

## STREAM SERIES

# desvendando a ferramenta

Git para iniciantes  
*O que você deveria saber?*

SERGIO CABRAL



[sergiocabral.com](http://sergiocabral.com)

# Índice

1. Perguntas e respostas .....	2
2. Criando um repositório .....	3
3. Comandos básicos .....	4
4. Onde estão os arquivos .....	5
5. <i>Branches e Tags</i> .....	6
6. Navegando entre os <i>commits</i> .....	7
6.1. O que é <i>HEAD</i> .....	8
7. Comunicação com o servidor .....	9
7.1. Um servidor local .....	10
8. Resolvendo conflitos .....	11
9. Aprendendo mais .....	12

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](http://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.



Figura 1.

## 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.



O GitHub tem no seu nome "Git" e isso causa alguma confusão. Mas há alternativas de serviços com o mesmo propósito de serem servidores para Git. Por exemplo, *GitLab*, *Bitbucket* e *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. Criando um repositório

### git init

Cria do zero um repositório Git.

The screenshot shows a terminal window titled "root :: pwsh :: my-super-app". The command `git init` is run in the directory `D:\Git\ebac-talk\my-super-app`. The output shows the creation of an empty repository. A blue callout box on the right side of the terminal window contains the following text:

- Cria uma pasta
- Após o `git init` a pasta continua sem conteúdo
- Mas na verdade tem uma pasta oculta .git

Red arrows point from the text in the callout box to the terminal output: one arrow points to the command `git init`, another points to the folder listing, and a third points to the ".git" entry in the second folder listing.

Figura 2.

### git clone

Cria uma cópia de um repositório existente.

The screenshot shows a terminal window titled "root :: pwsh :: ebac-talk". The command `git clone https://github.com/sergiocabral/Blockchain.Cabr0n.git` is run in the directory `D:\Git\ebac-talk`. The output shows the cloning process, including object enumeration, counting, compressing, receiving objects, and resolving deltas. A blue callout box on the right side of the terminal window contains the following text:

- Endereço do servidor Git
- Uma pasta com nome do repositório é criada

Red arrows point from the text in the callout box to the command being run and to the newly created folder entry in the folder listing.

Figura 3.



O banco de dados do Git é a 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.

The screenshot shows a terminal session in a Windows environment. The user is in the directory `D:\Git\ebac-talk\my-super-app` on the `master` branch. The session starts with some initial commands and then focuses on the workflow:

- Adding a file:** `echo "conteúdo qualquer" > my-file`
- Checking status:** `git status` (Shows "Untracked files: my-file")
- Staging the file:** `git add .\my-file`
- Checking status again:** `git status` (Shows "Changes to be committed: new file: my-file")
- Committing the change:** `git commit -m "uma mensagem descritiva aqui"`
- Viewing the log:** `git log` (Shows the commit message and details)

A red bracket highlights the command `git add .\my-file`, and a blue callout box points to it with the text: **O que está prestes a entrar no repositório é chamado de "stage", "staging area" ou "index"**.

Figura 4.

## 4. Onde estão os arquivos

Há três lugares onde seus arquivos podem estar.

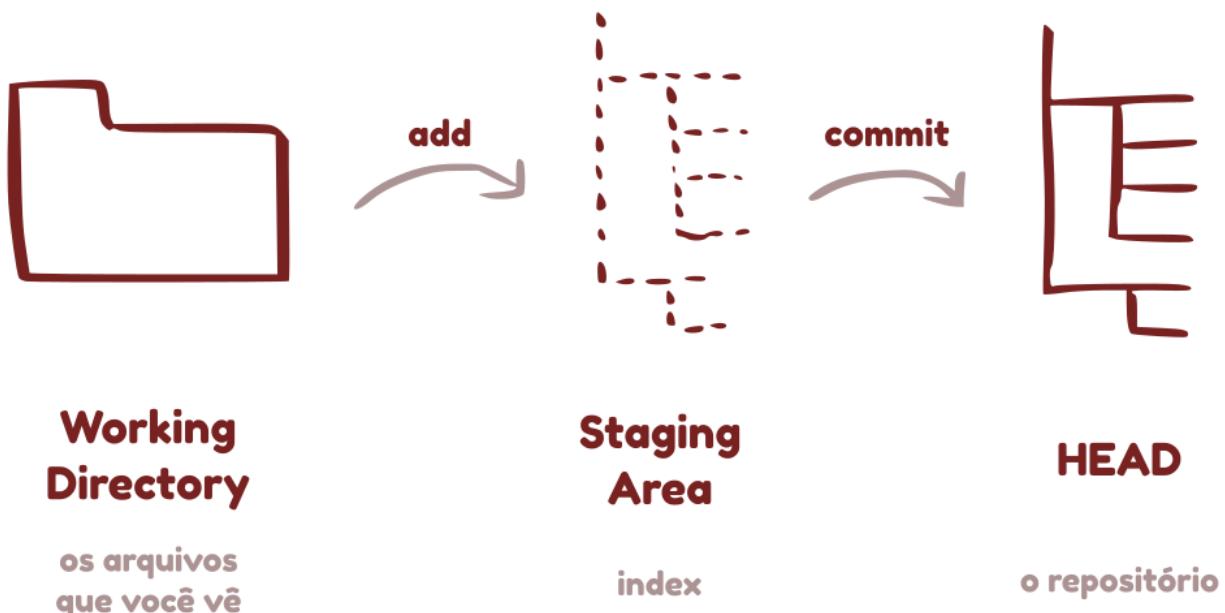


Figura 5.

