

## 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 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* nos links:

- [git.sergiocabral.com/ebook](http://git.sergiocabral.com/ebook) (versão on-line)
- [git.sergiocabral.com/ebook.pdf](http://git.sergiocabral.com/ebook.pdf) (versão para download em PDF)

Além disso, se quiser para praticar sem medo, acesse o playground de Git no link

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

The screenshot shows a web-based Git playground. At the top, there's a header bar with the title "Git Playground" and a URL field containing "git.sergiocabral.com/#free-remote". Below the header is a command line interface where several git commands have been entered and executed:

```
$ git commit  
$ git merge master  
$ git commit  
$ git checkout master  
$ git commit  
$ git commit  
$ enter git command
```

Below the command line, the status of the local repository is shown:

Local Repository  
HEAD: master

At the bottom, a detailed commit graph is displayed. The graph shows a sequence of commits connected by arrows pointing from right to left. Some commits are highlighted in purple, while others are grey. A yellow box labeled "hotfix" is placed above a specific commit. The HEAD pointer is at the top right, pointing to a purple commit. The master branch is also indicated at the top right. The graph includes commit IDs and short descriptions, such as "First commit" and "Merge". A box labeled "origin/master" is located near the bottom left of the graph area.

Figura 1.

# 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 2.

## 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 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 to the message "Initialized empty Git repository in D:/Git/ebac-talk/my-super-app/.git/", and another to the file ".git" in the directory listing.

Figura 3.

### 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 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 `git clone` and the resulting directory listing. One arrow points to the URL `https://github.com/sergiocabral/Blockchain.Cabr0n.git` and another to the newly created directory `Blockchain.Cabr0n`.

Figura 4.



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.

```
root :: pwsh :: my-super-app > D:\Git\ebac-talk\my-super-app >(master) ~# @> echo "conteúdo qualquer" > my-file
root :: pwsh :: my-super-app >(master) ~# @> git status
On branch master
No commits yet
Untracked files:
  (use "git add <file>..." to include in what will be committed)
    my-file
nothing added to commit but untracked files present (use "git add" to track)
root :: pwsh :: my-super-app >(master) ~# @> git add .\my-file
root :: pwsh :: my-super-app >(master) ~# @> git status
On branch master
No commits yet
Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
    new file:   my-file
root :: pwsh :: my-super-app >(master) ~# @> git commit -m "uma mensagem descritiva aqui"
[master (root-commit) f019806] uma mensagem descritiva aqui
1 file changed, 1 insertion(+)
create mode 100644 my-file
root :: pwsh :: my-super-app >(master) ~# @> git log
commit f019806e7923e3477da92378afb0175f7f0aefed (HEAD -> master)
Author: Sergio Cabral <git@sergiocabral.com>
Date:   Tue Nov 16 09:22:52 2021 -0300
    uma mensagem descritiva aqui
root :: pwsh :: my-super-app >(master) ~# @>
```

Figura 5.

## 4. Onde estão os arquivos

Há três lugares onde seus arquivos podem estar.

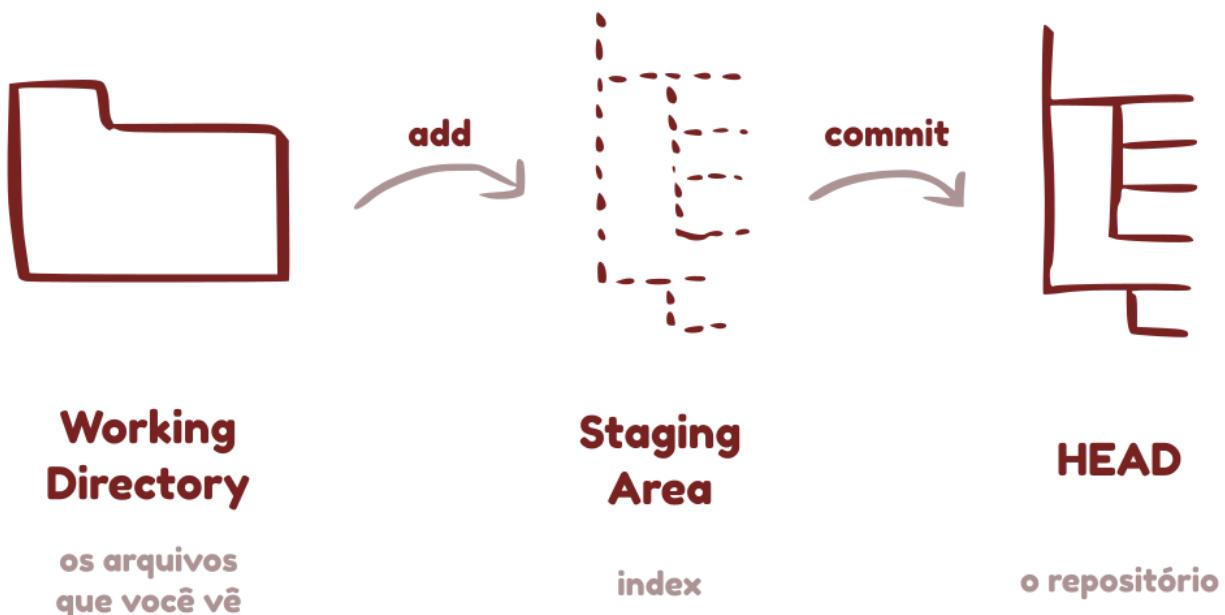


Figura 6.

