Apostila: Dominando o Uso de Branches no Git

Introdução ao Git e Branches

O **Git** é um sistema de controle de versão distribuído usado amplamente no desenvolvimento de software. Ele permite que você acompanhe alterações no código ao longo do tempo e, mais importante, que você trabalhe em diferentes funcionalidades de forma isolada, sem afetar o código principal. Para isso, o Git usa o conceito de **branches** (ramificações).

O Que é uma Branch?

Uma **branch** é uma linha de desenvolvimento paralela que permite trabalhar em uma funcionalidade ou correção de bug sem interferir diretamente na versão estável do projeto. Com branches, é possível testar novas ideias, corrigir erros, e desenvolver funcionalidades de forma organizada.

Por que Usar Branches?

- **Organização:** Permite manter o código organizado em diferentes linhas de desenvolvimento, como funcionalidades, correções de bugs e experimentos.
- **Segurança:** Altera apenas o código da branch atual, sem afetar a branch principal (geralmente chamada de main ou master).
- **Trabalho em equipe:** Evita conflitos ao permitir que vários desenvolvedores trabalhem simultaneamente em diferentes partes do projeto.
- **Facilidade para testar:** Permite testar novas funcionalidades e corrigir bugs sem risco de comprometer a base de código principal.
- **Revisão de código:** Facilita o processo de revisão de código, pois as mudanças podem ser integradas apenas após revisão.

Comandos Básicos de Branches

Criar uma Nova Branch

Para criar uma nova branch, use o comando:

```
git branch nome-da-branch
```

Isso cria a branch, mas você ainda precisa alternar para ela. Para fazer isso, use checkout.

Alternar para uma Branch Específica

Para alternar para uma branch já existente, use:

git checkout nome-da-branch

Criar e Alternar para uma Nova Branch Diretamente

Se você quiser criar uma nova branch e já alternar para ela, use:

git checkout -b nome-da-branch

Listar Todas as Branches

Para ver todas as branches no repositório (incluindo a branch atual), use:

git branch

Excluir uma Branch (após mesclagem)

Depois que uma branch for mesclada e não for mais necessária, você pode excluí-la:

git branch -d nome-da-branch

Forçar a Exclusão de uma Branch (sem mesclagem)

Se você precisar excluir uma branch sem fazer o merge, use o seguinte comando (cuidado, pois isso pode perder alterações):

git branch -D nome-da-branch

Trabalhando com Branches Remotas

Enviar uma Branch para o Repositório Remoto (Push)

Após criar e fazer commits em sua branch local, você pode enviá-la para o repositório remoto usando:

git push origin nome-da-branch

Atualizar Branches Locais com as do Repositório Remoto (Fetch)

O comando git fetch baixa as atualizações do repositório remoto, mas **não** as aplica à sua branch local automaticamente. Isso é útil para verificar as alterações realizadas por outros desenvolvedores sem modificar seu trabalho atual.

git fetch

Trazer Alterações do Repositório Remoto para Sua Branch Atual (Pull)

O git pull combina os efeitos do fetch e merge, ou seja, ele baixa as atualizações e já tenta integrá-las à sua branch local.

git pull origin nome-da-branch

Diferença entre fetch, merge, e pull

git fetch

- **O que faz:** Baixa as alterações do repositório remoto, mas não as integra à sua branch atual. Ele apenas atualiza o histórico do repositório local.
- **Quando usar:** Use fetch quando quiser verificar o que mudou no repositório remoto sem modificar o seu código local. Isso é útil quando você quer revisar as alterações feitas por outros desenvolvedores antes de incorporá-las.

Exemplo:

git fetch origin

git merge

- **O que faz:** Mescla as alterações de uma branch em outra. Após usar o fetch para obter as últimas alterações do repositório remoto, você pode usar merge para integrá-las na sua branch local.
- **Quando usar:** Use merge quando quiser incorporar mudanças de uma branch para a sua branch atual.

