

Git GitHub

SEGUNDO SEMESTRE DE 2023

Informações Úteis

- Site Git
 - <https://git-scm.com/>
- Documentação Git
 - <https://git-scm.com/doc>

Configurações Git

Nome de Usuário e Email

- Nome do usuário
 - `git config --global user.name "nomedeusuario"`
- Email do usuário
 - `git config --global user.email seuemaildogithub`

Levando em consideração que o email, tem que ser o mesmo do GitHub. E para ver essas mesmas configurações, é só digitar os comandos:

- Nome do usuário
 - `git config user.name`
- Email do usuário
 - `git config user.email`

Mudar a Branch (master -> main)

Agora para ver e configurar a branch padrão (master) para "main" é com o comando:

- Mudando para "main"
 - `git config --global init.defaultBranch main` (para visualizar tira "global e main")

Visualizar Configurações

Caso queira visualizar as configurações que foram feitas, é só inserir o comando:

- Visualizando configurações
 - `git config --global --list`

Autenticação via Token

Gerando o Token

Segue o passo a passo para chegar na geração do token dentro do GitHub.

- Passos
 - Foto de perfil - Settings - Developer settings (itens do canto esquerdo) - Tokens (classic) - Generate new token (superior direito) - Generate new token (classic) - senha do github

Depois disso você será direcionado a uma página para configurar o seu token. Abaixo tera algumas configurações a fazer.

- Note
 - Um nome de para que vai servir aquele token exemplo (Git Github)
- Expiration
 - Tempo de validade do token
- Select Scopes
 - Selecionar as permissões que o seu token vai ter dentro do seu github.

Após isso é só gerar o token, e o mesmo será mostrado na página a seguir, porém cuidado, o token ficará visível apenas uma vez. Ele será utilizado como a nossa “senha” na hora de tentar clonar um repositório.

Salvando o token na máquina

Como esse token só ficará visível uma vez para nós, podemos estar salvando ele em nossa máquina com os seguintes comandos:

- Caso compartilhe sua máquina com alguém, salve **temporariamente**
 - `git config --global credential.helper cache`
- Caso somente você use a máquina, salve **permanentemente**
 - `git config --global credential.helper store`

Após fazer um dos dois comandos, você terá que passar o token para o git por algum meio (como o git clone), e depois não será mais necessário ficar passando o token.

Autenticação Chave SSH

Gerando a Chave

Segue o passo a passo para chegar na geração da chave dentro do GitHub.

- Passos
 - Foto de perfil - Settings - SSH and GPG keys (itens do canto esquerdo)

Antes de gerar a chave, você pode estar verificando a documentação referente a essa parte das Chaves SSH. Como a verificação se já existe uma chave cadastrada no seu git.

Check out our guide to [generating SSH keys](#) or troubleshoot [common SSH problems](#).

[Sobre o SSH](#)

[Usar o encaminhamento de agente SSH](#)

[Gerenciar chaves de implantação](#)

[Verificar se há chaves SSH](#)

[Gerando uma nova chave SSH e adicionando-a ao agente SSH](#)
[Adicionar uma nova chave SSH à sua conta do GitHub](#)
[Testar a conexão SSH](#)
[Trabalhar com frase secreta da chave SSH](#)

Você pode estar consultando a documentação de acordo com o que você está necessitando.

Criando e Clonando Repositórios

Criando Repositório Local

Para estarmos criando um repositório dentro da nossa máquina, podemos estar utilizando dentro da pasta que gostaríamos de versionar, o comando:

- `git init`

Depois disso precisamos atrelar esse repositório local, a um repositório remoto.

- `git remote add origin URL`

Clonando Repositório Git

Para estar clonando um repositório já existente no GitHub, utilizamos o comando:

- `git clone URL nome-da-pasta`

Comandos Úteis

- Ver repositórios remotos vinculados
 - `git remote -v`

Salvando Alterações no Repositório Local

Depois de termos modificado algum arquivo dentro do repositório local, temos que salvar esse arquivo.

