STREAM SERIES

desvendando a ferramenta

Git para iniciantes

O que você deveria saber?

SERGIO CABRAL

sergiocabral.com

Índice

Este e-book ainda não foi terminado. Volte mais tarde! :)

Este conteúdo foi preparado para ser apresentado ao vivo na programação da EBAC sobre o tema Primeiros passos para se tornar um desenvolvedor.

Se tudo estiver certo, você consegue a versão atualizada desse *e-book* no link para download em git.sergiocabral.com/ebook.pdf

1. Perguntas e respostas

O que é o Git?

Um software de linha de comando conhecido como VCS ou SCM.

(D)VCS = (Distributed) Version Control System
SCM = Source Control Management

O que o Git faz?

Uma ferramenta que possibilita fazer alterações em um conjunto de arquivos e registrar cada alteração numa linha do tempo.

Por que preciso do Git?

Além de manter um histórico de versões, torna possível trabalhar com equipes distribuídas.

Lembre-se que o seu **eu** de amanhã será diferente do seu **eu** de hoje e vocês precisam trabalhar em equipe.

Como começo a usar o Git?

Você instala no modo *avançar-avançar* e transforma qualquer pasta em um repositório Git com o comando `git init`. Ou você pode usar o comando `git clone` para clonar um repositório existente.

Qual a relação do Git com GitHub?

São projetos independentes, mas são integrados. O GitHub é um servidor de repositórios, e o Git é um cliente para o servidor.

Como o GitHub tem no seu nome "Git" isso causa alguma confusão. Mas há outros serviços que se integram com o Git da mesma maneira, de modo que o GitHub não é a única escolha. Por exemplo:

- GitLab
- Bitbucket
- Azure DevOps

Por que alguns acham Git difícil de usar?

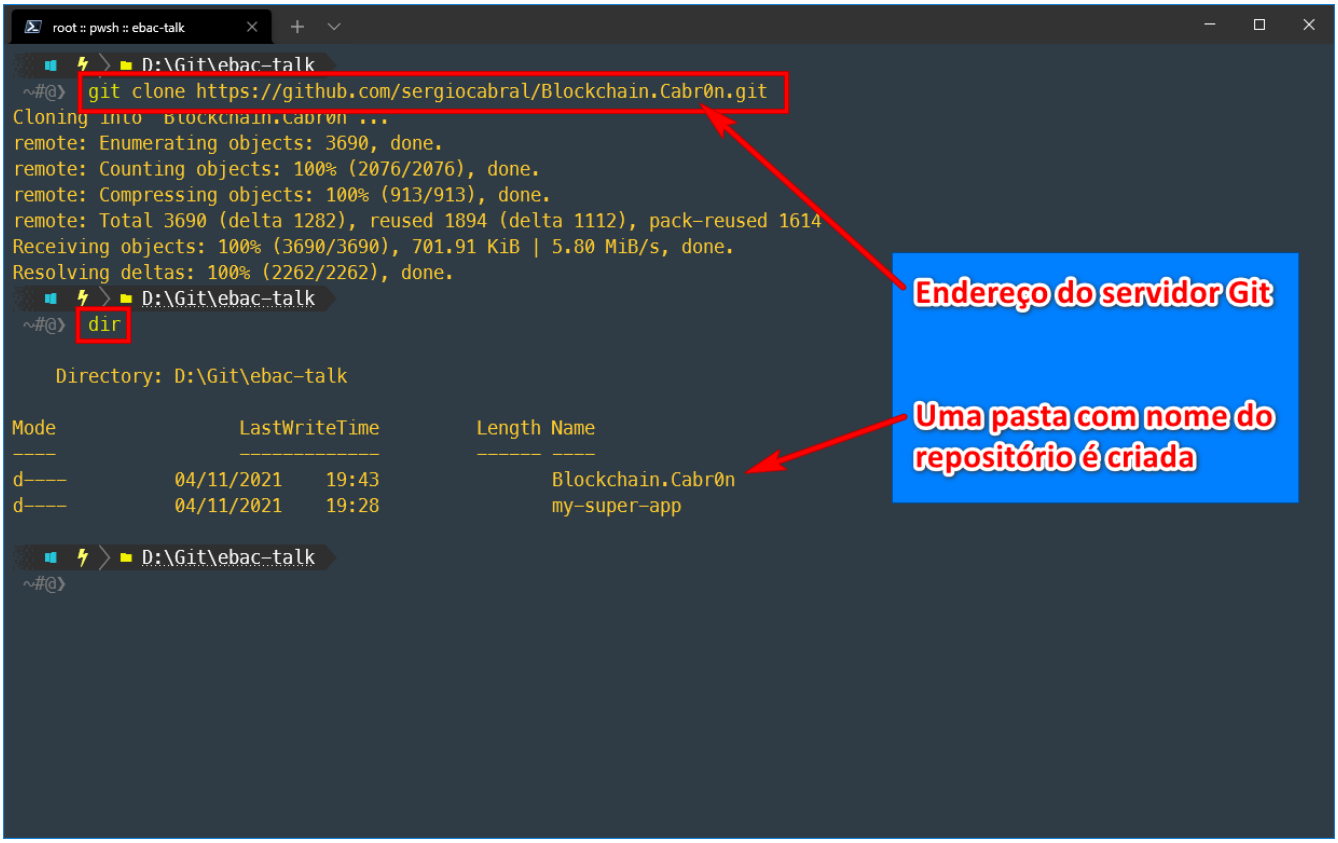
Porque o Git não tem interface gráfica; é uma ferramenta de linha de comando. Qualquer interface gráfica será uma camada intermediária entre o usuário e o Git.

