

GruPy SP apresenta:



# FS2W #5 INTRODUÇÃO AO GIT

**Samuel Sampaio**

**@SamukaSmk**

# PRAZER, EU SOU O SAMUKA!

- Administrador de Sistemas, desde 2010
- Desenvolvedor Python, desde 2014
- Entusiasta da Cultura DevOps e Cloud Computing
- Eterno estudante da vida e seus mistérios
- Escolinha do Professor Samuka

**Telegram, twitter, facebook, gmail:**

**@SamukaSMK**





# DOMINANDO A ARTE DO ~~JEET KUNE DO~~ GIT

Aqui você irá:

- Entender a diferença entre Git e GitHub;
- Se familiarizar com o Git e controlar as versões do seu código;
- Colaborar com outros desenvolvedores em projetos no GitHub;
- Hospedar um site estático no GitHub Pages;

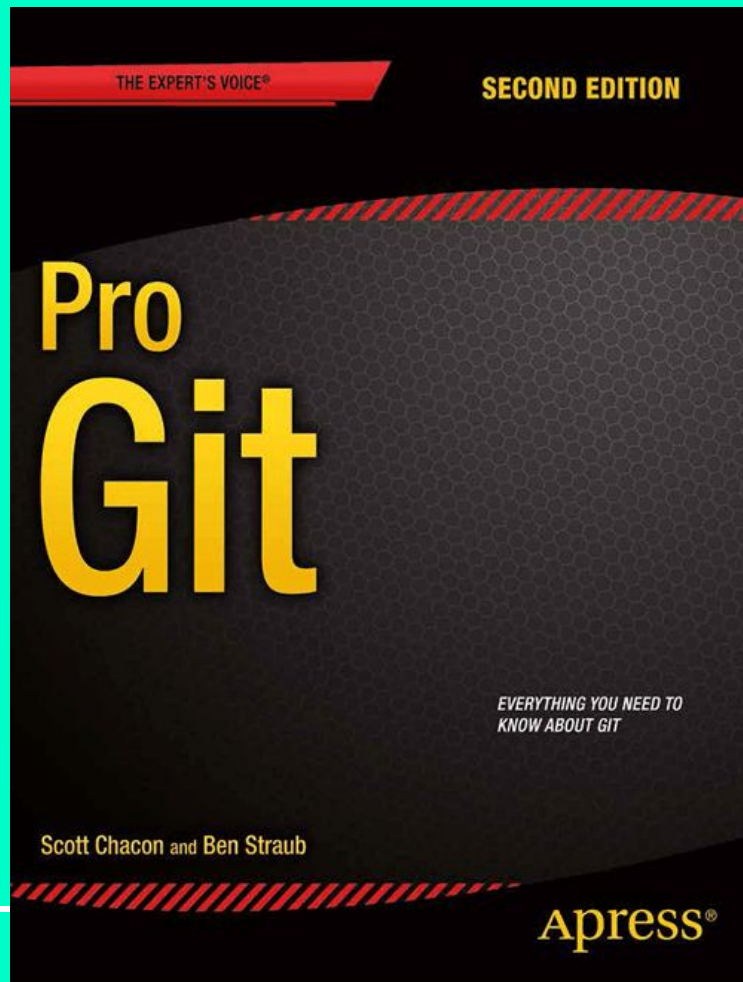
---

# LIVRO: PRO GIT

Por Scott CHACON; Ben STRAUB  
e a comunidade  
Editora Apress, 2014

Licença: [Creative Commons](#)  
[\(Attribution Non Commercial Share](#)  
[Alike 3.0\)](#)

Link para Download:  
<https://git-scm.com/book/en/v2>



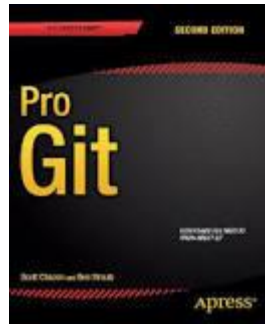
A pergunta que não quer calar:

O QUE É O GIT ?

GIT é uma ferramenta de controle de versão!

Obs: Não confunda Git com GitHub!

# O QUE É "CONTROLE DE VERSÃO", E PORQUE EU DEVERIA ME IMPORTAR?



Controle de versão é um sistema que registra alterações em um arquivo ou conjunto de arquivos ao longo do tempo para que você possa lembrar versões específicas mais tarde.

CHACON, SCOTT; STRAUB, BEN. TÍTULO: PRO GIT. EDITORA APRESS, 2014.

# FILOSOFIA DO CONTROLE DE VERSÃO:



**"Toda evolução de software é antes de tudo, uma história, descrita por algoritmos, através das linguagens de programação ou de máquina."**

**- SAMPAIO, SAMUEL MACIEL**

# É PARA QUE SERVE O GIT ?


- Controle de histórico do desenvolvimento;
- Organização de trabalho distribuído em equipe;
- Fácil resgate (rollback) de código;



# HISTÓRIA SOBRE O CONTROLE DE VERSÃO

## 0. Sem controle de versão

Gerenciando arquivos em pastas, gerando duplicidades, prática informal **muito comum** em gerência de **sites** apenas com **FTP**.



