

STREAM SERIES

desvendando a ferramenta

Git para iniciantes
O que você deveria saber?

SERGIO CABRAL



sergijocabral.com

Índice

1. Perguntas e respostas	2
2. Criando um repositório	3
3. Comandos básicos	4
3.1. Um pouco mais sobre <i>log</i>	5
4. Onde estão os arquivos	6
5. <i>Branches</i> e <i>Tags</i>	7
6. Navegando entre os <i>commits</i>	8
6.1. O que é <i>HEAD</i>	9
6.2. Como as coisas se relacionam	10
6.3. Restaurando arquivos	10
7. Comunicação com o servidor	11
7.1. Usando um servidor local	12
8. Unindo <i>branches</i>	13
8.1. Resolvendo conflitos	14
9. Aprendendo mais	15

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

Se quiser praticar sem medo, use esse playground para *Git*:

- git.sergiocabral.com
- Se quiser ver o [código-fonte](#)

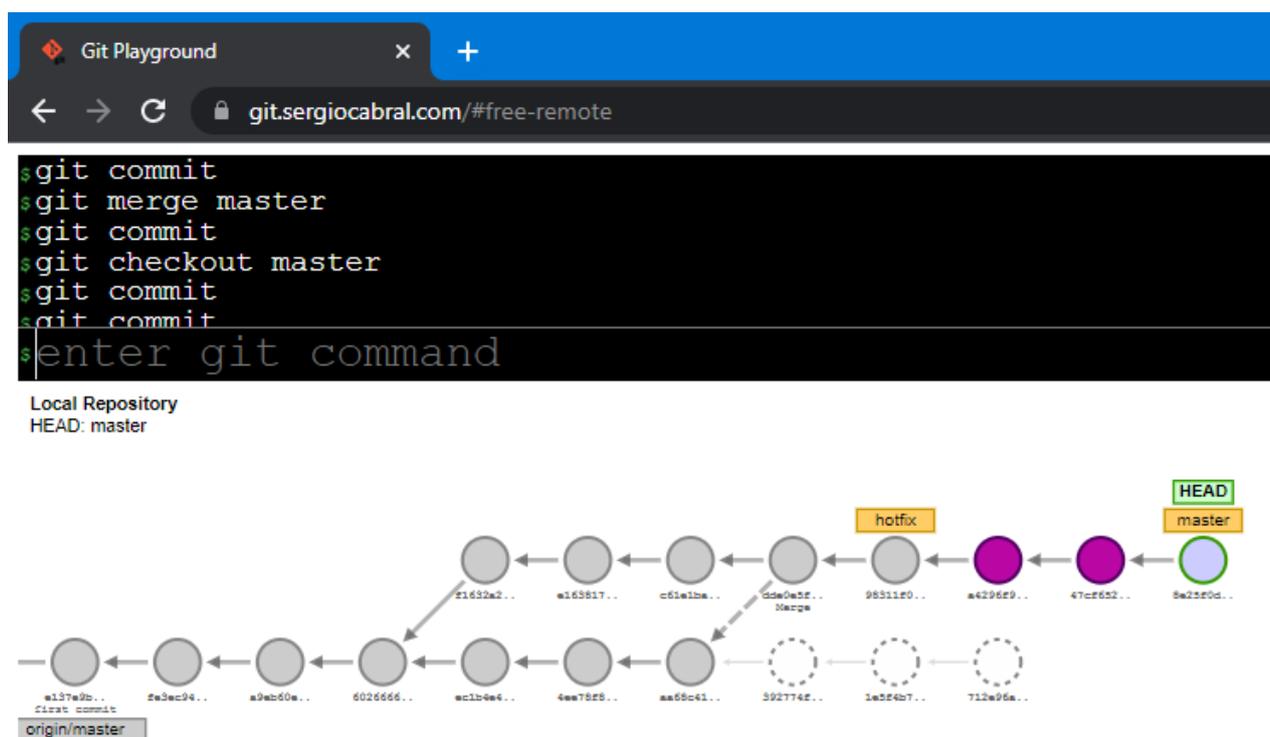


Figura 1. Playground para Git em git.sergiocabral.com

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. As etapas da evolução do projeto

