Optimización de Código

TEMA 2.2

Refactorización.

Refactorización o limpieza o reordenación de código sistemática, paralela y sin errores (calidad, claridad, legibilidad, mantenibilidad, productividad, ...), frente a optimización (algoritmo, operaciones, memoria, rendimiento, ...)

patrones, catálogos o métodos de refactorización (disponibles en los IDE

- Codificar con tabulación, sangrado, indentación, o endentación
- Usar nombres de objetos significativos.
- Extraer métodos, convirtiendo fragmentos de código en métodos con un nombre significativo.
- Encapsular atributos, convirtiendo atributo públicos en privados.
- Utilización de constantes, en lugar de valores fijos repetidos.
- Extraer subclases, para tareas separadas con sus propias propiedades.
- Extraer interfaz con métodos comunes a varias partes.

Malos Olores

- métodos largos
- clases grandes
- listas de parámetros largas
- uso excesivo de tipos primitivos
- clases triviales solo con get y set
- alternativas múltiples muy anidadas
- necesidad de comentarios excesivamente largos
- atributos de uso muy esporádico
- jerarquía de clases especulativa
- jerarquías de clases paralelas que hay que repetir

- clases exclusivamente intermediarias sin otra utilidad
- herencia de clases parcialmente innecesaria
- clases intrusas en parte privada de otras clases sin justificación
- paso de parámetros encadenado en secuencia
- clases sin apenas cometido o poca razón de existir
- cadena de cambios en clases muy cohesionadas o dependientes
- clases más dependientes externamente que de sí mismas
- El mismo código repetido en clases distintas
- bloques de datos tratados conjuntamente sin una clase dedicada

ciclo de refactorización (aplicar pruebas -> identificar malos olores -> refactorizar con herramientas -> volver a aplicar pruebas)

Referencias.

Refactoring

https://sourcemaking.com/refactoring/

Refactoring Guru

https://refactoring.guru/

Cómo funciona la refactorización en Java

https://codegym.cc/es/groups/posts/es.196.como-funciona-la-refactorizacion-en-java

Control de Versiones.

Un **Sistema de Control de versiones** o **Version Control System** (VCS) permite manejar los cambios en los archivos de código fuente de un proyecto software, aunque podrían ser de cualquier otra naturaleza, gestionando en un entorno de trabajo compartido el histórico de modificaciones en el código, con posibilidad de revertir los cambios en caso de producirse errores.

Tipos

Sistema **centralizado** de Control de versiones (CVCS), con enfoque cliente/servidor (Sistema de Versiones Concurrentes o CVS, Subversión o SVN, Perforce, SourceSafex, ...)

Sistema **distribuido** de Control de versiones (DVCS), con enfoque entre iguales (Git, Mercurial, Bazaar, Darcs, BitKeeper, ...)

