Minicurso de Git

instrutor: Luís Henrique Rodrigues

Iniciando com o Git

" git config --global user.name "
&
" git config --global user.email "

O que faz: Informa qual o nome e email cadastrados globalmente no sistema, ou seja, qual o nome e email pertencentes aos commits realizados.

Por que: Identificação e Rastreabilidade do Histórico.

Quando usar: Uma única vez, após instalar o Git, antes do primeiro commit em qualquer projeto.

Sintaxe para cadastrar as informações

git config --global user.name "seuNome"

git config --global user.email "seuEmail"

" git init "

O que faz: Cria um novo repositório Git na pasta atual.

Por que: É o primeiro passo para começar a rastrear seu projeto com Git.

Quando usar: No início de um novo projeto, ou quando você quer adicionar controle de versão Git a um projeto existente (que ainda não é um repositório).

" git status "

O que faz: Mostra o estado de rastreamento dos arquivos: quais estão modificados, na área de staging ou não rastreados.

Por que: É o seu "painel de controle", mostrando o que está pronto para ser commitado.

Quando usar: Frequentemente, para verificar o status do seu diretório de trabalho antes de adicionar ou commitar.

"git log"

O que faz: Exibe o histórico de commits da branch atual em ordem cronológica reversa (do mais novo para o mais antigo).

Por que: Permite ver o registro completo de todos os commits, incluindo o hash (ID), autor, data e mensagem, facilitando a depuração e o entendimento do projeto.

Quando usar: Frequentemente, para inspecionar o histórico de alterações: o que foi feito, por quem e quando.

"git log --oneline"

Mostra o histórico de forma concisa e rápida, exibindo apenas o hash curto e a mensagem de cada commit em uma única linha.

" git add 'arquivo.tipo' "
ou
" git add . "

O que faz: Adiciona arquivos à área de staging (preparação) para o próximo commit.

Por que: O Git rastreia apenas o que está na área de staging. Isso dá controle sobre quais mudanças vão para o commit.

Quando usar: Depois de fazer modificações em um ou mais arquivos e antes de fazer o commit. Use **"."** (ponto final) para adicionar todas as modificações.

" git commit -m "Mensagem aqui" "

O que faz: Grava as mudanças que estão na área de staging no histórico do repositório..

Por que: Cria um ponto de restauração permanente e significativo no tempo do projeto. A mensagem explica o que foi feito.

Quando usar: Depois de preparar as mudanças (git add) e você tem um conjunto de trabalho finalizado e coerente.

" git branch
&
git branch nomeDaBranch "

O que faz: O primeiro comando lista todas as branch's disponiveis localmente. O segundo cria uma nova branch local.

Por que: Os ramos são essenciais para isolar o trabalho (novas funcionalidades, correções) sem interferir no código principal de produção

Quando usar: Quando você precisa iniciar um trabalho isolado (nova funcionalidade, correção de bug, etc) que não deve afetar o código principal até estar pronto.

"git checkout -b nomeDaBranch
OU
git switch -c nomeDaBranch"

O que faz: Os dois comandos criam e mudam para a branch especificada. O diferencial do "switch" é que ele é mais moderno.

Por que: Permite que você crie e já entre na branch que você criou, fazendo dois apenas um unico comando.

Quando usar: Quando você precisa alternar o foco para trabalhar em uma funcionalidade ou ramo diferente.

"git cherry-pick hash_do_commit"

O que faz: Aplica um commit específico de uma branch para a branch atual.

Por que: evitar merges desnecessários, correções rápidas (hotfixes) e ajuda a manter o log da branch de destino limpa fazendo com que apenas os commit's necessários sejam transferidos.

Quando usar: Quando você precisa mover uma correção de bug ou uma pequena funcionalidade isolada rapidamente de um branch para outro, sem mesclar (fazer merge) todo o histórico do branch de origem.

"git restore nomeDoArquivo"

O que faz: Desfaz as modificações não commitadas (não rastreadas) em um arquivo específico no seu diretório de trabalho, voltando ao estado do último commit

Por que: Quando você fez alterações em um arquivo e decide que não quer mantê-las, precisando descartar o trabalho recente naquele arquivo

Quando usar: Quando você quer descartar as modificações não commitadas em um arquivo, voltando ao estado do último commit

"git stash"

O que faz: Salva temporariamente as modificações (arquivos modificados e staged) do seu diretório de trabalho, retornando a branch ao estado do último commit.

Por que: Permite limpar o diretório de trabalho instantaneamente (como um "salvamento temporário") para que você possa mudar de contexto, sem perder as alterações que ainda não estão prontas para um commit.

Quando usar: Quando você está trabalhando, mas precisa interromper o trabalho para corrigir um bug urgente ou trocar de branch, e você não quer fazer um commit incompleto

"git stash list"

O que faz: Exibe todos os "salvamentos" temporários (stashes) que você criou.

Por que: Permite ver e gerenciar a lista de trabalhos que foram guardados temporariamente.

Quando usar: Para verificar quais stashes estão disponíveis antes de aplicar ou remover um.

"git stash pop"

O que faz: Reaplica as mudanças do último stash ao diretório de trabalho e o remove da lista.

Por que: Traz as alterações de volta e remove o stash, indicando que aquele salvamento temporário foi totalmente processado e não será mais necessário.

Quando usar: Ao finalizar a interrupção e retornar ao trabalho, quando você quer reaplicar as mudanças salvas.