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.

```
root :: pwsh = my-super-app
D:\Git\ebac-talk
~#> mkdir my-super-app

Directory: D:\Git\ebac-talk

Mode                LastWriteTime         Length Name
----                -
d-----            04/11/2021   19:27             my-super-app

D:\Git\ebac-talk
~#> cd .\my-super-app\
D:\Git\ebac-talk\my-super-app
~#> git init
Initialized empty Git repository in D:/Git/ebac-talk/my-super-app/.git/
D:\Git\ebac-talk\my-super-app > |?master #
~#> dir
D:\Git\ebac-talk\my-super-app > |?master #
~#> dir -Attributes Hidden

Directory: D:\Git\ebac-talk\my-super-app

Mode                LastWriteTime         Length Name
----                -
d--h-              04/11/2021   19:28             .git

D:\Git\ebac-talk\my-super-app > |?master #
~#>
```

Cria uma pasta

Após o 'git init' a pasta continua sem conteúdo

Mas na verdade tem uma pasta oculta .git

Figura 3. Terminal demonstrando a criação de um repositório do zero.

git clone

Cria uma cópia de um repositório existente.

```
root :: pwsh = ebac-talk
D:\Git\ebac-talk
~#> git clone https://github.com/sergiocabral/Blockchain.Cabr0n.git
Cloning into 'Blockchain.Cabr0n'...
remote: Enumerating objects: 3690, done.
remote: Counting objects: 100% (2076/2076), done.
remote: Compressing objects: 100% (913/913), done.
remote: Total 3690 (delta 1282), reused 1894 (delta 1112), pack-reused 1614
Receiving objects: 100% (3690/3690), 701.91 KiB | 5.80 MiB/s, done.
Resolving deltas: 100% (2262/2262), done.
D:\Git\ebac-talk
~#> dir

Directory: D:\Git\ebac-talk

Mode                LastWriteTime         Length Name
----                -
d-----            04/11/2021   19:43     Blockchain.Cabr0n
d-----            04/11/2021   19:28             my-super-app

D:\Git\ebac-talk
~#>
```

Endereço do servidor Git

Uma pasta com nome do repositório é criada

Figura 4. Terminal demonstrando como fazer download de um repositório existente.



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`.
Você conhece também o arquivo `.git/info/exclude`?
Faz a mesma coisa, mas não precisa ser *commitado*.

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
D:\Git\ebac-talk\my-super-app > ?master # +1
~#@> 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)
D:\Git\ebac-talk\my-super-app > ?master # +1
~#@> git add .\my-file
D:\Git\ebac-talk\my-super-app > ?master # +1
~#@> git status
On branch master

No commits yet

Changes to be committed:
(use "git rm --cached <file>..." to unstage)
new file:   my-file

