Convención de Commits 1.0.0-beta.3

Resumen

La especificación de Commit Convencionales es una convención liviana que se aplica a los mensajes de los commits. Esta especificación proporciona un conjunto fácil de reglas para crear un historial de commits explícito; Esta convención se ajusta a **SemVer**, al describir las características, correcciones y cambios que rompen la compatibilidad en los mensajes de los commits.

El mensaje del commit debe estar estructurado de la siguiente forma:

```
<tipo>[ámbito opcional]: <descripción>

[cuerpo opcional]

[nota de pie opcional]
```

El commit contiene los siguientes elementos estructurales para comunicar la intención al consumidor de la librería:

1. **fix:** un commit de *tipo* `fix` corrige un error en la base del código (se correlaciona con `PATCH` en el versionado semántico).
2. **feat:** un commit de *tipo* `feat` introduce nuevas características en la base del código (se correlaciona con `MINOR` en el versionado semántico).
3. **BREAKING CHANGE:** un commit que contiene el texto `BREAKING CHANGE:` al inicio de su cuerpo opcional o la sección de nota de pie introduce un cambio que rompe la compatibilidad en el uso de la API (se correlaciona con `MAJOR` en el versionado semántico). Un cambio en el uso de la API puede ser parte de commits de *tipo*. e.g., a `fix:` , `feat:` & `chore:` todos tipos válidos, adicional a cualquier otro *tipo*.
4. Otros: *tipos* de commits distintos a `fix:` y `feat:` están permitidos, por ejemplo **@commitlint/config-conventional** (basado en **the Angular convention**) recomienda `chore:` , `docs:` , `style:` , `refactor:` , `perf:` , `test:` y otros. También recomendamos `improvement` para commits que mejorar una implementación actual sin añadir nuevas características ni corregir errores. Tenga presente que estos tipos no son impuestos por la especificación de commits convencionales y no tienen efecto implícito en el versionado semántico (a menos que incluyan el texto BREAKING CHANGE, lo cual NO es recomendado). Se puede agregar un ámbito al *tipo* de commit para proveer información contextual adicional y se escribe entre paréntesis, e.g., `feat(parser): add ability to parse arrays` .

Ejemplos

Mensaje de commit con descripción y cambio que rompe la compatibilidad en el cuerpo

```
feat: allow provided config object to extend other configs

BREAKING CHANGE: `extends` key in config file is now used for extending other config files
```

Mensaje de commit sin cuerpo

```
docs: correct spelling of CHANGELOG
```

Mensaje de commit con ámbito

```
feat(lang): added polish language
```

Mensaje de commit para una corrección usando un número de problema (opcional)

```
fix: minor typos in code

see the issue for details on the typos fixed

fixes issue #12
```

Especificación

Las palabras “DEBE” (“MUST”), “NO DEBE” (“MUST NOT”), “REQUIERE” (“REQUIRED”), “DEBERÁ” (“SHALL”), “NO DEBERÁ” (“SHALL NOT”), “DEBERÍA” (“SHOULD”), “NO DEBERÍA” (“SHOULD NOT”), “RECOMIENDA” (“RECOMMENDED”), “PUEDE” (“MAY”) Y “OPCIONAL” (“OPTIONAL”) en este documento se deben interpretar como se describe en **RFC 2119**.

1. Los commits DEBEN iniciarse con un tipo que consiste en un sustantivo `feat`, `fix`, etc., seguido de dos puntos y un espacio.
2. El tipo `feat` DEBE ser usado cuando un commit agrega una nueva característica a la aplicación o librería.
3. El tipo `fix` DEBE ser usado cuando el commit representa una corrección a un error en el código de la aplicación.
4. Se PUEDE añadir un ámbito opcional después del tipo. El ámbito es una frase que describe una sección de la base del código encerrada en paréntesis, e.g., `fix(parser):`
5. Una descripción DEBE ir inmediatamente después del tipo/ámbito inicial y es una descripción corta de los cambios realizados en el código, e.g., *fix: array parsing issue when multiple spaces*

were contained in string.