- `git add *` (para salvar tudo) or `git add nome-do-arquivo`
- `git commit -m "mensagem para descrever o commit"`
- `git log` (para ter mais detalhes do commit feito)

Desfazendo Alterações no Repositório Local

Caso queiramos desfazer alguma alteração que fizemos em algum arquivo, por exemplo: tínhamos já salvo em um commit um txt com anotações dentro, mas abrimos o txt depois, e apagamos o conteúdo, como posso voltar ao que estava antes?

- `git restore nome-do-arquivo`

Editando Mensagem do último commit

Caso queremos editar a mensagem do último commit feito, usamos o comando:

- `git commit --amend -m "editando a mensagem do commit"`

Retornando a Commits Anteriores

Podemos utilizar alguns comandos, para voltar a algum commit já realizado por nós. Ou seja, os commits feitos posteriormente a esse commit que iremos voltar, irão ser apagados, e o commit atual, será o que foi escolhido por nós com o hash. Deixando claro que esses comandos devem ser usados antes de enviarmos as alterações para o repositório remoto, se não pode haver conflitos.

Soft

Esse comando irá voltar no commit escolhido, e os arquivos dele já estarão prontos para serem commitados.

-
- `git reset --soft hash-do-commit-que-queremos-retornar`

Mixed

Esse comando faz o mesmo do que o anterior, porém os arquivos terão que ser adicionados na área de preparação (`git add .`) para ser salvo o commit

- `git reset --mixed hash-do-commit-que-queremos-retornar`

Hard

Esse comando irá voltar no commit escolhido, porém vai apagar os arquivos daquele commit.

- `git reset --hard hash-do-commit-que-queremos-retornar`

Outro comando Útil com Reset

Caso queira retirar algum arquivo da área de preparação, pode estar utilizando o seguinte código:

