Actividad 2.4: supuesto práctico sobre el versionado en un proyecto

Contexto

Eres el encargado del desarrollo de una aplicación llamada **"Task Manager Pro"** que se utiliza para gestionar tareas y proyectos en empresas. La aplicación está en constante evolución y has lanzado varias versiones con nuevas características y correcciones de errores. Estás utilizando **versionado semántico** para gestionar el proyecto, y se aproxima el lanzamiento de una nueva versión.

La versión actual es **1.2.3**, y el historial de versiones incluye las siguientes características:

- **Versión 1.2.0**: Añadió integración con calendarios externos (compatibilidad hacia atrás).
- **Versión 1.2.2**: Corrigió un problema con la exportación de datos.
- **Versión 1.2.3**: Introdujo un parche para corregir una vulnerabilidad de seguridad.

Ahora estás preparando una nueva versión que incluirá varias actualizaciones importantes:

1. **Nuevas características**: Integración con notificaciones automáticas por correo electrónico.
2. **Cambios que rompen la compatibilidad**: Reestructura completa de la API, lo que requiere a los usuarios ajustar sus integraciones actuales.

Tareas

1. **Preguntas de reflexión:**

¿Qué número de versión deberías asignar a la nueva versión del proyecto después de implementar las nuevas características y la reestructuración de la API? Justifica tu respuesta.
2.0.0 Al implementar cambios que requieren la reestructuración completa de la API, se la considera una versión MAYOR

¿Cómo deberían etiquetarse las versiones preliminares (alpha, beta, etc.) antes del lanzamiento oficial, y en qué momentos deberían utilizarse esas etiquetas?

Las versiones Alpha serían primero (2.0.0-alpha.1, 2.0.0-alpha.2, etc.), las versiones alpha están destinadas al equipo de desarrollo para realizar pruebas iniciales y descubrir errores básicos o problemas de estabilidad
Luego irían las versiones Beta (2.0.0-beta.1, 2.0.0-beta.2, etc.) una vez aprobadas las pruebas alpha, se almplia el gurpo de tester o usuarios para recibir retroalimentacion sobre el rendimiento, la usabilidad y la compatibilidad con otros sistemas.
Luego la version Release Candidate (2.0.0-rc.1, 2.0.0-rc.2, etc.) las RC se lanzan cuando la versión está prácticamente lista para producción, y los cambios se limitan a correcciones de detalles sin añadir nuevas características

2. **Ejercicios prácticos:**

Utiliza Git para simular el lanzamiento de la nueva versión. Realiza lo siguiente:

- Crea una nueva etiqueta para la versión `1.2.3` (la actual).
- Desarrolla la nueva característica de notificaciones automáticas en una rama llamada `feature-notifications`.
- Fusiónala en la rama `main` y actualiza el número de versión para reflejar los cambios que has realizado (incluyendo la reestructuración de la API que rompe la compatibilidad).
- Realiza el etiquetado adecuado para la nueva versión.

git tag -a v1.2.3 -m "Lanzamiento de la versión 1.2.3"
git checkout -b feature-notifications
git add .
git commit -m "Añadida la característica de notificaciones automáticas por correo electrónico"
git checkout main
git merge feature-notifications
git add version.txt
git commit -m "Actualizado el número de versión a 2.0.0 para reflejar cambios importantes"
git tag -a v2.0.0 -m "Lanzamiento de la versión 2.0.0"

Explica cómo manejarías las versiones "hotfix" si, después del lanzamiento de la versión 2.0.0, descubres un problema crítico que necesita corregirse inmediatamente. ¿Cómo gestionas el número de versión en ese caso?

Para corregir el problema critico de inmediato se crea una nueva rama específicamente para eso:

git checkout -b hotfix/2.0.1

Una vez solucionado el problema se agregan los cambios con una descripción acorde y se fusiona con el main:

git add . git commit -m "Corrige un problema crítico en la versión 2.0.0"
git checkout main
git merge hotfix/2.0.1

Se actualiza el numero de la version y el tag:

git add version.txt
git commit -m "Actualizado el número de versión a 2.0.1 para el hotfix"
git tag -a v2.0.1 -m "Lanzamiento de la versión 2.0.1 - Hotfix crítico"