Git Init Módulo III - Fundamentos do Git: Atualização, Organização e Colaboração

12/04/2023

Alexis Comesana Silvera
UDESC - Sistemas de Informação
São bento do Sul - Santa Catarina

SUMÁRIO

Módulo II - Trabalhando com Git **3**
- Git init 3
- Git status 4
- Git pull 6
- Git add 7
- Git commit 8
- Git checkout 10
- Git branch 12
- Git push 14

Módulo III - Trabalhando com Git

Git init

O comando **git init** é utilizado para criar um novo repositório Git em um diretório específico. Ele inicializa um novo repositório vazio, onde você poderá começar a versionar e controlar as alterações dos seus arquivos.

Quando você executa o **git init** em um diretório, o Git cria uma estrutura interna para armazenar as informações do repositório, como o controle de versões, histórico de **commits** e ramificações.

Para utilizar o **git init**, siga os seguintes passos:

1. Navegue até o diretório raiz do projeto onde deseja iniciar o repositório Git.
2. Abra o terminal ou prompt de comando nesse diretório.
3. Execute o comando "**git init**".

Após executar o **git init**, você verá uma mensagem informando que o repositório Git foi inicializado com sucesso. A partir desse momento, o diretório está sendo rastreado pelo Git e você poderá começar a adicionar e versionar seus arquivos.

É importante ressaltar que o **git init** cria um repositório local, ou seja, ele não está associado a um repositório remoto no Github ou em qualquer outra plataforma. O repositório local é onde você fará os **commits** e gerenciará as alterações antes de compartilhá-las com outras pessoas.

Após criar o repositório com **git init**, você poderá adicionar arquivos com o **git add**, fazer **commits** com **git commit** e, se desejar, conectar seu repositório local a um repositório remoto no Github usando "**git remote add origin <URL_do_repositório_remoto>**".

O **git init** é um dos primeiros comandos que você deve executar ao iniciar um novo projeto com Git. Ele estabelece a base do seu repositório e permite que você comece a controlar as versões dos seus arquivos de forma eficiente.

Agora que você conhece o **git init**, experimente utilizá-lo para iniciar um novo repositório Git em seus projetos. Ele será o ponto de partida para todo o controle de versões e colaboração que o Git oferece.

Git status

O comando **git status** é uma ferramenta essencial no Git que permite verificar o **status** do seu repositório local. Ele fornece informações úteis sobre as alterações que você fez nos arquivos e o que você precisa fazer a seguir.

O **git status** também mostra informações adicionais, como a ramificação atual em que você está trabalhando e se há algo para ser enviado ao repositório remoto.

Utilizar o **git status** regularmente é uma prática recomendada, pois ajuda a manter o controle das alterações feitas em seus arquivos. Ele permite que você veja rapidamente quais arquivos foram modificados, quais estão prontos para serem confirmados e quais ainda não estão sendo rastreados pelo Git.

Lembre-se de que o **git status** é apenas uma ferramenta de visualização do estado do seu repositório. Ele não faz alterações ou confirmações por si só, mas fornece informações valiosas para você entender o que está acontecendo em seu projeto.

Quando você executa o comando **git status**, o Git verifica todos os arquivos no seu repositório e exibe três possíveis estados para cada um deles:

4. Arquivos não rastreados:

   São arquivos novos que o Git não está acompanhando. Eles podem ser novos arquivos que você acabou de criar ou arquivos existentes que você ainda não informou ao Git que deseja monitorar.

   Exemplo: Se você criar um novo arquivo chamado script.js e ainda não o adicionar ao Git, o **git status** mostrará esse arquivo como "não rastreado".

5. Arquivos modificados:

   São arquivos que foram alterados desde o último **commit**. O Git detecta as modificações nos arquivos e mostra quais foram alterados.

   Exemplo: Se você fizer alterações no arquivo index.html, o **git status** indicará que esse arquivo foi modificado.

6. Arquivos preparados para **commit**:

   Esses são os arquivos que você modificou e adicionou ao "**staging area**". O "**staging area**" é uma área intermediária onde você prepara os arquivos antes de confirmá-los definitivamente com um **commit**.

   Exemplo: Após usar o comando **git add index.html**, o arquivo será colocado no "**staging area**" e o **git status** informará que ele está "pronto para **commit**".

É importante observar que o **git status** é uma ferramenta valiosa para acompanhar o estado do seu repositório e garantir que você esteja ciente das alterações realizadas. Ele oferece uma visão geral das modificações

pendentes e permite tomar decisões informadas sobre os próximos passos, como confirmar as alterações com **git commit** ou adicionar arquivos ao índice com **git add**.

