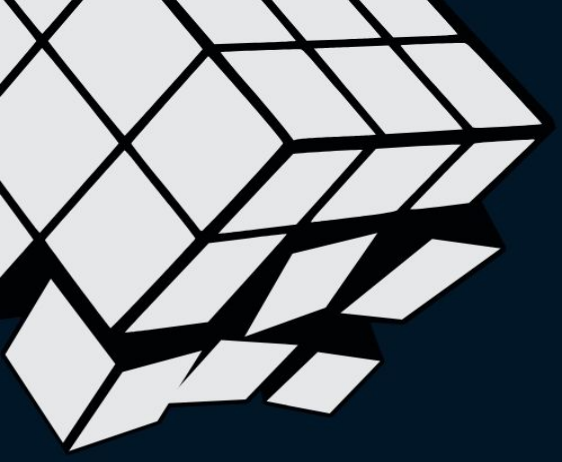


INTRODUCTION TO GIT & GITHUB

Table of Contents

```
  main
* novaFuncionalidade
```

Agora, crie um novo arquivo com conteúdo de demonstração:

```
echo "<h1>Meu Primeiro Branch de Funcionalidade</h1>" >
funcionalidade1.html
```

O comando acima irá criar o arquivo `funcionalidade1.html` com o conteúdo `<h1>Meu Primeiro Branch de Funcionalidade</h1>`.

Depois disso, prepare o arquivo e faça o commit:

```
git add funcionalidade1.html
git commit -m "Adicionar funcionalidade1.html"
```

O novo arquivo estará presente apenas no branch `novaFuncionalidade`. Se você mudar para o branch `main` e rodar o comando `ls` ou verificar o `git log`, verá que o arquivo não está lá.

Você pode verificar isso usando:

```
git log
```

Com isso, usamos vários dos comandos que vimos nos capítulos anteriores!

Comparando branches

Você também pode comparar dois branches com os comandos abaixo.

- Mostra os commits em `branchA` que não estão em `branchB`:

```
git log BranchA..BranchB
```

- Mostra as diferenças do que está em `branchA` mas não em `branchB`:

```
git diff BranchB...BranchA
```

Renomeando um branch

Se você criou um branch com o nome errado ou acha que o nome pode ser melhor, pode renomear com:

```
git branch -m nome-antigo nome-novo
```

Se quiser renomear o **branch atual**, basta rodar:

```
git branch -m nome-do-branch
```

Depois, se rodar `git branch` novamente, verá o nome correto.

Excluindo um branch

Para excluir um branch específico, execute:

```
git branch -d nome_do_branch
```

Isso exclui o branch apenas do seu repositório local. Se já tiver enviado o branch para o GitHub, use:

```
git push origin --delete nome_do_branch
```

Se quiser sincronizar seus branches locais com os remotos, rode:

```
git fetch
```

Conclusão

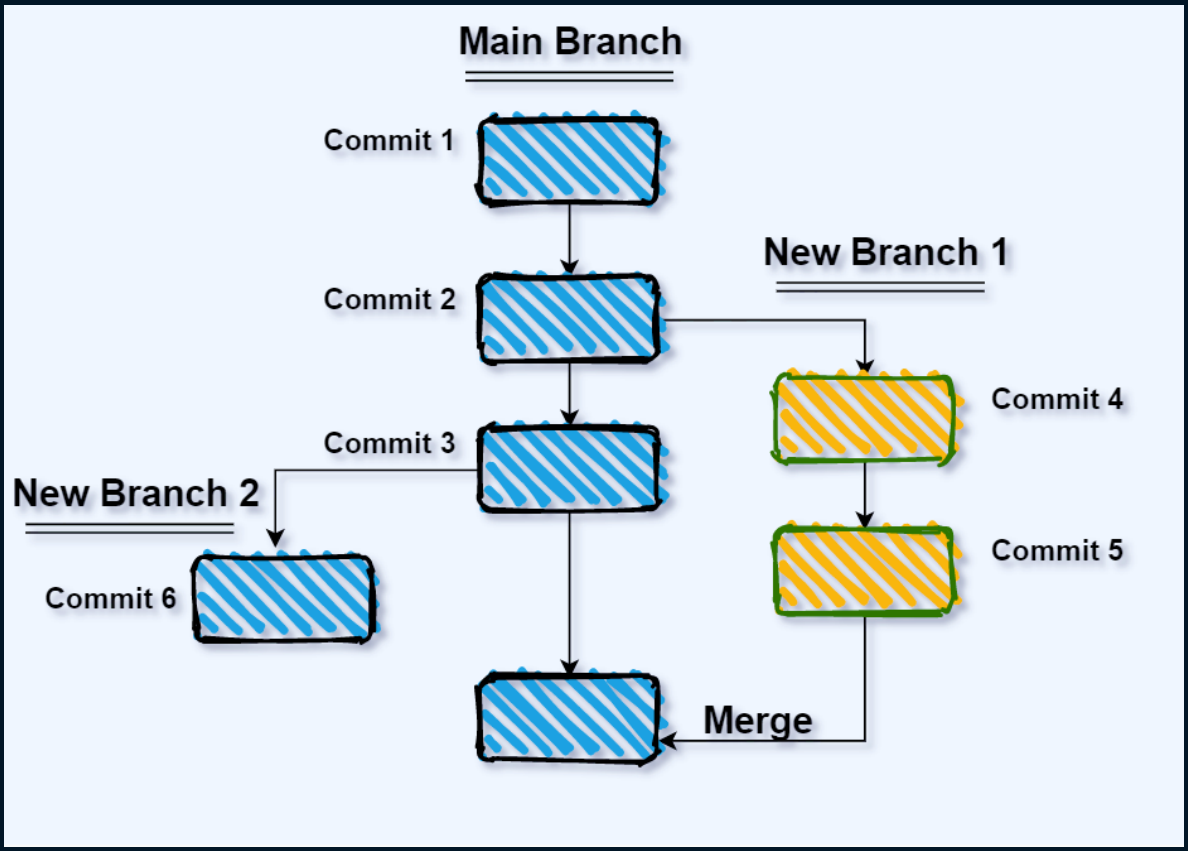
Com isso, nosso branch `novaFuncionalidade` está à frente do branch principal com 1 commit. Para trazer essas novas alterações para o branch principal, precisamos fazer o merge do branch `novaFuncionalidade` no `main`.

