

CAPÍTULO 1

Introdução

1.1 MANTENDO O HISTÓRICO DO CÓDIGO

Vida de programador não é fácil. Há sempre uma pressão por entregas rápidas de novas funcionalidades. Mas, apesar da pressa, é necessário prestar atenção no que estamos fazendo, mesmo se a alteração for pequena. Ao mexermos em um código existente é importante tomarmos cuidado para não quebrar o que já funciona.

Por isso, queremos mexer o mínimo possível no código. Temos medo de remover código obsoleto, não utilizado ou até mesmo comentado, mesmo que mantê-lo já nem faça sentido. Não é incomum no mercado vermos código funcional acompanhado de centenas de linhas de código comentado.

Sem dúvida, **é interessante manter o histórico** do código dos projetos, para entendermos como chegamos até ali. Mas manter esse histórico junto ao código atual, com o decorrer do tempo, deixa nossos projetos confusos,

poluídos com trechos e comentários que poderiam ser excluídos sem afetar o funcionamento do sistema.

Seria bom se houvesse uma maneira de navegarmos pelo código do passado, como uma *máquina do tempo* para código...

1.2 TRABALHANDO EM EQUIPE

Mas, mesmo que tivéssemos essa máquina do tempo, temos outro problema: muito raramente trabalhamos sozinhos.

Construir um sistema em equipe é um grande desafio. Nosso código tem que **se integrar de maneira transparente** e sem emendas com o código de todos os outros membros da nossa equipe.

Como podemos detectar que estamos alterando o mesmo código que um colega? Como mesclar as alterações que fizemos com a demais alterações da equipe? E como identificar conflitos entre essas alterações? Fazer isso manualmente, com cadernetas ou planilhas e muita conversa, parece trabalhoso demais e bastante suscetível a erros e esquecimentos.

Seria bom que tivéssemos um *robô de integração* de código, que fizesse todo esse trabalho automaticamente...

1.3 SISTEMAS DE CONTROLE DE VERSÃO

Existem ferramentas que funcionam como *máquinas do tempo* e *robôs de integração* para o seu código. Elas nos permitem acompanhar as alterações desde as versões mais antigas. Também é possível detectar e mesclar alterações nos mesmos arquivos, além de identificar conflitos, tudo de maneira automática.

Essas ferramentas são chamadas de **sistemas de controle de versão**.

Nesse tipo de ferramenta, há um **repositório** que nos permite obter qualquer versão já existente do código. Sempre que quisermos controlar as versões de algum arquivo, temos que informar que queremos rastreá-lo no repositório. A cada mudança que desejamos efetivar, devemos armazenar as alterações nesse repositório.

Alterações nos mesmos arquivos são mescladas de maneira automática sempre que possível. Já possíveis conflitos são identificados a cada vez que obtemos as mudanças dos nossos colegas de time.

Desde a década de 1990, existe esse tipo de ferramenta. Alguns exemplos de sistemas de controle de versão mais antigos são CVS, ClearCase, SourceSafe e SVN (que ainda é bastante usado nas empresas).

Em meados da década de 2000, surgiram sistemas de controle de versão mais modernos, mais rápidos e confiáveis, como Mercurial, Bazaar e, é claro, Git.

1.4 CONTROLE DE VERSÃO RÁPIDO E CONFIÁVEL COM GIT

O **Git** é um sistema de controle de versão que, pela sua estrutura interna, é uma máquina do tempo extremamente rápida e é um robô de integração bem competente.

Foi criado em 2005 por **Linus Torvalds**, o mesmo criador do Linux, que estava descontente com o BitKeeper, o sistema de controle de versão utilizado no desenvolvimento do *kernel* do Linux.

Hoje em dia, além do *kernel* do Linux, a ferramenta é utilizada em diversos outros projetos de código aberto. O Git também é bastante utilizado em empresas em todo o mundo, inclusive no Brasil.

Atualmente, conhecer bem como utilizar o Git é uma habilidade importante para uma carreira bem-sucedida no desenvolvimento de software.

1.5 HOSPEDANDO CÓDIGO NO GITHUB

Em 2008, foi criado o **GitHub**, uma aplicação Web que possibilita a hospedagem de repositórios Git, além de servir como uma rede social para programadores.

Diversos projetos de código aberto importantes são hospedados no GitHub como jQuery, Node.js, Ruby On Rails, Jenkins, Spring, JUnit e muitos outros.

1.6 O PROCESSO DE ESCRITA DESSE LIVRO

A utilização do Git não é restrita apenas ao desenvolvimento de software, muitos administradores de rede, por exemplo, utilizam o Git para manter o histórico de evolução de arquivos de configurações em servidores.

Acreditem, até mesmo a escrita desse livro foi feita utilizando o Git!