D:\Git\ebac-talk\my-super-app > ?master # +1
~#@> 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
D:\Git\ebac-talk\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
D:\Git\ebac-talk\my-super-app > ?master #
~#@>
```

O que está prestes a entrar no repositório é chamado de "stage", "staging area" ou "index"

Figura 5. Usando os comandos básicos do Git no terminal.

4. Onde estão os arquivos

Há três lugares onde seus arquivos podem estar.

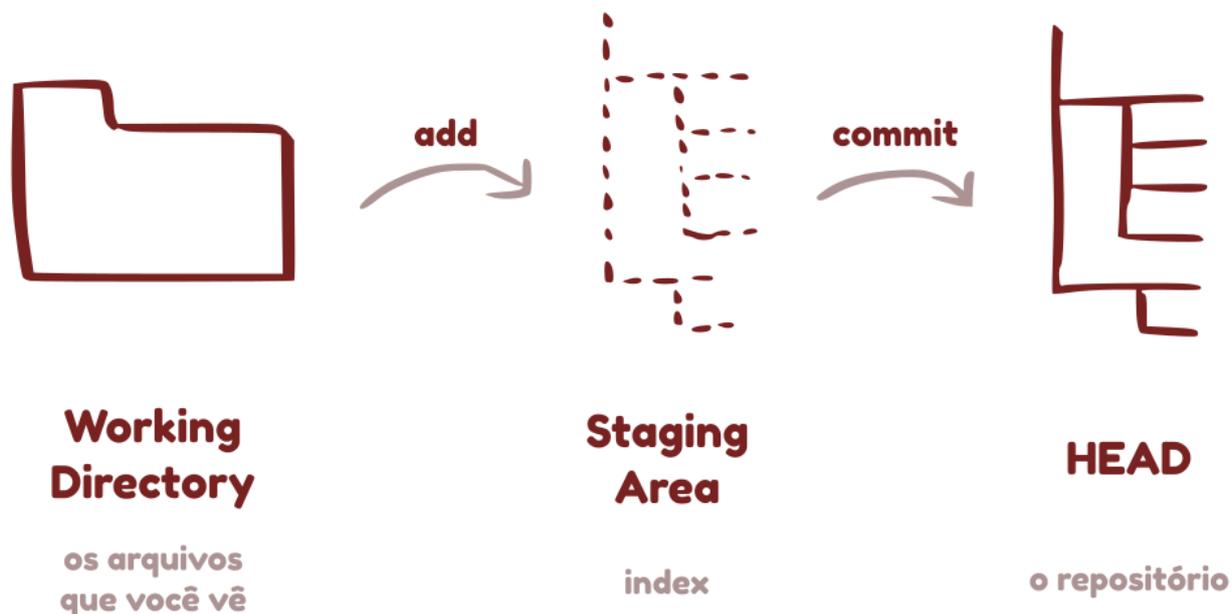


Figura 7. Conhecido como "As Três Árvores", no inglês, "The Three Trees", ou TTT.

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 8. Indicação de como cada *commit* sempre continuará existindo

Criar *tags*

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

Excluir *tags*

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



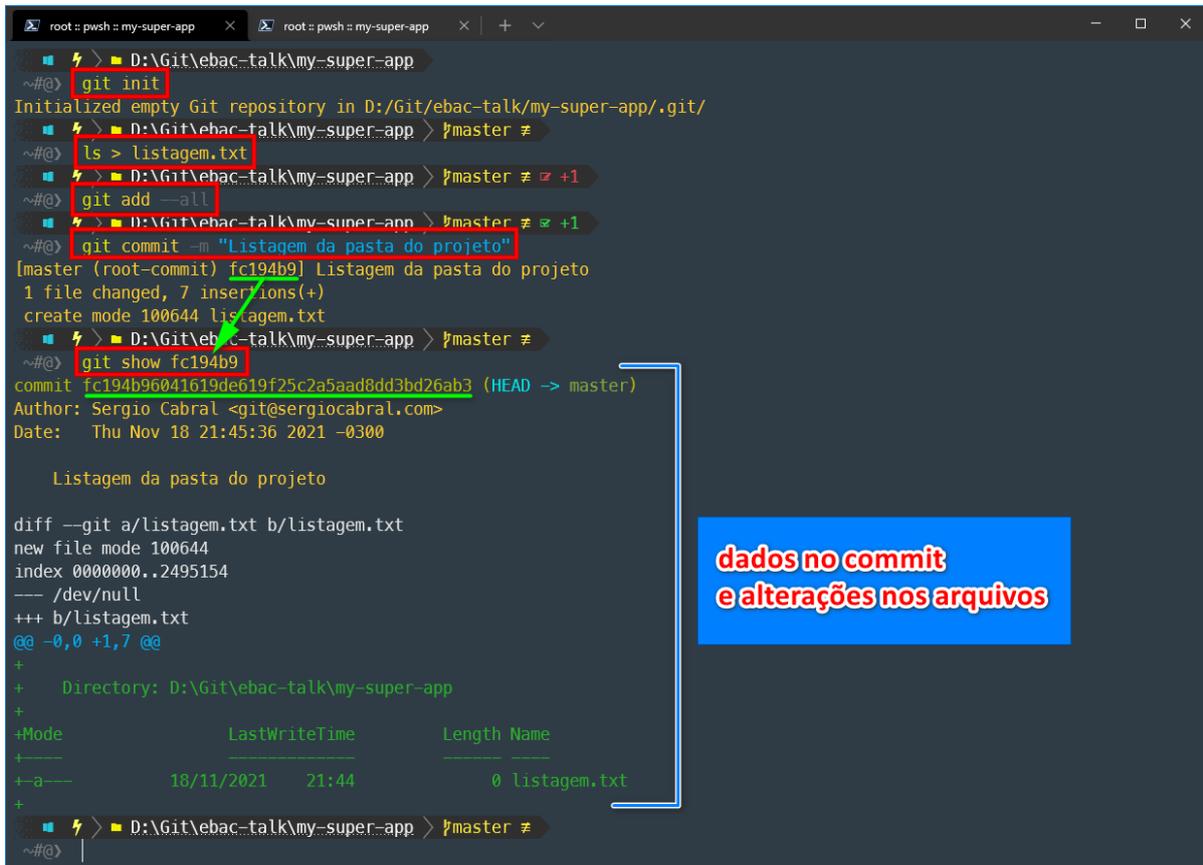
Branches e *tags* são semelhantes, mas a **diferença** está em que:

- 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.