6. Un cuerpo del commit más extenso PUEDE agregarse después de la descripción, dando información contextual adicional acerca de los cambios en el código. El cuerpo DEBE iniciar con una línea en blanco después de la descripción.
7. Una nota de pie PUEDE agregarse tras una línea en blanco después del cuerpo o después de la descripción en caso de que no se haya dado un cuerpo. La nota de pie DEBE contener referencias adicionales a los números de problemas registrados sobre el cambio del código (como el número de problema que corrige, e.g., `Fixes #13`).
8. Los cambios que rompen la compatibilidad con la API DEBEN ser indicados al inicio de la nota de pie o el cuerpo del commit. Un cambio que rompe la compatibilidad con la API DEBE contener el texto en mayúsculas `BREAKING CHANGE`, seguido de dos puntos y espacio.
9. Una descripción se DEBE proveer después de `BREAKING CHANGE:`, describiendo qué ha cambiado en la API, e.g., *BREAKING CHANGE: environment variables now take precedence over config files.*
10. La nota de pie DEBE contener solamente el texto `BREAKING CHANGE`, vínculos externos, referencias a problemas u otra metainformación.
11. Otros tipos distintos a `feat` y `fix` PUEDEN ser usados en los mensajes de los commits.

¿Por qué usar Commits Convencionales?

- Generación automática de CHANGELOGs.
- Determinación automática de los cambios de versión (basado en los tipos de commits).
- Comunicar la naturaleza de los cambios a los demás integrantes del equipo, el público o cualquier otro interesado.
- Ejecutar procesos de ejecución y publicación.
- Hacer más fácil a otras personas contribuir al proyecto, permitiendo explorar una historia de los commits más estructurada.

FAQ

¿Cómo puedo trabajar con los mensajes de los commits en la etapa inicial de desarrollo?

Recomendamos trabajar como si ya hubiera lanzado su producto. Típicamente *alguien*, incluso si son sus compañeros desarrolladores, están usando su producto. Ellos querrán saber qué se ha arreglado, qué se ha dañado, etc.

¿Se deben escribir los tipos de los commits en mayúscula o minúscula?

Cualquiera está bien, pero es mejor ser consistente.

¿Qué debo hacer si un commit encaja en más de un tipo de commit?

Regrese y haga múltiples commits de ser posible. Parte de los beneficios de los Commits Convencionales es la habilidad para hacer commits más organizados y así mismo PRs.

¿No desalienta esto el desarrollo y la iteración rápida?

Desalienta moverse rápido de una forma desorganizada. Ayuda a moverse rápido a largo plazo a través de proyectos con una gran variedad de contribuidores.

¿Pueden los Commits Convencionales llevar a los desarrolladores a limitar el tipo de commits que hacen ya que estarán pensando en los tipos previstos?

Los Commits Convencionales nos animan a hacer más de cierto tipo de commits como *fixes*. Adicionalmente, la flexibilidad de los Commits Convencionales permite a su equipo generar sus propios tipos y cambiarlos a lo largo del tiempo.

¿Cómo se relaciona esto con SemVer?

El tipo de commit `fix` se traduce a un cambio de versión `PATCH`. El tipo de commit `feat` se traduce a un cambio de versión `MINOR`. Commits con el texto `BREAKING CHANGE`, sin importar su tipo, se traducen a un cambio de versión `MAJOR`.

¿Cómo puedo versionar mis extensiones a la especificación de Commits Convencionales, e.g. `@jameswomack/conventional-commit-spec`?

Recomendamos usar SemVer para liberar su propia extensión a esta especificación (¡y lo animamos a hacer esta extensión!)

¿Qué debo hacer si por accidente uso un tipo de commit equivocado?

Cuando utiliza un tipo que es de la especificación pero no es el correcto, e.g. `fix` en lugar de `feat`

Antes de combinar o liberar el error, recomendamos usar `git rebase -i` para editar el historial de los commits. Después de que se ha liberado, la limpieza será distinta de acuerdo con las herramientas y procesos que usted use.

Cuando se usa un tipo que no está en la especificación, e.g. `feet` instead of `feat`

En el peor de los escenarios, no es el fin del mundo si aparece un commit que no cumple con las especificaciones de los commits convencionales. Simplemente, el commit será ignorado por las herramientas que se basen en esta especificación.

¿Deben todos los que contribuyen a mi proyecto usar esta especificación?

¡No! Si usa un flujo de trabajo basado en `squash` los líderes del proyecto pueden limpiar el mensaje en el momento en que se incorpora, sin agregar cargas adicionales a quienes contribuyen casualmente. Un flujo de trabajo común para esto es configurar su sistema de git para que haga el `squash` de manera automática de un pull request y presente al líder del proyecto un formulario para que ingrese el mensaje de commit correcto al momento de hacer el merge.