Lembre-se de executar o **git status** regularmente para estar ciente das modificações e do **status** do seu projeto. Isso ajudará você a manter um controle eficiente das suas alterações e evitar surpresas indesejadas.

Git pull

O comando **git pull** é uma ferramenta essencial no Git que permite atualizar o seu repositório local com as alterações feitas na nuvem (repositório remoto). Ele sincroniza o repositório local com as atualizações feitas por outros colaboradores do projeto ou com as alterações que você mesmo fez em outro computador.

O **git pull** é utilizado quando você precisa trazer as mudanças feitas por outras pessoas para o seu repositório local. É importante ressaltar que, antes de executar o **git pull**, é necessário ter feito o **git clone** do repositório remoto para criar uma cópia local em seu computador.

Quando você executa o comando **git pull**, o Git faz duas coisas:

1. Ele busca as alterações mais recentes do repositório remoto.
2. Ele mescla essas alterações com o seu repositório local.

Em alguns casos, pode haver conflitos durante a mesclagem devido às alterações conflitantes nos mesmos arquivos. O Git alertará sobre esses conflitos e permitirá que você os resolva manualmente antes de confirmar a mesclagem.

Lembre-se de que o **git pull** é uma ferramenta de atualização e mesclagem de alterações no repositório local. É importante utilizá-lo regularmente para manter o seu repositório sincronizado com as alterações feitas na nuvem.

Ao executar o comando **git pull**, o Git exibe informações sobre as alterações realizadas no repositório remoto e as alterações que foram mescladas com o seu repositório local. Ele também exibe informações sobre as alterações conflitantes que precisam ser resolvidas manualmente.

Utilizar o **git pull** é uma prática recomendada para manter o seu repositório atualizado com as últimas alterações feitas no projeto e evitar conflitos desnecessários.

Git add

O comando **git add** é utilizado no Git para adicionar modificações, novos arquivos ou diretórios ao índice do Git, também conhecido como "**staging area**". O "**staging area**" é uma área intermediária onde você prepara as alterações que serão incluídas no próximo **commit**.

O **git add** é uma etapa importante antes de fazer um **commit**, pois permite que você selecione as alterações específicas que deseja incluir no próximo snapshot do repositório.

Existem diferentes formas de utilizar o **git add**:

1. Adicionar um arquivo específico:

   Utilize o comando "**git add <nome-do-arquivo>**" para adicionar um arquivo específico ao "**staging area**". Por exemplo, para adicionar o arquivo "**script.js**", execute "**git add script.js**".

2. Adicionar todos os arquivos modificados:

   Utilize o comando "**git add .**" para adicionar todos os arquivos modificados ao "**staging area**". Isso inclui todos os arquivos que sofreram alterações desde o último **commit**.

3. Adicionar arquivos em um diretório específico:

   Utilize o comando "**git add <nome-do-diretório>/**" para adicionar todos os arquivos modificados dentro de um diretório específico ao "**staging area**". Por exemplo, para adicionar todos os arquivos modificados no diretório "**src/**", execute "**git add src/**".

O **git add** permite que você selecione cuidadosamente as alterações que deseja incluir no próximo **commit**. Isso é útil quando você está trabalhando em diferentes tarefas ou funcionalidades e deseja fazer **commits** separados para cada uma delas.

Após utilizar o **git add** para adicionar as modificações desejadas ao "**staging area**", você estará pronto para fazer um **commit** e registrar essas alterações no histórico do repositório.

Lembre-se de que o **git add** adiciona as alterações apenas ao "**staging area**", não ao repositório remoto. Para enviar as alterações para o repositório remoto, você precisará fazer um **commit** usando o comando **git commit**.

Git commit

O comando **git commit** é uma etapa crucial no fluxo de trabalho do Git. Ele é utilizado para confirmar as alterações feitas nos arquivos do seu repositório local, criando um novo ponto na história do projeto.

Ao fazer um **commit**, você está registrando as modificações que você fez nos arquivos, criando uma nova versão do seu projeto. Cada **commit** é acompanhado de uma mensagem descritiva que explica as alterações realizadas.

O **git commit** permite que você controle o histórico do seu projeto, facilitando a rastreabilidade e a colaboração com outros desenvolvedores. Cada **commit** representa um conjunto de alterações coesas e relacionadas que podem ser revisadas, revertidas ou mescladas posteriormente.

