TEC | Tecnológico de Costa Rica

ESCUELA DE INGENIERÍA EN MECATRÓNICA

MICROPROCESADORES Y MICROCONTROLADORES
MT7003

Grupo 01

PROFESOR: RODOLFO PIEDRA CAMACHO

Tarea 1
GitHub, Pytest y Flake 8

Integrantes:

ANTHONY CONEJO MÉNDEZ
DAVID L. GONZÁLEZ AGÜERO

I semestre 2024

1) **Explique la principal utilidad de git como herramienta de desarrollo de código.**

Git es una herramienta esencial para el control de versiones, que se utiliza ampliamente en el desarrollo de software para rastrear y administrar cambios en el código fuente, aunque también es aplicable a cualquier otro tipo de archivo que requiera seguimiento. Proporciona un registro detallado de las modificaciones, incluyendo la fecha, hora y comentarios descriptivos, lo que permite a los desarrolladores individuales o equipos colaborar de manera más efectiva.
En el contexto del desarrollo de código, la capacidad de Git para llevar un seguimiento meticuloso de cada cambio es muy aprovechable, puesto que los códigos a menudo requieren ajustes y pruebas que implican modificaciones frecuentes. Esto ayuda a identificar eficientemente dónde y por qué algo pudo haber salido mal, y quién estuvo involucrado en esos cambios, optimizando así el proceso de depuración y mejora continua del software.

2) **¿Qué es un branch?**

Un "branch" o rama en Git es una versión independiente del código que permite a los desarrolladores trabajar en nuevas características, correcciones o experimentos en paralelo al desarrollo principal, sin afectarlo. Facilita la colaboración y la organización al permitir múltiples líneas de trabajo que posteriormente pueden ser integradas al proyecto principal mediante una fusión.

3) **En el contexto de GitHub, ¿qué es un *pull request*?**

Un *pull request* es una propuesta para fusionar un conjunto de cambios de una rama a otra. Permite a los colaboradores revisar y discutir el conjunto propuesto de cambios antes de integrarlos en la base de código principal. Los *pull requests* muestran las diferencias entre el contenido en la rama fuente y el contenido en la rama objetivo, facilitando la revisión colaborativa de las modificaciones propuestas

4) **¿Qué es un commit?**

Un commit es una entrada (anotación) en el registro de git de los cambios aplicados a un archivo o conjunto de archivos. Su finalidad es registrar las versiones de los archivos. Además, cada *commit* está acompañado de un mensaje descriptivo proporcionado por el usuario que realizó el *commit*, el cual explica qué cambios se han realizado.

Los *commits* son fundamentales en el control de versiones porque permiten a los desarrolladores rastrear la historia de cambios en el proyecto, revertir a estados anteriores si es necesario, y colaborar con otros desarrolladores al compartir y fusionar cambios. Cada *commit* es identificado por un identificador único (usualmente un hash SHA-1), lo que permite referenciar de manera precisa a cada conjunto de cambios en el proyecto.

5) **Describa lo que sucede al ejectuar la siguiente operación: “git cherry-pick <SHA>”**

El comando git cherry-pick <SHA> en Git se utiliza para aplicar los cambios de un *commit* específico, identificado por su SHA (identificador único), a la rama actual. Esto crea un nuevo *commit* en la rama actual con esos mismos cambios. Es útil para incorporar cambios específicos de una rama a otra sin fusionar todas las modificaciones.

6) Explique qué es un "merge conflict" y cómo lo resolvería.

Un "merge conflict" ocurre en Git cuando dos ramas han realizado cambios incompatibles en el mismo segmento de un archivo y luego se intenta fusionar una rama con otra. Git no puede decidir automáticamente cuál cambio es el que se debe quedar, y, por lo tanto, detiene el proceso de fusión y marca el archivo como conflictivo, requiriendo intervención manual para resolver el conflicto.
Para resolver un conflicto de fusión, se puede utilizar un procedimiento como el de a continuación:

- **Identificar los archivos conflictivos**: Git despliega los archivos que tienen conflictos.

- **Examinar las líneas de código o el contenido conflictivo.**
- **Editar manualmente los archivos,** considerando los cambios que se quieren conservar en el commit.
- **Marcar el conflicto como resuelto,** mediante git add.
- **Confirmar la versión de los cambios realizados mediante *git commit.***

7) ¿Qué es una Prueba Unitaria o Unittest en el contexto de desarrollo de software?

Una Prueba Unitaria es un método de verificación en el desarrollo de software que comprueba que las partes individuales del código, como funciones o métodos, funcionan correctamente. Se enfoca en la unidad más pequeña de código que puede operar de forma independiente para asegurar que se comporta exactamente como se espera. Realizar pruebas unitarias ayuda a detectar errores rápidamente, garantiza que cada componente funcione bien por separado y contribuye a un código más robusto y fácil de mantener.

8) Bajo el contexto de pytest, ¿cuál es la utilidad de un "assert"?

El *assert* es una funcionalidad que revisa la condición booleana de verdadero o falso de la afirmación a la que se le aplica. Si da VERDADERO, se infiere que la afirmación aprobó la prueba. En caso de dar FALSO, la funcionalidad devuelve un valor de "fallido", además de detalles de la causa de la falla.

La utilidad de *assert* en *pytest* es permitir a los desarrolladores escribir casos de prueba de manera concisa y legible, comprobando que los resultados obtenidos en el código coincidan con los resultados esperados.

9) Mencione y explique 3 errores de formato detectables con Flake8

Flake8 es una herramienta de edición de texto de código en Python que puede detectar errores e imperfecciones de estilo. Como ejemplos, se pueden mencionar:

- **Espacios en blanco excesivos entre caracteres**: Tener múltiples espacios en blanco puede dificultar la lectura del código y desorganizar visualmente el texto, lo cual va en contra de las buenas prácticas de escritura de código limpio y legible.

- **Líneas demasiado largas**: Una línea de código que excede un cierto número de caracteres puede ser problemática, ya que no todos los editores de texto o interfaces de usuario pueden mostrarla completamente. Esto puede hacer que el código sea más difícil de leer y seguir, especialmente en entornos con limitaciones de espacio de visualización.

- **Importaciones no utilizadas**: Incluir bibliotecas o módulos que no se utilizan en el código agrega carga innecesaria y puede confundir a otros desarrolladores sobre las dependencias reales del proyecto. Es recomendable eliminar estas importaciones para mantener el código limpio y eficiente.

10) Explique la funcionalidad de parametrización de pytest.

La parametrización en *pytest* es una funcionalidad que permite ejecutar una misma prueba con distintos conjuntos de datos. Esto se hace para comprobar cómo se comporta una función o parte del código bajo diferentes condiciones sin tener que escribir una prueba separada para cada situación.

Para usar la parametrización, se emplea el decorador "***@pytest.mark.parametrize.***" Este decorador necesita dos argumentos: el primero es una cadena de texto con los nombres de los parámetros de la prueba separados por comas, y el segundo es una lista de tuplas, donde cada tupla contiene un conjunto de valores para esos parámetros.

A continuación, se muestra un ejemplo de parametrización:

```python
import pytest

@pytest.mark.parametrize("entrada, esperado", [
    (1, 2),
    (2, 3),
    (3, 4),
])
def test_suma_uno(entrada, esperado):
    assert entrada + 1 == esperado
```

En este ejemplo, la función ***test_suma_uno*** se ejecutará tres veces con diferentes valores para ***entrada*** y ***esperado.*** Esto significa que, en lugar de escribir tres funciones de prueba para cada par de valores, podemos escribir una sola función y probarla con varios datos.