```
root: pwsh: my-super-app x root: pwsh: my-super-app x | + v
~#(C) > D:\Git\ebac-talk\my-super-app
~#(C) git init
Initialized empty Git repository in D:/Git/ebac-talk/my-super-app/.git/
~#(C) > D:\Git\ebac-talk\my-super-app > ?master #
~#(C) ls > listagem.txt
~#(C) > D:\Git\ebac-talk\my-super-app > ?master # +1
~#(C) git add --all
~#(C) > D:\Git\ebac-talk\my-super-app > ?master # +1
~#(C) 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
~#(C) > D:\Git\ebac-talk\my-super-app > ?master #
~#(C) 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 0000000..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
+
~#(C) > D:\Git\ebac-talk\my-super-app > ?master #
~#(C) |
```

Figura 9. Como ver o conteúdo de um *commit*.

git checkout

O *Working Directory* vai ter o conteúdo do *commit* especificado.

git reset

A *Staging Area* vai ter 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~1 é o commit anterior ao último commit feito.
- HEAD^1 é o commit pai do último commit feito. Múltiplos pais acontecem como resultado de um merge.

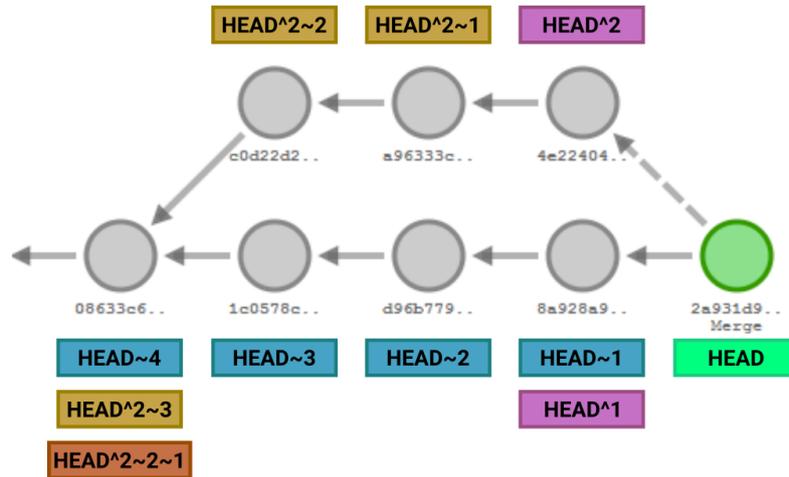


Figura 10. Formas de acessar commits usando o atalho HEAD.



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.