Não apenas esse, mas todos os livros da editora **Casa do Código** utilizam o Git como ferramenta de controle de versão, para manter o histórico de evolução dos capítulos. O GitHub também é utilizado para hospedagem dos repositórios dos livros.

CAPÍTULO 2

Tour prático

Neste capítulo, faremos um *tour* bem prático sobre como usar o Git para versionar nossos projetos. Não se preocupe com o significado dos comandos. No decorrer do livro, todos os comandos usados aqui serão explicados com profundidade.

2.1 INSTALANDO E CONFIGURANDO O GIT

Antes de utilizarmos o Git, é fundamental instalá-lo. Escolha a seguir o Sistema Operacional apropriado e mãos à obra!

Instalando no Windows

Acesse a seguinte URL, faça o download e instale a última versão disponível: <http://msysgit.github.io/>

A instalação é bastante simples. Escolha as opções padrão.

Serão instalados alguns programas, sendo o mais importante o **Git Bash**, que permite que o Git seja executado pela linha de comando no Windows.

Dê duplo clique no ícone do Git Bash e será aberto um terminal, com o seguinte *prompt* na linha de comando:

```
$
```

Esse prompt será seu amigo a partir de agora. Não tenha medo! Sempre que falarmos de terminal, estaremos falando do Git Bash.

Instalando no Mac

Baixe a última versão do instalador gráfico do Git para Mac OS X a partir do link: <https://code.google.com/p/git-osx-installer/downloads>

Abra um terminal e prepare-se para utilizar o Git!

Instalando no Linux

Para instalar o Git no Ubuntu, ou em uma outra distribuição baseada em Debian, execute em um terminal:

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

No Fedora, utilize:

```
$ sudo yum install git
```

Para as demais distribuições do Linux, veja o comando em: <http://git-scm.com/download/linux>

Configurações básicas

É importante nos identificarmos para o Git, informando nosso nome e e-mail. Em um terminal, execute os comandos a seguir:

```
$ git config --global user.name "Fulano da Silva"  
$ git config --global user.email fulanodasilva.git@gmail.com
```

Claro, utilize **seu nome e e-mail**!

A LINHA DE COMANDO

A maneira mais comum de usar Git é pela linha de comando, acessível através de um terminal. É o jeito que a maior parte dos bons profissionais do mercado utiliza o Git e será nossa escolha nesse livro.

GITHUB FOR WINDOWS

A maioria dos usuários do Windows não tem o hábito de utilizar o prompt de comandos, e prefere instalar alguma aplicação visual para trabalhar com o Git.

Uma destas aplicações é o **GitHub for Windows**, e mostraremos como utilizá-la no capítulo 11.

2.2 CRIANDO UM ARQUIVO TEXTO PARA VERSIONARMOS

Antes de utilizarmos o Git, vamos criar na sua **pasta pessoal**, um diretório chamado `citacoes` com um arquivo `filmes.txt`.

Dentro do arquivo `filmes.txt`, coloque o seguinte conteúdo:

```
"Não há certezas, apenas oportunidades." (V de Vingança)
"Diga 'olá' para meu pequeno amigo!" (Scarface)
```

PASTA PESSOAL

A pasta pessoal (*ou home directory*, em inglês) é o local dos arquivos de usuário como documentos, fotos, músicas etc.

Se você não souber onde é a pasta pessoal, digite o seguinte comando em um terminal:

```
$ echo ~
```

No Windows Vista, 7 ou 8, será algo como `C:\Users\Fulano\` ou, no Git Bash, `/c/Users/Fulano/`.

No Windows 2000, XP ou 2003, será algo como `C:\Documents and Settings\Fulano\` ou, no Git Bash, `/c/Documents and Settings/Fulano`.

No Linux, será `/home/fulano` e no Mac OS X `/Users/Fulano`.

2.3 VERSIONANDO SEU CÓDIGO COM GIT

Criando um repositório

Abra um terminal e vá até o diretório `citacoes`.

```
$ cd ~/citacoes
```

Para transformar o diretório atual em um repositório do Git, basta executar o comando `git init`:

```
$ git init
```

Deverá aparecer uma mensagem semelhante à seguinte:

```
Initialized empty Git repository in /home/fulano/citacoes/.git/
```

Pronto, o projeto já é um repositório Git vazio.

Observe que foi criada uma pasta oculta com o nome `.git`.

Rastreando o arquivo

Mas e o arquivo `filmes.txt`? Será que já está versionado?

Podemos ver a situação dos arquivos no repositório Git com o comando:

```
git status
```

A saída deverá ser algo como:

```
# On branch master
#
# Initial commit
#
# Untracked files:
#   (use "git add <file>..." to include in what will be
#   committed)
#
#    filmes.txt
nothing added to commit but untracked files present (use
"git add" to track)
```

Observe a mensagem anterior: ela indica que o arquivo `filmes.txt` ainda não foi rastreado pelo Git.