No próximo capítulo, você aprenderá como mesclar suas alterações de um branch para outro!

Uma coisa importante: antigamente, ao criar um novo repositório no GitHub, o nome padrão do branch era `master`. Agora, novos repositórios usam `main` como padrão, como parte do esforço do GitHub para usar termos mais inclusivos.

Git Merge

Quando os desenvolvedores terminam suas alterações, eles podem mesclar seus branches de funcionalidade ao branch principal e tornar essas funcionalidades disponíveis no site.



Se você seguiu os passos do capítulo anterior, seu branch `novaFuncionalidade` está à frente do branch principal com 1 commit. Para trazer essas novas alterações para o branch principal, precisamos fazer o merge do branch `novaFuncionalidade` no `main`.

Mesclando um branch

Siga estes passos:

- Primeiro, mude para o branch `main`:

```
git checkout main
```

- Depois, para mesclar o branch `novaFuncionalidade` e as alterações que criamos no capítulo anterior, execute:

```
git merge novaFuncionalidade
```

Saída:

```
Updating ab1007b..a281d25
Fast-forward
 funcionalidade1.html | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 funcionalidade1.html
```

Como você estava no branch `main` ao rodar o comando, o Git irá pegar todos os commits daquele branch e mesclá-los ao branch principal.

Agora, se rodar o comando `ls`, verá o novo arquivo `funcionalidade1.html`, e se verificar o histórico de commits com `git log`, verá o commit do branch `novaFuncionalidade` presente no branch principal.

Antes de fazer o merge, você pode usar o comando `git diff` para checar as diferenças entre seu branch atual e o branch que deseja mesclar. Por exemplo, se estiver no branch `main`, pode usar:

```
git diff novaFuncionalidade
```

Neste caso, o merge ocorreu sem conflitos, pois não havia conflitos. Porém, em projetos reais com várias pessoas, podem ocorrer conflitos.

Isso acontece quando alterações são feitas na mesma linha de um arquivo, ou quando um desenvolvedor edita um arquivo em um branch e outro exclui o mesmo arquivo.

Resolvendo conflitos

Vamos simular um conflito. Para isso, crie um novo branch:

```
git checkout -b demoConflito
```

Depois edite o arquivo `funcionalidade1.html`:

```
echo "<p>Demonstração de Conflito</p>" >> funcionalidade1.html
```

O comando acima adiciona `<p>Demonstração de Conflito</p>` ao final do arquivo. Você pode verificar o conteúdo com:

```
cat funcionalidade1.html
```

Saída:

```
<h1>Meu Primeiro Branch de Funcionalidade</h1>
<p>Demonstração de Conflito</p>
```

Você pode rodar `git status` e `git diff` para ver o que foi modificado antes de fazer o commit.

Depois, faça o commit da alteração:

```
git commit -am "Demo de Conflito 1"
```

Pull Requests

Você já sabe como mesclar alterações de um branch para outro no seu repositório Git local.

Para fazer o mesmo no GitHub, você precisa abrir um Pull Request (ou Merge Request, se estiver usando GitLab), ou PR para abreviar, e solicitar a mesclagem do seu branch de funcionalidade para o branch `main`.

