Guia para subir um projeto ao GitHub, evitando erros de permissão, renomeando branches e garantindo que tudo funcione sem problemas. Siga os passos abaixo:

Indice

* **Passo a Passo para Subir um Projeto ao GitHub** 
  + **1. Preparar o Projeto Local**
  + **2. Inicializar o Repositório Git**
  + **3. Renomear a Branch (se necessário)**
  + **4. Limpar o Histórico (Opcional)**
  + **5. Vincular ao Repositório Remoto**
  + **6. Autenticação sem Erros 403**
  + **Opção 2: Usar SSH (Recomendado)**
  + **7. Fazer o Primeiro Push**
  + **Solução de Problemas Comuns**
  + **Resumo dos Comandos Essenciais**
  + **Exemplo Prático**
* **Tutorial: Como Voltar um Projeto a uma Versão Anterior no Git**
  + **1. git checkout (Inspecionar uma versão antiga)**
  + **2. git revert (Desfazer alterações com segurança)**
  + **3. git reset (Reescrever histórico local)**
* **Configurações Úteis no Git para Produtividade (Detalhadas)**
  + **1. Rebase Automático ao Fazer Pull**
  + **2. Aliases para Comandos Frequentes**
  + **3. Arquivo Global de Ignorados**
  + **4. Autocorreção de Comandos**
  + **5. Editor Padrão para Commits**
  + **6. Assinatura de Commits com GPG**
  + **7. Diferença Visual com Ferramentas Externas**
* **Dicas Avançadas de Produtividade**

**Passo a Passo para Subir um Projeto ao GitHub**

**1. Preparar o Projeto Local**

* **Caso seja um projeto novo**:

bash

mkdir meu-projeto && cd meu-projeto

* **Caso já exista um projeto**:

bash

cd caminho/do/projeto

**2. Inicializar o Repositório Git**

bash

git init

**3. Renomear a Branch (se necessário)**

* Verifique o nome da branch atual:

bash

git branch

* Se for um um projeto novo, sem nenhum commit, faça as etapas até o primeiro commit e então será criado a Branch de nome “master”;
* Se a branch for master (antigo padrão), renomeie para main:

bash

git branch -M main

**4. Limpar o Histórico (Opcional)**

Se deseja **subir o projeto em um novo repositório sem histórico anterior**:

bash

rm -rf .git # Remove a pasta .git (cuidado: apaga todo o histórico local!)

git init # Reinicializa o Git

git add .

git commit -m "Primeiro commit"

**5. Vincular ao Repositório Remoto**

* Crie um **novo repositório vazio no GitHub** (sem README ou .gitignore).
* Copie o URL do repositório (HTTPS ou SSH). Exemplo:  
  https://github.com/seu-usuario/meu-projeto.git
* Adicione o remote ao projeto local:

bash

git remote add origin https://github.com/seu-usuario/meu-projeto.git

**6. Autenticação sem Erros 403**

**Opção 1: Usar Token de Acesso (PAT)**

1. Gere um **Personal Access Token (PAT)** no GitHub:  
   Settings > Developer Settings > Personal Access Tokens (Classic) > Marque repo.
2. Use o PAT como senha ao fazer o push.

**Opção 2: Usar SSH (Recomendado)**

1. Gere uma chave SSH:

bash

ssh-keygen -t ed25519 -C "seu-email@github.com"

1. Adicione a chave pública ao GitHub:  
   Settings > SSH and GPG Keys > Cole o conteúdo de ~/.ssh/id\_ed25519.pub.
2. Altere o remote para SSH:

bash

git remote set-url origin git@github.com:seu-usuario/meu-projeto.git

**7. Fazer o Primeiro Push**

bash

git add .

git commit -m "Primeiro commit"

git push -u origin main

**Solução de Problemas Comuns**

**Erro 403 (Permission Denied)**

* **Credenciais salvas**: Limpe-as:
  + Windows: Gerenciador de Credenciais > Remova entradas do GitHub.
  + Linux/Mac:

bash

git config --global --unset credential.helper

git credential-cache exit

* **Token ou SSH**: Use PAT ou configure SSH (Passo 6).

**Branch Não Encontrada**

* Verifique o nome da branch local (git branch).
* Force a criação da branch no remoto:

bash

git push -u origin main

**Projeto com Histórico Indesejado**

Para começar do zero em um novo repositório:

cd meu-projeto

rm -rf .git # Apaga o histórico local

git init

git add .

git commit -m "Novo início"

git remote add origin URL\_DO\_NOVO\_REPOSITORIO

git push -u origin main

**Resumo dos Comandos Essenciais**

**Ação Comando**

**Iniciar repositório git init**

**Renomear branch git branch -M main**

**Adicionar remote git remote add origin URL**

**Primeiro push git push -u origin main**

**Limpar histórico rm -rf .git**

**Exemplo Prático**

bash

# Passo 1: Preparar o projeto

cd ~/projetos/meu-app

# Passo 2: Inicializar Git

git init

# Passo 3: Renomear branch

git branch -M main

# Passo 4: Adicionar arquivos e commit

git add .

git commit -m "Versão inicial"

# Passo 5: Vincular ao GitHub

git remote add origin git@github.com:seu-usuario/meu-app.git

# Passo 6: Push (usando SSH)

git push -u origin main

Pronto! Seu projeto estará no GitHub sem erros de permissão. 🚀

**Tutorial: Como Voltar um Projeto a uma Versão Anterior no Git**

Voltar a uma versão anterior no Git envolve diferentes abordagens, dependendo do cenário. Aqui estão as principais:

**1. git checkout (Inspecionar uma versão antiga)**

* **Uso ideal**: Para explorar um commit antigo sem modificar o histórico.

bash

git log --oneline # Encontre o hash do commit desejado (ex: a1b2c3d)

git checkout a1b2c3d # Acesse o commit

git checkout -b novo-branch # Crie um branch para trabalhar nessa versão

**2. git revert (Desfazer alterações com segurança)**

* **Indicado para branches compartilhados** (ex: main ou develop), pois preserva o histórico.

