GIt

Configurar git en línea de comandos:

1. Ejecuta el comando git config --global user.name "usuario" para establecer tu nombre de usuario en Git.
2. Ejecuta el comando git config --global user.email "email@ejemplo.com" para establecer tu correo. Asegúrate de usar el correo vinculado a tu cuenta de git.
3. Puedes verificar si se ha establecido correctamente la configuración con el comando git config --list

Comandos

git init: Inicializa un nuevo repositorio en la carpeta actual. Este comando crea un directorio oculto llamado ".git" que contiene la información del repositorio.

git status: Muestra el estado actual del repositorio, incluyendo archivos modificados y nuevos que aún no se han agregado al repositorio.

git add [archivo]: Agrega un archivo específico al área de preparación. Los archivos en el área de preparación serán incluidos en el próximo commit. También se puede hacer “git add .” para añadir todos los archivos nuevos y con cambios.

git commit -m "mensaje": Crea un commit con los cambios actuales del área de preparación y agrega un mensaje para describir los cambios.

git push: Actualiza los archivos del repositorio de acuerdo al commit que se hizo.

git clone [url]: Clona un repositorio existente en la url especificada en la carpeta actual.

git pull: Descarga y combina los cambios del repositorio remoto con el repositorio local.

Branch: es una línea de desarrollo independiente de la principal llamada "main" (anteriormente master). Puedes crear una nueva rama para implementar nuevas características o solucionar un bug sin afectar el código en la rama principal.

Pasos para usar git branch:

1. **Crea una nueva rama**: Para crear una nueva rama, utiliza el comando "git branch (nombre de la rama)" en la terminal. Por ejemplo, "git branch principal2" crea una nueva rama llamada "principal2".
2. **Cambia a la nueva rama**: Para trabajar en la nueva rama, debes cambiarte a ella. Puedes hacerlo utilizando el comando "git checkout (nombre de la rama)" en la terminal. Por ejemplo, "git checkout principal2" cambia a la rama "principal2".

3. **Haz cambios en la nueva rama**: Los cambios que hagas en la nueva rama no van a afectar a la principal.

Merge: Merge es el proceso de combinar una rama con otra. Por ejemplo, una vez que se ha completado una nueva característica en una rama de desarrollo, se puede fusionar esa rama con la rama principal para incluir los cambios en la versión principal del código.

Cómo usar git merge: Una vez que hayas completado los cambios en la nueva rama, puedes fusionarla con la rama principal. Para hacerlo, debes estar en la rama principal y utilizar el comando "git merge (nombre de la rama)" en la terminal. Por ejemplo, "git merge principal2" fusiona la rama "principal2" con la rama principal.

Resolver conflictos: Cuando intentas fusionar 2 ramas y ambas ramas han modificado el mismo archivo de manera independiente hay un conflicto.

Pasos para resolver conflictos:

1. **Identifica el conflicto:** Git marcará los archivos con conflictos con una serie de marcadores especiales que indican dónde comienzan y terminan los conflictos.
2. **Abre el archivo con conflicto:** Abre el archivo en un editor de código y busca los marcadores de conflicto.
3. **Decide qué cambios mantener:** Revisa el código en los marcadores de conflicto y decide qué cambios debes mantener. Puedes eliminar los marcadores de conflicto y los cambios que no deseas mantener.
4. **Haz un commit**: Una vez que hayas decidido qué cambios debes mantener, debes hacer un commit para finalizar el proceso de resolución de conflictos.
5. **Continua con el merge**: Una vez que has hecho el commit, puedes continuar con el merge si no hay más conflictos.

Try with resources

Try-with-resources: Es una característica que permite automatizar el cierre de recursos adquiridos dentro de un bloque try. Con esto, se pueden declarar recursos en el bloque try y automáticamente se cierran los recursos al finalizar el bloque try, ya sea que se produzca una excepción o no. Sirve para no tener que escribir código adicional para que los recursos se cierren correctamente, esto hace que haya menos posibilidad de errores relacionados con recursos no cerrados.
Ejemplo sin try-with-resources:

```
FileReader fr = new FileReader(path);
    BufferedReader br = new BufferedReader(fr);
    try {
        return br.readLine();
    } finally {
        br.close();
        fr.close();
    }
```

Ejemplo con try-with-resources.

```
try (FileReader fr = new FileReader(path);
     BufferedReader br = new BufferedReader(fr)) {
           return br.readLine();
}
```

Collections

List: Representa una lista ordenada de elementos, donde cada elemento tiene un índice. La clase más común que la implementa es ArrayList, que es una clase dinámica y permite acceder a los elementos por índice. También existe LinkedList, que es una implementación de una lista enlazada y tiene un mejor rendimiento en operaciones de inserción o eliminación en el medio de la lista.

Queue: Representa una estructura de datos de cola (FIFO) y es implementado por diversas clases como LinkedList o ArrayList, permite añadir elementos al final de la cola y extraer elementos desde el principio.

