BOAS PRÁTICAS PARA ESTRUTURAR O PROJETO COM COMMITS

Passo 1 - Nome da Branch

Sempre que for modificar algum código ou criar algum código, o ideal é sempre separar por branch, assim fica mais fácil entender o que foi feito antes de atualizar no projeto principal. Eu gosto de usar as iniciais do projeto e uma sequência numérica, por exemplo sistema-hotel-pet ficaria (SHP-1).

```
# Padrão:
<id-da-sua-tarefa>/<super-resumo-da-feature>

#Exemplo de criação:
SHP-1/criar-projeto
```

Se precisa verificar a branch é só dar o comando abaixo

```
# Exemplo:
git checkout -b TL-100/create-post-api
```

Isso é uma boa estrutura, pois se usar o trello ou o jira, conseguimos ter os IDs das tarefas relacionados com a branch, fazendo com que o projeto seja fácil de entender por qualquer desenvolvedor.

Passo 2 - Utilizar Padrões de Commit

Manter um padrão nos commits ajuda e muito tem uma boa organização das tarefas, não adianta criar uma branch bem escrita e criar commits aleatórios dentro dela, então manter o padrão ajuda e muito na hora de entender o que foi feito. Gosto de abordar os conceitos usados pelo Angular pois conseguimos dividir em **"tipo(escopo): descrição".** Temos uma convenção como os detalhes abaixo:

- **feat:** Um novo recurso para a aplicação, e não precisa ser algo grande, mas apenas algo que não existia antes e que a pessoa final irá acessar.
- **fix:** Correções de bugs
- **docs:** Alterações em arquivos relacionados à documentação
- **style:** Alterações de estilização, formatação etc

- **refactor:** Um código de refatoração, ou seja, que foi alterado, que tem uma mudança transparente para o usuário final, porém uma mudança real para a aplicação
- **perf:** Alterações relacionadas à performance
- **test:** Criação ou modificação de testes
- **chore:** Alterações em arquivos de configuração, build, distribuição, CI, ou qualquer outra coisa que não envolva diretamente o código da aplicação para o usuário final.

```
# Exemplo

feat(post): criar nova integracao com a API

ou

feat : criar nova integracao POST com a API

test: adicionar testes a nova inregracao
```

Como podemos notar no exemplo iniciamos com o tipo seguindo com o escopo ou não e depois uma descrição da funcionalidade.

Passo 3 - Padrão de Título na Pull Request

Na pull request podemos unir a convenção da branch com o commit, assim criaremos um contexto bem explicado dá nova alteração.

```
# Padrão:
[<id-da-sua-tarefa>] tipo(escopo): descrição

# Exemplo:
[SHP-1] feat(post): criar nova integracao com a API
```

Além do título, devemos também incluir na pull request sempre uma breve descrição e alguns checklist do que foi feito.

segue um exemplo de descrição detalhada sobre a pull

```
## Tipo de modificações
```

```
## Descrição

## Screenshots ou Prints se tiver

## Links das tarefas

## Checklist

## Dependências usadas nessa etapa
```

Explicando:

- **Item 1:** Tipo da alteração, se é bug fix, feature, chore ou uma release (para o caso de fazer releases com alguma branch que não é a "main")
- **Item 2:** Descrição com mais detalhes, principalmente se esse recurso altera pontos fundamentais do sistema. Pontos esses que nós precisaremos lembrar no futuro, uma vez que não podemos confiar em nossas memórias
- **Item 3:** Se for possível e fizer sentido, capturas de telas que explicam melhor o recurso
- **Item 4:** Os links para as tarefas na aplicação que gerencia as estórias e tarefas
- **Item 5:** Checklist básico para subir uma PR:
    - Menos de 400 linhas
    - Revisão no próprio código antes de abrir PR
    - Todos os testes existentes passaram
    - Comentários em lugares necessários foram escritos
    - Criação de testes para o novo recurso
- **Item 6:** Outras pull requests que são dependentes, por exemplo: alguma pull request com uma API do backend que é necessária ser entregue antes de subir uma tela no frontend.

Fonte de onde peguei esse estilo de padrão e no qual achei bem bacana:

https://www.tabnews.com.br/guscsales/uma-maneira-de-organizar-suas-branches-commits-e-pull-requests