Exercícios de Fixação sobre Git e GitHub

Esses exercícios têm como objetivo ajudar a fixar os conceitos sobre Git e GitHub, essenciais para o controle de versões e colaboração em projetos de desenvolvimento de software. Responda os seguintes exercícios para treinar e testar seus conhecimentos.

**Exercício 1: Configuração Inicial do Git**

1. Como configurar seu nome e e-mail no Git? Escreva os comandos e explique a importância de cada um.

git config --global user.name "Seu Nome"

git config --global user.email "seuemail"

git config --global --list

só usar esses comandos para configurar seu email no git.

**Exercício 2: Comandos Básicos do Git**

2. O que faz o comando 'git init'? Explique a diferença entre 'git init' e 'git clone'.

O comando git init é utilizado para inicializar um repositório Git vazio em um diretório existente.

git init cria um repositório vazio local.

git clone copia um repositório existente (geralmente remoto) e cria uma cópia local com todos os arquivos e histórico.

**Exercício 3: Criando e Mudando de Branch**

3. Como você criaria uma nova branch chamada 'desenvolvimento' e mudaria para ela? Escreva os comandos e explique o que cada um faz.

git branch desenvolvimento -Esse comando é usado para criar, listar e excluir branches no Git.

git checkout desenvolvimento -Este comando é usado para mudar de branch no Git. Ele altera a sua área de trabalho para refletir o estado da branch especificada.

git switch -c desenvolvimento -Este comando foi introduzido para tornar o processo de troca de branches mais simples e direto.

**Exercício 4: Adicionando e Commitando Arquivos**

4. Após criar um arquivo de texto, como você o adicionaria ao Git e faria um commit? Escreva os comandos e explique o que acontece após cada comando.

git status - Isso mostrará os arquivos não rastreados.

git add . - Para adicionar todos os arquivos não rastreados.

git commit -m "Adiciona" -Pode realizar o commit com uma mensagem do que foi feito.

git log - Para garantir que o commit foi feito corretamente.

**Exercício 5: Trabalhando com Repositórios Remotos**

5. Explique os comandos 'git remote add', 'git push', 'git pull' e 'git fetch'. Quando e por que devemos usar cada um deles?

git remote add - Usar quando você está criando um repositório local que ainda não está conectado a um repositório remoto. O porque usar: Para associar seu repositório local a um repositório remoto, permitindo a sincronização de mudanças entre eles.

git push - Usar após fazer commits no seu repositório local e querer compartilhar ou armazenar as alterações no repositório remoto. Usar porque Para atualizar o repositório remoto com suas mudanças locais.

git pull - Quando você quer obter as mudanças mais recentes de um repositório remoto e usar porque é para garantir que seu repositório local esteja sincronizado com as últimas alterações feitas.

git fetch -Usar Quando você quer verificar se há atualizações no repositório remoto, mas não quer integrar essas mudanças automaticamente na sua branch local e usar porque é para baixar as últimas alterações do repositório remoto sem modificar seu código local diretamente.

**Exercício 6: Resolvendo Conflitos**

6. O que é um conflito de merge? Como você resolveria um conflito de merge entre duas branches no Git?

Um conflito de merge ocorre quando o Git não consegue automaticamente integrar mudanças feitas em diferentes branches.

Como resolver:

Tentar fazer o merge com git merge <nome-da-branch>.

Verificar os arquivos conflitantes com git status.

Abrir os arquivos conflitantes e resolver manualmente os conflitos.

Adicionar os arquivos resolvidos ao staging com git add.

Fazer o commit do merge com git commit.

Verificar se o merge foi resolvido corretamente com git status e git log.

**Exercício 7: Usando GitHub**

7. Como você cria um repositório no GitHub e o conecta ao seu repositório local? Escreva os comandos necessários para isso.

git init

git add .

git commit -m ""

git remote add origin "link do git com o repositório"

git push -u origin main

**Exercício 8: Histórico de Commits**

8. Como você visualiza o histórico de commits no Git? Quais comandos você usaria para ver o histórico de commits e detalhes de um commit específico?

git log

git log -n

**Exercício 9: Trabalhando com Forks**

9. Explique o que é um fork no GitHub e qual é o fluxo de trabalho típico ao trabalhar com forks em projetos colaborativos.

No GitHub, fork é um processo que cria uma cópia independente de um repositório existente sob sua conta de usuário.

Fazer o fork do repositório no GitHub.

Clonar o fork para o seu ambiente local.

Adicionar o repositório original como remoto (upstream).

Sincronizar o fork com as mudanças do repositório original.

Criar uma nova branch para implementar funcionalidades ou correções.

Fazer alterações, commitar e enviar para o seu fork.

**Exercício 10: Revertendo Commits**

10. O que o comando 'git revert' faz? Qual a diferença entre 'git revert' e 'git reset'?

git revert - é utilizado para criar um novo commit que desfaz as alterações feitas por um commit anterior.

git reset - é usado para mover a referência do HEAD e da branch atual para um commit anterior.

diferença:

git revert:

Cria um novo commit que desfaz as alterações de um commit anterior.

Não altera o histórico de commits.

Seguro para repositórios compartilhados, pois preserva o histórico.

Não afeta arquivos no diretório de trabalho.

Usado para desfazer mudanças em commits já compartilhados sem afetar os outros colaboradores.

git reset:

Move o HEAD para um commit anterior, podendo remover commits do histórico.

Pode alterar o histórico de commits.

Pode afetar o diretório de trabalho dependendo da opção usada (--soft, --mixed, --hard).

Perigoso em repositórios compartilhados, pois pode causar perda de dados se usado em commits já compartilhados.