Conventional Commits

Resumo

A especificação do Conventional Commits é uma convenção simples para utilizar nas mensagens de commit. Ela define um conjunto de regras para criar um histórico de commit explícito, o que facilita a criação de ferramentas automatizadas. Esta convenção segue o **SemVer**, descrevendo os recursos, correções e modificações que quebram a compatibilidade nas mensagens de commit.
A mensagem do commit deve ser estruturada da seguinte forma:

```
<tipo>[escopo opcional]: <descrição>

[corpo opcional]

[rodapé opcional]
```

O commit contém os seguintes elementos estruturais, para comunicar a intenção ao utilizador da sua biblioteca:

1. **fix:** um commit do *tipo* `fix` soluciona um problema na sua base de código (isso se correlaciona com `PATCH` do versionamento semântico).
2. **feat:** um commit do *tipo* `feat` inclui um novo recurso na sua base de código (isso se correlaciona com `MINOR` do versionamento semântico).
3. **BREAKING CHANGE:** um commit que contém o texto `BREAKING CHANGE:`, no começo do texto do corpo opcional ou do rodapé opcional, inclui uma modificação que quebra a compatibilidade da API (isso se correlaciona com `MAJOR` do versionamento semântico). Uma BREAKING CHANGE pode fazer parte de commits de qualquer *tipo*.
4. Outros: *tipos* adicionais são permitidos além de `fix:` e `feat:`, por exemplo **@commitlint/config-conventional** (baseado na **Convenção do Angular**) recomenda-se `chore:`, `docs:`, `style:`, `refactor:`, `perf:`, `test:`, entre outros.

Também recomendamos `improvement` para commits que melhoram uma implementação atual sem adicionar um novo recurso ou consertar um bug.
Observe que esses tipos não são obrigatórios pela especificação do Conventional Commits e não têm efeito implícito no versionamento semântico (a menos que incluam uma BREAKING CHANGE). Um escopo pode ser adicionado ao tipo do commit, para fornecer informações contextuais adicionais

e está contido entre parênteses, por exemplo `feat(parser): adiciona capacidade de interpretar arrays.`

Exemplos

Mensagem de commit com descrição e modificação que quebra a compatibilidade no corpo

```
feat: permitir que o objeto de configuração fornecido estenda outras configurações

BREAKING CHANGE: a chave `extends`, no arquivo de configuração, agora é utilizada
para estender outro arquivo de configuração
```

Mensagem de commit com ! opcional para chamar a atenção para quebra a compatibilidade

```
chore!: remove Node 6 da matriz de testes

BREAKING CHANGE: removendo Node 6 que atinge o final de vida em Abril
```

Mensagem de commit sem corpo

```
docs: ortografia correta de CHANGELOG
```

Mensagem de commit com escopo

```
feat(lang): adiciona tradução para português brasileiro
```

Mensagem de commit de uma correção utilizando número de ticket (opcional)

```
fix: corrige pequenos erros de digitação no código

veja o ticket para detalhes sobre os erros de digitação corrigidos

closes issue #12
```

Especificação

As palavras-chaves “DEVE” (“MUST”), “NÃO DEVE” (“MUST NOT”), “OBRIGATÓRIO” (“REQUIRED”), “DEVERÁ” (“SHALL”), “NÃO DEVERÁ” (“SHALL NOT”), “PODEM” (“SHOULD”), “NÃO PODEM” (“SHOULD NOT”), “RECOMENDADO” (“RECOMMENDED”), “PODE” (“MAY”) e “OPCIONAL” (“OPTIONAL”), nesse documento, devem ser interpretados como descrito na **RFC 2119**.

1. A mensagem de commit DEVE ser prefixado com um tipo, que consiste em um substantivo, `feat`, `fix`, etc., seguido por um escopo OPCIONAL, e OBRIGATÓRIO terminar com dois-pontos e um espaço.
2. O tipo `feat` DEVE ser usado quando um commit adiciona um novo recurso ao seu aplicativo ou biblioteca.
3. O tipo `fix` DEVE ser usado quando um commit representa a correção de um problema em seu aplicativo ou biblioteca.
4. Um escopo PODE ser fornecido após um tipo. Um escopo DEVE consistir em um substantivo que descreve uma seção da base de código entre parênteses, por exemplo, `fix(parser):`

