O que são conflitos?

Contextualizando..

Um conflito ocorre no contexto da colaboração em desenvolvimento de software quando pelo menos duas pessoas trabalham simultaneamente no mesmo arquivo. Isso pode ocorrer quando cada pessoa está fazendo alterações independentes na mesma porção do código. Quando chega a hora de mesclar, ou seja, integrar suas implementações na branch principal do projeto, podem surgir problemas.
O conflito se manifesta quando as alterações feitas por uma pessoa no arquivo estão em conflito direto com as alterações feitas por outra pessoa. Isso significa que o sistema de controle de versão não pode determinar automaticamente como combinar essas alterações de forma coesa.
Para resolver um conflito, é necessário realizar uma atualização no seu próprio código para incorporar as alterações mais recentes da branch principal. Isso envolve reconciliar manualmente as diferenças entre as versões do arquivo e escolher quais alterações devem ser mantidas. É um processo cuidadoso que requer atenção aos detalhes para garantir que o código final seja funcional e coerente.
Em resumo, um conflito ocorre quando várias pessoas editam o mesmo arquivo ao mesmo tempo, resultando em alterações conflitantes que precisam ser resolvidas manualmente, incorporando as mudanças mais recentes da branch principal em sua própria implementação. Essa prática garante que o código permaneça atualizado e funcional durante o processo de mesclagem.

Assumindo que a implementação foi feita em uma branch e o conflito aconteceu na hora que foi fazer o merge com a branch principal do projeto, por exemplo, a main/master. Para resolver, basta sair da branch local e ir para a branch principal, fazer a mudança de branch usando o comando: **git checkout nome_da_branch_destino.**
Uma vez na branch de destino(a principal branch do projeto) usamos o comando para levar as atualizações da branch principal para a branch local: **git pull origin nome_da_branch_principal.**
Com isso, temos as atualizações na branch local. Agora, acessamos a IDE com código fonte que foi implementado e resolvemos manualmente o conflito. Depois preparamos um arquivo para ir para o repositório remoto.

```
git add nome_do_arquivo
git commit -m “mensagem da correção de conflito”
git push
```

Na prática …

Passo 1: Criar uma Branch para Implementar

Suponha que estamos trabalhando em um projeto Java e decidimos criar uma nova funcionalidade para calcular a média de números em uma classe chamada

CalculoMedia.java. Primeiro, criamos uma nova branch chamada feature-media para implementar essa funcionalidade.

git branch feature-media

git checkout feature-media

Passo 2: Implementar as Alterações na Branch Feature

Agora, começamos a implementar a funcionalidade de cálculo da média na classe CalculoMedia.java. Enquanto fazemos isso, outra pessoa da equipe também está trabalhando na mesma classe na branch principal (main).

```
// Em CalculoMedia.java na branch feature-media
public class CalculoMedia {
    public double calcularMedia(double[] numeros) {
        // Implementação do cálculo da média
        return 0.0;
    }
}
```

Passo 3: Conflito Gerado

Enquanto estamos trabalhando na nossa branch feature-media, a outra pessoa fez alterações na mesma classe na branch main. Por exemplo, eles adicionaram um método diferente:

```
// Em CalculoMedia.java na branch main
public class CalculoMedia {
    public void outroMetodo() {
        // Outra funcionalidade
    }
}
```

Agora, tentamos mesclar nossa branch feature-media de volta à branch main usando o comando:

git checkout main
git merge feature-media

Nesse ponto, o sistema de controle de versão (como o Git) detecta um conflito porque ambas as branches alteraram o mesmo arquivo na mesma área.

Passo 4: Resolver o Conflito

Para resolver o conflito, abrimos o arquivo CalculoMedia.java e o encontramos com marcações que indicam as áreas conflitantes. O arquivo pode se parecer com isso:

```
// Em CalculoMedia.java
public class CalculoMedia {
    public double calcularMedia(double[] numeros) {
        // Implementação do cálculo da média
        return 0.0;
    }

<<<<<<< HEAD
    public void outroMetodo() {
        // Outra funcionalidade
    }
=======
    // Nossa implementação
    public void nossaFuncao() {
        // Nossa funcionalidade
    }
>>>>>>> feature-media
}
```

Aqui, as marcações <<<<<<< HEAD, =======, e >>>>>>> feature-media indicam as partes conflitantes. Precisamos escolher quais alterações manter.

Vamos resolver o conflito mantendo ambas as funções. O arquivo fica assim:

```
// Em CalculoMedia.java
public class CalculoMedia {
    public double calcularMedia(double[] numeros) {
        // Implementação do cálculo da média
        return 0.0;
    }

    public void outroMetodo() {
        // Outra funcionalidade
    }

    // Nossa implementação
    public void nossaFuncao() {
        // Nossa funcionalidade
    }
}
```

Passo 5: Completar a Mesclagem

Após resolver o conflito manualmente, salvamos o arquivo. Agora, podemos continuar com o processo de mesclagem:

```
# Adicionar o arquivo resolvido
```

```
git add CalculoMedia.java

# Confirmar as mudanças
git commit -m "Resolver conflito em CalculoMedia.java"

# Completar a mesclagem
git merge --continue
```

Agora, as alterações da branch feature-media foram mescladas com sucesso na branch main, e o conflito foi resolvido. A classe CalculoMedia.java contém ambas as funcionalidades das duas branches.

Gostou? curte. compartilha e segue meu canal rs
by : https://github.com/ADRIELYNOVAIS