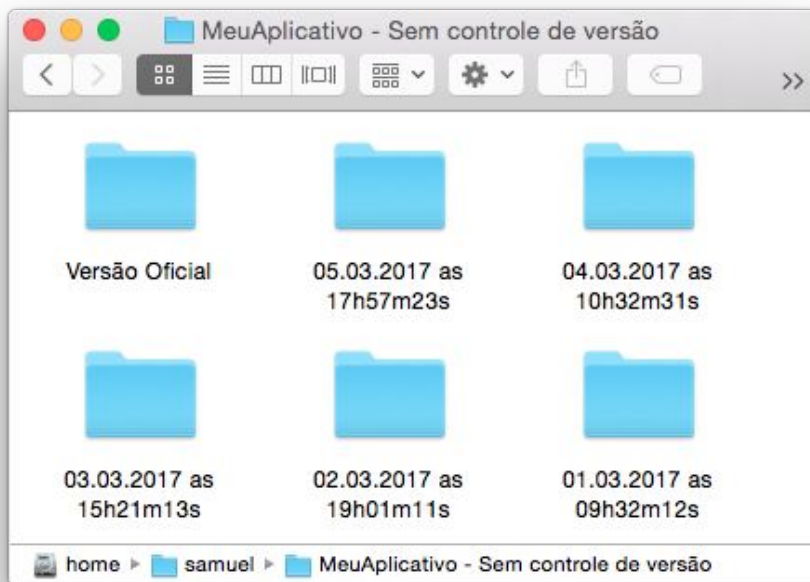
---

## 0. SEM CONTROLE DE VERSÃO

O método de controle de versão de muitas pessoas é copiar os arquivos para outro diretório (talvez um diretório com carimbo de tempo, se eles forem espertos).

**Isso é muito comum, porque é tão simples. Porém é fácil esquecer em qual diretório você está e acidentalmente sobrescrever o arquivo errado ou copiar arquivos que não quer.**

CHACON, SCOTT; STRAUB, BEN. TÍTULO: PRO GIT. EDITORA APRESS, 2014.



# PROBLEMA #1: ARMAZENAR APENAS ALTERAÇÕES DOS ARQUIVOS

Banco de dados de versionamento

## Primeira versão:

Criação do arquivo (meu\_primeiro\_arquivo.txt)

- **Adicionado** linhas #1, #2, #3 e #4

```
meu_primeiro_arquivo.txt
@@ -0,0 +1,4 @@
+1
+2  Arquivo de exemplo de diffs
+3
+4  1: Olá, essa é a primeira alteração feita neste arquivo
```

# PROBLEMA #1: ARMAZENAR APENAS ALTERAÇÕES DOS ARQUIVOS

## Banco de dados de versionamento

### Segunda versão:

Alteração do arquivo (meu\_primeiro\_arquivo.txt)

- **Adicionado** linhas #5 e #6

### Primeira versão:

Criação do arquivo (meu\_primeiro\_arquivo.txt)

- **Adicionado** linhas #1, #2, #3 e #4

```
meu_primeiro_arquivo.txt
@@ -2,3 +2,5 @@
2      2      Arquivo de exemplo de diffs
3      3
4      4      1: Olá, essa é a primeira alteração feita neste arquivo
+5
+6      2: Já esta segunda linha, é a segunda vez que altero aqui...
```

```
+ meu_primeiro_arquivo.txt
@@ -0,0 +1,4 @@
+1
+2      Arquivo de exemplo de diffs
+3
+4      1: Olá, essa é a primeira alteração feita neste arquivo
```

# PROBLEMA #1: ARMAZENAR APENAS ALTERAÇÕES DOS ARQUIVOS

## Banco de dados de versionamento

### Terceira versão:

Alteração do arquivo (meu\_primeiro\_arquivo.txt)

- **Removido** linhas #4, #5 e #6
- **Adicionado** linha #4

```
meu_primeiro_arquivo.txt
@@ -1,6 +1,4 @@
1      1
2      2  Arquivo de exemplo de diffs
3      3
-4
-5
-6
+4  Nesta terceira alteração, apaguei todas as outras linhas
```

### Segunda versão:

Alteração do arquivo (meu\_primeiro\_arquivo.txt)

- **Adicionado** linhas #5 e #6

```
meu_primeiro_arquivo.txt
@@ -2,3 +2,5 @@
2      2  Arquivo de exemplo de diffs
3      3
4      4  1: Olá, essa é a primeira alteração feita neste arquivo
+5
+6  2: Já esta segunda linha, é a segunda vez que altero aqui...
```

### Primeira versão:

Criação do arquivo (meu\_primeiro\_arquivo.txt)

- **Adicionado** linhas #1, #2, #3 e #4

```
meu_primeiro_arquivo.txt
@@ -0,0 +1,4 @@
+1
+2  Arquivo de exemplo de diffs
+3
+4  1: Olá, essa é a primeira alteração feita neste arquivo
```

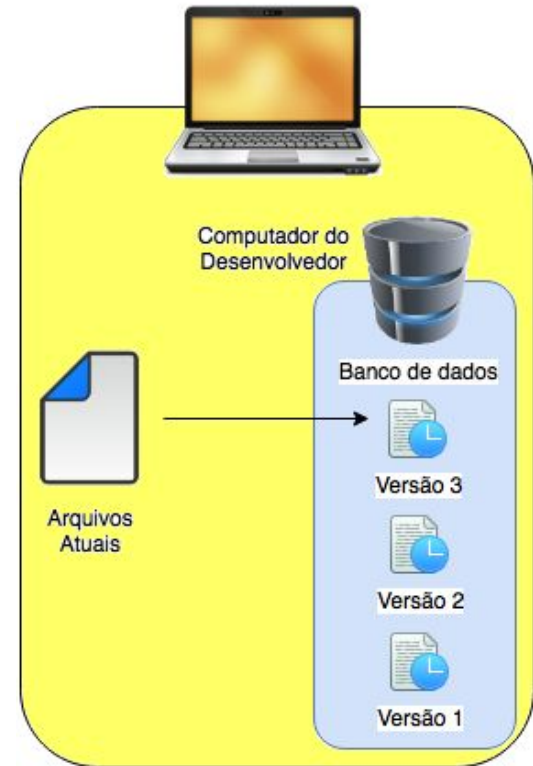
# PROBLEMA #1: ARMAZENAR APENAS ALTERAÇÕES DOS ARQUIVOS

## 1. SISTEMAS LOCAIS DE CONTROLE DE VERSÃO (VCS)

Para lidar com este problema, programadores há muito tempo desenvolveram VCSs locais, que tem um banco de dados simples, que mantém todas as alterações nos arquivos sob controle de revisão. CHACON, SCOTT; STRAUB, BEN. TÍTULO: PRO GIT. EDITORA APRESS, 2014.