Exemplo:

git merge origin/main

git pull (Fetch + Merge)

- **O que faz:** O comando git pull combina as funcionalidades de fetch e merge em um único comando. Ele baixa as alterações do repositório remoto e as aplica à sua branch local.
- **Quando usar:** Use pull para atualizar rapidamente a sua branch com as alterações mais recentes do repositório remoto.

Exemplo:

git pull origin main

Exercícios Práticos

Exercício 1: Criando e Mudando para uma Nova Branch

Objetivo: Criar uma branch chamada nova-feature e alternar para ela.

Passos:

1. No terminal, execute o seguinte comando para criar e mudar para a nova branch:

   ```
   git checkout -b nova-feature
   ```

2. Verifique se a nova branch foi criada e está ativa:

   ```
   git branch
   ```

   A branch nova-feature deve estar destacada com um *.

Exercício 2: Editando e Commitando na Nova Branch

Objetivo: Fazer alterações e registrar essas mudanças na branch nova-feature.

Passos:

1. Crie um arquivo chamado exemplo.txt:

   ```
   echo "Conteúdo inicial" > exemplo.txt
   ```

2. Adicione o arquivo ao controle do Git:

   ```
   git add exemplo.txt
   ```

3. Realize o commit:

   ```
   git commit -m "Adicionando o arquivo exemplo.txt"
   ```

Exercício 3: Mesclando uma Branch no Projeto Principal

Objetivo: Mesclar as alterações da branch nova-feature na branch principal main.

Passos:

1. Volte para a branch principal:

   ```
   git checkout main
   ```

2. Mescle a branch nova-feature na main:

   ```
   git merge nova-feature
   ```

3. Se houver conflitos, edite os arquivos conforme necessário e finalize com:

   ```
   git add .
   git commit -m "Resolvido conflito ao mesclar nova-feature"
   ```

Exercício 4: Branch de Correção de Erros (Hotfix)

Objetivo: Criar uma branch de correção urgente e enviar ao repositório remoto.

Passos:

1. Crie uma branch chamada hotfix-erro-login:

   ```
   git checkout -b hotfix-erro-login
   ```

2. Corrija o erro no arquivo apropriado.
3. Adicione as mudanças ao controle de versão:

   ```
   git add .
   git commit -m "Corrigido erro na função de login"
   ```

4. Envie a branch ao repositório remoto:

   ```
   git push origin hotfix-erro-login
   ```

Exercício 5: Branch de Desenvolvimento com Submódulo

Objetivo: Criar uma branch com um módulo separado para que um desenvolvedor possa trabalhar sem acessar todo o projeto.

Passos:

1. Crie uma branch chamada modulo-relatorios:

   ```
   git checkout -b modulo-relatorios
   ```

2. Adicione o submódulo para o novo módulo:

   ```
   git submodule add <URL_DO_REPO> app/Modules/Relatorios
   ```

3. Commit e push:

   ```
   git commit -m "Adicionado submódulo de relatórios"
   git push origin modulo-relatorios
   ```

Dicas de Boas Práticas com Branches

Nomeando Branches

- **Exemplos bons de nomes:**
  - feature/login-page
  - bugfix/erro-validacao-email
  - hotfix/fix-crash-on-startup

Estrutura de Commits

- Mantenha commits pequenos e frequentes.
- Sempre que possível, use uma mensagem clara no commit, como:
    - feat: Adicionando a página de login
    - fix: Corrigido erro de validação de e-mail

Revisões de Código

- Abra Pull Requests (PRs) para cada feature ou correção de bug. Isso permite que o código seja revisado antes de ser integrado ao repositório principal.

Conclusão

O uso de branches é essencial para manter um fluxo de desenvolvimento organizado e eficiente. Com o Git, você pode isolar alterações, testar novas funcionalidades, corrigir bugs e trabalhar de forma colaborativa com outros desenvolvedores, sem correr o risco de comprometer o projeto principal. Continue praticando para se tornar um mestre no uso de branches no Git!

Caso precise de mais detalhes ou exemplos adicionais, estou à disposição!