Terminología

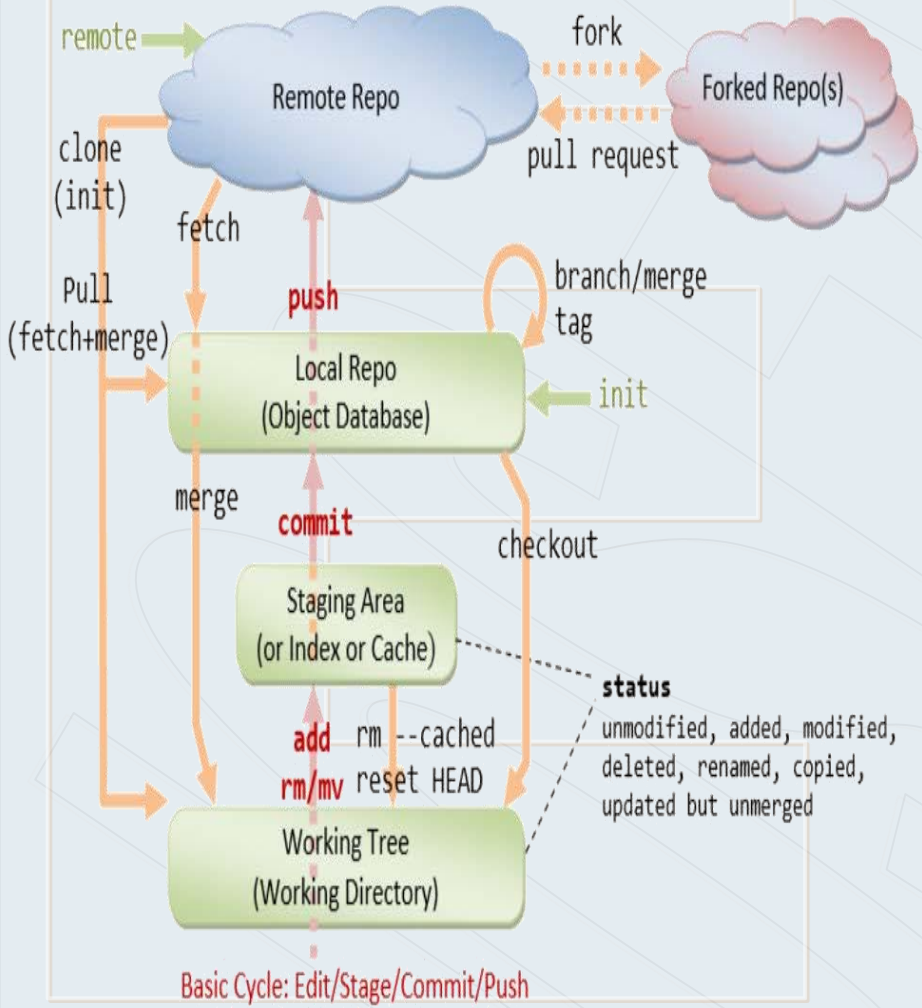
- **Repositorio**, **deposit**, **depot** (archivos, histórico, changuelog, ...) / **Módulo** (Proyecto).
- **Revión** o **versión** (head, ...) / **Etiqueta** o **rótulo** (tag) / Linea base (baseline).
- **Rama** o **bifurcación** (branch: main, ...) / **Integración** o **fusión** (merge).
- **Cambio** (change, diff, delta) / Conflicto / Resolución.
- **Publicación** o **punto de confirmación** (commit, check in, install, submit) / **Despliegue** o **restablecimiento** (checkout).
- **Sincronización** (pull / push).
- **Exportación** (export) e **importación** (import).

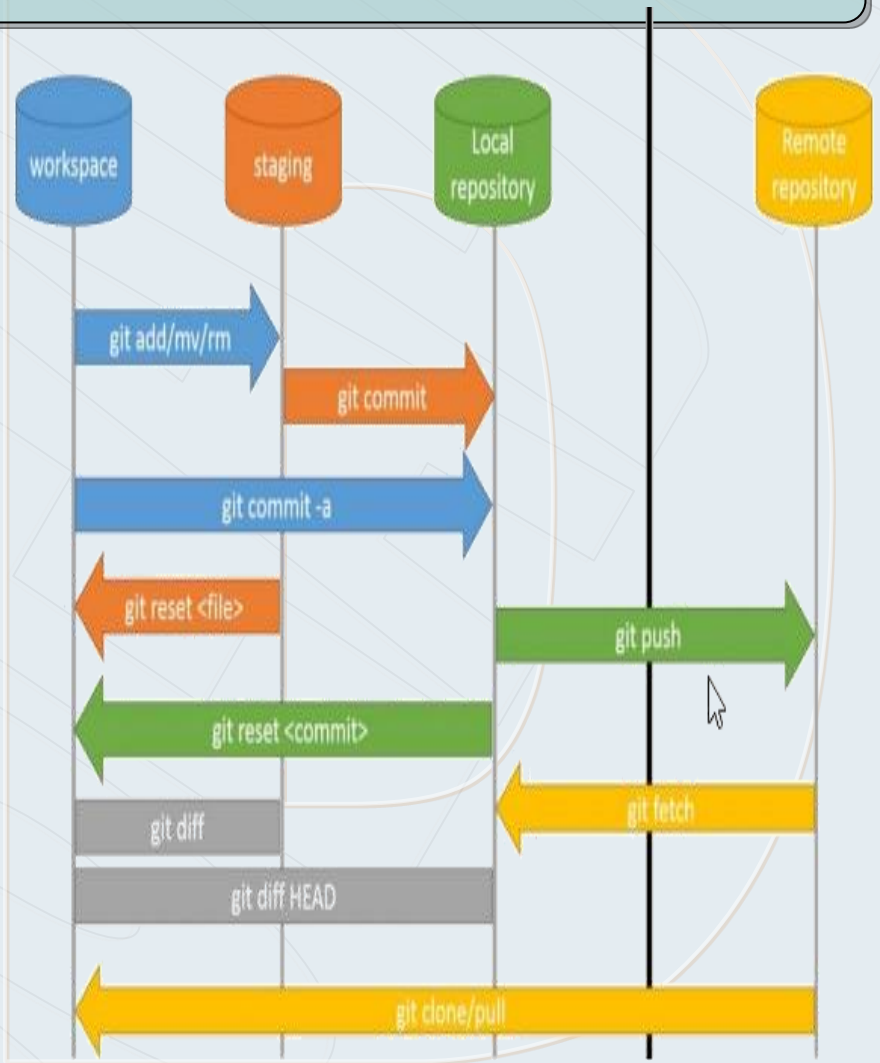
Git (Linus Torvalds - 2005) es un sistema de control de versiones distribuido (DVCS), con Interfaz de comandos (git bash para windows), y también posibilidad de interfaz gráfica (git-gui), a través de comandos para la gestión de cambios.