Uma vez que os arquivos são alterados intencionalmente, apenas as mudanças dos arquivos são salvas no banco de dados.

A Desvantagem desse modelo é o banco de dados é único, estático no computador de cada desenvolvedor. O compartilhamento ainda seria do modo convencional, comprimindo em um arquivo só e enviando a outros desenvolvedores.



# HISTÓRIA SOBRE O CONTROLE DE VERSÃO

## 0. Sem controle de versão

Gerenciando arquivos em pastas, gerando duplicidades, prática informal **muito comum** em gerência de **sites** apenas com **FTP**.

## 1. Sistemas Locais de Controle de Versão (VCS)

Em 1972, Marc J. Rochkind desenvolveu o **SCCS** na Bell Labs, rodando em um mainframe IBM System/370. Outros: RCS<sub>(1982)</sub>, PVCS<sub>(1985)</sub>, QVCS<sub>(1991)</sub>

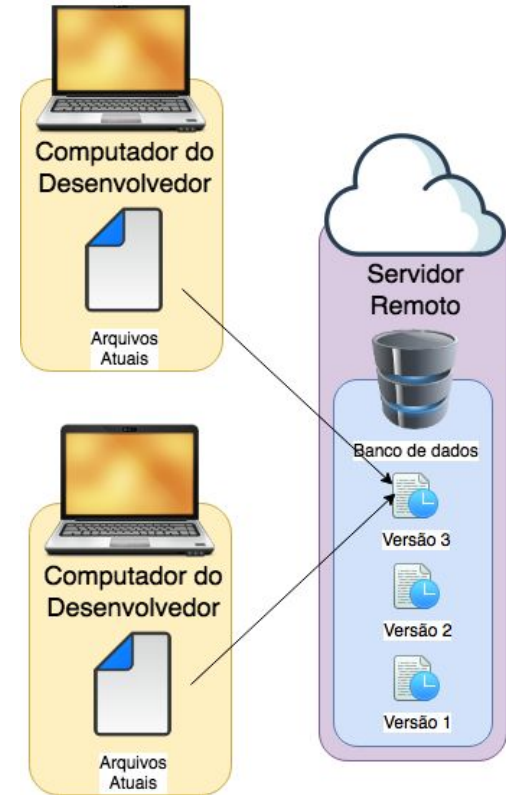
# PROBLEMA #2: TRABALHO CENTRALIZADO UM ÚNICO PONTO

## 2. SISTEMAS CENTRALIZADOS DE CONTROLE DE VERSÃO (CVCS)

O CVCS é similar aos sistema anteriores, porém com a vantagem do banco de dados estar compartilhado, através de um servidor. Permitindo fácil colaboração, no compartilhamento de códigos e trazendo um avanço para as práticas de Desenvolvimento de software.

A desvantagem é que todos os desenvolvedores dependem que o servidor de banco de dados central funcione perfeitamente, se der problema, ninguém pode colaborar ou salvar as alterações de versão.

Se for corrompido e não tiver backups, você perde absolutamente toda a história do projeto.





# HISTÓRIA SOBRE O CONTROLE DE VERSÃO

## 0. Sem controle de versão

Gerenciando arquivos em pastas, gerando duplicidades, prática informal **muito comum** em gerência de **sites** apenas com **FTP**.

## 2. Sistemas Centralizados de Controle de Versão (CVCS)

Em 1986, nascia o **CVS** por um dos times da GNU. Só apenas nos anos 2000 que surgiu o **Apache Subversion (SVN)** pela CollabNet. Outros: ClearCase(1992), CMVC(1994), Perforce Helix(1995)

## 1. Sistemas Locais de Controle de Versão (VCS)

Em 1972, Marc J. Rochkind desenvolveu o **SCCS** na Bell Labs, rodando em um mainframe IBM System/370. Outros: RCS(1982), PVCS(1985), QVCS(1991)

# PROBLEMA #3: TRABALHO DISTRIBUÍDO, COM INÚMERAS RAMIFICAÇÕES

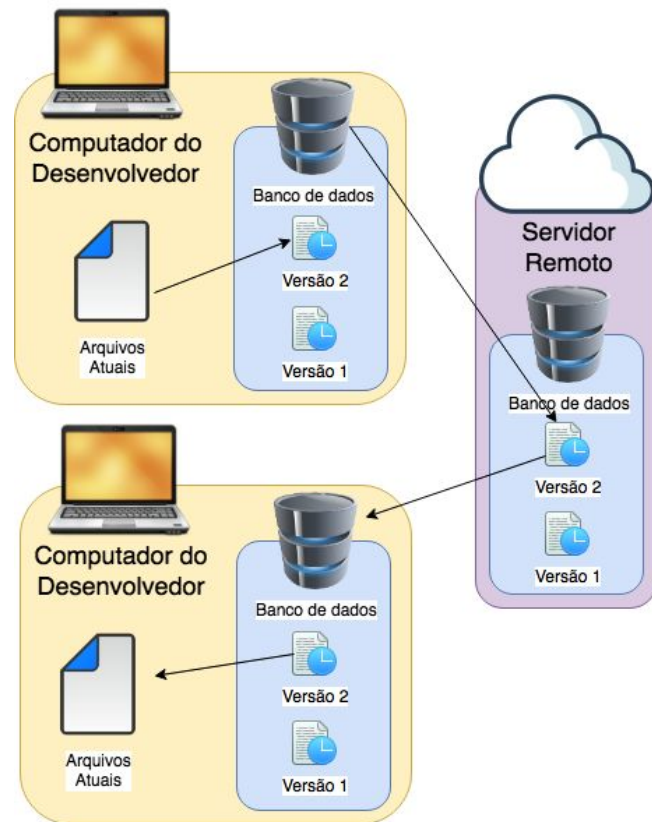
## 3. SISTEMAS DISTRIBUÍDOS DE CONTROLE DE VERSÃO (DVCS)

A grande inovação dos DVCS, é implementação de **bancos de dados distribuídos**, em todas as pontas.