"git stash apply"

O que faz: Reaplica as mudanças do último stash ao diretório de trabalho, mas mantém o stash na lista.

Por que: Permite que o stash seja reutilizado ou reaplicado em outros branches ou locais do repositório.

Quando usar: Quando você precisa aplicar o mesmo conjunto de alterações salvas em mais de uma branch.

"git reset --hard hash_do_commit"

O que faz: Move o HEAD e o branch atual para um commit anterior, descartando todas as alterações posteriores.

Por que: Permite voltar ao estado exato de um ponto seguro do projeto (o "ponto de restauração"), desfazendo tudo o que veio depois.

Quando usar: Quando o projeto está em um estado quebrado e você precisa descartar o trabalho recente para voltar a um commit funcional.

"git branch -D nomeDaBranch"

O que faz: Força a exclusão de uma branch local, mesmo que ele não tenha sido mesclado (incorporado) em outro branch.

Por que: Usar quando tiver que excluir uma branch que não está sendo mais usada ou que foi criada para fazer uma hotfix.

Quando usar: Quando você tem certeza que o branch não é mais necessário e deseja descartar seu histórico local, mesmo que ele não tenha sido mesclado.

Git com GitHub

"git clone URLdoRepositorio"

O que faz: Copia um repositório remoto inteiro (ex: do GitHub) para sua máquina local.

Por que: É o primeiro comando usado para começar a trabalhar em um projeto existente.

Quando usar: No início de um novo projeto que já está hospedado em um servidor remoto.

"git branch -M main"

O que faz: Força a renomeação do branch atual. Usado para mudar o nome de "master" para "main"

Por que: Renomear e padronizar: Altera o nome do branch principal

Quando usar: Quando você está no branch master (ou qualquer outro) e precisa renomeá-lo para main (ou outro nome).

"git remote add origin url-do-seu-repositorio"

O que faz: Cria uma conexão entre o seu repositório local e um repositório Git em um servidor (remoto).

Por que: Estabelecer a ponte de colaboração: Define o endereço para onde suas mudanças serão enviadas (**git push**) e de onde as mudanças de outros serão baixadas (**git pull**).

Quando usar: Após o git init em um novo projeto, para vincular sua cópia local de código à hospedagem na nuvem

"git push -u origin nomeDaBranch"

O que faz: Publica a branch local no remoto e configura o rastreamento

Por que: Define o destino padrão, permitindo que futuros comandos git push e git pull funcionem sem argumentos adicionais, indo diretamente para a branch remota correta.

Quando usar: No primeiro push de uma branch recém-criada, para que ela se associe à sua cópia remota

"git push origin HEAD:nomeDaBranch"

O que faz: Envia o estado atual do seu repositório local (representado por HEAD) para uma branch específica no repositório remoto.

Por que: Forçar um destino diferente: Permite enviar sua branch atual (local) para criar ou atualizar uma branch remota com um nome diferente, sem precisar trocar de branch localmente.

Quando usar: Quando você precisa enviar commits de uma branch **local** para uma branch **remota** com um nome diferente, ignorando a configuração de rastreamento

"git pull"

O que faz: Busca as mudanças do repositório remoto e as mescla (combina) com o seu branch local.

Por que: Garante que sua cópia local esteja atualizada com as últimas alterações dos colegas de equipe.

Quando usar: Antes de começar a trabalhar em uma nova tarefa e periodicamente para evitar conflitos grandes.

"git pull --rebase"

O que faz: Baixa as alterações do repositório remoto e reaplica seus commits locais no topo dessas novas mudanças.

Por que: Ele garante que o seu histórico de commits permaneça em uma linha reta, evitando a criação de commits de merge desnecessários.

Quando usar: Ao sincronizar sua branch de trabalho com o remoto, quando você já tem commits locais e quer integrar as mudanças de forma limpa, dando a impressão que seu trabalho foi feito por último.

"git merge nomeDaBranch"

O que faz: Combina o histórico de um branch (ex: feature-x) no seu branch atual (ex: main)

Por que: Incorpora o trabalho finalizado de um branch secundário de volta ao branch principal.

Quando usar: Depois que uma funcionalidade estiver pronta e testada, para trazê-la para o branch principal.

"git push origin --delete nomeDaBranch"

O que faz: Exclui um branch do repositório remoto

Por que: Limpar o ambiente remoto, removendo branches que já foram mesclados ou que são obsoletos, mantendo o servidor organizado.

Quando usar: Depois de mesclar e excluir um branch localmente, este comando é usado para limpar a cópia remota dele.

Conventional Commits

Tipos

feat: Usada quando é implementada um novo recurso para o usuário

fix: Indica uma correção de bug

docs: Alteração apenas na documentação do projeto

style: Modificações que não afetam o significado do codigo (espaços em branco, formatação, ponto e virgulas ausentes, trocar a cor do botão, ect)

refactor: Alteração de código que não corrige um bug nem adiciona um recurso

perf: Alteração de código que melhora o desempenho

test: Adição de testes ou correção dos existentes

chore: Outras alterações que não afetam o código-fonte ou arquivos de teste, como atualizações de dependências e mudanças em ferramentas de build.

Exemplos de mensagem

feat: adicionar botão de login na página inicial

fix(login): corrigir erro ao autenticar usuário

docs(readme): atualizar instruções de instalação e requisitos

style: aplicar padronização de aspas duplas no código

refactor: mover arquivos de configuração para /config

chore: adicionar pasta 'dist' ao .gitignore