### 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 7.

## 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 and a file diff. The commit history shows the creation of a file named 'listagem.txt' with the message 'Listagem da pasta do projeto'. A blue callout box points to this commit with the text 'dados no commit e alterações nos arquivos'. Below the commit history, a 'diff' command is shown comparing 'listagem.txt' from the index ('a') and working directory ('b'). The diff output shows the file was created with mode 100644 and content 'Listagem da pasta do projeto'. A blue callout box points to the diff output with the same text 'dados no commit e alterações nos arquivos'.

```
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
```

Figura 8.

## 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. Múltiplos pais acontecem como 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.

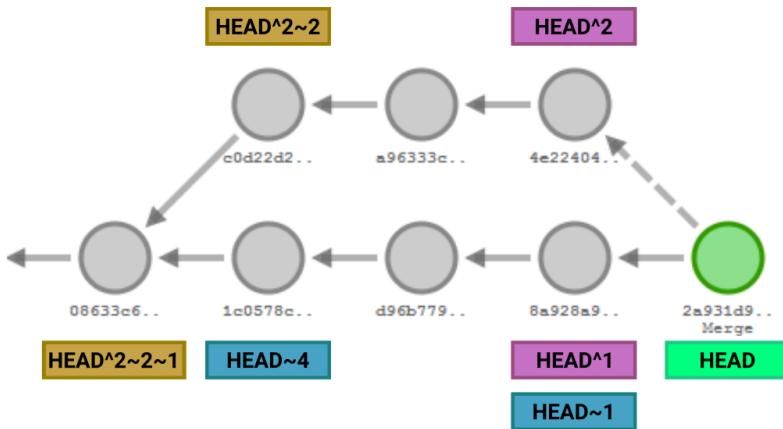


Figura 9.

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 screenshot shows a terminal window with several command-line sessions:

- The first session shows an error: "fatal: No configured push destination. Either specify the URL from the command-line or configure a remote repository using `git remote add <name> <url>`". A blue box highlights the command `git push`. A red box highlights the error message. A blue callout box says "O Git vai sugerindo os comandos a digitar".
- The second session shows the configuration of a remote repository: "git remote add servidor\_github https://github.com/sergiocabral/my-super-app.git". A red box highlights the command `git remote add`. A blue callout box says "Pode ser que apareça uma janela de autenticação do GitHub nesse momento, mas eu cancelei e digitou usuário e access token no terminal".
- The third session shows the attempt to push the current branch: "git push servidor\_github main". A red box highlights the command `git push`. A blue callout box says "nas próximas vezes basta 'git push'".
- The fourth session shows the successful push: "Everything up-to-date".

Figura 10.