5. Uma descrição DEVE existir depois do espaço após o prefixo tipo/escopo. A descrição é um breve resumo das alterações de código, por exemplo, *fix: problema na interpretação do array quando uma string tem vários espaços*.
6. Um corpo de mensagem de commit mais longo PODE ser fornecido após a descrição curta, fornecendo informações contextuais adicionais sobre as alterações no código. O corpo DEVE começar depois de uma linha em branco após a descrição.
7. Um rodapé de uma ou mais linhas PODE ser fornecido depois de uma linha em branco após o corpo. O rodapé DEVE conter informações adicionais sobre o commit, por exemplo, pull-requests, revisores, modificações que quebram a compatibilidade, com uma informação adicional por linha.
8. A modificação que quebra a compatibilidade DEVE ser indicadas logo no início da seção do corpo ou no início de uma linha na seção de rodapé. Uma modificação que quebra a compatibilidade DEVE consistir de um texto em maiúsculas BREAKING CHANGE, seguido por dois-pontos e um espaço.
9. Uma descrição DEVE ser fornecida após o texto "BREAKING CHANGE:", descrevendo o que mudou na API, por exemplo, *BREAKING CHANGE: as variáveis de ambiente agora têm preferência sobre os arquivos de configuração*.
10. Além de `feat` e `fix`, outro tipo PODE ser usados em suas mensagens de commit.
11. Cada bloco de informação que compõem o commit convencional NÃO DEVE ser tratado como sensível a maiúscula e minúscula pelos implementadores, com exceção de BREAKING CHANGE, que DEVE ser maiúscula.
12. Um `!` PODE ser acrescentado antes do `:` no prefixo tipo/escopo, para chamar a atenção para modificações que quebram a compatibilidade. `BREAKING CHANGE: description` também DEVE ser incluído no corpo ou no rodapé, junto com o `!` no prefixo.

Porque utilizar Conventional Commits

- Criação automatizada de CHANGELOGs.
- Determinar automaticamente um aumento de versionamento semântico (com base nos tipos de commits).
- Comunicar a natureza das mudanças para colegas de equipe, o público e outras partes interessadas.
- Disparar processos de build e deploy.
- Facilitar a contribuição de outras pessoas em seus projetos, permitindo que eles explorem um histórico de commits mais estruturado.

Perguntas Frequentes

Como devo lidar com mensagens de commit na fase inicial de desenvolvimento?

Recomendamos que você proceda como se já tivesse lançado o produto. Normalmente *alguém*, mesmo que seja seus colegas desenvolvedores, está usando seu software. Eles vão querer saber o que foi corrigido, o que quebra etc.

Os tipos no título das mensagens commit são maiúsculos ou minúsculos?

Qualquer opção pode ser usada, mas é melhor ser consistente.

O que eu faço se o commit estiver de acordo com mais de um dos tipos?

Volte e faça vários commits sempre que possível. Parte do benefício do Conventional Commits é a capacidade de nos levar a fazer commits e PRs mais organizados.

Isso não desencoraja o desenvolvimento rápido e a iteração rápida?

Desencoraja a movimentação rápida de forma desorganizada. Ele ajuda você a ser capaz de se mover rapidamente a longo prazo em vários projetos com vários colaboradores.

O Conventional Commits leva os desenvolvedores a limitar o tipo de commits que eles fazem porque estarão pensando nos tipos fornecidos?

O Conventional Commits nos encorajam a fazer mais commits de tipos específicos, por exemplo correções. Além disso, a flexibilidade do Conventional Commits permite que sua equipe crie seus próprios tipos e altere ao longo do tempo.

Qual a relação com o SemVer?

Commits do tipo `fix` devem ser enviados para releases `PATCH`. Commits do tipo `feat` devem ser enviados para releases `MINOR`. Commits com `BREAKING CHANGE` nas mensagens, independentemente do tipo, devem ser enviados para releases `MAJOR`.

Como devo versionar minhas extensões utilizando a especificação do Conventional Commits Specification, e.g. `@jameswomack/conventional-commit-spec`?

Recomendamos utilizar o **SemVer** para liberar suas próprias extensões para esta especificação (e incentivamos você criar essas extensões!)

O que eu faço se acidentalmente usar o tipo errado de commit?

Quando você usou um tipo da especificação, mas não do tipo correto, por exemplo `fix` em vez de `feat`

Antes do merge ou relase com o erro, recomendamos o uso de `git rebase -i` para editar o histórico do commit. Após o release, a limpeza será diferente de acordo com as ferramentas e processos que você utiliza.

Quando você usou um tipo que *não* é da especificação, por exemplo `feet` em vez de `feat`

Na pior das hipóteses, não é o fim do mundo se um commit não atender à especificação do Conventional Commits. Significa apenas que o commit será ignorado por ferramentas baseadas nessa especificação.

Todos os meus colaboradores precisam usar a especificação do Conventional Commits?

Não! Se você usar um workflow de git baseado em squash, os mantenedores poderão limpar as mensagens de commit à medida que forem fazendo novos merges, não adicionando carga de trabalho aos committers casuais. Um workflow comum para isso é fazer com que o git faça squash dos commits automaticamente de um pull request e apresente um formulário para o mantenedor inserir a mensagem do commit apropriada para o merge.