**PADRÕES DE COMMITS**

**O que é um commit?**

Um commit é uma ação de salvar alterações feitas no código em um sistema de controle de versão (como o Git). Cada commit registra o que foi alterado e por que. Ele funciona como um “checkpoint” no histórico do projeto.

**O que são commits semânticos?**

Os commits semânticos são uma convenção de mensagens de commit que define um padrão para escrever descrições claras e organizadas. Isso ajuda tanto os desenvolvedores quanto ferramentas automatizadas a entender rapidamente o tipo de alteração realizada.

De acordo com a documentação do Conventional Commits, os commits semânticos seguem uma convenção simples que orienta a escrita das mensagens de commit de forma padronizada. Essa prática permite criar um histórico de alterações claro e organizado, facilitando tanto o entendimento do que foi modificado quanto a utilização de ferramentas automatizadas para análise e deploy do código.

A principal vantagem desse padrão é que ele ajuda você e sua equipe a compreender rapidamente quais mudanças foram realizadas em cada commit. Em vez de mensagens genéricas, cada commit possui uma palavra-chave e, opcionalmente, um emoji, que indicam de forma direta o tipo de alteração feita no código.

**Por exemplo:**

**Alterações de código**: modificações na lógica ou no comportamento do programa.

**Atualização de pacotes ou dependências**: quando bibliotecas ou ferramentas são adicionadas ou atualizadas.

**Documentação**: mudanças em arquivos de documentação, comentários ou READMEs.

**Alterações visuais ou de interface**: ajustes na aparência do sistema ou layout.

**Testes**: criação, modificação ou correção de testes automatizados.

Ao utilizar esta convenção, cada commit se torna auto explicativo, permitindo que qualquer pessoa que veja o histórico do projeto saiba imediatamente o propósito de cada alteração, sem precisar investigar o código linha por linha.

Em resumo, seguir padrões de commits semânticos aumenta a clareza, facilita a colaboração e contribui para a manutenção eficiente do projeto a longo prazo.

* **Clareza**: qualquer pessoa que leia o histórico consegue entender rapidamente o que foi feito.
* **Organização**: facilita a identificação de tipos de alterações sem precisar abrir o código.
* **Automação**: ferramentas de integração contínua (CI/CD) podem usar essas informações para gerar versões do software ou relatórios automaticamente.
* **Colaboração**: em equipes grandes, commits claros evitam confusão e melhoram a comunicação.

**Tipo e descrição**

O commit semântico possui elementos estruturais chamados tipos, que indicam a intenção da alteração realizada no código. Cada tipo de commit ajuda o time a entender rapidamente o propósito de cada mudança.

**Tipos de commits**

**feat** – Indica que o commit adiciona um novo recurso (feature) ao projeto.

Está relacionado ao MINOR do versionamento semântico, ou seja, mudanças que adicionam funcionalidades sem quebrar compatibilidade existente.

**fix** – Indica que o commit corrige um problema ou bug no código.

Está relacionado ao PATCH do versionamento semântico, usado para correções que não adicionam novas funcionalidades.

**docs** – Indica alterações somente na documentação, como no README ou outros arquivos de documentação.

Não inclui alterações no código.

**test** – Indica alterações em testes, seja criando, modificando ou removendo testes unitários.

Não inclui alterações no código funcional.

**build** – Indica mudanças em arquivos de build ou dependências do projeto, como scripts de compilação ou pacotes.

**perf** – Indica alterações relacionadas a performance do código, como otimizações que tornam o sistema mais rápido ou eficiente.

**style** – Indica alterações de formatação do código, como ajustes de identação, pontos e vírgulas, trailing spaces ou lint.

Não inclui mudanças na lógica do código.

**refactor** – Indica refatorações de código, ou seja, melhorias na estrutura ou organização do código sem alterar sua funcionalidade.

Pode incluir mudanças de layout, reorganização de funções ou otimizações de performance.

**chore** – Indica tarefas administrativas ou de manutenção do projeto, como atualizar pacotes, configurar o gitignore ou alterar arquivos de configuração.

Não inclui alterações no código funcional.

**ci** – Indica alterações relacionadas à integração contínua (Continuous Integration), como ajustes em pipelines, scripts de teste ou deploy automático.

**raw** – Indica alterações em arquivos de configuração, dados ou parâmetros do sistema, incluindo features experimentais ou ajustes de ambiente.

**cleanup** – Indica a remoção de código desnecessário, como trechos comentados, arquivos obsoletos ou redundantes, melhorando a legibilidade e manutenção do projeto.

**remove** – Indica a exclusão de arquivos, diretórios ou funcionalidades obsoletas, ajudando a reduzir a complexidade e manter o projeto organizado.

**Padrão de Commits – SquadTech**

Para manter um histórico de commits claro, organizado e padronizado, todos os membros do grupo SquadTech devem seguir a convenção abaixo. Não utilizaremos emojis nos commits, mas todos devem ser escritos de forma semântica e padronizada, facilitando o entendimento das alterações por toda a equipe.

Estrutura obrigatória do commit:

**<tipo>(ID da User Story)**

**Descrição curta e objetiva da alteração**

**<tipo>:** indica o tipo de alteração realizada. Deve ser um dos tipos definidos pelo padrão de commits semânticos (sem emojis).

**ID da User Story:** referência ao código da User Story ou tarefa relacionada no backlog do projeto.

**Descrição curta e objetiva**: uma frase resumida explicando claramente a alteração realizada.

feat(US102) - Adiciona página de login com validação de email

docs(US110) - Atualiza README com instruções de instalação

chore(US120) - Atualiza dependências do projeto e arquivo gitignore

**Boas Práticas de Commits**

**Um commit = uma alteração**

Cada commit deve corresponder a uma única mudança ou objetivo no projeto.

**Exceção para alterações relacionadas**

Se duas mudanças pequenas estiverem diretamente relacionadas e não puderem ser separadas:

Indique ambos os tipos de alteração no título do commit.

Explique cada alteração detalhadamente no corpo do commit.

**Exemplo:**

feat/fix(US205)

Descrição detalhada:

- feat: Implementada a funcionalidade de filtro por categoria nos resultados de busca.

- fix: Corrige bug que impedia a pesquisa por palavras-chave contendo acentos.

Use sempre o tempo imperativo:

“Adiciona”, “Corrige”, “Remove”, “Atualiza”, “Refatora”, etc.

Inclua sempre a referência à User Story:

Facilita rastreabilidade, revisão e manutenção do histórico do projeto.

**OBS: Commits fora do backlog**

Quando um commit não estiver diretamente relacionado a nenhuma tarefa da backlog, deverá ser utilizado o código de tipo 00. Nesse caso, é obrigatório apresentar uma descrição detalhada do que foi feito e em qual parte do projeto a alteração foi realizada.

Esse padrão garante rastreabilidade e facilita a compreensão de mudanças que, embora não vinculadas a uma tarefa planejada, impactam diretamente o código ou a documentação.