- `git reset nome-do-arquivo`

Enviando e Baixando Alterações com Rep Remoto

Enviando alterações para repositório remoto

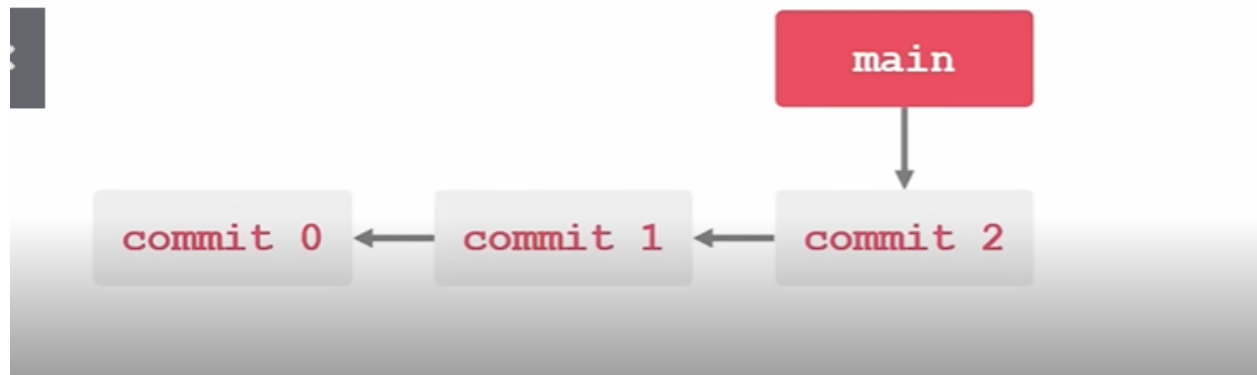
- `git push -u origin main`

Baixando alterações do repositório remoto

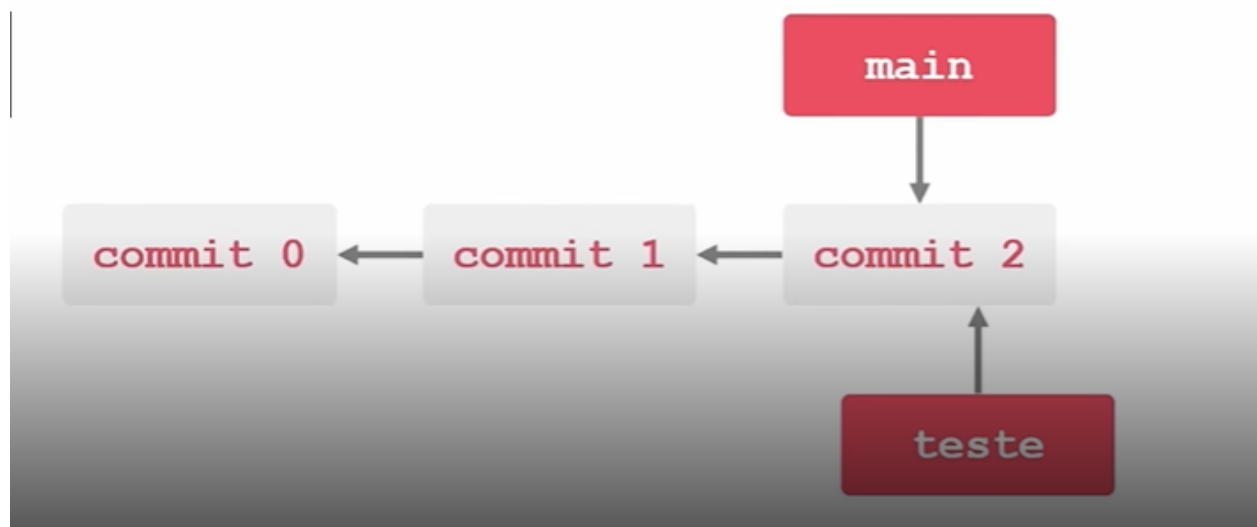
- `git pull`

Trabalhando com Branches

Trabalhando com Branches

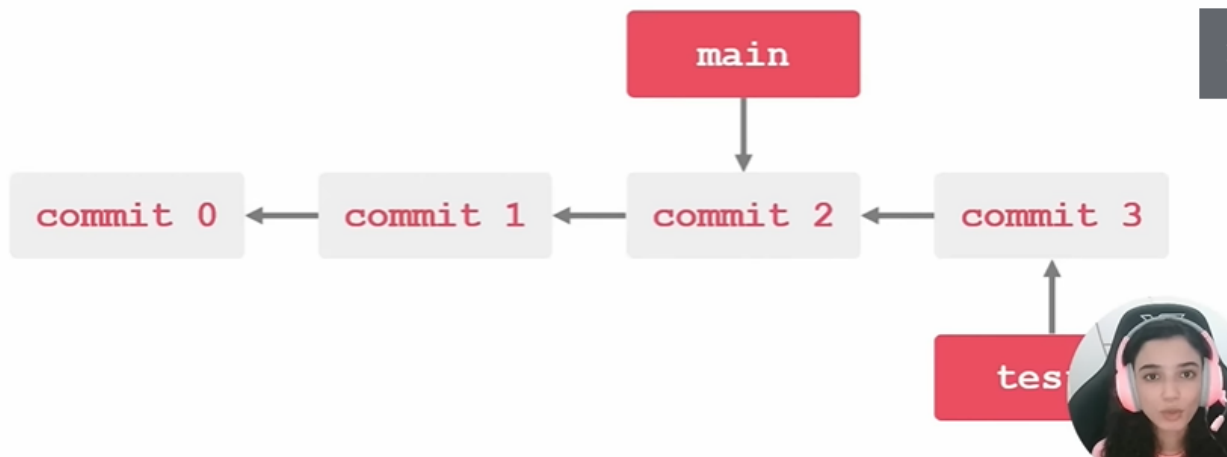


Trabalhando com Branches



Quando criada uma branch paralela, ela se inicia apontando para o mesmo commit que a branch main.

