Revisão de estudos Git

Tags

Aprendendo sobre Git e Github

FLUXO BÁSICO DE TRABALHO COM GIT

1. Cria diretório;
2. Inicia o git dentro do diretório que foi criado anteriormente → **git init .**
3. Edita ou cria os arquivos
4. Valida se o status da alterações → **git status**
5. Por vias de boa prática, crie uma branch alternativa utilizando → **git checkout -b “nome_branch_alt”**, dessa forma podemos fazer commits para um branch alternativa e só depois avaliar através do PR se podemos levar as alterações para a branch main.
6. Adiciona os arquivos a fila de commit → **git add <nome_do_arquivo>**
7. Valida novamente se efetuou a adição anterior → **git status**
8. Efetua o commit da fila de commit organizada → **git commit -m “nome_do_commit”**;
9. Valida novamente se o commit foi corretamente → **git status**
10. Caso tenha um repositório local e queira conectar com um repositório remoto (GitHub por exemplo), use o comando → **git remote add origin <link do repositorio>**
11. Em seguida após conecta-lo faça o push para a branch alternativa → **git push -u origin <nome da branch secundaria>**
12. Verifique a alteração no seu diretório GitHub ou outro que estiver utilizando, e execute o ‘compare & pull request’ o famoso PR
13. Após criar o pull request no GitHub, tendo feito tudo corretamente precisará apenas executar um merge do PR e seus últimos ajustes no

código estarão devidamente versionados na sua main branch do seu projet

RESUMO GERAL DE COMANDOS GIT

1. O que é Commit?
    a. Fazendo uma analogia é como se fosse um save progress ou checkpoint. Nos quais você pode retornar a cada um desses checkpoints quando achar necessário. Comitou? Criou um checkpoint e assim por diante. Dessa forma, criando versões do código ou seja, o versionamento.
2. Comandos básicos:
    a. cd → Muda de diretório, exemplo: cd Desktop/ ← muda o diretório para o caminho da pasta Desktop;
    b. ls → Lista os itens contidos dentro do diretório que foi determinado, como por exemplo o desktop;
    c. pwd → Retorna o caminho completo de onde você está (print work director);
    d. ls -a → Mostra todos os itens contidos incluindo os ocultos;
    e. cd .. → Desloca para a pasta anterior de onde você está atualmente;
    f. cd . → É a própria pasta onde você está;
    g. mkdir → Cria um nova pasta chamada ‘nome_pasta’, exemplo: mkdir nome_pasta
    h. cat → Exibe o que tem contido dentro de qualquer arquivo editável, exemplo: cat nome_arquivo.txt
    i. git init . → cria um repositório git vazio dentro do caminho
    j. rm -rf . git/ → apaga a pasta/repositório vazio caso haja alguma erro, etc.
        i. -rf é um acrescimo de comando que força apagar sem nenhum tipo de interrupção, ou seja, rf = recursão force. Ou seja, cuidado isso pode apagar qualquer coisa sem ter o que fazer. * **CUIDADO AO UTILIZAR***
    k. git status → Dá um resumo geral da branch utilizada

    i. untracked files → são arquivos que ainda não foram comitados ou seja não há versionamento.

l. git add → Adiciona o arquivo que eu quero versionar com o commit. Exemplo: git add nome_arquivo.txt , aqui você adiciona esse arquivo no commit.

m. git commit -m → Fazer um commit e nomea-lo por exemplo: git commit -m “comandos bash”

n. git config —global user.email “email@.com.br” - > assim você seta qual email/user sera definido para todos os repositórios.

o. git config —global user.email ou git config —global user.name → verifica qual email ou nome está setado atualmente.

p. git log → Verifica os logs de todos os commits já feitos

q. . → diretório atual

r. .. → diretório anterior ao atual

s. * → se refere a tudo. Por exemplo: ‘git add comandos_*’ , esse comando adicionaria para fila de commit todo os arquivos dentro ta pasta atual q tenham nome q contenham ‘comandos_’.

t. git diff → mostra a diferença ou alterações entre commits, por exemplo: ‘git diff comandos_git.txt’

u. git reset → Altera o status de um arquivo de staged para unstaged, ou seja, remove ele da fila de add para o proximo commit. Exemplo: git reset ‘nome_arquivo.txt’

v. git checkout -b → Cria uma branch ou ramificação da branch main ou principal, exemplo: git checkout -b ‘nome_branch_secundaria’.
p.s: Utiliza-se o termo -b é só quando a branch não existe ainda.

w. git branch → Exibe quais as branchs existentes e qual branch você está no momento.

x. git merge → Unifica as duas branchs, exemplo: git merge secundaria.

    i. Vale notar que, se fizemos merge de uma branch **secundaria_1** com a main e depois um merge de outra branch **secundaria_2** também com a main, tanto a branch **secundaria_1** como a branch **secundaria_2** estarão inalteradas, pois só fizemos as alterações

presentes em cada branch para main e não unindo as duas branchs secundarias antes de mergear com a main.

ii. a ideia tb é sempre sua branch pessoal fazer merge com a main pra levar as alteracoes da SUA BRANCH, entao caso outra pessoa faça um merge entre a branch dela com a nova main branch (alterada pelo seu avanço) pode assim ocorrer um conflito.

y. git branch -d <nome_branch> → Apaga a branch selecionada.

z. git remote add origin → Conecta o repositório local com o repositório remoto no GitHub, Gitlab, etc.

aa. git remote remove origin → remove o link com repo pra vc trocar de projeto por exemplo

ab. git push - u origin <branch que quer enviar a alteração> → Esse comando envia todas as alterações da branch local da main para branch remota <origin>. Ou seja, envia as coisas alteradas no ultim commit para a repo remota.

i. **GIT PUSH ENVIA ALTERACOES DE X PARA, EXEMPLO: origin main, quer dizer que está enviando da main (local) para remota (origin).**

ac. touch <nome do arquivo> → criar uma arquivo vazio

ad. git pull - u origin <branch que quer puxar a alteração> → então você quer RECEBER alterações da origin.

ae. git clone <link do repositório> → clona todo o repositório para sua máquina local.