

Git Branching

A palavra Branching significa desvio da linha principal de desenvolvimento, enquanto continuamos trabalhando em partes do projeto, sem mexer no projeto raiz, e depois de feito o que é necessário, podemos unir essas partes ao projeto principal. Porém, é um processo caro e que exige muito tempo quando se trata de grandes projetos.

No entanto, o modelo de *branching* do Git quebrou todos os paradigmas, trazendo leveza ao *branching*. O Git lida de forma estratégica, tornando o processo rápido e criando um fluxo melhor para o projeto. Compreender a forma como o Git *branching* trabalha pode gerar uma melhora significativa no desenvolvimento do projeto.

O intuito das *branches* é proteger o projeto principal das novas atualizações e facilitar o processo colaborativo de desenvolvimento, tornando-o mais dinâmico e melhorando seu fluxo.

Para entender como o Git faz as ramificações, precisamos entender como ele armazena seus dados. O Git armazena uma série de *snapshots*, capturando o estado de algo em um ponto específico no tempo.

Com o *commit*, o Git armazena um objeto de *commit* que contém um ponteiro para o *snapshot* do conteúdo que preparei. Neste objeto, também irão conter informações pessoais minhas, como e-mail e nome. Basicamente, cada *commit* do Git captura cada momento específico do meu projeto, incluindo informações sobre quem fez a mudança, como foi feita, quando foi feita e como se conecta a outros *commits*.

Quando se faz um *commit*, o Git cria uma árvore que lista os arquivos e os *blobs* associados a esses arquivos, e armazena essa árvore no *commit*. O *commit* também inclui informações sobre quem fez a mudança e o histórico dos *commits* anteriores.

Quando criamos uma ramificação, criamos um ponteiro para movimentar. Para criar uma ramificação, basta executar o seguinte comando:

A terminal window with a dark background and three light gray window control buttons in the top-left corner. The text 'git branch' is displayed in a monospaced font, with 'git' in green and 'branch' in yellow.

```
git branch
```

Este comando cria a ramificação e aponta para o mesmo commit da ramificação atual.

Existe uma forma do Git saber em qual ramificação estamos. Ele mantém um ponteiro chamado HEAD, que aponta para a ramificação local na qual estou atualmente.

O comando `git branch` não nos tira da ramificação master; ele apenas cria uma nova. Para fazer de fato essa mudança, podemos usar:

A terminal window with a dark background and three light gray window control buttons in the top-left corner. The text 'git checkout' is displayed in a monospaced font, with 'git' in green and 'checkout' in yellow.

```
git checkout
```

Em uma versão mais recente do Git, podemos usar o comando `git switch`.

Com o `git log`, podemos ver onde os ponteiros das ramificações estão apontados. Essa opção é chamada de :

A terminal window with a dark background and three light gray window control buttons in the top-left corner. The text `git --decorate` is displayed in a light green monospace font.

```
git --decorate
```


Porém, o Git só mostra onde ele acha que estamos interessados. Para vermos todo o histórico de commits, temos que executar o comando:

A terminal window with a dark background and three light gray window control buttons in the top-left corner. The text `git log --all` is displayed in a light green monospace font.

```
git log --all
```

É importante lembrar que, quando trocamos de ramificação, podemos alterar os arquivos no nosso diretório. Se trocarmos para uma ramificação mais antiga, o diretório será revertido ao estado em que estava na última vez que fizemos o commit. Caso o Git não consiga executar a mudança corretamente, ele não permitirá que troquemos de ramificação.

Para criar uma nova ramificação e ao mesmo tempo alternar para a mesma, podemos usar:



```
git checkout -b <nomedarama>
```

Ao trocar de ramificação, o Git ajusta seu diretório de trabalho para refletir o estado dos arquivos no último commit dessa ramificação, fazendo alterações automaticamente.

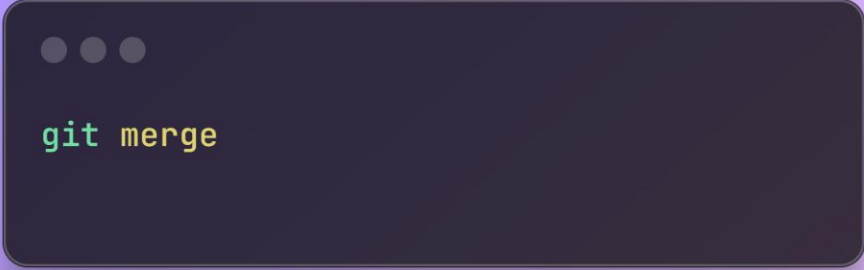


```
git branch -d
```

Usando este comando, podemos excluir ramificações locais do Git.

Mesclagem Básica

Se quisermos mesclar uma ramificação como fluxo principal do projeto (master/main), devemos verificar a ramificação que desejamos mesclar e então executar o comando:

A terminal window with a dark background and three window control buttons in the top left corner. The text 'git merge' is displayed in a monospaced font, with 'git' in green and 'merge' in yellow.

```
git merge
```

Conflitos de Mesclagem Básicos

Os conflitos podem ocorrer quando alteramos a mesma parte do mesmo arquivo de forma diferente nas duas ramificações, e o Git não consegue mesclá-las de forma limpa. Neste momento, ele pausa o processo enquanto resolvemos os problemas.

