O Anel da Codificação

E a Batalha dos Commits

Saiba quais são os principais comandos git e como resolver conflitos de commits

Diana Alves

Guia Prático de Git:

O que é Git?

Git é um sistema de controle de versão amplamente utilizado para o gerenciamento eficiente de projetos de software. Ele permite que equipes de desenvolvimento colaborem de forma eficaz, rastreiem alterações no código-fonte e revertam para versões anteriores quando necessário.

01

Configurações Iniciais

Primeiros Passos

Configuração:

Para começar a usar o Git, é necessário configurar seu nome de usuário e endereço de e-mail, o que pode ser feito com os seguintes comandos:

Comandos:

- git config --global user.name "Seu Nome"
- git config --global user.email "seu@email.com"

02

Inicializando um Repositório

Iniciando o Repositório

Ao trabalhar em um novo projeto, é fundamental iniciar um repositório Git para rastrear suas alterações. Vejamos mais comandos para essa etapa inicial:

Clonando um Repositório Existente

Se você não estiver criando um projeto totalmente novo e precisar começar a trabalhar em um repositório existente, pode cloná-lo para o seu ambiente local. Use o comando git clone seguido do URL do repositório.
Isso criará uma cópia local do repositório, permitindo que você trabalhe em seu código e mantenha-se atualizado com as alterações feitas por outros colaboradores:

```
➢ git clone "url_do_repositorio"
```

Iniciando um Novo Repositório

Se você estiver começando um novo projeto e deseja versionar seu código com o Git, use o comando git init no diretório do seu projeto, Isso criará um novo repositório Git localmente, pronto para rastrear suas alterações:

```
➢ git init
```

03

Principais comandos

Comandos Mais Usados

São aqueles usados com mais frequência:

Adicionando Arquivos ao Repositório:

Para adicionar arquivos ao repositório, você pode usar o comando git add. Por exemplo, para adicionar todos os arquivos modificados à área de preparação, use:

- `git add .`

Verificando o Status do Repositório:

Para visualizar o status atual do seu repositório Git, use o comando git status:

- `git status`

Verificando o Histórico de Commits:

Esse comando irá exibir uma lista de commits em ordem cronológica reversa, começando pelo commit mais recente. Cada commit incluirá detalhes como o hash do commit, autor, data e mensagem de commit.

- `git log`

04

Commits

Criando Commits

Depois de adicionar os arquivos que deseja incluir em um commit, você pode criar um novo commit usando o comando git commit:

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

Após o commit você pode enviar suas alterações locais para o repositório, através do seguinte comando :

Substitua "branch-name" pelo nome da branch que você deseja enviar para o repositório remoto. "Origin" é o apelido (alias) padrão do repositório remoto. Se você configurou um apelido diferente, substitua "origin" por esse apelido.

- `git push origin "branch-name"`

Agora se o que você precisa é puxar as alterações mais recentes do repositório remoto para o seu repositório local, será o seguinte comando:

- `git pull origin "branch-name"`

Navegando nos Commits

O comando git branch no Git é usado para listar, criar ou excluir branches (ramificações) em seu repositório.

- git branch

Se você fornecer um nome de branch como argumento, ele cria uma nova branch com esse nome:

Aqui está como você pode usar o comando git branch para criar uma nova branch com um nome específico:

- git branch "nome_da_ramificacao"

Se precisar recuperar uma versão anterior do seu código, você pode usar o comando git checkout seguido do hash do commit desejado.
Isso mudará seu diretório de trabalho para o estado em que estava no momento do commit especificado:

- git checkout "hash_do_commit"

05

Conflitos nos Commits

Resolvendo conflitos

Ao realizar um merge que faz a mescla das alterações da ramificação especificada de volta para a ramificação atual.

- git merge "nome_da_ramificacao"

Pode acontecer conflitos e esses conflitos de merge ocorrem quando duas ou mais ramificações possuem alterações conflitantes no mesmo trecho de código. O Git não consegue determinar automaticamente qual versão deve prevalecer, deixando para o usuário resolver manualmente.

```
<<<<<<< HEAD
Código na branch atual
=======
Código na outra branch
>>>>>>> outra_branch
```

Nesse exemplo, <<<<<<< HEAD, ======= e >>>>>>> outra_branch são marcadores que indicam as versões conflitantes do código. O trecho entre <<<<<<< HEAD e ======= representa as alterações na branch atual, enquanto o trecho entre ======= e >>>>>>> outra_branch representa as alterações na outra branch.

Resolvendo conflitos

Para resolver o conflito, você deve editar manualmente o arquivo para decidir qual versão das alterações deve ser mantida ou combinada. Após editar o arquivo, você precisa remover os marcadores de conflito e quaisquer linhas indesejadas adicionadas pelo Git.

Após resolver todos os conflitos em todos os arquivos afetados, você precisa adicioná-los novamente à área de preparação usando git add e, em seguida, finalizar o merge com git commit.

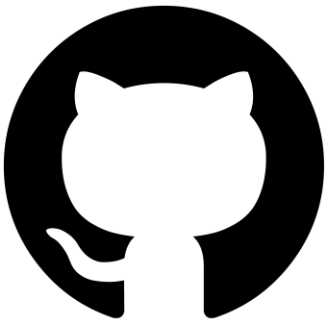
É importante resolver conflitos com cuidado, garantindo que o código resultante seja funcional e mantenha a integridade do projeto.

Agradecimentos

OBRIGADO POR LER ATÉ AQUI

Esse Ebook foi gerado por IA, diagramado por humano e inspirado através das aulas do Bootcamp Fundamentos de IA para Devs com a aula dada pelo professor Felipe Aguiar.

Esse conteúdo foi gerado com fins didáticos de construção, não foi realizado uma validação cuidadosa humana no conteúdo e pode conter erros gerados por uma IA.



Para mais esclarecimentos e dúvidas entre nesses links disponíveis.

https://github.com/felipeAguiarCode/prompts-recipe-to-create-a-ebook

https://github.com/dianalves