Trabalhando com Branches



A partir daí, a branch **teste** vai apontar para os novos commits, porém a branch **main**, vai permanecer do mesmo jeito, com isso pode se estar realizando testes em uma branch a parte, que não vai se relacionar com a branch principal.

Criando uma Branch paralela

- `git checkout -b teste(nome da branch)`

Com isso, estaremos trocando da branch que estamos, para a branch teste, o mesmo se faz para trocar para branch main novamente.

Se estamos então em uma nova branch, podemos trabalhar como se estivéssemos na branch principal, só que sem afetá-la.

Vendo últimos commits de cada branch

- `git branch -v`

Mesclando as branch

Caso seu teste ou seja lá o que for, tenha dado certo, e você quiser que as alterações na branch **teste**, seja passada para a branch **main**;

Estando dentro da branch **main** utilizamos:

- `git merge teste`

Excluindo Branch

Agora quando nao ha mais necessidade da branch teste, podemos excluí-la, primeiro podemos listar todas as branch que temos com o comando:

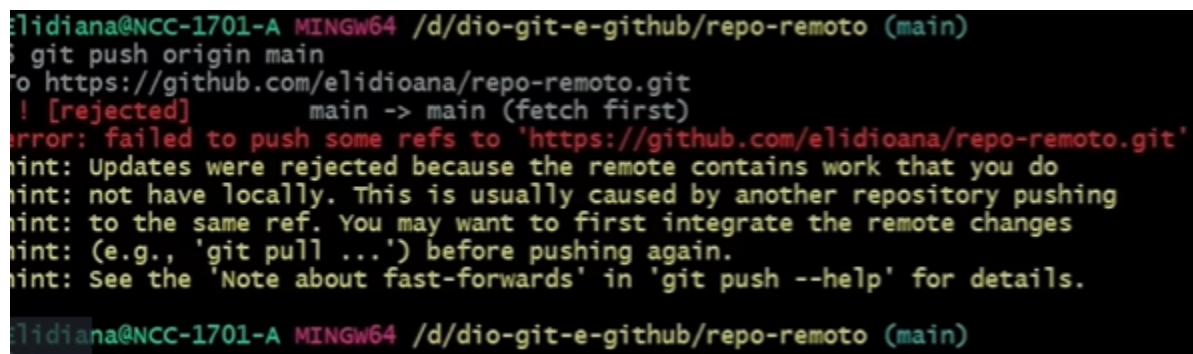
- `git branch`

Agora para excluir:

- `git branch -d teste`

Tratando Conflitos

Imagine que você e um amigo estão trabalhando no mesmo repositório, e os dois acabam alterando a mesma linha de código. E quando tenta enviar, acaba gerando um conflito com o outro que enviou.



```
elidiana@NCC-1701-A MINGW64 /d/dio-git-e-github/repo-remoto (main)
$ git push origin main
to https://github.com/elidioana/repo-remoto.git
! [rejected]        main -> main (fetch first)
error: failed to push some refs to 'https://github.com/elidioana/repo-remoto.git'
hint: Updates were rejected because the remote contains work that you do
hint: not have locally. This is usually caused by another repository pushing
hint: to the same ref. You may want to first integrate the remote changes
hint: (e.g., 'git pull ...') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
elidiana@NCC-1701-A MINGW64 /d/dio-git-e-github/repo-remoto (main)
```

Com isso utilizamos o:

- `git pull`

```
elidiana@NCC-1701-A MINGW64 /d/dio-git-e-github/repo-remoto (main)
$ git pull
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), 697 bytes | 11.00 KiB/s, done.
From https://github.com/elidioana/repo-remoto
   25ef81a..6fd16d3  main      -> origin/main
Auto-merging README.md
CONFLICT (content): Merge conflict in README.md
Automatic merge failed; fix conflicts and then commit the result.
```

