**Salvando sua sexta com Git Revert**

O **Git Revert** é um comando usado para desfazer as alterações introduzidas por um commit específico, criando um novo commit que anula (ou "reverte") essas alterações. Diferentemente de comandos como git reset, o git revert não modifica o histórico existente, o que o torna uma abordagem mais segura e adequada para repositórios compartilhados.

**Como Funciona o Git Revert?**

Quando você executa git revert, o Git gera um novo commit que aplica o inverso das alterações feitas pelo commit especificado. Isso significa que, mesmo revertendo um commit, o histórico do repositório continua linear e intacto, preservando o registro de todas as mudanças feitas no passado.

**Quando Usar Git Revert?**

1. **Desfazer Alterações de Forma Segura**:
   * Quando você precisa desfazer um commit em um repositório compartilhado sem reescrever o histórico.
2. **Manter o Histórico Intacto**:
   * Ideal em situações onde o repositório já foi compartilhado e outros desenvolvedores dependem do histórico existente.
3. **Corrigir um Commit Problemático**:
   * Caso um commit específico tenha introduzido um bug ou uma mudança indesejada, você pode revertê-lo sem impactar os commits subsequentes.

**Sintaxe do Git Revert**

**1. Reverter um Commit Específico:**

git revert <hash-do-commit>

* Substitua <hash-do-commit> pelo hash do commit que você deseja reverter.
* Isso cria um novo commit que inverte as mudanças introduzidas pelo commit especificado.

**2. Reverter Vários Commits (Sequencialmente):**

Você pode reverter vários commits um por um:

git revert <hash1> <hash2> <hash3>

**3. Reverter um Intervalo de Commits:**

Para reverter um intervalo de commits:

git revert <hash-inicial>..<hash-final>

* Isso reverte todos os commits no intervalo especificado.

**4. Reverter sem Abrir o Editor:**

Use a flag -m para evitar o editor de texto ao fazer o revert:

git revert -m 1 <hash-do-commit>

**Exemplo de Uso**

Imagine o seguinte histórico de commits:

A - B - C - D

* O commit C introduziu um bug.
* Para reverter o commit C, execute:
* git revert <hash-do-C>

O histórico resultante será:

A - B - C - D - E

* O commit E contém as mudanças opostas às feitas por C.

**Reverter Commits de Merge**

Reverter um commit de merge requer um cuidado especial. Você precisa especificar o **parent** (pai) do merge que deseja manter.

**Comando:**

git revert -m <parent-number> <hash-do-merge-commit>

* O **parent-number** indica qual branch pai deve ser considerado como base.
  + 1 para o primeiro pai (normalmente o branch de destino do merge).
  + 2 para o segundo pai (normalmente o branch que foi mesclado).

**Resolvendo Conflitos Durante o Revert**

Se houver conflitos ao executar o git revert, o Git pausará o processo e solicitará que você resolva os conflitos manualmente. O fluxo é o seguinte:

1. Resolva os conflitos nos arquivos listados.
2. Marque os arquivos como resolvidos:
3. git add <arquivos-resolvidos>
4. Finalize o revert:
5. git revert --continue

**Diferença entre Git Revert e Git Reset**

| **Aspecto** | **Git Revert** | **Git Reset** |
| --- | --- | --- |
| **Histórico** | Preserva o histórico. | Pode reescrever o histórico. |
| **Novo Commit** | Cria um novo commit. | Não cria um novo commit. |
| **Uso em Equipe** | Seguro para repositórios compartilhados. | Pode causar problemas em repositórios compartilhados. |
| **Desfaz** | Alterações de um commit específico. | Todo o histórico de commits afetados. |

**Casos de Uso Reais**

1. **Bug Introduzido no Último Commit**: Se o commit mais recente introduziu um bug, você pode revertê-lo:
2. git revert HEAD
3. **Desfazer Commits Antigos**: Caso um bug seja identificado em um commit antigo, você pode revertê-lo sem alterar os commits mais recentes.
4. **Reverter Alterações de Um Merge**: Se um merge foi feito erroneamente, use o comando com a flag -m para revertê-lo.

**Conclusão**

O **Git Revert** é uma ferramenta poderosa e segura para desfazer mudanças específicas em um repositório Git, especialmente em repositórios compartilhados onde o histórico não deve ser reescrito. Ele preserva a integridade do histórico, criando um novo commit para anular alterações indesejadas, tornando-o ideal para corrigir problemas sem impactar o trabalho de outros desenvolvedores.