### 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.  
Também chamado *index* ou *stage*.

### 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*.  
Isso é um hash de um commit qualquer: `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*.



Figura 6.

## Criar tags

- `git tag <nome-da-tag> -m "<descritivo-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

## git show

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

The screenshot shows a terminal window with two tabs. The current tab displays a git commit history:

```
git init
Initialized empty Git repository in D:/Git/ebac-talk/my-super-app/.git/
ls > listagem.txt
git add --all
git commit -m "Listagem da pasta do projeto"
[master (root-commit) fc194b9] Listagem da pasta do projeto
 1 file changed, 7 insertions(+)
 create mode 100644 listagem.txt
git show fc194b9
commit fc194b96041619de619f25c2a5aad8dd3bd26ab3 (HEAD -> master)
Author: Sergio Cabral <git@sergiocabral.com>
Date:   Thu Nov 18 21:45:36 2021 -0300

Listagem da pasta do projeto

diff --git a/listagem.txt b/listagem.txt
new file mode 100644
index 000000..2495154
--- /dev/null
+++ b/listagem.txt
@@ -0,0 +1,7 @@
+
+ Directory: D:/Git/ebac-talk/my-super-app
+
+Mode           LastWriteTime      Length Name
+----          -----          ---- 
+a---          18/11/2021    21:44          0 listagem.txt
```

A blue callout box points to the commit message and author information with the text "dados no commit e alterações nos arquivos".

Figura 7.

## 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 com que também seu *Working Directory* tenha o conteúdo do *commit* especificado. Você pode perder dados aqui!

Às vezes o *checkout* pode falhar com a mensagem "Aborting".

Provavelmente existem *arquivos conflitantes* 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. Descartar as alterações com `git reset --hard` e `git clean -fd`.  
Você pode perder dados aqui!

## 6.1. O que é *HEAD*

*HEAD* é um atalho. Por exemplo:

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

*HEAD* poderia ser substituído por outros atalhos, como o nome de um *branch* ou *tag*, ou o *hash* de um *commit* específico.



Assim como você, *HEAD* só conhece seu passado. Não existem atalhos para um *commit* à frente.

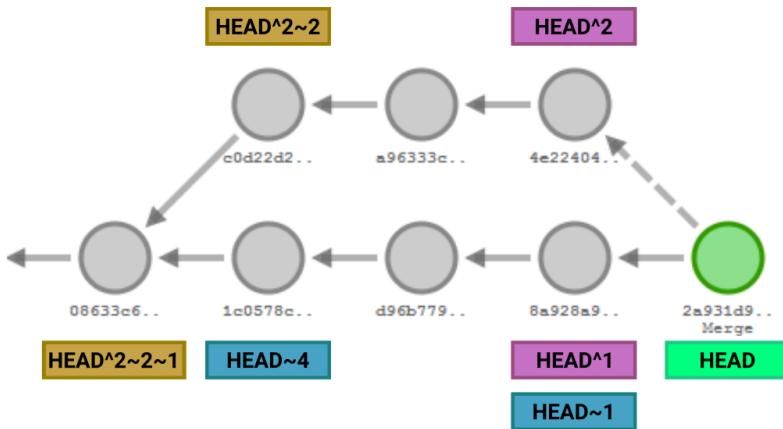


Figura 8.

Fazendo uma analogia, pense que...

- *HEAD* é você.
- *branch* é uma casa no estilo motorhome.
- *reset* vai estacionar o motorhome (o *branch*) em outro lugar.  
Mas como você (o *HEAD*) é o motorista, você acaba indo junto com a casa (o *branch*).
- *checkout* é uma forma de você (o *HEAD*) viajar a pé para algum lugar.
- *detached HEAD* é uma forma de dizer que você (o *HEAD*) é um sem-teto depois que viajou (com *checkout*) para um lugar que não tem casa (um *branch*).

# 7. Comunicação com o servidor

## git push

Envia seu *branch* atual para o servidor.

Mas o servidor precisa estar configurado no seu repositório local.  
Essa configuração é feita com `git remote`.