Para que o arquivo seja **rastreado**, devemos executar o seguinte comando:

```
git add filmes.txt
```

Agora, se executarmos `git status` novamente, teremos a seguinte saída:

```
# On branch master
#
# Initial commit
#
# Changes to be committed:
#   (use "git rm --cached <file>..." to unstage)
#
#    new file:   filmes.txt
#
```

Gravando o arquivo no repositório

O resultado anterior mostra que o conteúdo do arquivo `filmes.txt` já está sendo rastreado pelo Git, mas ainda não foi gravado (ou *comitado*, em uma linguagem mais técnica) no repositório.

Para gravarmos as mudanças no repositório, devemos executar o comando:

```
git commit -m "Arquivo inicial de citacoes"
```

Observe que foi invocado o comando `git commit`, com a opção `-m` para informar a mensagem do commit.

Deve ter aparecido algo como a seguinte mensagem:

```
[master (root-commit) 8666888] Arquivo inicial de citacoes
1 file changed, 2 insertions(+)
create mode 100111 filmes.txt
```

Se executarmos `git status` novamente, teremos:

```
# On branch master
nothing to commit, working directory clean
```

Alterando o arquivo

Insira mais uma linha no arquivo `filmes.txt`, com o conteúdo:

"Hasta la vista, baby." (Exterminador do Futuro 2)

Depois disso, se executarmos `git status` novamente, podemos observar que há uma nova mudança para ser rastreada:

```
# On branch master
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in
working directory)
#
#    modified:   filmes.txt
#
no changes added to commit (use "git add" and/or
"git commit -a")
```

Rastreando e gravando as alterações no repositório

Para rastreamos a modificação, devemos executar o comando `git add` novamente:

```
git add filmes.txt
```

Com a modificação rastreada, podemos gravá-la no repositório, com o comando `git commit`:

```
git commit -m "Inserindo nova citacao"
```

Devemos ter uma resposta parecida com:

```
[master 7878787] Inserindo nova citacao
1 file changed, 1 insertion(+)
```

Verificando alterações realizadas

Para verificar o histórico das alterações gravadas no repositório, podemos executar o comando `git log`:

```
$ git log
```

A saída será parecida com:

```
commit 787878700000000000000000000000000000  
Author: Fulano da Silva <fulanodasilva.git@gmail.com>  
Date: Fri Apr 11 21:21:31 2014 -0300
```

Inserindo nova citacao

```
commit 86668880000000000000000000000000  
Author: Fulano da Silva <fulanodasilva.git@gmail.com>  
Date: Fri Apr 11 21:21:31 2014 -0300
```

Arquivo inicial de citacoes

Pronto! Temos um repositório criado com as alterações no arquivo `filmes.txt` devidamente gravadas. Mas e agora?

2.4 COMPARTILHANDO SEU CÓDIGO ATRAVÉS DO GitHub

Para que o mundo possa descobrir nosso incrível projeto, temos que compartilhá-lo na internet. Para isso, utilizaremos uma aplicação web chamada GitHub.

Criando uma conta no GitHub

O primeiro passo é criar uma conta no GitHub. Para projetos de código aberto, não há custo nenhum! Com um navegador, acesse: <https://github.com/>

Preencha seu nome, e-mail e escolha uma senha.

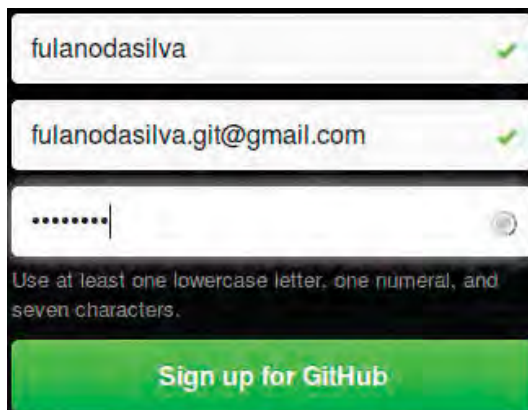


A screenshot of the GitHub sign-up form. It features three input fields: the first contains 'fulanodasilva' with a green checkmark; the second contains 'fulanodasilva.git@gmail.com' with a green checkmark; the third contains a masked password '.....' with a green checkmark. Below the password field is a text requirement: 'Use at least one lowercase letter, one numeral, and seven characters.' At the bottom is a large green button labeled 'Sign up for GitHub'.

Figura 2.1: Criando conta no GitHub

Então, selecione o plano apropriado e finalize o cadastro, clicando em *Finish Signup*.