Ao início do trabalho, o desenvolvedor obtém o histórico de evolução do projeto de software, **atualizando o banco de dados** da sua máquina **local**, com o histórico existente do **banco de dados do servidor central**. (*git clone, git fetch & git pull*)

Ao término do trabalho, o desenvolvedor **atualiza de volta o banco de dados** do servidor **central**, com as **alterações feitas no banco de dados** da sua máquina **local**. (*git add, git commit e git push*)

Isso provém um meio de se trabalhar em paralelo com diversas equipes, em inúmeras ramificações. (*branches*)



# HISTÓRIA SOBRE O CONTROLE DE VERSÃO

## 0. Sem controle de versão

Gerenciando arquivos em pastas, gerando duplicidades, prática informal **muito comum** em gerência de **sites** apenas com **FTP**.

## 1. Sistemas Locais de Controle de Versão (VCS)

Em 1972, Marc J. Rochkind desenvolveu o **SCCS** na Bell Labs, rodando em um mainframe IBM System/370. Outros: RCS(1982), PVCS(1985), QVCS(1991)

## 2. Sistemas Centralizados de Controle de Versão (CVCS)

Em 1986, nascia o **CVS** por um dos times da GNU. Só apenas nos anos 2000 que surgiu o **Apache Subversion (SVN)** pela CollabNet. Outros: ClearCase(1992), CMVC(1994), Perforce Helix(1995)

## 3. Sistemas Distribuídos de Controle de Versão (DVCS)

Em 1996, foi desenvolvido o Code Co-op, pela Reliable Software. Em 2005 finalmente nasceu o Git, por Linus Torvald e Junio Hamano. Outros: BitKeeper(1998), Darcs(2002), Mercurial(2005), GNU Bazaar(2005)

# MÃOS À OBRA

[http://rogerdudler.github.io/gti-guide/index.pt\\_BR.html](http://rogerdudler.github.io/gti-guide/index.pt_BR.html)

# INSTALANDO O GIT

<https://git-scm.com/book/pt-br/v1/Primeiros-passos-Instalando-Git>

No Linux Ubuntu, Debian, Mint:

```
$ sudo apt-get install git
```

No Linux RedHat, CentOS, Fedora

```
$ sudo yum install git-core
```

No Windows:

<https://git-for-windows.github.io/>

No Mac OSX:

- Via [home brew](#): `$ brew install git`
- Via [pacote](http://sourceforge.net/projects/git-osx-installer/): <http://sourceforge.net/projects/git-osx-installer/>
- Via [ports](#): `$ sudo port install git-core +svn +doc +bash_completion +gitweb`

# CONFIGURANDO O GIT

1. Definindo o nome do usuário (que aparecerá no commit):

```
$ git config --global user.name "Samuel Sampaio"
```

2. Definindo o e-mail (que aparecerá no commit)

```
$ git config --global user.email samukasmk@gmail.com
```

A pergunta que não quer calar #2:

O QUE É O GITHUB ?

GitHub é um serviço de git, disponível via internet.

Ele possui uma interface administrativa web poderosa para o trabalho em colaboração open source.

**<http://github.com>**

VAMOS CONTAR UMA  
HISTÓRIA, COM  
CONTROLE DE VERSÃO?





# CRIE UM NOVO REPOSITÓRIO ABERTO: GIT-BRUCLEE

## Create a new repository

A repository contains all the files for your project, including the revision history.

Owner

 samukasmk ▾

Repository name

git-brucelee

Great repository names are short and memorable. Need inspiration? How about **fictional-spork**.

Description (optional)



**Public**

Anyone can see this repository. You choose who can commit.



**Private**

You choose who can see and commit to this repository.



**Initialize this repository with a README**

This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.

Add .gitignore: **None** ▾

Add a license: **None** ▾



Create repository

# LICENÇAS DE SOFTWARE MAIS FAMOSAS

SUA  
PRÓPRIA

APACHE 2.0

MIT

BSD

GPL 2.1

GPL 3.0

LGPL 3.0

AFFERO GPL

<https://tldrlegal.com/license/>

# DEFININDO UM REPOSITÓRIO GIT

## 1. CLONANDO UM REPOSITÓRIO EXISTENTE DO SERVIDOR

```
$ git clone git@github.com:samukasmk/git-brucelee.git
```

```
$ cd git-brucelee
```

---

## 1. CRIANDO REPOSITÓRIO DE UMA PASTA LOCAL EXISTENTE

```
$ cd /caminho/para/pasta/do/projeto
```

```
$ git init
```

# PRIMEIRAS ALTERAÇÕES

## 2. CRIANDO UM ARQUIVO (README.MD)

\$ **vim** **README.md**

**ESQ** **i**

```
# 0 Voo do Dragão (1973) - Bruce Lee  
- Cena 1: Tang Lung aparece no aeroporto
```

**ESQ** : **wq**

# STATUS: ARQUIVOS NÃO RASTREADOS

(UNTRACKED FILES)