Bash

git revert a1b2c3d # Cria um novo commit desfazendo o commit especificado

git revert oldest-commit^..newest-commit # Reverte um intervalo de commits

**3. git reset (Reescrever histórico local)**

* **Use com cautela**: Altera o histórico local. Ideal para branches não compartilhados.

bash

git reset --soft a1b2c3d # Mantém alterações no staging area

git reset --mixed a1b2c3d # Mantém alterações no working directory (padrão)

git reset --hard a1b2c3d # Descarta todas as alterações após o commit

**Configurações Úteis no Git para Produtividade (Detalhadas)**

**1. Rebase Automático ao Fazer Pull**

Evite merges desnecessários e mantenha o histórico linear:

bash

git config --global pull.rebase true

* **Descrição**: Configura o Git para usar rebase ao invés de merge durante git pull.
* **Caso de Uso**: Ideal para branches locais onde você quer um histórico limpo.
* **Nota**: Em caso de conflitos, resolva-os durante o rebase e continue com git rebase --continue.

**2. Aliases para Comandos Frequentes**

Simplifique comandos repetitivos no .gitconfig:

bash

git config --global alias.st "status -s" # Status resumido

git config --global alias.co "checkout" # Atalho para checkout

git config --global alias.br "branch" # Listar branches

git config --global alias.cm "commit -m" # Commit com mensagem rápida

git config --global alias.lg "log --graph --abbrev-commit --pretty=format:'%C(yellow)%h%Creset -%C(red)%d%Creset %s %Cgreen(%cr) %C(bold blue)<%an>%Creset'"

* **Uso do git lg**: Exibe o histórico em formato gráfico e colorido.
* **Exemplo de Saída**:

\* a1b2c3d - (HEAD -> main) Adiciona feature X (2 hours ago) <João>

\* d4e5f6a - Corrige bug Y (4 hours ago) <Maria>

**3. Arquivo Global de Ignorados**

Configure um .gitignore\_global para ignorar arquivos em todos os projetos (ex: .DS\_Store, .log):

bash

touch ~/.gitignore\_global # Crie o arquivo

git config --global core.excludesfile ~/.gitignore\_global

* **Conteúdo Sugerido**:

# IDEs/Editores

.vscode/

.idea/

# Sistemas Operacionais

.DS\_Store

Thumbs.db

# Logs e Dependências

\*.log

node\_modules/

**4. Autocorreção de Comandos**

Habilite sugestões automáticas para erros de digitação:

bash

git config --global help.autocorrect 20 # 20 décimos de segundo para executar a correção

* **Exemplo**: Se você digitar git comit, o Git executará git commit após 2 segundos.

**5. Editor Padrão para Commits**

Defina seu editor preferido para mensagens de commit:

bash

git config --global core.editor "code --wait" # VS Code

# Alternativas:

# git config --global core.editor "nano" # Nano

# git config --global core.editor "vim" # Vim

**6. Assinatura de Commits com GPG**

Adicione verificação de autenticidade aos commits:

bash

git config --global commit.gpgsign true # Habilita assinatura

git config --global user.signingkey "SUA\_CHAVE\_GPG\_AQUI" # Chave GPG

* **Como Gerar uma Chave GPG**:

bash

gpg --full-generate-key # Siga as instruções no terminal

gpg --list-secret-keys --keyid-format LONG # Liste suas chaves

**7. Diferença Visual com Ferramentas Externas**

Configure o Git para usar seu editor/IDE para resolver conflitos:

bash

git config --global diff.tool vscode

git config --global difftool.vscode.cmd "code --wait --diff $LOCAL $REMOTE"

* **Uso**:

bash

git difftool HEAD~1 # Compara com a versão anterior

**Dicas Avançadas de Produtividade**

**Rebase Interativo**

Reescreva o histórico de commits localmente:

bash

git rebase -i HEAD~3 # Edite os últimos 3 commits

* **Opções durante o rebase**:
  + squash: Combina commits.
  + reword: Altera mensagens.
  + edit: Modifica um commit.

**Stash Inteligente**

Salve alterações temporárias e aplique-as posteriormente:

bash

git stash push -m "Mensagem descritiva" # Salva com uma mensagem

git stash list # Lista todos os stashes

git stash apply stash@{0} # Aplica um stash específico

**Push Padrão com Tracking**

Defina o branch atual como padrão para git push:

bash

git push -u origin main # Configura o tracking

git config --global push.default current # Define o padrão para todos os projetos

**Habilite Cores no Terminal**

Melhore a legibilidade de comandos como git status e git diff:

bash

git config --global color.ui true

**Resumo das Configurações Essenciais**

| **Comando** | **Descrição** |
| --- | --- |
| git config --global pull.rebase true | Evita merges desnecessários. |
| git config --global alias.st "status -s" | Status resumido. |
| git config --global core.excludesfile ~/.gitignore\_global | Ignora arquivos globalmente. |
| git config --global help.autocorrect 20 | Corrige comandos digitados errado. |

Com essas configurações, você otimiza seu fluxo de trabalho, mantém o histórico limpo e reduz a chance de erros. 🚀

New chat