**1. Supongamos que en un repositorio GIT hiciste un commit y olvidaste un archivo. Explica cómo se soluciona si hiciste push, y cómo si aún no hiciste. De ser posible, que quede solo un commit con los cambios.**

**Respuesta a la pregunta :**

1. Si ya hiciste el push y olvidaste un archivo: En este caso, tu commit ya está en el repositorio remoto, pero te falta incluir un archivo. Puedes solucionarlo de la siguiente manera:

   a. Primero, agrega el archivo que faltaba:
   **git add <nombre_del_archivo>**

   b. Luego, crea un nuevo commit con ese archivo:
   **git commit -m "Agrega archivo olvidado"**

   c. Ahora, para fusionar estos dos commits en uno sólo, puedes hacer un "rebase interactivo":
   **git rebase -i HEAD~2**

   Este comando abre un editor de texto con los últimos dos commits. Cambia la palabra "pick" del segundo commit a "squash" (o "s" para abreviar). Esto hará que los dos commits se fusionen en uno solo. d. Guarda y cierra el editor. Git te pedirá que edites el mensaje del commit fusionado. Modifica el mensaje para que refleje todos los cambios. e. Finalmente, haz un forzado del push para reemplazar el historial remoto con tu historial local:

   **git push --force**

2. Si aún no has hecho el push y olvidaste un archivo: En este caso, el commit aún no se ha subido al repositorio remoto, así que es más sencillo solucionarlo: a. Primero, agrega el archivo que faltaba:

   **git add <nombre_del_archivo>**

   b. Luego, modifica el último commit para incluir ese archivo:

   **git commit --amend -m "Commit con todos los cambios"**
   Este comando reemplaza el último commit con uno nuevo que incluye el archivo que faltaba.

   c. Ahora puedes hacer el push sin problemas:

   **git push**

En ambos casos, el resultado final será un solo commit en el repositorio remoto que incluye todos los cambios.

La principal diferencia es que, si ya habías hecho el push, debes usar el rebase interactivo y luego forzar el push para reescribir el historial remoto. En cambio, si aún no habías hecho el push, puedes simplemente modificar el último commit con git commit --amend y luego hacer el push normalmente.

## 2. Si has trabajado con control de versiones ¿Cuáles han sido los flujos con los que has trabajado?

**Respuesta a la pregunta:**

He trabajado con control de versiones utilizando principalmente el flujo de trabajo conocido como **Feature Branches**. En este enfoque, cada desarrollador crea una nueva rama a partir de la rama principal (master o main) para desarrollar una funcionalidad específica o realizar un cambio. Una vez que la nueva funcionalidad está completa y probada, se integra de nuevo a la rama principal mediante un pull request o merge request.

Este flujo es especialmente útil para mantener la estabilidad de la rama principal, ya que permite desarrollar y probar nuevas características en un entorno aislado antes de fusionarlas con el código base. Además, facilita la revisión de código y la colaboración en equipo, ya que cada cambio es revisado antes de ser integrado.

## 3.¿Cuál ha sido la situación más compleja que has tenido con esto?

**Respuesta a la pregunta:**

Una de las situaciones más complejas que he enfrentado fue cuando trabajaba en un proyecto con un equipo grande. Teníamos múltiples ramas de desarrollo activas, algunas de las cuales se habían desviado significativamente de la rama principal. Cuando llegó el momento de integrar todos los cambios, surgieron numerosos conflictos de fusión que tuvimos que resolver manualmente. Algunas de las dificultades que enfrentamos fueron:

- Múltiples desarrolladores realizan cambios en las mismas áreas del código.
- Falta de comunicación y coordinación entre los miembros del equipo.
- Problemas de sincronización entre las diferentes ramas de desarrollo.
- Dificultad para mantener una visión clara del estado general del proyecto.

Para resolver esta situación, implementamos las siguientes estrategias:

- Mejorar la comunicación y la coordinación entre los miembros del equipo.
- Establecer pautas y procesos más claros para la integración de cambios.
- Implementar revisiones de código más rigurosas y pruebas automatizadas.
- Realizar sesiones de resolución de conflictos en grupo para llegar a un consenso.

Después de aplicar estas medidas, logramos estabilizar el proyecto y mejorar la gestión del control de versiones, lo que nos permitió avanzar de manera más efectiva en el desarrollo.

## 4. ¿Qué experiencia has tenido con los microservicios?

**Respuesta cuarta Pregunta:**

He tenido la oportunidad de desarrollar microservicios en Python utilizando tanto Django Rest Framework como Flask, cada uno con sus propias fortalezas.

Django REST Framework ha sido una herramienta fundamental en muchos de mis proyectos. Su capacidad para crear APIs RESTful de manera rápida y eficiente, combinada con las sólidas bases de Django, lo convierte en una elección ideal para microservicios que requieren un alto nivel de estructura y escalabilidad. En particular, he aprovechado:

- **Serializadores:** Para transformar objetos de mi modelo en representaciones JSON fácilmente consumibles por otros servicios.

- **Vistas basadas en clases:** Para organizar mi lógica de negocio de forma clara y modular.