## 3. ANALISANDO AS ALTERAÇÕES DO NOVO ARQUIVO

\$ **git status**

```
On branch master
```

```
Initial commit
```

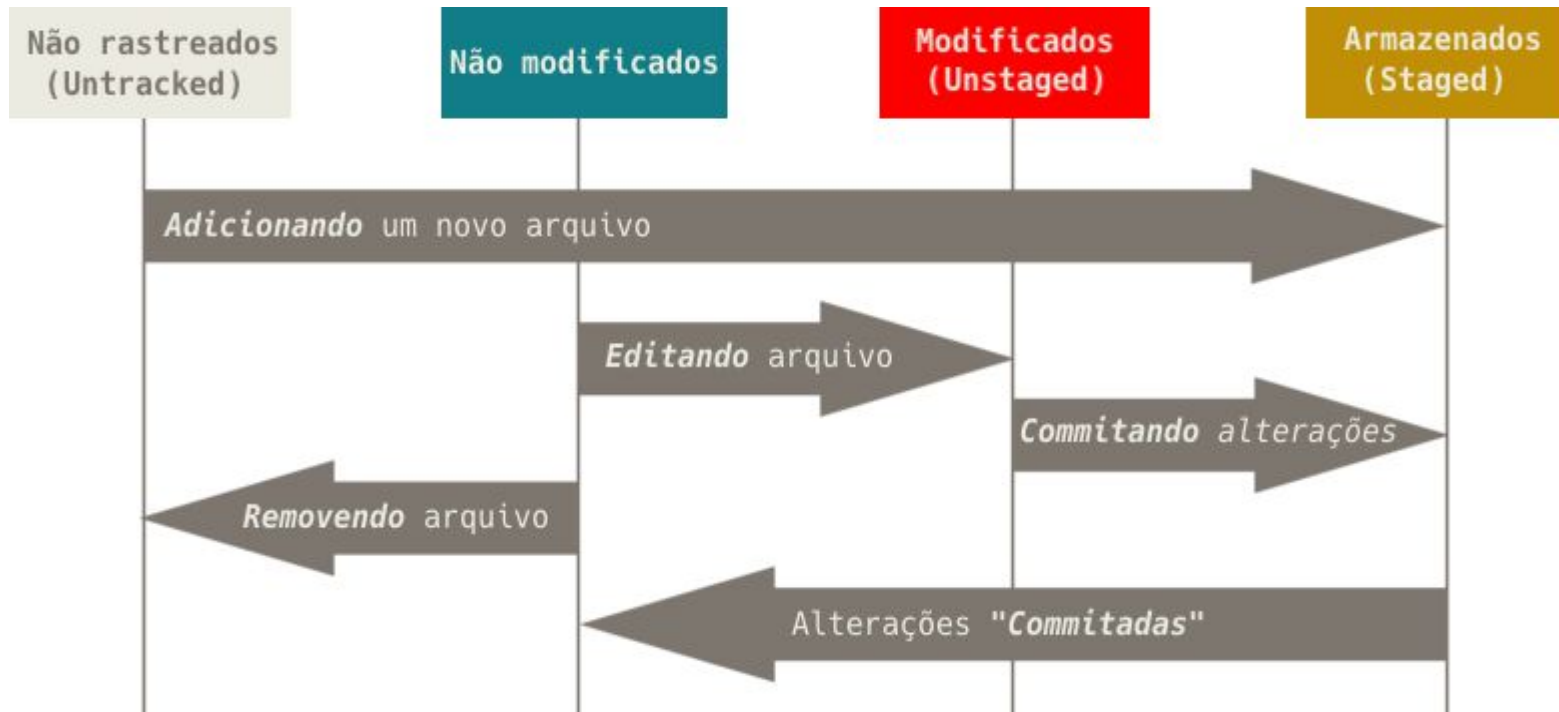
```
Untracked files:
```

```
(use "git add <file>..." to include in what will be committed)
```

```
README.md
```

```
nothing added to commit but untracked files present (use "git add" to track)
```

# STATUS DE VERSIONAMENTO



DE: WORKING DIRECTORY  
PARA: STAGING AREA

#### 4. ADICIONANDO AS ALTERAÇÕES A ÁREA (STAGING)

```
$ git add README.md
```

```
$ git add .
```

# STATUS: ALTERAÇÕES ADICIONADAS (STAGED)

## 5. ANALISANDO AS ALTERAÇÕES ADICIONADAS AO PROXIMO COMMIT

\$ **git status**

```
On branch master

Initial commit

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

    new file:   README.md
```



# GERANDO NOSSA PRIMEIRA VERSÃO

## 6. SALVANDO AS ALTERAÇÕES EM UM COMMIT

```
$ git commit -m "Cena: 1"
```



# GERANDO NOSSA PRIMEIRA VERSÃO

## 7. VISUALIZANDO A PILHA DE COMMITS (HEAD)

\$ **git log**

```
commit 7e258471dadea46f8cf63c8c47c20cd057d69d20
```

```
Author: SamukaSMk <samukasmk@gmail.com>
```

```
Date: Sat Mar 11 00:28:13 2017 -0300
```

```
Cena 1
```

```
commit b5fed86aca1f2c403e8ae7c38eec6faaffd5bce3
```

```
Author: Samuel Sampaio <samukasmk@gmail.com>
```

```
Date: Sat Mar 11 00:25:47 2017 -0300
```

```
Initial commit
```

# EXEMPLO DE GRAFO: O VOO DO DRAGÃO 1973



Commit



Branch: master

Por: Bruce Lee



Cena: 4



Cena: 3



Cena: 2



Cena: 1



**Tang Lung (Bruce Lee)** derrota **Colt (Chuck Norris)** depois de uma luta disputada.



**Tang Lung (Bruce Lee)** e **Colt (Chuck Norris)** se encaram, ambos se preparam para lutar.



**Tang Lung (Bruce Lee)** passa vergonha no restaurante do aeroporto por não saber italiano



**Tang Lung (Bruce Lee)** aparece no aeroporto, viajando de Hong Kong até Roma, com seu jeito tímido e perdido

# EXERCITANDO O FLUXO DE COMMITS

REPITA DOS PASSOS: 3. AO 7. (3 VEZES)

ADICIONANDO CADA CENA A UM COMMIT

```
$ vim README.md
```

```
$ git status
```

```
$ git add .
```