6.2. Como as coisas se relacionam

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*).

6.3. Restaurando arquivos

Como já dito, usar `--` em alguns comandos limitam o contexto a um arquivo. Então...

`git checkout <commit> -- <nome-do-arquivo>`

Restaura a versão de um arquivo presente em um *commit* e grava no *Working Directory*.

`git reset <commit> -- <nome-do-arquivo>`

Restaura o arquivo na *Staging Area*.

É mais fácil entender quando dizemos que remove o arquivo da *Staging Area*.



- Usar `git add` faz a *Staging Area* corresponder ao *Working Directory*.
- Usar `git reset` faz a *Staging Area* corresponder ao *HEAD* (o repositório).

Nos casos acima, o *commit* é opcional. Quando não informado assume que seja *HEAD*. Ao invés de *commit*, pode ser um *branch*, uma *tag* ou *HEAD*.

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**.

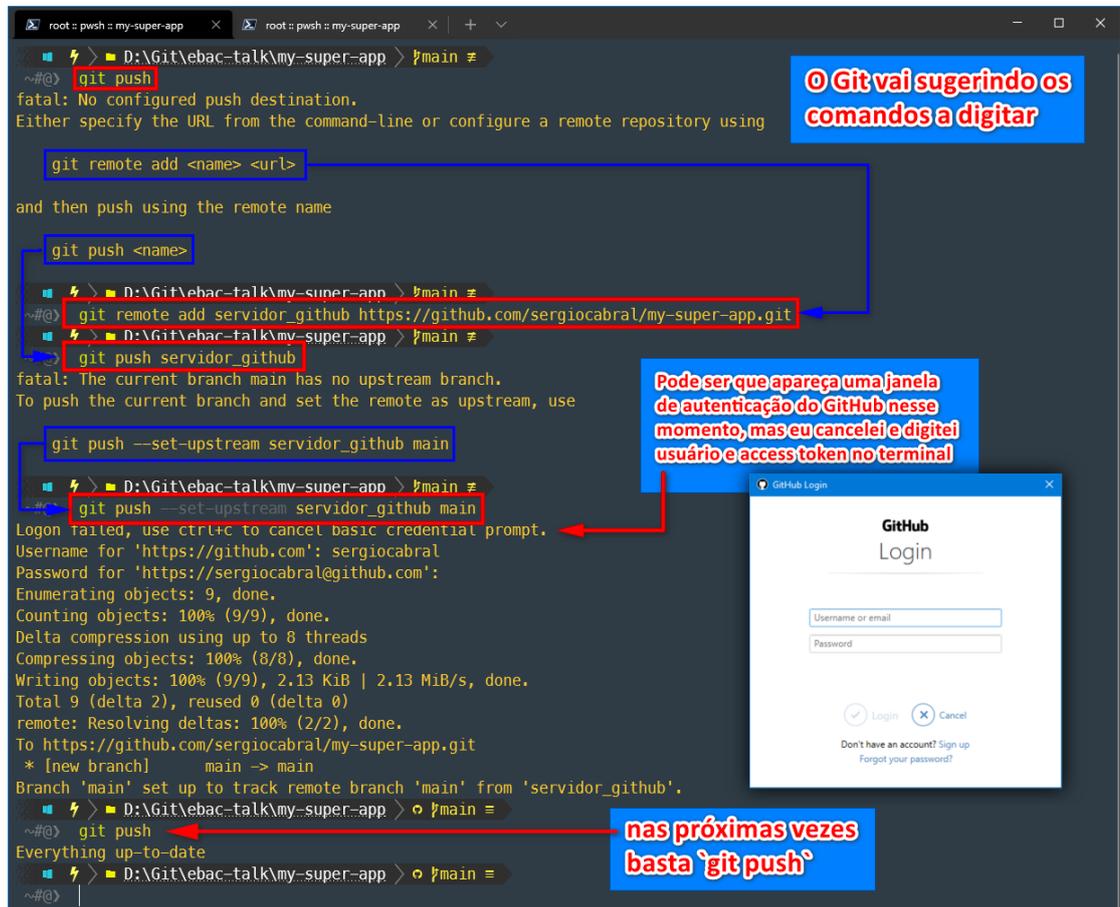


Figura 11. Ao enviar um branch para o servidor o Git vai dando dicas.



O **GitHub** não aceita que você digite sua senha. Você precisa 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. Usando um servidor local

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