Ao realizar um **commit**, o Git verifica os arquivos que estão no "**staging area**" (área intermediária) e cria um novo ponto de controle, associando as modificações aos arquivos afetados.

É importante ressaltar que o **git commit** apenas registra as alterações no repositório local. Para enviar essas alterações ao repositório remoto (como o GitHub), é necessário utilizar o comando **git push**.

Ao executar o comando **git commit**, algumas práticas comuns são:

1. Utilizar uma mensagem de **commit** descritiva e concisa: A mensagem deve resumir as alterações feitas e fornecer informações úteis sobre o **commit**.
2. Separar as alterações em **commits** lógicos: É recomendado fazer **commits** atômicos, ou seja, cada **commit** deve conter alterações coesas e relacionadas. Isso facilita a revisão e o entendimento das alterações.
3. Utilizar uma convenção para as mensagens de **commit**: Algumas equipes seguem convenções específicas para as mensagens de **commit**, como prefixos indicando o tipo de alteração (ex: "feat:" para uma nova funcionalidade, "fix:" para correção de bug, etc.).

O comando **git commit** é essencial para o controle de versões e para manter um histórico claro e organizado das alterações em seu projeto. Utilize-o regularmente para registrar suas modificações e documentar o progresso do desenvolvimento.

Git checkout

O comando **git checkout** é uma ferramenta poderosa no Git que permite alternar entre **branches**, criar novas **branches**, restaurar arquivos e até mesmo revisitar **commits** anteriores.

A principal finalidade do **git checkout** é permitir que você navegue pelo histórico do seu repositório Git e trabalhe em diferentes pontos da linha do tempo do projeto. Com ele, você pode alternar entre **branches** existentes, criar novas **branches** para desenvolver recursos ou corrigir bugs, além de restaurar arquivos para versões anteriores.

Aqui estão alguns usos comuns do comando **git checkout**:

1. Alternar para uma branch existente:

   Utilize o comando "**git checkout <nome-da-branch>**" para mudar para uma **branch** específica. Por exemplo, "**git checkout develop**" irá trocar para a **branch** chamada "**develop**". Isso permite que você trabalhe em uma linha de desenvolvimento diferente do ramo principal.

2. Criar uma nova **branch** e alternar para ela:

   Use o comando "**git checkout -b <nome-da-nova-branch>**" para criar uma nova **branch** e imediatamente mudar para ela. Por exemplo, "**git checkout -b feature-novo-botao**" criará uma nova **branch** chamada "feature-novo-botao" e você será movido para essa nova **branch**.

3. Revisitar um **commit** anterior:

Com o comando "**git checkout <hash-do-commit>**", você pode revisitar um **commit** específico do histórico do seu repositório. Isso é útil quando você precisa verificar um estado anterior do projeto ou analisar o código em um ponto específico da linha do tempo.

4. Restaurar arquivos:

   Utilize o comando "**git checkout -- <nome-do-arquivo>**" para restaurar um arquivo para a versão mais recente commitada no repositório. Isso descarta as alterações não confirmadas do arquivo e o retorna ao estado do último **commit**.

5. Alternar entre **branches** com arquivos não confirmados:

   Quando você tem alterações não confirmadas em sua **branch** atual e deseja alternar para outra **branch**, pode usar o comando "**git checkout -f <nome-da-outra-branch>**". Isso força a mudança de **branch**, descartando todas as alterações não confirmadas.

O **git checkout** é uma ferramenta versátil e flexível para navegar e manipular o histórico do seu repositório Git. Ele permite que você trabalhe em diferentes contextos, explore **commits** anteriores e gerencie as mudanças nos arquivos.

É importante usar o comando **git checkout** com cuidado, pois ele pode descartar alterações não salvas. Certifique-se de ter salvado todas as alterações importantes antes de alternar entre **branches** ou revisitar **commits** anteriores.

Agora que você conhece o **git checkout**, aproveite ao máximo essa funcionalidade para explorar diferentes partes do seu projeto, criar novas **branches** e restaurar arquivos quando necessário.

Git branch

O comando **git branch** é usado para criar, listar, renomear e excluir **branches** em um repositório Git. Uma **branch** é uma linha de desenvolvimento independente, permitindo que você trabalhe em diferentes recursos, correções de bugs ou experimentos sem interferir diretamente no código da **branch** principal.

Ao criar uma **branch**, você está essencialmente criando uma cópia do seu repositório em um determinado ponto. Isso permite que você trabalhe em alterações específicas sem afetar o código da **branch** principal, geralmente chamada de "main" ou "master".