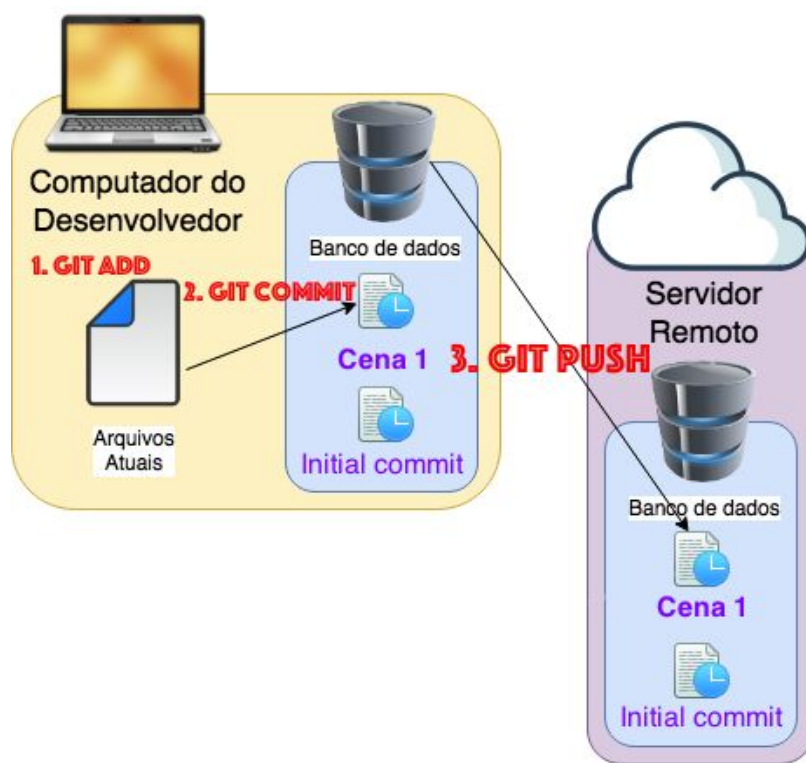
```
$ git status
```

```
$ git commit -m "Cena: 2"
```

```
$ git log
```

## 8. ENVIADOS OS NOVOS COMMITS AO REPOSITÓRIO REMOTO (GITHUB)

\$ **git push origin master**



# SE ORIENTANDO PELO GITHUB

📁 samukasmk / **git-brucelee**

👁 Unwatch ▾

1

★ Star

0

🍴 Fork

0

<> Code

🔔 Issues 0

🔗 Pull requests 0

📁 Projects 0

📖 Wiki

⚡ Pulse

📊 Graphs

⚙ Settings

Um filme escrito, atuado e dirigido por Bruce Lee

Edit

[Add topics](#)