The terminal window shows the following sequence of commands:

```
git push
fatal: No configured push destination.
Either specify the URL from the command-line or configure a remote repository using
  git remote add <name> <url>
and then push using the remote name
  git push <name>
git remote add servidor_github https://github.com/sergiocabral/my-super-app.git
git push servidor_github
fatal: The current branch main has no upstream branch.
To push the current branch and set the remote as upstream, use
  git push --set-upstream servidor_github main
git push --set-upstream servidor_github main
Logon failed, use ctrl+c to cancel basic credential prompt.
Username for 'https://github.com': sergiocabral
Password for 'https://sergiocabral@github.com':
Enumerating objects: 9, done.
Counting objects: 100% (9/9), done.
Delta compression using up to 8 threads
Compressing objects: 100% (8/8), done.
Writing objects: 100% (9/9), 2.13 KiB | 2.13 MiB/s, done.
Total 9 (delta 2), reused 0 (delta 0)
remote: Resolving deltas: 100% (2/2), done.
To https://github.com/sergiocabral/my-super-app.git
 * [new branch]      main -> main
Branch 'main' set up to track remote branch 'main' from 'servidor_github'.
```

A callout box points to the GitHub login dialog with the text: "O Git vai sugerindo os comandos a digitar". Another callout box points to the password field with the text: "Pode ser que apareça uma janela de autenticação do GitHub nesse momento, mas eu cancelei e digitei usuário e access token no terminal".

Figura 9.



O GitHub não aceita que você digite sua senha. Você precisar criar um *personal access token*. Para saber como, visite <https://docs.github.com/pt/authentication/keeping-your-account-and-data-secure/creating-a-personal-access-token>.

## git pull

Faz download de todos os novos dados no servidor, e atualiza o *branch* atual, onde você está.

O comando `git pull` é equivalente a dois comandos, `git fetch` seguido de `git merge` (ou `git rebase` dependendo da sua configuração). O último comando é aquele que vai atualizar seu *branch* atual para corresponder ao *branch* remoto.

Então, se quiser fazer o download sem atualizar o *branch* local, use `git fetch`.

## 7.1. Um servidor local

Um servidor remoto não precisa estar na internet, mas pode ser qualquer pasta que já seja um repositório Git.

The screenshot shows a terminal window with two tabs: 'root :: pwsh :: my-personal-repo' and 'root :: pwsh :: my-super-app'. The 'my-super-app' tab displays a command-line session:

```
⚡ > D:\Git\ebac-talk
~# cd ..\my-super-app
⚡ > D:\Git\ebac-talk\my-super-app > master ✘
~# git log
commit fc194b96041619de619f25c2a5aad8dd3bd26ab3 (HEAD -> master)
Author: Sergio Cabral <git@sergiocabral.com>
Date:   Thu Nov 18 21:45:36 2021 -0300

Listagem da pasta do projeto
⚡ > D:\Git\ebac-talk\my-super-app > master ✘
~# cd ..
⚡ > D:\Git\ebac-talk
~# git clone .\my-super-app my-personal-repo-of-super-app
Cloning into 'my-personal-repo-of-super-app'...
done.
⚡ > D:\Git\ebac-talk
~# cd ..\my-personal-repo-of-super-app
⚡ > D:\Git\ebac-talk\my-personal-repo-of-super-app > master ✘
~# git log
commit fc194b96041619de619f25c2a5aad8dd3bd26ab3 (HEAD -> master, origin/master, origin/HEAD)
Author: Sergio Cabral <git@sergiocabral.com>
Date:   Thu Nov 18 21:45:36 2021 -0300

Listagem da pasta do projeto
⚡ > D:\Git\ebac-talk\my-personal-repo-of-super-app > master ✘
~#
```

Annotations in red boxes highlight specific commands and output:

- 'git log' in the first section.
- 'git clone .\my-super-app my-personal-repo-of-super-app'
- 'git log' in the second section.

Annotations in blue boxes provide explanatory text:

- 'o repositório clonado faz referência a branches remotos com prefixo 'origin'' (The cloned repository refers to remote branches with the prefix 'origin').
- 'Para trocar o prefixo 'origin' use 'git remote rename origin novo\_nome'' (To change the prefix 'origin' use 'git remote rename origin novo\_nome').

Figura 10.

Seria possível definir uma pasta na rede local como servidor. Então a equipe de desenvolvedores faz `git clone` dessa pasta e envia as alterações com `git push`.

Como ninguém vai alterar os arquivo dessa pasta diretamente com *commits*, ela não precisa ter *Working Directory*, que pode ser removido por...

- Reppositórios novos, criados do zero:
  1. Criar o repositório usando `git init --bare`.
- Reppositórios já existentes:
  1. Ativar o modo *bare* com `git config core.bare true`
  2. Apagar o conteúdo da pasta, exceto a `.git`
  3. Mover o conteúdo da pasta `.git` para a pasta do repositório.
  4. Apagar a pasta `.git` que agora estará vazia.



## 8. Resolvendo conflitos

**git merge**

abc

**git rebase**

abc

## 9. Aprendendo mais

**git --help**

abc

[git-scm.com/doc](http://git-scm.com/doc)

abc

[Livro Pro Git](#)

abc