Set: Representa un conjunto de elementos, donde no se permiten elementos repetidos. La clase más común que la implementa es HashSet, que es una implementación de un conjunto que permite almacenar elementos sin orden y sin elementos repetidos. También existe LinkedHashSet que es una implementación de un conjunto que permite almacenar elementos sin elementos repetidos y mantiene el orden de inserción.

Map: Representa un mapa de clave-valor. La clase más común que la implementa es HashMap, que permite almacenar elementos mediante parejas de clave-valor y buscar elementos mediante su clave. También existe TreeMap, que es una implementación ordenada de un mapa, donde las claves están ordenadas.

Multicatch

Manejo de excepciones: El manejo de excepciones se realiza mediante el uso de bloques try-catch. El bloque try contiene el código que puede generar una excepción, y el bloque catch contiene el código que maneja esa excepción.

Multicatch: El concepto de "multicatch" se refiere a la capacidad de manejar varias excepciones diferentes en un único bloque catch. En lugar de tener varios bloques catch para manejar cada una de las excepciones, se pueden especificar varias excepciones en un único bloque catch. Esto puede mejorar la legibilidad del código y evitar la duplicación de código.
Ejemplo:

```
try {
    FileInputStream file = new FileInputStream("example.txt");
} catch (FileNotFoundException | IOException e) {
    System.out.println("Error: " + e.getMessage());
}
```

Arquitecturas

Microservicios: Arquitectura en la cual una aplicación se divide en pequeños servicios independientes, cada uno con una responsabilidad específica. Cada servicio se comunica con los demás a través de una interfaz establecida, como una API. Los microservicios son desplegados y escalados de manera independiente, lo que permite una mayor flexibilidad y escalabilidad.

Monolítica: Arquitectura en la cual todas las funcionalidades de una aplicación se encuentran en un solo código y en un único paquete de despliegue. Esto significa que si se desea actualizar o escalar una parte específica de la aplicación, se debe actualizar o escalar toda la aplicación.

Diferencias:
Escalabilidad: Los microservicios son más fáciles de escalar de manera independiente que las monolíticas.

Flexibilidad: Los microservicios permiten a los equipos trabajar de manera independiente y con tecnologías diferentes (Diferentes lenguajes de programación, frameworks, etc.

Facilidad de mantenimiento: Los microservicios permiten a los equipos trabajar en pequeñas partes de la aplicación, lo que facilita la depuración y resolución de problemas.

Complejidad: Una arquitectura de microservicios puede ser más compleja que una arquitectura monolítica debido a la necesidad de manejar y coordinar múltiples servicios.

Scrum

1. Establecer un equipo multidisciplinario: El equipo debe estar compuesto por personas con habilidades diferentes que puedan trabajar juntas para completar un objetivo común.
2. Establecer un Scrum Master: Este es el miembro del equipo responsable de facilitar el proceso de Scrum y de asegurar que el equipo siga las reglas y los procesos de Scrum.
3. Establece un Responsable de Producto: Este es el miembro del equipo que tiene la responsabilidad de representar a los intereses del negocio y priorizar el trabajo del equipo.
4. Establecer un Sprint: Un sprint es un período de tiempo en el que el equipo trabaja para completar un conjunto específico de tareas. Los sprints suelen durar entre una y cuatro semanas.
5. Crear una bitácora de producto: Este es una lista de tareas u objetivos que el equipo debe completar durante el proyecto. La bitácora debe evolucionar al igual que el proyecto.El Responsable de Producto es el responsable de mantenerla actualizada.
6. Planear el sprint: Se hace una reunión para planificar el próximo Sprint (un periodo de tiempo corto, generalmente una semana o dos, durante el cual se desarrolla una parte del proyecto) y seleccionar las tareas de la bitácora que se van a trabajar durante el Sprint.
7. Realizar una reunión diaria de Scrum: Debe tardar aproximadamente 15 minutos y durante esta reunión, el equipo se reúne para discutir el progreso del sprint y planificar el trabajo para el día siguiente. En la reunión se deben contestar las siguientes preguntas: ¿Qué hice ayer para ayudar a completar el trabajo del sprint? ¿Qué voy a hacer hoy para ayudar a completar el

trabajo del sprint? ¿Hay algún impedimento que me impida completar mi trabajo?

8. Realizar una revisión de sprint: Al final de cada sprint, el equipo se reúne para revisar el trabajo completado y planificar el próximo sprint.
9. Realizar una reunión de retrospectiva: Al final de cada sprint, el equipo se reúne para reflexionar sobre cómo mejorar el proceso de Scrum y el trabajo en equipo.
10. Implementar un sistema de medición: El equipo debe establecer un sistema para medir el progreso y el rendimiento del proyecto.
11. Iniciar el siguiente sprint: Tomando en cuenta lo aprendido, es importante que el equipo continúe reflexionando y mejorando el proceso.

MVC