📄 5 commits

🌿 1 branch

📦 0 releases

👤 1 contributor

📄 Apache-2.0

Branch: master ▾

New pull request

Create new file

Upload files

Find file

Clone or download ▾



samukasmk Cena: 4

Latest commit 900dd7d 17 seconds ago

📄 LICENSE

Initial commit

5 minutes ago

📄 README.md

Cena: 4

16 seconds ago

📖 README.md

## O Voo do Dragão (1973) - Bruce Lee

- Cena 1: Tang Lung aparece no aeroporto

# VISUALIZANDO OS COMMITS (PELO GITHUB)

 samukasmk / git-brucelee

 Unwatch ▾


1

★ Star

0

 Fork

0

 Code

 Issues 0

 Pull requests 0

 Projects 0

 Wiki

 Pulse

 Graphs

 Settings

Branch: master ▾

 Commits on Mar 11, 2017



**Cena: 4**

samukasmk committed 3 minutes ago



900dd7d



**Cena: 3**

samukasmk committed 3 minutes ago



75c7b4e



**Cena: 2**

samukasmk committed 4 minutes ago



1c8b699



**Cena: 1**

samukasmk committed 6 minutes ago



eda44f8



**Initial commit**

samukasmk committed 7 minutes ago



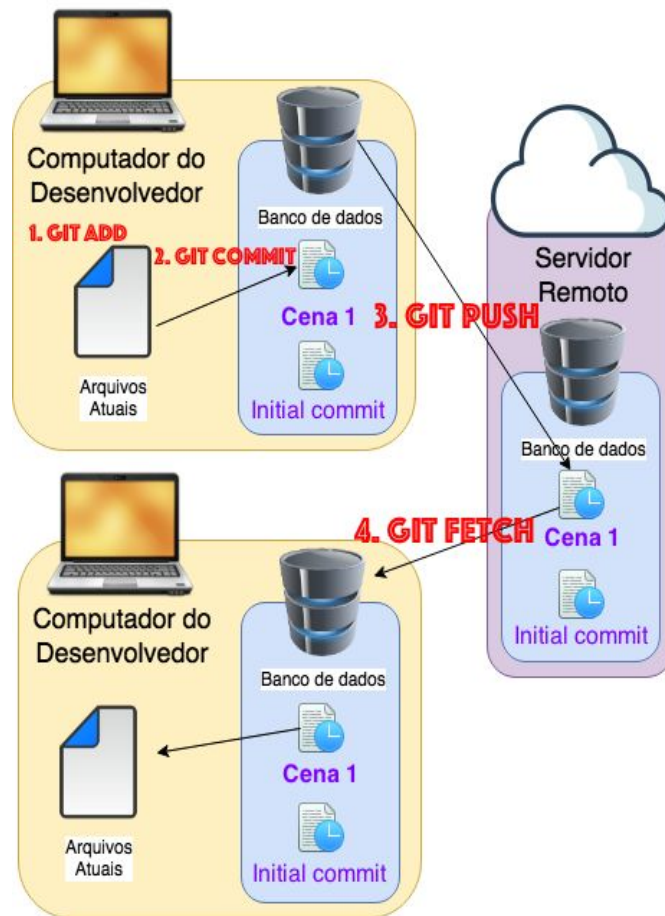
7aef102



# 9. OBTENDO OS COMMITS DO REPOSITÓRIO REMOTO (GITHUB)

PARA O MEU REPOSITÓRIO LOCAL

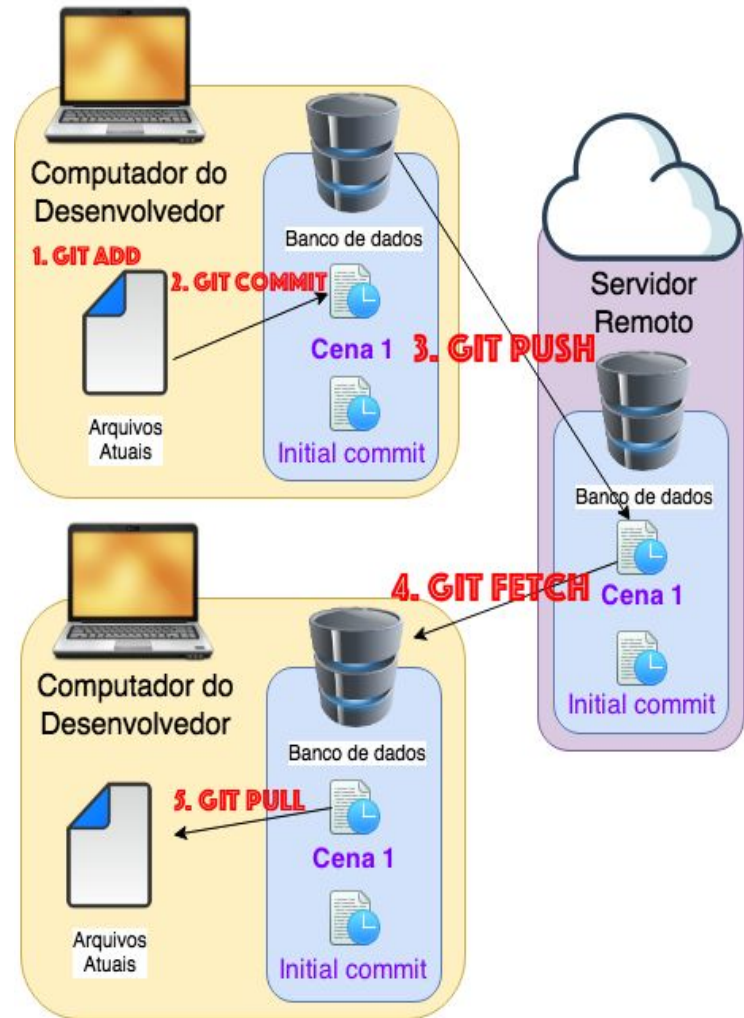
\$ **git** **fetch**





# 10. APLICANDO OS NOVOS COMMITS PARA A PASTA DE TRABALHO ATUAL

\$ **git pull**



E COMO EU FAÇO  
PARA PUBLICAR UM WEBSITE,  
DE GRAÇA, COM O GITHUB PAGES?

# ENTENDA O QUE SÃO BRANCHS (RAMIFICAÇÕES)



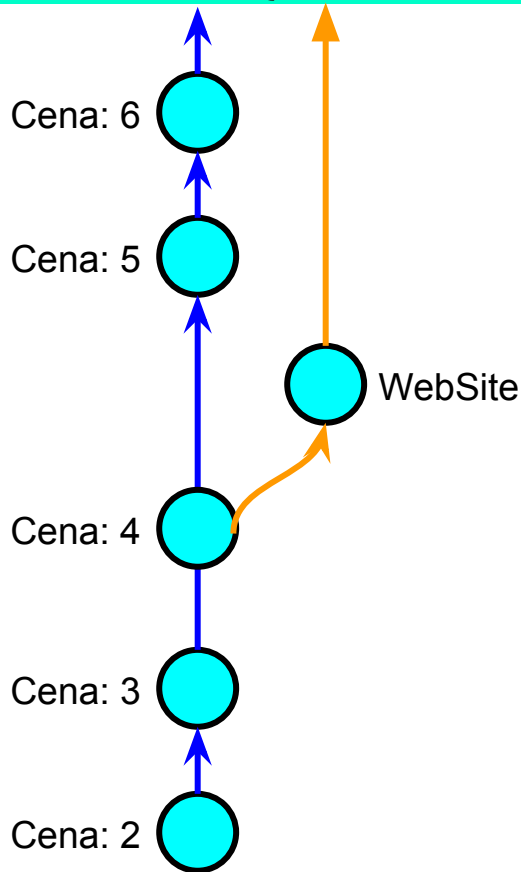
Commit



Branch:  
master



Branch:  
gh-pages



**Mais um outro commit no branch: master**

**Outro commit no branch: master**

**Publicação do site em uma nova ramificação (branch): gh-pages**



**Tang Lung (Bruce Lee) derrota Colt (Chuck Norris)** depois de uma luta disputada.



**Tang Lung (Bruce Lee) e Colt (Chuck Norris) se encaram, ambos se preparam para lutar.**

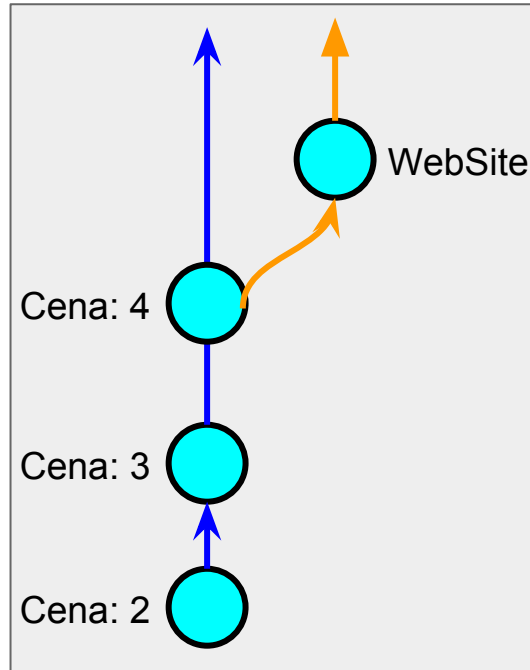


**Tang Lung (Bruce Lee) passa vergonha no restaurante do aeroporto por não saber italiano**

# HOSPEDANDO WEBPAGES DE GRAÇA!

## 11. CRIANDO UMA RAMIFICAÇÃO PARA O GITHUB PAGES

```
$ git checkout -b gh-pages
```



## 12. CRIANDO A PÁGINA PRINCIPAL DO SEU SITE (INDEX.HTML)

\$ vim index.html

ESQ i

```
<html>
<head></head>

<body>
  <h1>O Voo do Dragão (1973) - Bruce Lee</h1>

  <li>Cena 1: Tang Lung aparece no aeroporto</li>
  <li>Cena 2: Tang Lung passa vergonha no restaurante por não falar italiano</li>
  <li>Cena 3: Tang Lung e Colt se preparam para o combate</li>
  <li>Cena 4: Colt é derrotado por Tang Lung em uma luta disputada</li>

</body>

</html>
```

