Documentação da Estratégia de Branches - Equipe Chronos

1. Objetivo

O objetivo desta estratégia é **organizar o fluxo de desenvolvimento** de modo que cada nova **funcionalidade**, **correção de bug** ou **melhoria** do projeto seja desenvolvida em uma **branch isolada**, garantindo:

- Maior **controle de versão** e **segurança**;
- **Facilidade na revisão de código** (*code review*);
- **Redução de conflitos** no merge;
- Histórico mais **limpo e rastreável** de mudanças.

---

2. Estrutura de Branches

Branches principais

Branch	Função	Regras
`main`	Contém o código **estável** e **em produção**.	Somente merges aprovados e testados.
`dev`	Ambiente de **integração** de novas features antes do release.	Base para dar merge em branches de feature já concluídas.

---

3. Fluxo de Trabalho (Workflow)

O fluxo de desenvolvimento segue 6 etapas principais.

Etapa 1: Criar uma nova branch de funcionalidade

Sempre que for iniciar uma nova tarefa (funcionalidade, ajuste, etc.), crie uma nova branch a partir da branch `dev`.

Comando:

```
git checkout develop
git pull origin develop
git checkout -b feature/nome-da-funcionalidade
```

Etapa 2: Desenvolver a funcionalidade

Dentro da branch criada, implemente o código referente à nova funcionalidade.
Faça *commits* frequentes e bem descritos.

Exemplo de commit:

```
git add .
git commit -m "feat: adiciona tela de login com validação de credenciais"
```

Convenção de commits (recomendada):

- `feat:` → nova funcionalidade
- `fix:` → correção de bug
- `refactor:` → melhoria de código (sem alterar funcionalidade)
- `docs:` → atualização de documentação
- `style:` → alterações de formatação
- `test:` → inclusão ou ajuste de testes
- `chore:` → tarefas de build, CI/CD, etc.

Etapa 3: Atualizar a branch antes do merge

Antes de abrir um *Pull Request (PR)*, atualize sua branch com a versão mais recente da `dev` para evitar conflitos:

```
git checkout develop
git pull origin develop
git checkout feature/nome-da-funcionalidade
git merge develop
```

Se houver conflitos, resolva-os e confirme o merge localmente.

Etapa 4: Criar um Pull Request

Crie um **Pull Request** da sua branch (`/nome_da_funcionalidade`) para a branch `dev`.

No GitHub:

1. Vá até a aba **Pull requests**;
2. Clique em **New Pull Request**;
3. Compare sua branch (`/nome_da_funcionalidade`) com `dev`;
4. Preencha o título e descrição do PR (explique o que foi feito);
5. Solicite a revisão de outro desenvolvedor.

Etapa 5: Revisão e Aprovação

Durante a revisão:

- Outro desenvolvedor revisa o código;
- Caso necessário, são feitos comentários ou ajustes;
- Após aprovação e testes bem-sucedidos, o merge é realizado na `dev`.

Etapa 6: Merge e Deploy

➢ Para ambientes de homologação:

- A branch `dev` é usada para deploy de testes (staging/homolog).

➢ Para produção:
Quando tudo estiver testado e aprovado, é criada uma branch de release:

```
 git checkout develop
git checkout -b release/1.0.0
```

- Após os testes finais, o merge é feito em:
    - `main` → código pronto para produção;
    - `dev` → para manter o histórico sincronizado.

---

4. Boas Práticas

- **1 branch = 1 funcionalidade**
  → Evite misturar escopos diferentes na mesma branch.
- **Sempre derive de `dev`** (nunca de `main`).
- **Mantenha commits pequenos e claros.**
- **Evite merges diretos na `main`.**
- **Revise o código de outros colegas** regularmente.

Apague branches antigas após o merge:

```
 git branch -d feature/nome-da-funcionalidade
git push origin --delete feature/nome-da-funcionalidade
```

---

5. Benefícios da Estratégia

✅ Isolamento de código: cada feature é desenvolvida sem afetar o restante do projeto.
✅ Controle e rastreabilidade: commits e merges bem documentados.
✅ Colaboração eficiente: revisões claras e controle de qualidade.
✅ Fluxo seguro até a produção: evita deploys acidentais de código instável.

---

6. Resumo Final

Etapa	Ação	Branch Base	Resultado
Criar funcionalidade	`git checkout -b nome_da_funcionalidade`	`dev`	Cria nova branch isolada
Desenvolver	commits descritivos	feature branch	Código implementado
Atualizar branch	`merge dev`	develop	Evita conflitos
Revisar e aprovar	Pull Request	develop	Validação por pares
Publicar	merge para main	release	Deploy de produção