



# Git: Criação de Branches, Pull Requests e Commits

## Contexto

Neste guia, abordaremos os padrões e boas práticas recomendadas para a criação de branches, commits e Pull Requests (PRs) no Git. Este documento visa padronizar e otimizar o processo de desenvolvimento, garantindo clareza e consistência nas alterações de código, o que facilita a revisão e a colaboração entre os membros do time. Serão detalhados os tipos de elementos estruturantes que classificam as alterações de código, bem como as convenções para nomeação de branches e a estrutura de commits.

## Escopo e não escopo

<b>Dentro do Escopo:</b>	<ul style="list-style-type: none"><li>• boas práticas de <b>escrita de commits</b>;</li><li>• boas práticas de <b>nomenclatura de branches</b>;</li><li>• estabelecer a criação de <b>commits e PR atômicos no dbt</b></li></ul>
<b>Fora do Escopo:</b>	<ul style="list-style-type: none"><li>• detalhamento de ferramentas utilizadas</li><li>• como criar uma branch</li><li>• como criar um commit</li><li>• como criar um Pull Request (PR)</li></ul>

 Para conhecer mais detalhes sobre Git e Github, acesse esta página  [Git/Github](#)

## Tipos de elementos estruturantes

Elementos estruturantes classificam o tipo de tarefa ou alteração proposta para revisão em um Pull Request. Podemos aplicar esses elementos na nomenclatura de branches, título de PRs, e descrições de commits.

O padrão de elementos estruturantes que propomos seguir são os seguintes prefixos:

**FEAT** - Nova funcionalidade/feature que seja visível para o usuário. Por exemplo: um novo campo de uma tabela, nova tabela, um novo model no dbt ou view etc.

**CHORE** - Qualquer alteração que adiciona coisas no código, mas não alteram código tangível pelo o usuário. Por exemplo: separar ou adicionar blocos de código SQL em subquery ou CTEs etc.

**FIX** - Conserto de código/funcionalidade quebrada;

**REFACT** - Reorganizar código, sem alteração de funcionalidade que seja visível para o usuário final. Por exemplo: alterar nome de objetos ou models, alterar nomes de variáveis e parâmetros etc.

**STYLE** - Alteração de estilo de código. Por exemplo: remover espaço em branco, editar formatação em bloco de código etc.

**DOC** - Alteração ou adição de documentação em descrição de tabelas e colunas ou comentários para documentar o código.

## Padrões de Nomenclatura de Branches

Recomenda-se a seguinte abordagem para criação de branches, nesta ordem de prioridade e **em inglês**:

1. Prefixo por task do Jira (e.g. AE-555), assim temos integrado à task o status do PR (tendo como premissa a integração do Jira e Github habilitada);
2. Na ausência de uma task, utilizar o mesmo prefixo da alteração (refact, chore, feat, etc)

Prefixo por task do Jira (e.g. AE-555):

```
PREFIX0-JIRA/coloque-descricao-aqui
```

Exemplos:

✓ AE-555/feat-add-new-model-tickets-arcotech

✗ AE-155-add-new-model-tickets-arcotech

✗ AE-155

✗ novo-modelo-esteira-pedidos-coc

Prefixo por Tipo de Task (Badge):

```
PREFIX0-TIP0/coloque-descricao-aqui
```

Exemplos:

✓ FEAT/add-new-model-tickets-arcotech

✗ feat-add-new-model-tickets-arcotech

✗ add-new-model-tickets-arcotech

## 💡 Motivação para Padrões de Nomenclatura de Branches [🔗](#)

Manter uma nomenclatura consistente para branches ajuda a organizar o fluxo de trabalho e facilita o rastreamento do progresso em relação a tarefas específicas. Quando as branches são nomeadas de forma clara e padronizada, é mais fácil entender o contexto e o objetivo de cada branch, além de simplificar a integração com sistemas de gerenciamento de projetos, como o Jira. Isso assegura uma navegação mais intuitiva e eficiente dentro do repositório, promovendo uma colaboração mais fluida e menos sujeita a erros.

## Padrões de Escrita de Commits [🔗](#)

As boas práticas de commits seguem esta lógica básica, **em inglês**:

```
tipo_de_prefixo: <verbo no infinitivo> <contexto da alteração>
```

Exemplo de commit no dbt:

✓ feat: add new tickets model from arcotech domain

✗ added tickets model

✗ feat: added new tickets model from arcotech domain

### 📘 E por que verbo no infinitivo?

Tenha em mente a frase ***If applied, this commit will..***

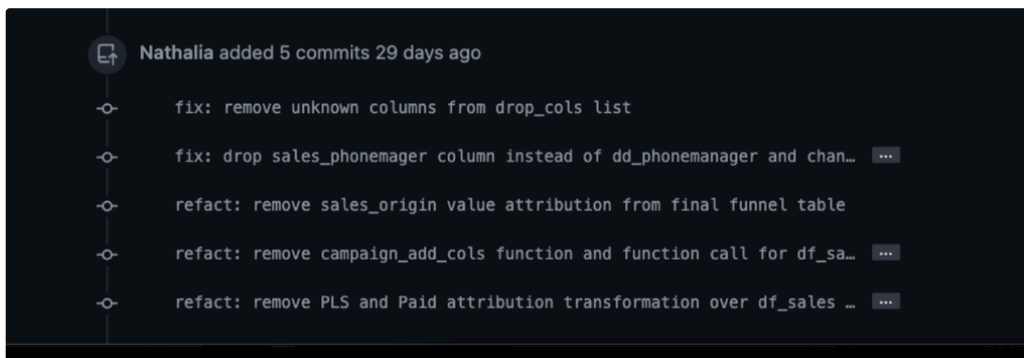
Exemplo: *If applied, this commit will* **add PLS/non-PLS dimension in calls view**

[Este artigo](#) fala mais sobre isso.

## 💡 Motivação para Padrões de Commits [🔗](#)

Utilizar verbos no infinitivo e criar commits atômicos são práticas que promovem clareza e consistência. Essas práticas ajudam a equipe a entender rapidamente a intenção por trás de cada mudança, facilitam o processo de revisão e tornam mais simples

reverter alterações específicas sem afetar outras funcionalidades.



Exemplo de sequência de commits atômicos abertos em um mesmo PR.

## Padrões de Escrita de PRs [↗](#)

Os padrões de escrita de PR seguem este padrão, **em português**:

1. Para título de PRs:

```
tipo_de_prefixo: <verbo no infinitivo> <contexto da alteração>
```

Exemplo:

✗ novo campo xyz na tabela abc do modelo 123

✓ feat: adicionar campo xyz na tabela abc do modelo 123

2. Para descritivo de PRs, seguir o template designado por Data Platform Engineering.

### 🔗 Motivação para Padrões de Escrita de PRs [↗](#)

A adoção de padrões de escrita para Pull Requests (PRs) é crucial para assegurar uma comunicação clara e eficiente dentro da equipe. Um título bem estruturado, utilizando prefixos como FEAT ou FIX, oferece uma visão rápida e precisa sobre a natureza das alterações propostas. Além disso, a descrição detalhada do PR fornece o contexto necessário para os revisores entenderem as mudanças, seus objetivos e quaisquer impactos potenciais. Isso não apenas facilita o processo de revisão, mas também documenta as decisões de design e implementação, servindo como uma referência útil para futuras manutenções e melhorias no código.

## Referências [↗](#)

1. [👁️ How to Write a Git Commit Message](#)
2. [📄 Conventional Commits](#)