos/iniciar-repositorio-git-primer-commit.html
  - http://rogerdudler.github.io/git-guide/index.es.html

Documentación

- Texto escrito que acompaña a los proyectos de cualquier tipo.
- Tipos de documentación:
  - Documentación de las especificaciones
  - Documentación del diseño
  - Documentación del código fuente
  - Documentación de usuario final

Documentación de las especificaciones

- Definida por el estándar IEEE 830
- Debe incluir:
  - Introducción
    - Define los objetivos del software
  - Descripción de la información
    - Descripción detallada de la aplicación incluyendo software y hardware necesarios.
  - Descripción funcional
    - Diagramas que describen las funciones del sistema.
  - Descripción del comportamiento
    - Describe cómo reaccionará el software ante sucesos externos y controles internos
  - Criterios de validación
    - Describe límites de rendimiento, clases de pruebas, etc.

Documentación del diseño

- Describe las estructuras de datos:
  - clases,
  - métodos y sus atributos
  - bases de datos
- Define las funciones:
  - datos de entrada y salida,
  - tareas
  - …

Documentación de usuario final

- Describe cómo utilizar la aplicación desde el punto de vista de usuario especializado y no especializado.

- *"IEEE 1063-2001 Standard for Software User Documentation"* propone una estructura para el manual de usuario

Documentación del código fuente

- Documentar el código fuente de los programas aclara su funcionamiento y legibilidad.
- Facilita la corrección de errores
- Facilita la incorporación de nuevas funcionalidades
- Incluye:
  - Qué hace una clase o método o una variable
  - Uso previsto de cada variable
  - Qué algoritmo resuelve algún problema,
  - Qué se debería revisar si hubiera tiempo, …
- Existen herramientas para automatizar la documentación del código fuente
  - javadoc, PHPDoc, JSDoc

Documentación: javadoc

- Utilidad de Java para extraer y generar documentación en formato HTML desde el código fuente.
- Estructura de comentarios en ***Java***:
  - Comentarios de línea: "\\" al principio de la línea

    \\ *esto es un comentario de una línea*
  - Comentarios de varias líneas: "/* texto "*/"

    /* Esto es un comentario

    de varias líneas */
  - Comentarios para ***Javadoc***: "/** texto */

    /** Esto es un comentario

    * de varias líneas

    * para javadoc */

Documentación: javadoc

- Comentarios de documentación para ***JavaDoc***:
  - Deben colocarse antes de la declaración de una clase, atributo, método o constructor.
  - Cada línea comienza con el carácter '*'
  - Dentro de los delimitadores /** ….. */ pueden escribirse etiquetas HTML
  - Los comentarios pueden incluir etiquetas de javadoc (@) para documentar ciertos aspectos concretos de la clase

Documentación: javadoc

- Etiquetas de javadoc (@):

ETIQUETA	DESCRIPCIÓN
@author	Autor de la clase. Sólo para las clases
@version	Versión de la clase. Sólo para las clases
@see	Referencia a otra clase de la API o del proyecto.
@param	Descripción de parámetro. Una etiqueta por cada parámetro.
@return	Descripción de lo que devuelve. Sólo si no es *void*.
@throws	Descripción de la excepción que puede propagar. Una etiqueta por cada tipo de excepción.
@deprecated	Marca el método como obsoleto.
@since	Nº de versión desde la que existe el método

Documentación: javadoc

- Generar la documentación:
  - En Eclipse:

    Menú ***Project*** -> ***Generate Javadoc***
  - En Netbeans:

    Menú ***Run*** -> ***Generate Javadoc***