Veja alguns comandos comuns relacionados ao git branch:

1. Criar uma nova **branch**:

   O comando "**git branch <nome-da-branch>**" cria uma nova branch com o nome especificado. Por exemplo, "**git branch feature-novo-botao**" cria uma **branch** chamada "**feature-novo-botao**" a partir do ponto atual do repositório.

2. Listar as branches:

   O comando "**git branch**" lista todas as **branches** disponíveis no repositório. A **branch** atual é destacada com um asterisco (*).

3. Alternar para uma **branch**:

   O comando "**git checkout <nome-da-branch>**" permite alternar para uma **branch** específica. Por exemplo, "**git checkout feature-novo-botao**" mudará para a **branch** chamada "feature-novo-botao".

4. Renomear uma **branch**:

O comando "**git branch -m <nome-atual> <novo-nome>**" renomeia uma **branch** existente. Por exemplo, "**git branch -m feature-novo-botao feature-botao**" renomeará a **branch** "**feature-novo-botao**" para "**feature-botao**".

5. Excluir uma **branch**:

   O comando "**git branch -d <nome-da-branch>**" exclui uma **branch** que já foi mesclada em outra **branch**. Por exemplo, "**git branch -d feature-novo-botao**" excluirá a **branch** "**feature-novo-botao**" do repositório.

O **git branch** é uma ferramenta poderosa para organizar e controlar diferentes fluxos de trabalho em um projeto. Permite que você trabalhe em paralelo em diferentes recursos, isole correções de bugs e mantenha um histórico claro de cada etapa do desenvolvimento.

Ao utilizar o **git branch**, é importante ter em mente as seguintes considerações:

- Sempre crie uma nova **branch** para cada recurso ou tarefa separada que você estiver trabalhando. Isso facilitará o gerenciamento e a organização do seu código.
- Mantenha as **branches** atualizadas e faça fusões (**merges**) regulares com a **branch** principal para incorporar as alterações de outros colaboradores e manter uma base de código consistente.
- Evite criar muitas **branches** sem necessidade. Remova ou exclua as **branches** que não estão mais sendo utilizadas para manter o repositório limpo e organizado.

Lembre-se de utilizar o comando **git branch** para gerenciar suas **branches** de forma eficiente e organizar seu fluxo de trabalho no Git.

Git push

O comando **git push** é utilizado para enviar as alterações feitas no repositório local para um repositório remoto, como o GitHub. Ele sincroniza o seu repositório local com o repositório remoto, garantindo que todas as alterações sejam compartilhadas e estejam disponíveis para outros colaboradores.

Quando você realiza um **git push**, as alterações nos arquivos que foram confirmadas com o comando **git commit** são enviadas para o repositório remoto. Isso inclui novos **commits**, atualizações nos arquivos existentes e até mesmo exclusões.

Ao enviar as alterações para o repositório remoto, é necessário especificar a **branch** na qual você deseja fazer o **push**. Normalmente, a **branch** padrão é a **branch** principal, chamada "**main**" ou "**master**".

O **git push** é uma etapa essencial para a colaboração em projetos. Ele permite que outros membros da equipe acessem e visualizem as alterações feitas por você. Além disso, ao compartilhar o seu código com o repositório remoto, você cria um backup seguro e mantém um histórico completo de todas as modificações feitas.

É importante lembrar que, antes de executar o **git push**, é necessário ter realizado o **git commit** para registrar as alterações localmente. Assim, você garante que está enviando uma versão consistente do seu projeto.

Algumas considerações ao utilizar o **git push**:

1. Certifique-se de ter a permissão necessária para fazer o **push** para o repositório remoto. Em alguns casos, é necessário ser um colaborador autorizado ou o proprietário do repositório.

2. Verifique se você está no **branch** correto antes de executar o **git push**. Utilize o comando **git branch** para visualizar as **branches** disponíveis e **git checkout <nome-do-branch>** para alternar entre as **branches**.
3. Ao fazer o **git push** pela primeira vez para um repositório remoto, é possível que você precise autenticar sua conta para estabelecer a conexão segura entre o repositório local e o remoto. Siga as instruções fornecidas pelo Git para realizar a autenticação.

Lembre-se de realizar o **git push** regularmente para manter o repositório remoto atualizado e garantir que as suas alterações sejam compartilhadas com outros membros da equipe. Isso facilitará a colaboração e garantirá que todos tenham acesso às versões mais recentes do projeto.