2. Como criar ou baixar um repositório Git

Resultado do `git init`.

Exemplo 1.

Resultado do `git clone`.



Exemplo 2.

O Git grava seu banco de dados na pasta `.git`.

3. Comandos básicos

`git status`

Estou numa pasta gerenciada pelo Git?
Quais arquivos que **não estão** no repositório?

Para ignorar certos arquivos, adicione-o ao `.gitignore`.

`git add <nome-do-arquivo>`

Marca os arquivos que **vão entrar** no repositório.

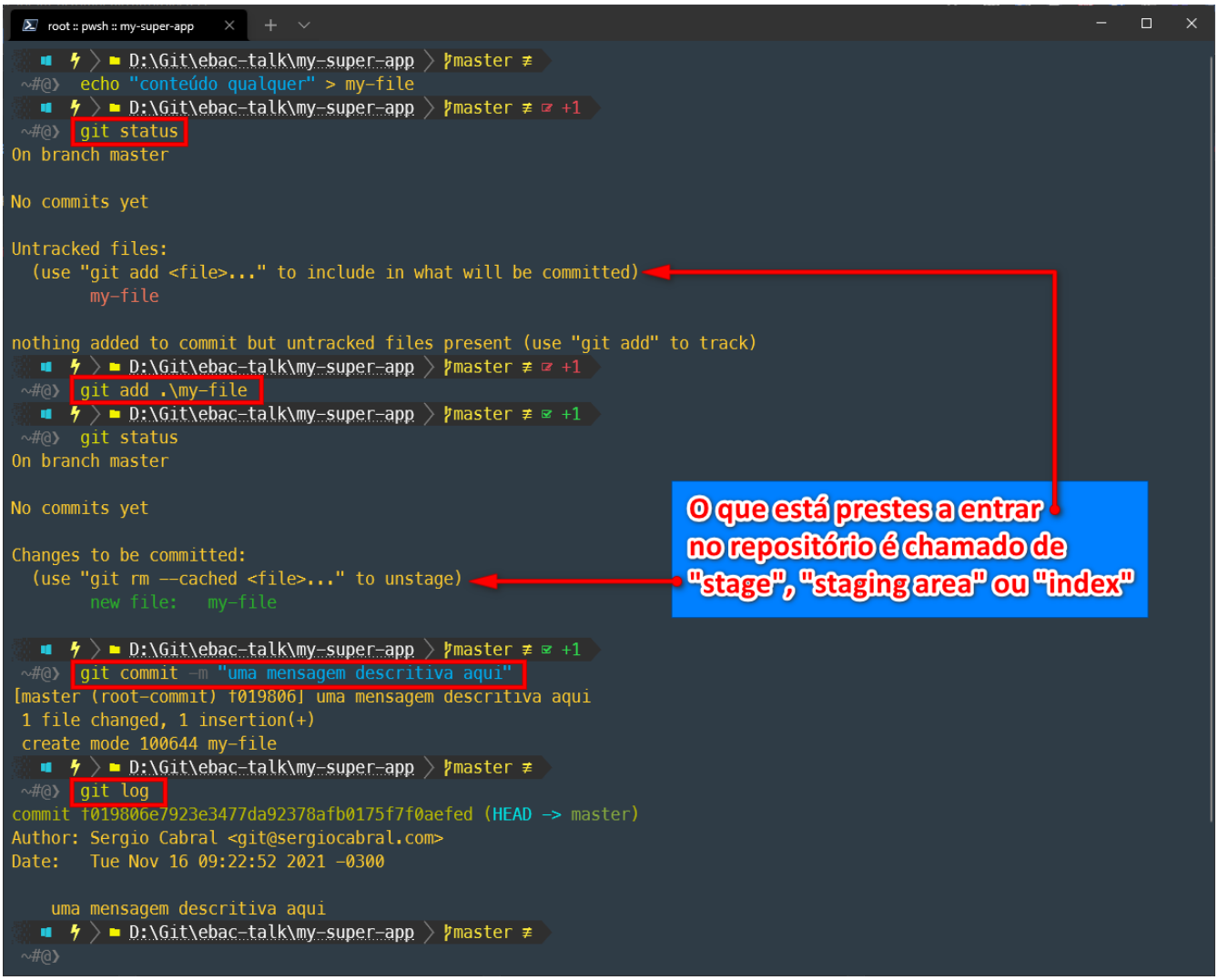
`git commit -m "<mensagem>"`

Faz com que os arquivos **entrem** no repositório.

O Git não faz *commit* de pastas, somente de arquivos.

`git log`

Mostra o histórico do que **entrou** no repositório.



Exemplo 3.

4. Onde estão os arquivos

Há três lugares onde seus arquivos podem estar.

Working Directory

O que está na pasta do seu projeto.
Gerenciado pelo sistema operacional.

Staging Area

O que vai para o repositório.
Próximo *commit* ainda não feito.

HEAD

O repositório.
Último *commit* feito e pai do próximo *commit*.

5. *Branches* e *Tags*

Commits sempre são um hash, que é uma sequência de 40 caracteres gerado com *SHA1*.
Por exemplo: `f019806e7923e3477da92378afb0175f7f0aefed`

Branches e *tags* são apenas nomes para *commits*.
Ao apagar *branches* ou *tags* nenhum dado é afetado ou perdido.

Criar *branches*

- `git branch <nome-do-branch>`
- `git checkout -b <nome-do-branch>`

Excluir *branches*

- `git branch -D <nome-do-branch>`

Quando se usa `-d` (minúsculo) o Git não vai excluir branches que apontam para commits que nunca sofreram *merge*.

Exemplo 4.

Criar *tags*

- `git tag <nome-da-tag> -m "<descri□□o-opcional>"`

Excluir *tags*

- `git tag -d <nome-da-tag>`

Branches e *tags* são semelhantes, mas a diferença está em que:

- Um *branch* acompanha cada *commit* feito a partir dele.
- Uma *tag* é fixada em um *commit* na corrente do tempo.

6. Navegando entre os *commits*

`HEAD` é um atalho.