- **Permisos y autenticación:** Para asegurar que solo los usuarios autorizados puedan acceder a mis APIs.

**Aplicaciones en Ciberseguridad**

En el ámbito de la ciberseguridad, he construido microservicios con Flask para tareas como:

- **Técnicas de Web Scraping:** necesarias para hacer reportes automatizados de ciberseguridad
- **Respuesta a incidentes:** Automatizando tareas como la contención de amenazas y la restauración de sistemas.
- **automatización de tareas repetitivas**: llenado automático de base de datos mediante el integración y consumo de plataformas externas

La flexibilidad de Flask me ha permitido crear servicios altamente personalizados y eficientes, adaptados a las necesidades específicas de cada caso.

**Aplicaciones en Plataformas Financieras**
Para desarrollar microservicios en plataformas financieras, he optado por Django Rest Framework debido a su enfoque en la seguridad y su ecosistema maduro. Algunos ejemplos de mis proyectos incluyen:

**Gestión de riesgos:** Evaluando el riesgo crediticio de los clientes y aplicando modelos de scoring.
**Análisis de datos:** Extrayendo y analizando grandes volúmenes de datos financieros para tomar decisiones estratégicas.

**Django REST Framework** ha sido esencial para crear APIs robustas y escalables que cumplen con los estrictos requisitos de seguridad de la industria financiera.

**Beneficios de los microservicios**

a mi me gusta mucho usar micro servicios ya que la arquitectura de microservicios ha aportado numerosos beneficios:

- **Escalabilidad**: Cada microservicio puede escalar de forma independiente, lo que permite hacer frente a picos de demanda de manera eficiente.

- **Resiliencia:** Los fallos en un microservicio no afectan al funcionamiento de otros servicios

- **Despliegue continuo:** Los microservicios pueden desplegarse de forma frecuente y con menor riesgo.
- **Tecnología heterogénea:** Cada microservicio puede utilizar la tecnología más adecuada para su tarea específica.

**Docker: el complemento perfecto por eso me gusta desarrollar en Docker también.**

La combinación de microservicios y Docker ha sido clave para lograr una alta disponibilidad y portabilidad en mis proyectos. Docker me ha permitido:

- Encapsular cada microservicio en un contenedor aislado.
- Simplificar el proceso de despliegue y orquestación.
- Asegurar la consistencia del entorno de ejecución.

Mi experiencia con Django REST Framework y microservicios usando Flask me ha permitido construir sistemas altamente escalables, resilientes y adaptables a las necesidades de los negocios modernos.

## 5. ¿Cuál es tu servicio favorito de GCP o AWS? ¿Por qué?

A mí personalmente, me parecen muy interesantes ambos proveedores de servicios en la nube, tanto GCP como AWS. Cada uno tiene sus fortalezas y se adapta a diferentes necesidades. Creo que la elección ideal depende mucho del proyecto específico y de las prioridades del cliente.

Durante mi trayectoria profesional, he tenido la oportunidad de trabajar en diversos proyectos en la nube, y he visto cómo cada caso requiere un análisis detallado. Por ejemplo, participé en una migración de AWS a GCP para Equifax. **En ese proyecto, la decisión se inclinó hacia GCP principalmente por factores económicos**, ya que lograron negociar condiciones muy favorables. Sin embargo, es importante destacar que el costo no siempre es el factor determinante.

**Al evaluar qué plataforma elegir, considero fundamental analizar los siguientes aspectos:**

- **Costo total de propiedad:** No solo el precio inicial, sino también los costos a largo plazo, considerando el consumo de recursos, las tarifas de transferencia de datos y las posibles escaladas.
- **Escalabilidad:** Es crucial asegurarse de que la plataforma pueda adaptarse a las necesidades cambiantes del proyecto, ya sea creciendo o reduciendo la infraestructura.
- **Facilidad de uso:** La interfaz, la documentación y la comunidad de usuarios son factores clave para agilizar el desarrollo y la gestión de la aplicación.
- **Integración con otras herramientas**: Es importante evaluar cómo se integra la plataforma con otras herramientas que ya estamos utilizando, como bases de datos, sistemas de CI/CD, etc.
- **Regiones:** La disponibilidad de la plataforma en diferentes regiones geográficas puede ser un factor determinante para cumplir con requisitos de latencia o normativas.
- **Seguridad:** Ambos proveedores ofrecen robustas medidas de seguridad, pero es importante comparar las características específicas de cada uno y evaluar cuál se ajusta mejor a las necesidades del proyecto.
- **Características específicas:** Cada plataforma ofrece servicios especializados que pueden ser cruciales para un proyecto en particular. Por ejemplo, si necesitamos realizar análisis de datos a gran escala, GCP podría ser una mejor opción gracias a BigQuery.

**La elección entre GCP y AWS es una decisión estratégica que requiere una evaluación cuidadosa de las necesidades del proyecto y del cliente**. No existe una respuesta única, ya que cada caso es diferente. Lo más importante es realizar un análisis detallado y seleccionar la plataforma que mejor se adapte a las necesidades específicas.