1 OBJETIVOS

- Comprender el concepto de control de versiones y cómo los sistemas como Git gestionan diferentes versiones de un proyecto.
- Implementar y manipular estructuras de datos como arreglos y tablas hash para almacenar y gestionar eficientemente la información versionada.
- Desarrollar habilidades en programación en C, centrándose en el manejo de memoria, punteros, y eficiencia algorítmica.
- Simular operaciones de un sistema de control de versiones para obtener una comprensión más profunda de su funcionamiento interno.

2 DESCRIPCIÓN DEL PROYECTO

μGit(ugit) es un simulador de un sistema de control de versiones que emula funcionalidades básicas de Git. Este proyecto requiere implementar un sistema que permita a los usuarios realizar operaciones como agregar archivos, crear commits, ver el historial de versiones, y cambiar entre versiones, utilizando estructuras de datos como arreglos y tablas hash en el lenguaje C. La finalidad del proyecto es proporcionar a los estudiantes una experiencia práctica en la manipulación de estructuras de datos para resolver problemas reales y en la implementación de conceptos fundamentales de control de versiones.

3 NORMATIVAS DE CODIFICACIÓN EN C

CONSISTENCIA EN EL ESTILO DE CÓDIGO

- Usar nombres de variables y funciones descriptivos y seguir un esquema de nomenclatura coherente (e.g., snake_case o camelCase).
- Mantener la consistencia en la indentación y espaciado (uso de 4 espacios por nivel de indentación recomendado).

MODULARIDAD

- Dividir el código en funciones pequeñas y manejables que realicen una única tarea.
- Evitar funciones que superen las 50 líneas de código, a menos que sea estrictamente necesario.

DOCUMENTACIÓN

- Identificar cada código con el programador
- Incluir comentarios claros y precisos para explicar la funcionalidad de bloques de código complejos.
- Documentar cada función con comentarios que expliquen su propósito, parámetros de entrada, valores de retorno y posibles errores.

GESTIÓN DE ERRORES

- Implementar mecanismos robustos para la gestión de errores, asegurándose de que todos los errores posibles sean capturados y manejados adecuadamente.

USO EFICIENTE DE MEMORIA

- Evitar el uso innecesario de variables globales.
- Liberar memoria dinámicamente asignada cuando ya no sea necesaria para evitar fugas de memoria.

OPTIMIZACIÓN

- Escribir código optimizado en términos de tiempo y espacio, priorizando la eficiencia sin sacrificar la claridad.

4 EJEMPLO DE USO:

Supongamos que estamos desarrollando un pequeño proyecto de software y queremos usar μGit para gestionar las versiones.

Inicialización del repositorio:

El usuario ejecuta ugit init, lo que inicializa un nuevo repositorio.

Agregar archivos:

El usuario crea un archivo main.c y ejecuta ugit add main.c. Este comando añade el archivo al área de preparación (staging area).

Crear un commit:

El usuario ejecuta ugit commit -m "Initial commit" para crear un commit que captura el estado actual del proyecto (solo main.c en este caso).

Ver el historial de commits:

Al ejecutar ugit log, se muestra el historial de commits con información como el identificador del commit y el mensaje asociado ("Initial commit").

Cambiar a una versión anterior:

Si el usuario quiere volver a un estado anterior del proyecto, puede ejecutar ugit checkout <commit_id>, donde <commit_id> es el identificador del commit al que desea volver.

Flujo de Ejecución:

```
$ ugit init
Initialized empty uGit repository

$ ugit add main.c
File 'main .c' added to the staging area

$ ugit commit -m "Initial commit"
[ master ( root - commit ) abcd1234 ] Initial commit
 1 file changed , 10 insertions (+)
 create mode 100644 main .c

$ ugit log
commit abcd1234 ( HEAD -> master )
Author : Your Name < your.email@example.com >
Date : Fri Aug 15 14:05:29 2025 -0300

     Initial commit

$ ugit checkout abcd1234
Switched to commit abcd1234
```

En este ejemplo, μGit gestiona los archivos y versiones de una manera similar a Git, pero utilizando estructuras de datos básicas como arreglos y tablas hash, implementadas en C. Este enfoque permite a los estudiantes aplicar conceptos fundamentales de estructuras de datos en un contexto práctico y relevante.

5 ENTREGA Y EVALUACIÓN

- **Código Fuente:** El código fuente debe estar bien documentado y seguir las normativas de codificación establecidas.

- **Presentación Oral:** Los estudiantes deben presentar su proyecto, destacando las decisiones técnicas tomadas, los desafíos enfrentados y cómo se abordaron.

Este proyecto ayudará a los estudiantes a consolidar su comprensión de las estructuras de datos en C y aplicar estos conceptos a un problema realista y relevante como el control de versiones, mientras refuerzan la importancia de la originalidad en el desarrollo de software.

La tarea es de a tres personas. La versión que se evaluará será la que esté presente en el repositorio designado para entrega el 22 de septiembre a las 23:59 hrs.
Las presentaciones se realizarán a partir de la clase siguiente.