- `HEAD` é uma referência ao último *commit* feito.
- `HEAD~1` é o *commit* anterior ao último *commit* feito.
- `HEAD^1` é o *commit* pai do último *commit* feito. Em geral *commits* tem no máximo 2 pais, quando é resultado de um *merge*.

Acima, `HEAD` poderia ser substituído pelo nome de um *branch* ou *tag*, ou pelo *hash* de um *commit* específico.

`git show`

Exibe o conteúdo do *commit* e o que foi modificado.

`git checkout`

Faz com que seu `Working Directory` tenha o conteúdo do *commit* especificado.

`git reset`

Faz com que sua `Staging Area` tenha o conteúdo do *commit* especificado.
Usar `git reset --hard` faz também com que seu `Working Directory` tenha o conteúdo do *commit* especificado.
Você pode perder dados aqui.

Fazendo uma analogia, pense que...

- *HEAD* é você.
- *Branch* é uma casa, mas no estilo motorhome.
- *checkout* é uma forma de você (*HEAD*) viajar a pé para algum lugar.
- *reset* muda o lugar em que o motorhome (*Branch*) está estacionado. Mas como você (*HEAD*) é o motorista, você acaba indo junto com a casa (*Branch*).

Dependendo do lugar para onde viaja (*checkout*), você (*HEAD*) pode se tornar um sem-teto (*detached HEAD*).

Às vezes `git checkout` pode falhar com a mensagem *"Aborting"*.
Provavelmente existem arquivos no seu *Working Directory* com nomes iguais aos arquivos no *commit* especificado.

Opções:

1. Fazer *commit* das alterações com `git add --all` e `git commit -m mensagem`.
   Atenção! Arquivos em .gitignore serão ignorados.
2. Descarta as alterações com `git reset --hard` e `git clean -df`.
   Você pode perder dados aqui!

7. Comunicação com o servidor

`git push`

abc

- Protege seu repositório local contra danos.
- Não precisa ser um servidor remoto.

`git pull`

abc

`git fetch`

abc

8. Resolvendo conflitos

`git merge`

abc

`git rebase`

abc

9. Aprendendo mais

`git --help`

abc

git-scm.com/doc

abc

Livro Pro Git

abc