Documentação de Fluxo de Branches e Padrão de Commits

Objetivo

Este documento descreve o fluxo de branches e o padrão de commits que deve ser seguido em um projeto usando **Git** para controle de versão. Este processo é utilizado em um monorepo com frontend e backend separados, e um ambiente Docker configurado para desenvolvimento e produção.

Estrutura de Branches

O fluxo de branches visa garantir um desenvolvimento organizado, minimizando conflitos e garantindo a qualidade do código. Abaixo está a descrição das principais branches utilizadas no projeto:

1. `main` (ou `master`)

- **Objetivo**: Contém o código **estável**, pronto para produção.
- **Regras**:
  - Somente código que passou por testes e revisão de código deve ser merged nesta branch.
  - Nunca desenvolva diretamente na `main`.

2. `dev`

- **Objetivo**: Branch de **integração**. Serve para consolidar as mudanças das branches de funcionalidades antes de serem lançadas em produção.
- **Regras**:
  - Todas as novas funcionalidades e correções devem ser integradas aqui antes de serem testadas e movidas para produção.

3. `feature/[nome-da-funcionalidade]`

- **Objetivo**: Branch para **desenvolver novas funcionalidades** ou alterações no projeto.
- **Regras**:
  - Criada a partir da `dev` e deve ser mergeada de volta na `dev` quando a funcionalidade estiver pronta.
  - Exemplo de nome: `feature/frontend-nova-pagina` ou `feature/backend-nova-api`.

4. `hotfix/[descricao-do-hotfix]`

- **Objetivo**: Para correções de **bugs críticos** que precisam ser feitas diretamente na `main`.
- **Regras**:
  - Criada a partir da `main` para corrigir erros que estão impactando a produção.
  - Após a correção, o hotfix deve ser merged tanto na `main` quanto na `dev`.

5. ***`release/[versao]`***

- **Objetivo**: Preparar o código para a próxima versão de produção.
- **Regras**:
    - Criada a partir da `dev` quando a versão está pronta para ser testada e validada antes de ser lançada.
    - A branch de release é onde são feitos os ajustes finais e as correções de bugs.
    - Exemplo de nome: `release/v1.0.0`.

Fluxo de Trabalho

O fluxo de trabalho aqui descrito é projetado para facilitar o desenvolvimento colaborativo e o gerenciamento de mudanças no código:

1. **Criação de Branches de Funcionalidade**
    - Quando você ou outro membro da equipe começar a trabalhar em uma nova funcionalidade, crie uma branch específica a partir da `dev`.

1.
```
git checkout dev
git checkout -b feature/frontend-nova-pagina
```
2. ou
3.
```
git checkout dev
git checkout -b feature/backend-nova-api
```

4. **Desenvolvimento Independente**
    - Trabalhe nas suas funcionalidades nas branches específicas.
    - Durante o desenvolvimento, crie **commits pequenos** e frequentes para facilitar o controle das mudanças.

1. **Realizando Pull Requests (PRs)**
    - Quando uma funcionalidade estiver pronta, crie um **pull request** para mergeá-la na branch `dev`.
    - Certifique-se de que todas as dependências (frontend e backend) funcionem bem no ambiente de desenvolvimento antes de solicitar a revisão de código.

1. **Revisão de Código**
    - As mudanças devem ser revisadas antes de serem integradas à `dev`. Isso garante que o código esteja bem estruturado e sem erros graves.

1. **Merge na Branch `dev`**
    - Após aprovação, a branch de funcionalidade será mergeada na `dev`.

1. **Testes de Integração**
    - Teste as funcionalidades de forma integrada, garantindo que as partes frontend e backend funcionem corretamente.
    - Utilize **Docker** para rodar os containers de ambos os serviços localmente.

1. **Preparação para Produção (Release)**
    - Quando a branch `dev` estiver pronta para a produção, crie uma branch de **release** a partir dela para ajustes finais.

1. `git checkout dev`
   `git checkout -b release/v1.0.0`
   - Realize testes finais e correções de bugs antes de fazer o merge na `main`.

1. **Merge para `main`**
   - Após a validação da versão, faça o merge da branch de **release** na `main` e marque a versão como `v1.0.0`.

1. **Hotfixes**
   - Se um bug crítico for encontrado na produção, crie uma branch **hotfix** a partir da `main`, faça a correção, e depois faça o merge tanto na `main` quanto na `dev`.

1. `git checkout main`
   `git checkout -b hotfix/corrigir-bug`

Padrão de Commits

Um padrão de commits claro é essencial para garantir que o histórico de mudanças do código seja legível e útil. Siga este padrão para criar commits organizados e compreensíveis.

Formato de Commit:

Use o seguinte formato para mensagens de commit:

```
<tipo>(<escopo>): <mensagem>
```

Onde:

- **`tipo`**: Tipo da mudança que está sendo feita.
- **`escopo`**: Área afetada pela mudança (pode ser opcional em alguns casos).
- **`mensagem`**: Descrição breve e objetiva da mudança.

Tipos Comuns de Commit:

- **`feat`**: Nova funcionalidade (feature).
- **`fix`**: Correção de bugs.
- **`chore`**: Tarefas gerais de manutenção (por exemplo, ajustes no Dockerfile ou configuração).
- **`docs`**: Mudanças em documentação (ex: README, comentários no código).
- **`style`**: Mudanças que não afetam a lógica do código, como formatação e espaçamento.
- **`refactor`**: Refatoração do código, ou seja, mudanças no código que não alteram o comportamento, mas melhoram a estrutura.
- **`test`**: Mudanças relacionadas a testes (ex: adicionar ou corrigir testes).

Exemplos de Mensagens de Commit:

- `feat(frontend): adicionar página de login`
- `fix(backend): corrigir erro ao salvar no banco de dados`
- `chore(docker): atualizar docker-compose.yml para a versão mais recente`

Boas Práticas de Commit:

- Faça **commits pequenos** e **frequentes** para cada mudança lógica.
- Escreva mensagens de commit claras e objetivas, de modo que qualquer pessoa possa entender rapidamente o que foi alterado.
- Evite commits grandes e com muitas mudanças ao mesmo tempo.

Exemplo de Fluxo Completo

1. Você cria uma branch para uma funcionalidade no frontend:
2. ```
   git checkout dev
   git checkout -b feature/frontend-nova-pagina
   ```
3. Faz alterações no código do frontend e realiza commits com mensagens claras:
4. ```
   git commit -m "feat(frontend): adicionar página de login"
   git commit -m "fix(frontend): corrigir bug no formulário de login"
   ```
5. Quando a funcionalidade estiver pronta, cria um pull request para a `dev`.
6. Após aprovação e merge, crie uma branch de release:
7. ```
   git checkout dev
   git checkout -b release/v1.0.0
   ```
8. Realiza testes finais e faz o merge na `main` após garantir que tudo está funcionando corretamente.

Conclusão

Este fluxo de trabalho e padrão de commits ajuda a garantir que o desenvolvimento no monorepo seja organizado, com uma boa separação de funcionalidades, e com um histórico claro de mudanças. Seguindo estas práticas, o processo de integração entre o frontend, backend e o ambiente Docker se torna mais simples e menos propenso a erros.
   ```