Os passos para abrir um Pull Request são:

- Se você está trabalhando em um projeto open source do qual não é mantenedor, primeiro faça um fork do repositório conforme explicado no capítulo 21. Pule esta etapa se você for o mantenedor do repositório.
- Depois, clone o repositório localmente com o comando `git clone`:

```
git clone git@github.com:seu_usuario/seu_repositorio
```

- Crie um novo branch com o comando `git checkout`:

```
git checkout -b nome_do_branch
```

- Faça suas alterações no código

- Prepare as alterações com `git add`:

```
git add .
```

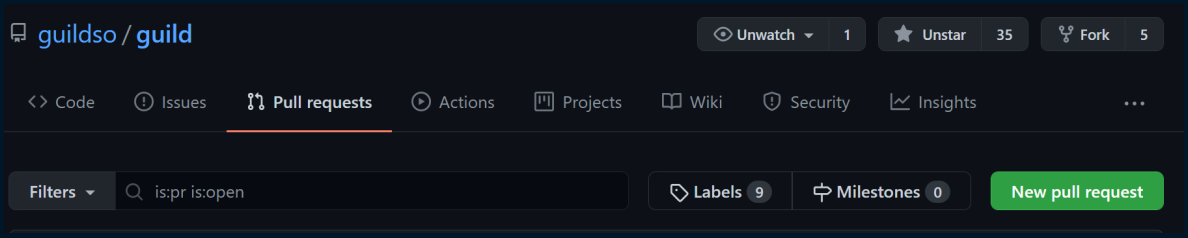
- E então faça o commit com `git commit`:

```
git commit -m "Mensagem do Commit"
```

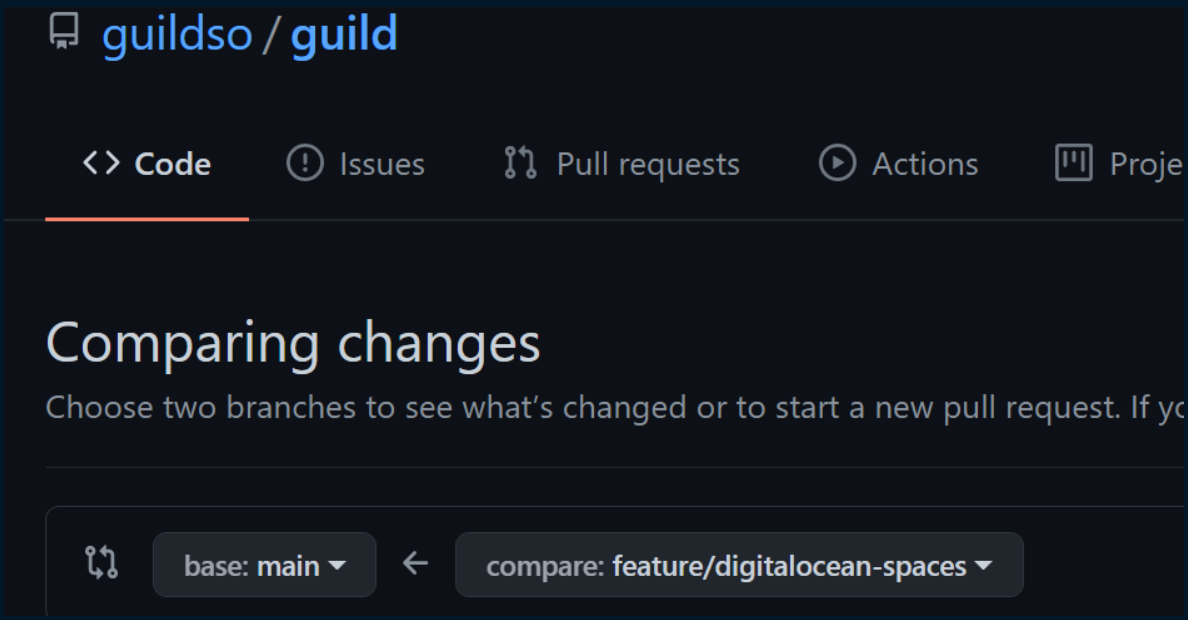
- Depois envie seu novo branch para o GitHub com `git push`:

```
git push origin nome_do_branch
```

- Após isso, acesse o repositório no GitHub e clique no botão `Pull Requests` e depois no botão verde `New pull request`:



- Lá, escolha o branch para o qual deseja mesclar e o branch de onde deseja mesclar:



- Depois revise as alterações, adicione um título e descrição e clique no botão de criar.
- Se estiver trabalhando em um projeto com múltiplos colaboradores, selecione alguns revisores. Revisores são pessoas que você gostaria que revisassem seu código antes de ele ser mesclado ao branch `main`.

Para uma representação visual de todo o processo, confira este tutorial passo a passo:

- Como enviar seu primeiro Pull Request no GitHub

Git e VS Code

Por mais que eu goste de usar o terminal para minhas tarefas diárias, no final das contas prefiro realizar várias tarefas em uma única janela (GUI) ou fazer tudo diretamente pelo terminal.

No passado, eu usava editores de texto (vim, nano, etc.) no terminal para editar o código dos meus repositórios e depois utilizava o cliente git para commitar minhas alterações. Mas depois migrei para o Visual Studio Code para gerenciar e desenvolver meu código.

Recomendo que você confira este artigo sobre por que usar o Visual Studio. É um artigo do próprio site do Visual Studio.

Por que você deve usar o Visual Studio

O Visual Studio Code possui gerenciamento de controle de versão integrado (SCM) e inclui suporte ao Git nativamente. Muitos outros provedores de controle de versão estão disponíveis através de extensões no Marketplace do VS Code. Ele também suporta múltiplos provedores de controle de versão simultaneamente, permitindo abrir todos os seus projetos ao mesmo tempo e fazer alterações sempre que necessário.