www.conventionalcommits.org /pt-br/v1.0.0/

Conventional Commits 1.0.0

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 baseadas na especificação. Esta convenção se encaixa com 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(s) opcional(is)]
```

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 no rodapé opcional o texto `BREAKING CHANGE:`, ou contém o símbolo `!` depois do tipo/escopo, introduz 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 `build:`, `chore:`, `ci:`, `docs:`, `style:`, `refactor:`, `perf:`, `test:`, entre outros.
5. Outros *rodapés* diferentes de `BREAKING CHANGE: <descrição>` podem ser providos e seguem uma convenção similar ao git trailer format.

Observe que esses tipos adicionais não são exigidos pela especificação do Conventional Commits e não têm efeito implícito no versionamento semântico (a menos que incluam uma `BREAKING CHANGE`). Um escopo pode ser fornecido 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 rodapé

```
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 ! para chamar a atenção para quebra a compatibilidade

```
feat!: envia email para o cliente quando o produto é enviado
```

Mensagem de commit com escopo e ! para chamar a atenção para quebra a compatibilidade

```
feat(api)!: envia email para o cliente quando o produto é enviado
```

Mensagem de commit com ! e BREAKING CHANGE no rodapé

```
chore!: remove suporte para Node 6

BREAKING CHANGE: refatorar para usar recursos do JavaScript não disponíveis no Node 6.
```

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

Revisado por: Daniel Nass
Refs #133
```

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, símbolo 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 corpo de mensagem de commit é livre e PODE consistir em infinitos parágrafos separados por uma nova linha.
8. PODE(M) ser fornecidos um ou mais rodapés, uma linha em branco após o corpo. Cada rodapé DEVE consistir em um token de palavra, seguido por um separador `:<espaço>` ou `<espaço>#`, seguido por um valor de uma string (isso é inspirado pelo git trailer convention).
9. Um token de rodapé DEVE usar `-` no lugar de espaços em branco, por exemplo, `Acked-by` (isso ajuda a diferenciar a seção de rodapé de um corpo de vários parágrafos). Uma exceção é feita para `BREAKING CHANGE`, que PODE também ser usado como um token.
10. O valor de um rodapé PODE conter espaços e novas linhas, e a análise (parsing) DEVE terminar quando o próximo token/separador de rodapé válido for encontrado.
11. BREAKING CHANGES DEVEM ser indicadas após o tipo/escopo de uma mensagem de commit, ou como uma entrada no rodapé.
12. Se incluída como um rodapé, uma alteração de quebra DEVE consistir no texto em maiúsculas QUEBRAR ALTERAÇÃO, seguido por dois pontos, espaço e descrição, por exemplo, *BREAKING CHANGE: as variáveis de ambiente agora têm precedência sobre os arquivos de configuração*.
13. Se incluído no prefixo de tipo/escopo, as `BREAKING CHANGES` DEVEM ser indicadas por um `!` imediatamente antes de `:`. Se o símbolo `!` for usado, `BREAKING CHANGE:` PODE ser omitido da seção de rodapé, e a descrição da mensagem de commit DEVE ser usada para descrever a `BREAKING CHANGE`.
14. Tipos diferentes de `feat` e `fix` PODEM ser usados em suas mensagens de commit, por exemplo, *docs: documentos de referência atualizados*
15. As unidades de informação que compõem o Conventional Commits NÃO DEVEM ser tratadas com distinção entre maiúsculas e minúsculas pelos implementadores, com exceção de `BREAKING CHANGE` que DEVE ser maiúscula.
16. `BREAKING-CHANGE` DEVE ser sinônimo de `BREAKING CHANGE`, quando usado como um token em um rodapé.

Porque utilizar Conventional Commits

- Criação automatizada de CHANGELOGs.
- Determinar automaticamente alterações no 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 melhor estruturado.

Perguntas Frequentes

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

Recomendamos que você prossiga 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, novas features, breaking changes 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 de forma mais organizada.

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 múltiplos 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 release 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.

Como o Conventional Commits trata os commits de reversão?

Reverter código pode ser complicado: você está revertendo vários commits? se você reverter um recurso, a próxima versão deve ser um patch?

O Conventional Commits não o força a definir um comportamento de reversão. Em vez disso, deixamos isso para que os autores de ferramentas usem a flexibilidade de *tipos* e *rodapés* para desenvolver sua própria lógica para lidar com reversões.

Uma recomendação é usar o tipo `revert` e um rodapé que referencia os SHAs de commit que estão sendo revertidos:

```
revert: nunca mais falaremos do incidente do miojo

Refs: 676104e, a215868
```