O ``git mergetool`` pode nos guiar pelos conflitos, caso ocorram.

Depois de realizar a mesclagem, o Git precisa confirmar se o processo foi bem-sucedido. Se confirmado por nós mesmos, ele irá marcar o arquivo como resolvido. Após dar ``git status``, podemos verificar se os conflitos foram resolvidos.

Em seguida, com todo o processo feito com sucesso, podemos dar um ``git commit`` para finalizar o commit de mesclagem.

Gerenciamento de Ramificações

Ao darmos o comando `git branch`, podemos ver uma lista de nossas ramificações. Caso executarmos `git branch -v`, podemos ver o último commit em cada ramificação.

Já as opções `--merged` e `--no-merged` podem filtrar a lista de ramificações mescladas e não mescladas. Para ver quais foram mescladas, podemos dar o comando `git branch --merged`.

Alterando o Nome de uma Ramificação

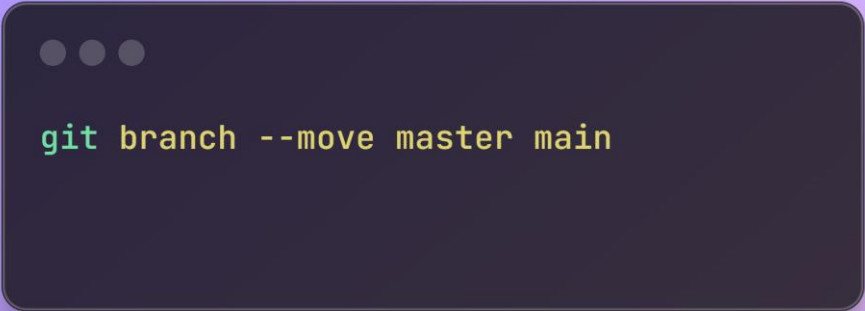
Não podemos renomear ramificações que ainda estão em uso ou colocar seus nomes como `master`, `main` ou `mainline`. Para alterar o nome mantendo todo o histórico, podemos fazer:

A terminal window with a dark background and three small circles in the top-left corner. The text `git branch --move` is displayed in a light green monospace font.

```
git branch --move
```

Mudança do Nome da Ramificação Master

Quando entramos em uma empresa, precisamos verificar o nome da ramificação, se é `main`, `master`, etc. Para mudar, podemos dar um:



```
git branch --move master main
```

Trocando assim o `master` pelo `main`.

Ramificações de Longa Duração

Ramificações de longa duração são projetadas para existir por um período prolongado e geralmente servem para o desenvolvimento contínuo de uma funcionalidade ou para manutenção ao longo do tempo. Elas diferem das ramificações temporárias ou de curto prazo, que são usadas para tarefas específicas e são frequentemente mescladas e removidas após a conclusão.

Ramificações como `master`, `main`, ou `develop` geralmente são mais estáveis e têm menos mudanças bruscas em comparação com ramificações temporárias.

Essas ramificações são úteis para organizar o trabalho e manter o código gerenciável, especialmente em projetos grandes e complexos.

Branches de Tópico

Um *branch* de tópico é um *branch* de curta duração que você cria e usa para um único recurso ou trabalho relacionado.

Seu trabalho está separado em silos, onde todas as alterações naquele **branch** estão relacionadas a um único tópico. Isso facilita a revisão do código e outras tarefas.

Você pode manter as alterações lá por minutos, dias ou meses, e mesclá-las quando estiverem prontas, independentemente da ordem em que foram criadas ou trabalhadas.

Branches Remotos

Branches remotos (ou ramificações remotas) são ramificações que existem em repositórios remotos. Elas permitem que múltiplas pessoas colaborem em um projeto, compartilhando e sincronizando seu trabalho com outras pessoas.

O comando ``git pull`` é basicamente uma combinação do ``git fetch`` e ``git merge`` em um só comando. Ele atualiza seu **branch** local com as mudanças do **branch** remoto e as mescla automaticamente. Isso pode ser conveniente, mas às vezes usar ``git fetch`` e ``git merge`` separadamente dá mais controle e clareza sobre as mudanças que estão sendo integradas ao seu **branch**.

Para deletar um **branch** remoto depois que ele foi mesclado (ou se não for mais necessário), você usa o ``git push`` com a opção ``--delete``.

Pushing

Quando você deseja compartilhar uma *filial* com o mundo, você precisa enviá-la para um *controle remoto* ao qual você pode ter acesso de gravação. Suas *filiais* locais não são sincronizadas automaticamente com os *controles remotos* para os quais você escreve — você deve enviar explicitamente os *ramos* que deseja compartilhar. Dessa forma, você pode usar *ramificações privadas* para trabalhos que não deseja compartilhar e enviar apenas as *ramificações de tópico* com as quais deseja colaborar.

