1. Criando uma Nova Branch

Branches são usadas para desenvolver funcionalidades isoladas umas das outras. Para criar uma nova branch:

`git checkout -b nome-da-branch`

Substitua `nome-da-branch` pelo nome descritivo da sua nova funcionalidade ou correção.

1.1 Convenções para Nomes de Branches

Use nomes descritivos e siga um padrão para facilitar a organização. Aqui estão algumas convenções comuns:

- `feature/nome-da-feature`: Para novas funcionalidades.
  - Exemplo: `feature/adicionar-login`
- `fix/nome-da-correcao`: Para correções de bugs.
  - Exemplo: `fix/corrigir-erro-login`
- `hotfix/nome-do-hotfix`: Para correções urgentes em produção.
  - Exemplo: `hotfix/corrigir-falha-seguranca`
- `docs/nome-da-documentacao`: Para atualizações na documentação.
  - Exemplo: `docs/atualizar-readme`
- `refactor/nome-do-refactor`: Para refatorações de código.
  - Exemplo: `refactor/melhorar-performance`
- `chore/nome-da-tarefa`: Para tarefas de manutenção ou configuração.
  - Exemplo: `chore/atualizar-dependencias`
- `test/nome-do-teste`: Para adicionar ou corrigir testes.
  - Exemplo: `test/adicionar-testes-login`

---

2. Fazendo Alterações e Commit

Depois de fazer alterações no código, você precisa adicionar essas mudanças ao histórico do Git:

1. Verifique as alterações feitas:
   `git status`
2. Adicione as alterações para o próximo commit:
   `git add .`

3. Faça o commit das alterações com uma mensagem descritiva:
   `git commit -m "Descrição das alterações"`
4. Depois faça o git push:
   `git push`

2.1 Boas Práticas para Mensagens de Commit

Mensagens de commit devem ser claras e descritivas. Use o padrão **Conventional Commits** para facilitar a leitura do histórico:

- **Tipos comuns**:
  - `feat`: Para novas funcionalidades.
    - Exemplo: `feat: adicionar autenticação via Google`
  - `fix`: Para correções de bugs.
    - Exemplo: `fix: corrigir erro ao validar senha`
  - `docs`: Para alterações na documentação.
    - Exemplo: `docs: atualizar instruções de instalação`
  - `refactor`: Para refatorações de código.
    - Exemplo: `refactor: melhorar legibilidade do código`
  - `chore`: Para tarefas de manutenção.
    - Exemplo: `chore: atualizar pacotes npm`
  - `test`: Para adicionar ou corrigir testes.
    - Exemplo: `test: adicionar testes de autenticação`

---

3. Subindo a Branch para o Repositório Remoto

Para enviar sua branch local para o repositório remoto no GitHub:

`git push origin nome-da-branch`

Isso criará a branch no GitHub e enviará suas alterações.

---

4. Criando um Pull Request (PR)

Um Pull Request é uma solicitação para mesclar sua branch com a branch `main` (ou outra branch principal). Siga os passos:

5. No GitHub, vá até o repositório.

6. Clique em **"Pull Requests"** > **"New Pull Request".**
7. Selecione sua branch e a branch `main`.
8. Adicione uma descrição e crie o PR.

---

5. Mesclando a Branch com o Main

Depois que o PR for revisado e aprovado, você pode mesclar as alterações:

9. No GitHub, vá até o PR.
10. Clique em "Merge Pull Request".
11. Confirme a mesclagem.

---

6. Atualizando o Repositório Local

Depois de mesclar a branch, atualize seu repositório local:

```
git checkout main
git pull origin main
```

Isso trará as alterações mais recentes para sua máquina.

---

7. Boas Práticas

- Sempre crie uma nova branch para cada nova funcionalidade ou correção.
- Use nomes de branches descritivos e siga as convenções.
- Escreva mensagens de commit claras e descritivas.
- Revise o código antes de mesclar com a branch `main`.