ESQ : wq

## 13. CRIANDO UM COMMIT COM O CONTEÚDO DO WEB SITE

```
$ git add index.html
```

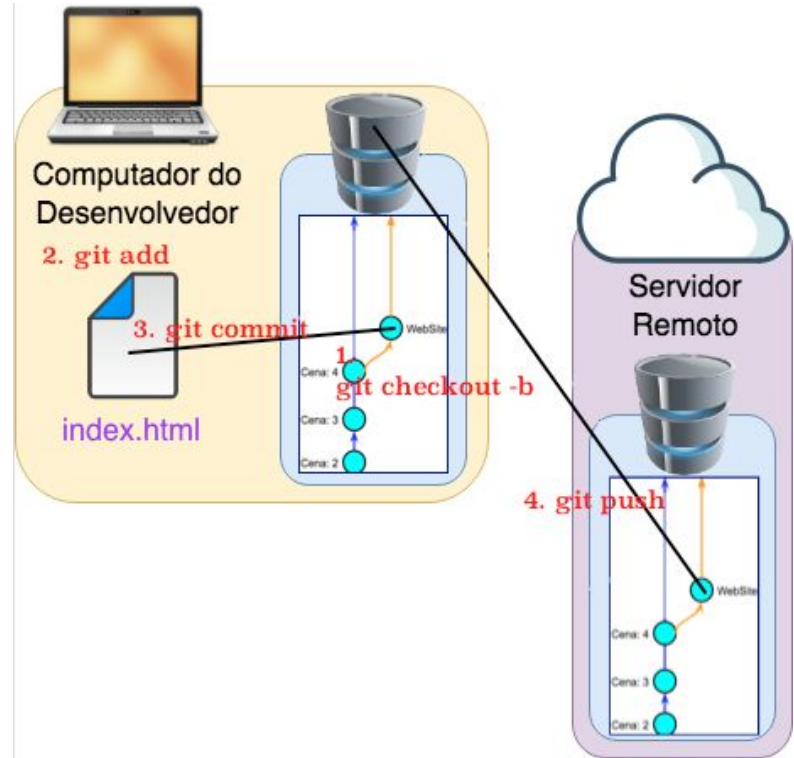
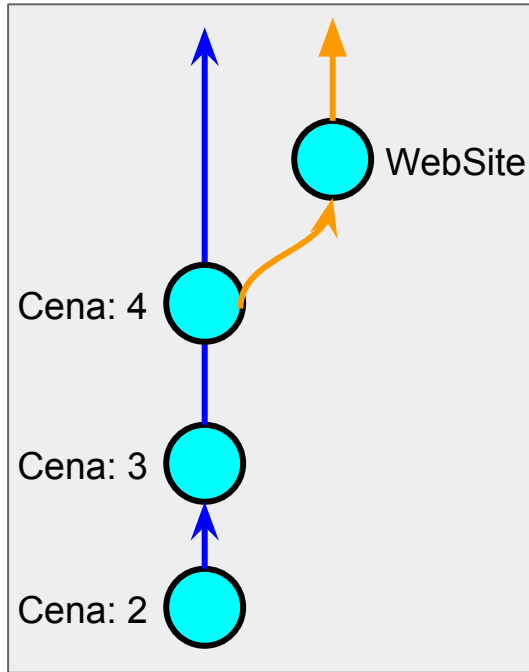
```
$ git commit -m "WebSite v1"
```

The screenshot shows the GitHub interface for the repository 'samukasmk / git-brucelee'. At the top, there are buttons for 'Unwatch', 'Star' (0), and 'Fork' (0). Below this is a navigation bar with links for 'Code', 'Issues' (0), 'Pull requests' (0), 'Projects' (0), 'Wiki', 'Pulse', 'Graphs', and 'Settings'. A dropdown menu indicates the current branch is 'gh-pages'. The main section is titled 'Commits on Mar 11, 2017' and lists four commits. Each commit entry includes a profile picture, the commit title, the author 'samukasmk', the time 'committed 2 hours ago', a green checkmark, a file icon, the commit hash, and a code icon.

Commit Title	Author	Time	Commit Hash
WebSite	samukasmk	committed 2 hours ago	b5273af
Cena: 4	samukasmk	committed 2 hours ago	900dd7d
Cena: 3	samukasmk	committed 2 hours ago	75c7b4e
Cena: 2	samukasmk	committed 2 hours ago	1c8b699

# 14. ENVIADOS OS NOVOS COMMITS AO REPOSITÓRIO REMOTO (GITHUB)

\$ **git push origin gh-pages**



ACESSANDO SEU SITE PELA URL

<http://LOGIN.github.io/REPOSITORIO>

EXEMPLO DE URL DO MEU REPOSITÓRIO

<http://samukasmk.github.io/git-brucelee>



# OBRIGADO!

# DÚVIDAS?

**SAMUEL SAMPAIO**

Telegram: @SamukaSMK

Facebook: @SamukaSMK

Twitter: @SamukaSMK

Email: [samuel@smk.net.br](mailto:samuel@smk.net.br)

QUER PARTICIPAR DA ESCOLINHA

DO PROFESSOR SAMUKA ?

DEIXE SEUS CONTATOS

---