Como podemos ver, ainda temos um conflito na penúltima linha.

```
<<<<<< HEAD
# Repositório Local

Descrição inserida no repositório local depois do clone.
=====
# Repositório Remoto Depois do Clone

Descrição inserida no repositório remoto depois do clone.
>>>>>> 6fd16d372b0443a0d50444a8bd0d7bcae275d1bb]
```

Com essa imagem, temos primeiramente alterações feitas na máquina local, mas que não foram aceitas pelo git, por conta do conflito; e depois alterações feitas no repositório remoto, que puxamos com o “pull”.

O que temos que fazer agora é decidir o que tem que ficar.

```
# Repositório Local

Descrição inserida no repositório local depois do clone.
```

Com isso, podemos salvar o arquivo, e enviar para o repositório remoto novamente.

Trabalhando com Branche - Comandos Úteis

Baixar alterações mas não mesclar

Caso você queira baixar as alterações do repositório remoto, mas ainda não queira mesclar com o repositório local pode estar se utilizando o fetch.

- git fetch origin main

Se utilizarmos o comando

- `git diff main(nossa branch local) origin/main(branch remota principal)`

Conseguimos ver a diferença entre as duas.

E caso queremos mesclar as duas agora

- `git merge origin/main`

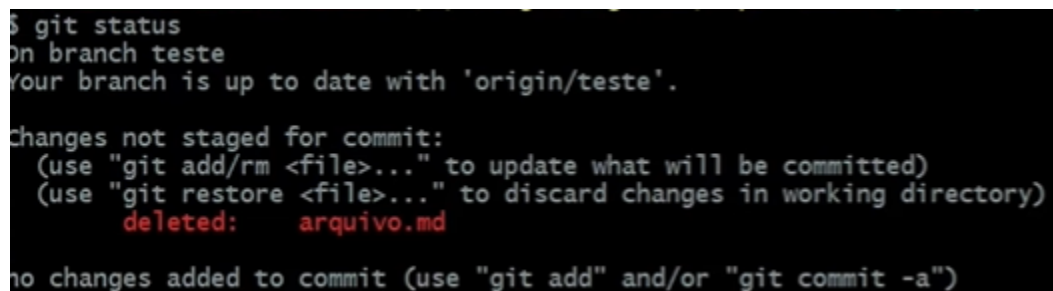
Clonar um repositório com várias branch

Caso precise clonar um repositório que há várias branch, podemos escolher apenas uma branch.

- `git clone URL --branch nome-da-branch --single-branch`

Comando Stash

Imaginemos que estamos em uma branch teste fazendo algumas alterações. Se utilizarmos o “`git status`” vai mostrar estas modificações.

A screenshot of a terminal window with a dark background and light-colored text. The output of the 'git status' command is shown. It indicates the current branch is 'teste' and that it is up to date with 'origin/teste'. Below this, it states 'changes not staged for commit:' and provides instructions on how to stage or discard changes. A specific change is listed as 'deleted: arquivo.md'. At the bottom, it says 'no changes added to commit (use "git add" and/or "git commit -a")'.

```
$ git status
On branch teste
Your branch is up to date with 'origin/teste'.

changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        deleted:    arquivo.md

no changes added to commit (use "git add" and/or "git commit -a")
```

Agora por algum motivo precisamos criar uma nova branch, mas não queremos enviar essas modificações.

Então podemos arquivar essa modificação da branch teste com o comando:

- git stash
- Listar modificações arquivadas
 - git stash list

Com isso podemos criar uma nova branch, e usá-la a nosso querer. Quando terminarmos de usá-la, e voltarmos para a branch teste, podemos “desarquivar” as modificações arquivadas com o comando:

- git stash pop

Caso queira voltar com as alterações arquivadas, e excluir as alterações recentes da branch. Ou:

- git stash apply

Caso seja apenas para trazer a modificação arquivada