Se você tiver uma branch chamada *serverfix* na qual deseja trabalhar com outras pessoas, poderá empurrá-la para cima da mesma forma que empurrou seu primeiro branch. Execute `git push <remote> <branch>`:

Fetch e Branches de Rastreamento Remoto


Ao fazer um fetch, novas *branches de rastreamento remoto* são criadas, mas não são editáveis localmente. Para trabalhar nelas, você pode criar uma nova branch local baseada na *branch de rastreamento remoto* usando `git checkout -b <local-branch> <remote-branch>`.

Tracking Branches

Tracking Branches são branches locais que têm uma relação direta com uma branch remota. Quando estamos em uma **tracking branch** e digitamos ``git pull``, o Git saberá de qual servidor buscar e em qual branch mesclar.

Quando você clona um repositório, ele geralmente cria uma branch ``master`` que rastreia ``origin/master``. Contudo, é possível configurar outras **tracking branches** que rastreiem diferentes branches remotas ou que não rastreiem a ``master``. Para isso, pode-se usar o comando ``git checkout -b <branch> <remote>/<branch>``, e o Git oferece a abreviação ``--track`` para essa operação comum.

Quando fazemos o **checkout** de uma branch local a partir de uma remota, o Git cria automaticamente uma **tracking branch**, que permite operações simplificadas como ``git pull``. Podemos configurar outras **tracking branches** para rastrear diferentes branches remotas ou renomeá-las. Para verificar o status das **tracking branches**, podemos usar o ``git branch -vv``, que mostra se suas branches estão adiantadas ou atrasadas em relação ao remoto.



```
git checkout --track origin/serverfix
Branch serverfix configurada para rastrear a branch remota
serverfix de origin.
Mudado para uma nova branch 'serverfix'
```

Deletando Branches Remotas

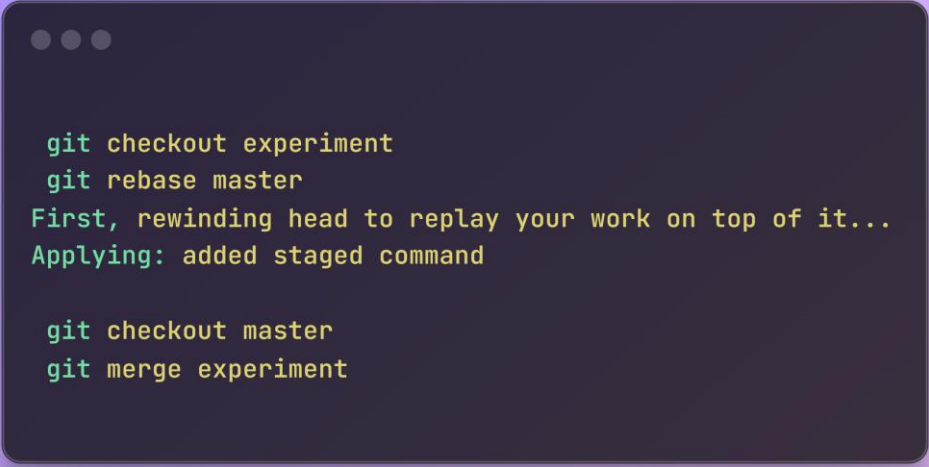
Para deletar uma branch remota que não é mais necessária, podemos executar ``git push origin --delete <branch>``. Isso remove o ponteiro da branch no servidor, mas os dados podem ser recuperados temporariamente antes da coleta de lixo do Git.

Integrando mudanças de uma branch em outra

Existem duas formas de integrar as mudanças de uma branch em outra: usando o merge ou o rebase.

Merge: O merge é uma forma de integrar mudanças de diferentes branches no Git. Ele realiza um merge de três vias entre os dois snapshots mais recentes das branches e o ancestral comum mais recente, criando um novo snapshot e commit que une os históricos de ambas as branches.

Rebase: Com o comando ``rebase``, podemos pegar todas as mudanças realizadas e commitadas em uma branch e reaplicá-las em uma branch diferente.

A terminal window with a dark background and light green text. It shows a sequence of Git commands and their output. The commands are: 'git checkout experiment', 'git rebase master', 'git checkout master', and 'git merge experiment'. The output includes 'First, rewinding head to replay your work on top of it...' and 'Applying: added staged command'.

```
git checkout experiment
git rebase master
First, rewinding head to replay your work on top of it...
Applying: added staged command

git checkout master
git merge experiment
```

A escolha entre merge ou rebase depende da preferência da equipe que executa o projeto e do contexto geral.

Podemos perceber que o Git surgiu para agregar muito aos nossos projetos e serviços. Há uma vasta gama de conteúdos sobre os comandos que podemos executar, e o Git branching é apenas a ponta do iceberg, mas de extrema importância para nós. O que antes era caro e difícil de ser executado, hoje, com um simples comando, facilita a produção de milhares de projetos.