Elementos

- **Modelo de almacenamiento Git**.
  - **Directorio de trabajo** o **working directory** (readme.md, .Git, .gitignore, ...).
  - **Zona intermedia** o **zona de preparación** o **index-staging**.
  - **Repositorio Local** o **git directory** / **Repositorio Remoto** (GitHub, GitLab, BitBucket) o repositorio Git de proyectos alojados en la web que funcionan como plataformas de desarrollo colaborativo de software construidas sobre el sistema de Git.
- **Flujo de trabajo** / **Ciclo de vida** (iniciar/clonar, editar/cambiar, preparar/reiniciar, confirmar/restablecer, empujar/traer).
- **Estados de archivos** (untracked, unmodified, modified, staged).

Control de Versiones.





Referencias.

Taller de Git y GitHub desde cero

https://informatica.ucm.es/data/cont/media/www/pag-66886/TallerGitGitHub.pdf

Using Git in Apache NetBeans

https://netbeans.apache.org/tutorial/main/kb/docs/ide/git/

Markdown (sintaxis de escritura y formato básicos)

https://docs.github.com/es/get-started/writing-on-github/getting-started-with-writing-and-formatting-on-github/basic-writing-and-formatting-syntax

Basic Syntax The Markdown elements outlined in the original design document

https://www.markdownguide.org/basic-syntax/

Documentación.

Documentación de código, basada en comentarios, delimitados por marcas y símbolos de caracteres con determinado significado.

Formatos

- **Texto plano** con comentarios sin formato o con formato libre, de línea o de bloque.
- **Texto enriquecido** (markdown, reStructuredText, asciidoc, javadoc, ...) con comentarios con formato, mediante lenguaje de marcado ligero de documentos, convertible a otros formatos de documentos (HTML, XHTML, PDF, EPUB, LaTeX, ...).

Javadoc es un formato enriquecido de comentarios (al principio de las clases y los métodos), a base de etiquetas (de clases e interfaces, y de constructores y métodos) diversas (autor, versión, referencia, parámetros, retorno, herencia, código, ...), siguiendo las reglas y estándares de documentación, que permite generar la documentación técnica de la API de las aplicaciones en formato HTML.

Las etiquetas de **Javadoc** van precedidas por @, estos son los mas usados:

ETIQUETA	DESCRIPCIÓN
@author	Autor de la clase. Solo para las clases.
@version	Versión de la clase. Solo para clases.
@see	Referencia a otra clase, ya sea del API, del mismo proyecto o de otro. Por ejemplo: @see cadena @see paquete.clase#miembro @see enlace
@param	Descripción parámetro. Una etiqueta por cada parámetro.
@return	Descripción de lo que devuelve. Solo si no es void. Podrá describir valores de retorno especiales según las condiciones que se den, dependiendo del tipo de dato
@throws	Descripción de la excepción que puede propagar. Habrá una etiqueta throws por cada tipo de excepción.
@deprecated	Marca el método como obsoletc. Solo se mantiene por compatibilidad.
@since	Indica el nº de versión desde la que existe el método.

Referencias.

Documentación oficial Javadoc

https://www.oracle.com/es/technical-resources/articles/java/javadoc-tool.html