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 contains the following command sequence:

```
⚡ > 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 the command 'git log' in the first section and the line 'Cloning into...' in the second section. A green arrow points from the 'git log' command to the 'Cloning into...' line. Two blue arrows point upwards from the bottom of the annotations to a callout box containing the text: 'o repositório clonado faz referência a branches remotos com prefixo 'origin'' and 'Para trocar o prefixo 'origin' use 'git remote rename origin novo\_nome'.'

Figura 11.

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.

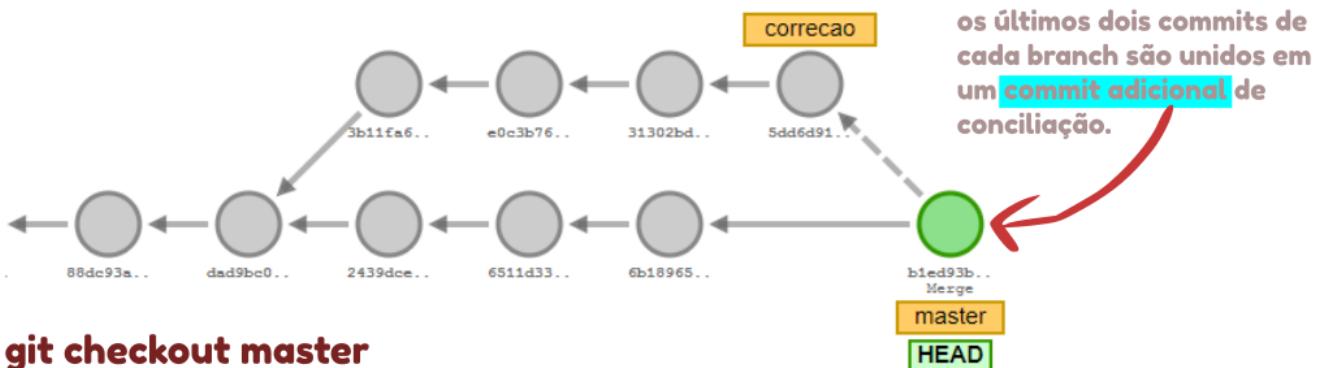


# 8. Resolvendo conflitos

Resolver conflitos envolve juntar alterações de dois *branches* em um.  
Há dois comandos para se fazer isso...

## git merge

Analisa a diferença entre dois *branches* e cria um *commit* que conciliar as diferenças.



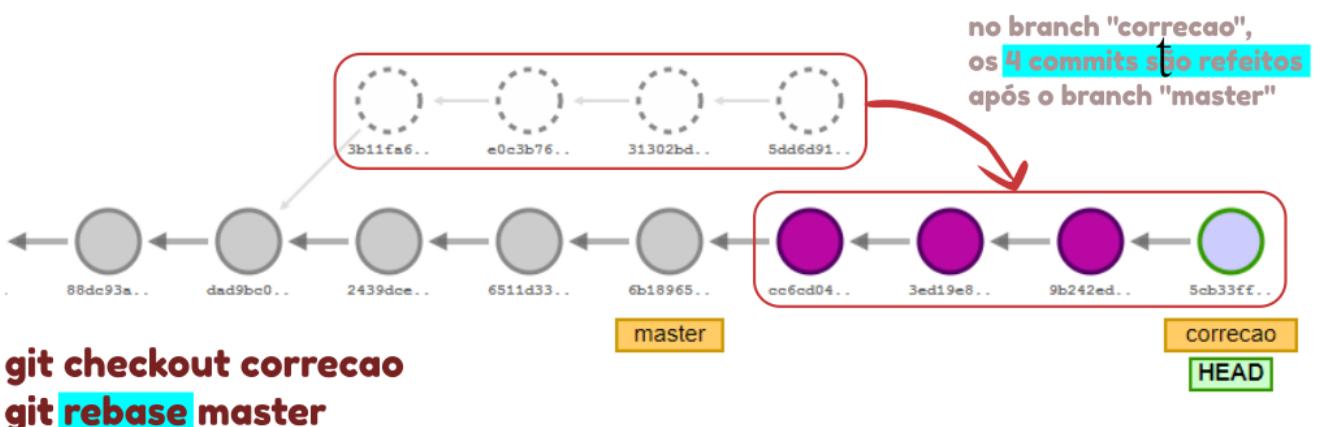
**git checkout master  
git merge correcao**

Figura 12.

## git rebase

Verifica quais *commits* do *branch* atual não existem no *branch*.

Então percorre cada *commit* e os aplica individualmente no *branch* de destino, que será a nova 'base' para o *branch* atual.



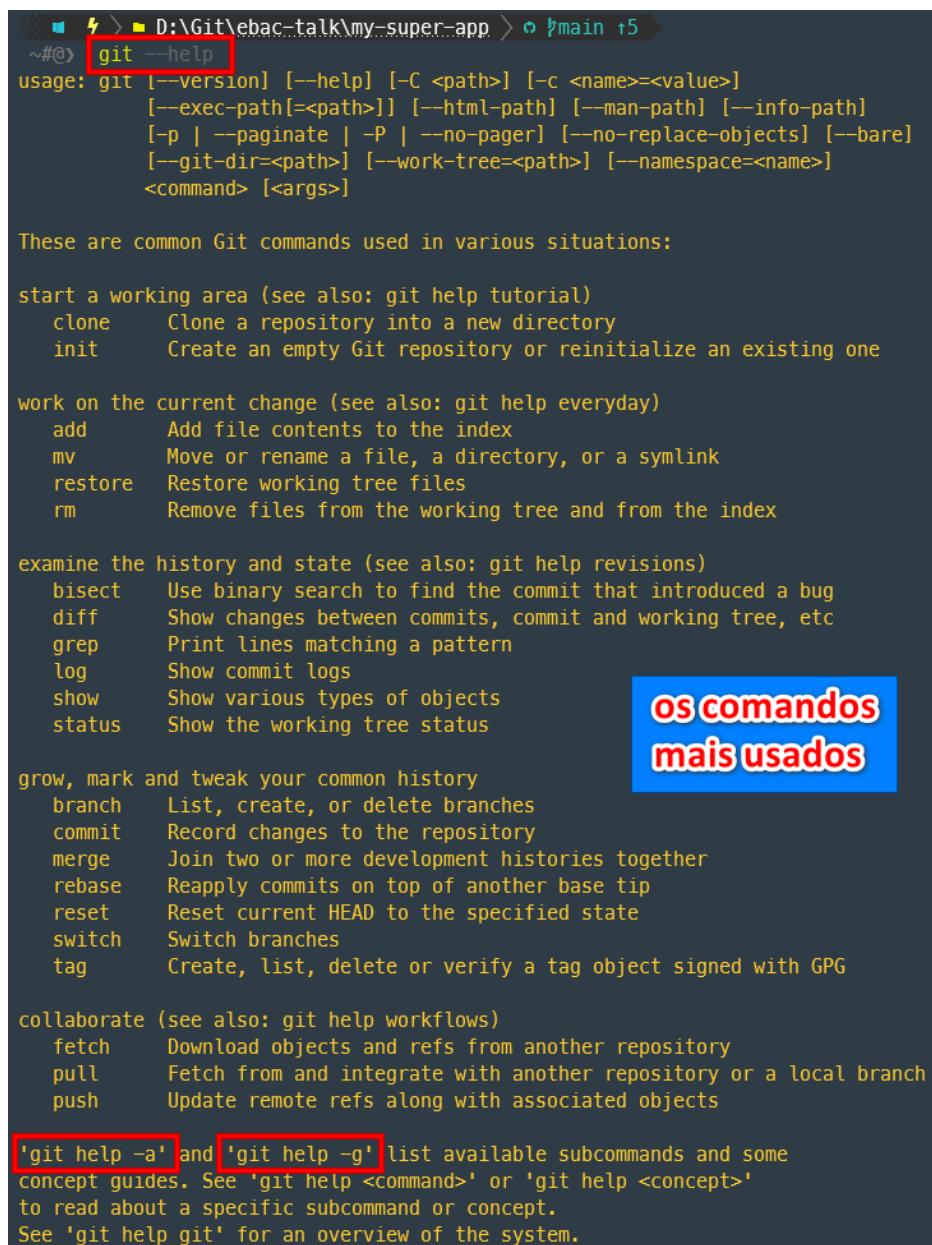
**git checkout correcao  
git rebase master**

Figura 13.

# 9. Aprendendo mais

## git --help

Já é um bom ponto de partida. Exibe os **comandos mais usados**.



