Parte I - Preguntas Teóricas

1) ¿Explique la principal utilidad de git como herramienta de desarrollo de código?

Git es un proyecto de código abierto que considera una herramienta especialmente útil en el desarrollo de código ya que es un sistema de control de versiones de código fuente distribuido (no depende de un servidor central) donde los cambios se guardan de manera incremental y permite tener un registro de los cambios, retroceder a versiones anteriores y añadir funcionalidades.

Git demuestra una capacidad de rendimiento superior a la de otros sistemas. Esto se refleja en la frecuencia de cambios y fusión de versiones anteriores, lo que aumenta la eficiencia del software. Todo esto se almacena en el repositorio y está protegido por diferentes algoritmos para asegurar el código y los cambios realizados; garantizando el funcionamiento sin fallas. Git es flexible en numerosos aspectos, tales como la capacidad de manejar múltiples flujos de desarrollo no lineales y la eficiencia que se logra en cualquier proyecto, grande o pequeño.

2) ¿Qué es un branch?

En Git, una rama o *branch* es una versión del código de un proyecto. Permite trabajar en nuevas funciones o corregir errores sin afectar la rama principal. Trabajar utilizando las ramas de desarrollo de Git ayuda a mantener el orden en el control de versiones y permite manipular el código de forma segura.

3) En el contexto de github. ¿Qué es un Pull Request?

Un *Pull Request* es una funcionalidad de GitHub que permite a los colaboradores proponer cambios en el código. Cuando alguien realiza modificaciones en una rama (o en un fork) y desea que esas mejoras sean integradas en la rama principal del proyecto, abre un *Pull Request*. De esta manera, se genera un espacio de discusión donde se facilita que otros miembros del equipo revisen y comenten la propuesta. Una vez aprobados, los cambios pueden integrarse en la rama destino.

4) ¿Qué es un commit?

Un *commit* en Git es un comando que crea una copia de seguridad de los cambios realizados en un proyecto. Es una de las funciones principales de Git y se considera una práctica fundamental. Este comando captura los cambios preparados en un momento determinado, los guarda y genera una nueva versión.

5) Describa lo que sucede al ejectuar las siguientes operaciones: “git fetch” “git rebase origin/master”

El comando “*git fetch”* se conecta al repositorio remoto y descarga las actualizaciones (como nuevas ramas o cambios en las existentes) sin aplicarlas directamente en el repositorio local. Este comando comprueba si existen actualizaciones disponibles sin transferirlos al git local.

Por otro lado, el comando “*git rebase origin/master*” se utiliza para trasladar los commits de la rama actual a la punta actual de la rama master. En lugar de crear una fusión (*merge*) que integre los cambios, el rebase "reaplica" los commits uno por uno sobre la versión actualizada de master, lo que resulta en un historial lineal y más limpio.

6) Explique que es un "*merge conflict*" o "*rebase conflict*" en el contexto de tratar de hacer *merge* a un *Pull Request* o de completar una operación *git rebase*.

Un "*merge conflict*" o "*rebase conflict*" ocurre cuando Git no puede determinar de forma automática cómo combinar cambios realizados en la misma parte de un archivo desde diferentes ramas o commits. Un "*merge conflict*" puede ocurrir, por ejemplo, cuando dos o más desarrolladores han modificado la misma línea o bloque de código en diferentes ramas, y al momento de hacer *merge* a cada *Pull Request*, estos son incompatibles entre sí. Un desarrollador debe revisar el conflicto, decidir que cambios conservar y resolverlo de forma manual.

Asimismo, puede ocurrir al completar una operación *git rebase* si los commits a rebasar modifican el mismo código que ha cambiado en la nueva base. Al igual que en el *merge*, es necesario resolver el conflicto manualmente para continuar el proceso.

En ambos casos, la resolución implica editar los archivos conflictivos, eliminar las marcas que Git inserta para señalar las diferencias, y luego continuar con el proceso de *merge* o *rebase* una vez que los conflictos han sido solucionados.

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

Un *Unittest* o prueba unitaria es una metodología de testing en el desarrollo de software que se centra en comprobar de forma aislada el funcionamiento correcto de las unidades más pequeñas de código, generalmente funciones o métodos individuales. Estas pruebas permiten detectar errores en etapas tempranas y garantizar que cada parte del programa cumpla con su especificación. Los unittests se pueden ejecutar de forma automática, y funcionan como una forma de documentar el comportamiento esperado de las unidades de código.

8) Bajo el contexto de pytest. ¿Cuál es la utilidad de un "assert"?

En el contexto de pytest, el *assert* se utiliza para verificar que una condición se cumpla durante la ejecución de una prueba. Si la condición evaluada resulta falsa, pytest genera automáticamente un error en el test, indicando la discrepancia entre el resultado esperado y el obtenido. Esto simplifica la validación de resultados, ya que no es necesario escribir código adicional para comparar valores o generar mensajes de error personalizados.

9) ¿Qué es Flake 8?

Flake8 es una herramienta de análisis estático para código Python que se utiliza para detectar errores de sintaxis, problemas de estilo y complejidad en el código. Combina *PyFlakes*, que detecta errores lógicos y problemas de sintaxis; *Pycodestyle*, que revisa el cumplimiento de las convenciones de estilo de Python; y *McCabe*, que calcula la complejidad ciclomática del código para identificar posibles áreas complicadas.

10) Explique la funcionalidad de parametrización de pytest.

La parametrización en pytest permite ejecutar una misma función de test con diferentes conjuntos de datos, lo que facilita la cobertura de múltiples escenarios sin tener que escribir múltiples funciones de prueba. Con esto se evita repetir pruebas muy similares, es posible probar distintos casos de forma sistemática y facilita la identificación de fallos pues cada combinación de datos se ejecuta como un test independiente.