```
root :: pwsh :: my-personal-repo x root :: pwsh :: my-super-app x + v
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 ≡
~#@>
```

o repositório clonado faz referência a branches remotos com prefixo 'origin'

Para trocar o prefixo 'origin' use 'git remote rename origin novo_nome'

Figura 12. É possível clonar um repositório a partir de outro repositório local.

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`.

Repositórios usados como servidores não recebem *commits*, então **não precisa ter Working Directory**, que pode ser removido por...



- Repositórios novos, criados do zero:
 1. Criar o repositório usando `git init --bare`.
- Repositórios já existentes:
 1. Ativar o modo *bare* com `git config core.bare true`
 2. Apagar o conteúdo da pasta, exceto a `.git`

8. Unindo *branches*

Unir *branches* envolve **juntar alterações de dois *branches*** (ou mais) em um único *branch*. Há dois comandos para se fazer isso...

git merge

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

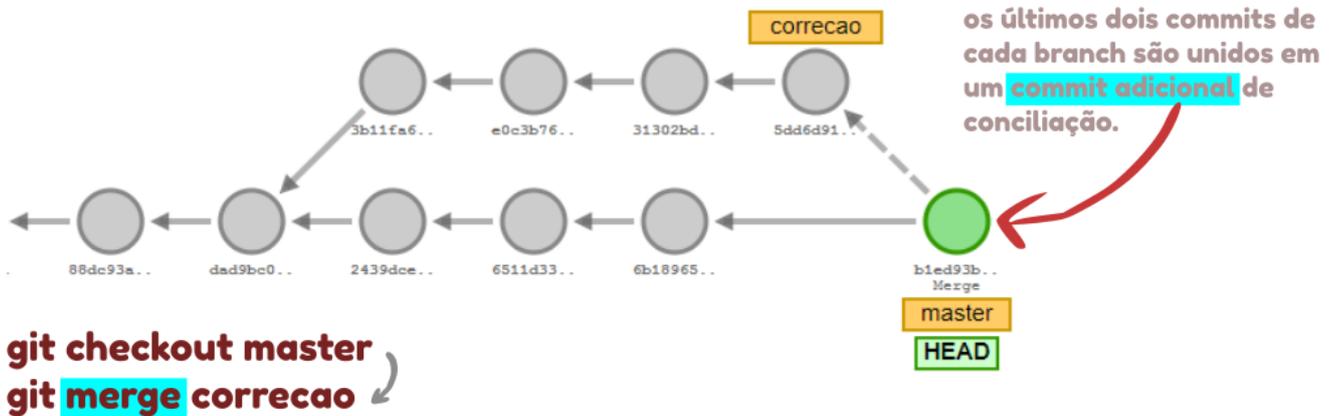


Figura 13. O comportamento do merge.

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.

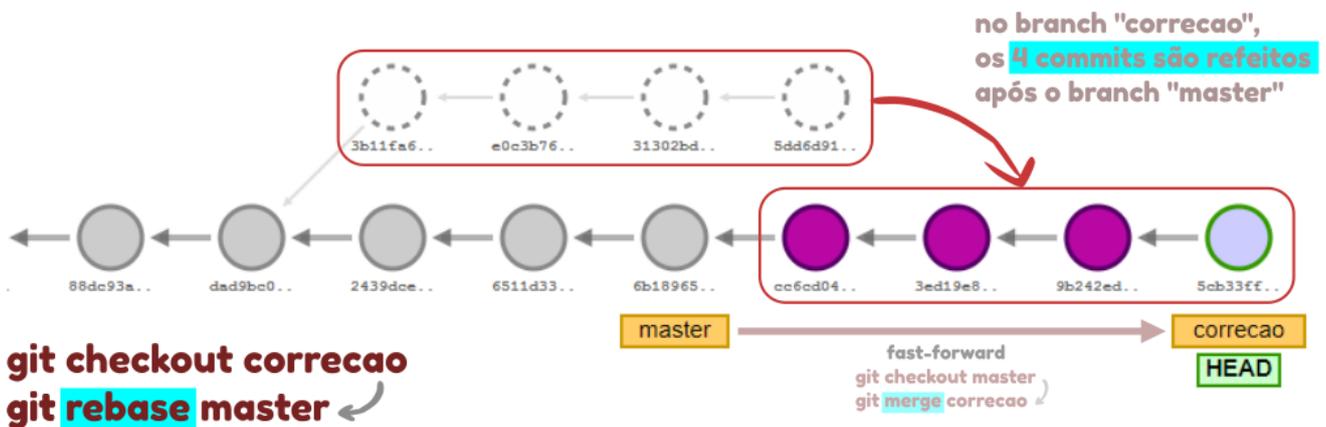


Figura 14. O comportamento do rebase.

Em caso de conflito nos arquivos...



- Usando **git merge** você faz a resolução uma única vez.
- Usando **git rebase** você pode ter que fazer a resolução para cada *commit* do *branch* atual que está sendo enviado para frente do *branch* de destino.

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 18. Terminal exibindo a saída do comando de ajuda do Git.

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*

Uma **manual completo** com tudo que você precisa saber sobre *Git*.

A versão brasileira está em processo de tradução colaborativa no *GitHub*. Se você quiser ajudar, souber inglês e entender o mínimo de *Git*, então aparece por lá.