```
git --help
usage: git [--version] [--help] [-C <path>] [-c <name>=<value>]
           [--exec-path[=<path>]] [--html-path] [--man-path] [--info-path]
           [-p | --paginate | -P | --no-pager] [--no-replace-objects] [--bare]
           [--git-dir=<path>] [--work-tree=<path>] [--namespace=<name>]
           <command> [<args>]

These are common Git commands used in various situations:

start a working area (see also: git help tutorial)
  clone      Clone a repository into a new directory
  init       Create an empty Git repository or reinitialize an existing one

work on the current change (see also: git help everyday)
  add        Add file contents to the index
  mv         Move or rename a file, a directory, or a symlink
  restore    Restore working tree files
  rm         Remove files from the working tree and from the index

examine the history and state (see also: git help revisions)
  bisect    Use binary search to find the commit that introduced a bug
  diff      Show changes between commits, commit and working tree, etc
  grep      Print lines matching a pattern
  log       Show commit logs
  show      Show various types of objects
  status    Show the working tree status

grow, mark and tweak your common history
  branch   List, create, or delete branches
  commit   Record changes to the repository
  merge    Join two or more development histories together
  rebase   Reapply commits on top of another base tip
  reset   Reset current HEAD to the specified state
  switch  Switch branches
  tag     Create, list, delete or verify a tag object signed with GPG

collaborate (see also: git help workflows)
  fetch   Download objects and refs from another repository
  pull    Fetch from and integrate with another repository or a local branch
  push    Update remote refs along with associated objects

'git help -a' and 'git help -g' list available subcommands and some
concept guides. See 'git help <command>' or 'git help <concept>'
to read about a specific subcommand or concept.
See 'git help git' for an overview of the system.
```

**os comandos  
mais usados**

Figura 14.

## git <comando> -help

Ajusta básica com os **parâmetros mais utilizados do comando**.

## git <comando> --help

Ajusta completa com **todos os parâmetros possíveis do comando**. Vai abrir uma página de documentação em HTML ou manual do sistema (*MAN*) se estiver no Linux.

## Livro Pro Git

Tudo mesmo que você precisa saber sobre Git está nesse livro.

A versão brasileira está em processo de tradução colaborativa. Se você souber inglês e quiser ajudar, aparece por lá.