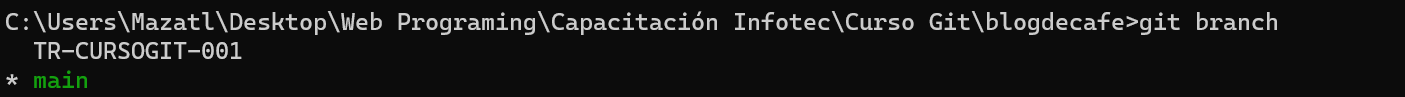
Texto

Descripción generada automáticamenteTexto

Descripción generada automáticamente

Resumen

Especificación de versiones semánticas (SemVer)

Las palabras clave "DEBE", "NO DEBE", "REQUERIDO", "RECOMENDADO", "PUEDE" y "OPCIONAL" en este documento son debe interpretarse como se describe en RFC 2119.El software que utiliza el control de versiones semántico DEBE declarar una API pública. Esta API podría declararse en el propio código o existir estrictamente en la documentación. Independientemente de cómo se haga, DEBE ser preciso y completo.

1. Un número de versión normal DEBE adoptar la forma X.Y.Z, donde X, Y y Z son números enteros no negativos, y NO DEBEN contener ceros a la izquierda. X es la versión principal, Y es la versión secundaria y Z es la versión de parche. Cada elemento DEBE aumentar numéricamente. Por ejemplo: 1.9.0 -> 1.10.0 -> 1.11.0.
2. Una vez que se ha lanzado un paquete versionado, el contenido de esa versión NO DEBE modificarse. Cualquier modificación DEBE publicarse como una nueva versión.
3. La versión principal cero (0.y.z) es para el desarrollo inicial. Cualquier cosa PUEDE cambiar en cualquier momento. La API pública NO DEBE considerarse estable.
4. La versión 1.0.0 define la API pública. La forma en que se incrementa el número de versión después de este lanzamiento depende de esta API pública y de cómo cambia.
5. La versión del parche Z (x.y.Z | x > 0) DEBE incrementarse si solo se introducen correcciones de errores compatibles con versiones anteriores. Una corrección de errores se define como un cambio interno que corrige el comportamiento incorrecto.
6. La versión secundaria Y (x.Y.z | x > 0) DEBE incrementarse si se introduce una funcionalidad nueva compatible con versiones anteriores en la API pública. DEBE incrementarse si alguna funcionalidad de la API pública está marcada como obsoleta. PUEDE incrementarse si se introducen nuevas funcionalidades o mejoras sustanciales en el código privado. PUEDE incluir cambios de nivel de parche. La versión del parche DEBE restablecerse a 0 cuando se incrementa la versión secundaria.

¿Por qué utilizar el control de versiones semántico?

Sin el cumplimiento de algún tipo de especificación formal, los números de versión son esencialmente inútiles para la gestión de dependencias. Al dar un nombre y una definición clara a las ideas anteriores, es fácil comunicar sus intenciones a los usuarios de su software. Una vez que estas intenciones son claras, finalmente se pueden hacer especificaciones de dependencia flexibles (pero no demasiado flexibles).

Un ejemplo simple, considere una biblioteca llamada "Firetruck". Requiere un paquete versionado semánticamente llamado "Ladder". En el momento en que se crea Firetruck, Ladder tiene la versión 3.1.0. Dado que Firetruck usa algunas funciones que se introdujeron por primera vez en 3.1.0, puede especificar con seguridad la dependencia de Ladder como mayor o igual a 3.1.0 pero menor a 4.0.0. Ahora, cuando las versiones 3.1.1 y 3.2.0 de Ladder estén disponibles, puede publicarlas en su sistema de administración de paquetes y saber que serán compatibles con el software dependiente existente.

Como desarrollador responsable, por supuesto, querrá verificar que las actualizaciones de los paquetes funcionen como se anuncia. El mundo real es un lugar desordenado; no hay nada que podamos hacer al respecto, excepto estar atentos. Lo que puede hacer es dejar que Semantic Versioning le brinde una forma sensata de lanzar y actualizar paquetes sin tener que implementar nuevas versiones de paquetes dependientes, lo que le ahorrará tiempo y molestias.