**Completed**
Set up a personal account

**Step 2:**
Choose your plan

Choose your personal plan

Plan	Cost	Private repos	
Large	\$50/month	50	<button>Choose</button>
Medium	\$22/month	20	<button>Choose</button>
Small	\$12/month	10	<button>Choose</button>
Micro	\$7/month	5	<button>Choose</button>
Free	\$0/month	0	<button>Chosen</button>

Don't worry, you can cancel or upgrade at any time.

☐ **Help me set up an organization next**
Organizations are separate from personal accounts and are best suited for businesses who need to manage permissions for many employees.
[Learn more about organizations.](#)

Finish sign up

Figura 2.2: Selecionando plano no GitHub

Se necessário, verifique o e-mail.

Criando um repositório no GitHub

Agora podemos criar um repositório remoto, que ficará disponível para todos da internet. Para isso, clique no botão *New Repository* após acessar: <https://github.com/>



Figura 2.3: Novo repositório no GitHub

No *Repository name*, devemos preencher o nome do repositório remoto. No nosso caso, vamos preencher com “citacoes”. Deixe o repositório como *Public*, para que qualquer pessoa consiga ver o seu código. As demais opções podem ficar com os valores padrão. Finalmente, devemos clicar em *Create repository*.

Figura 2.4: Criando repositório no GitHub

Pronto, já foi criado um repositório vazio lá no GitHub.

Apontando seu projeto para o GitHub

Devemos agora apontar o repositório da nossa máquina para o repositório do GitHub.

Em um terminal, certifique-se de estar no diretório `citacoes`, que tem o repositório local:

```
$ cd ~/citacoes
```

Então, execute o comando `git remote`, conforme o que segue:

```
$ git remote add origin https://github.com/fulanodasilva/citacoes.git
```

Não deixe de **alterar** `fulanodasilva` para o **seu usuário do GitHub**. Não deve aparecer nenhuma mensagem.

Com o comando anterior, apontamos o nome `origin` para o repositório lá do GitHub.

Enviando as alterações para o GitHub

Com o repositório remoto configurado, podemos enviar nossas mudanças para o GitHub e, por consequência, para todo o mundo.

Para isso, basta executar o comando `git push`, da seguinte forma:

```
$ git push origin master
```

Com o comando anterior, enviamos as alterações para o repositório remoto configurado com o nome `origin`.

Forneça seu usuário e senha do GitHub quando solicitado. Deverá aparecer algo semelhante à seguinte saída:

```
Username for 'https://github.com': fulanodasilva
Password for 'https://fulanodasilva@github.com':
Counting objects: 6, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (4/4), done.
Writing objects: 100% (6/6), 609 bytes | 0 bytes/s, done.
Total 6 (delta 1), reused 0 (delta 0)
To https://github.com/fulanodasilva/citacoes.git
 * [new branch]      master -> master
```

Vá até a página do seu projeto no GitHub: <https://github.com/fulanodasilva/citacoes>

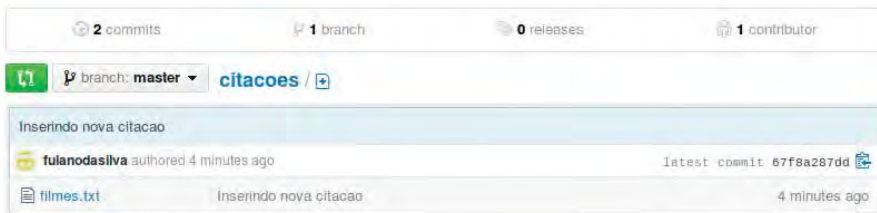


Figura 2.5: Página do projeto no GitHub

Observe que o arquivo que você enviou já está disponível para qualquer pessoa da internet. Avise seu primo, sua vizinha, todo mundo!

É possível ver todas as alterações no projeto até agora (no caso, foram duas), através do endereço: <https://github.com/fulanodasilva/citacoes/commits/master>

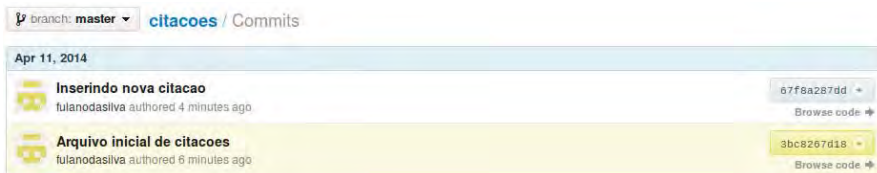


Figura 2.6: Listando alterações gravadas no GitHub

Se clicarmos na última alteração, por exemplo, é possível ver as mudanças que foram feitas. Fascinante, não?



Figura 2.7: Detalhando uma alteração gravada no GitHub

Obtendo projeto do GitHub

Com o projeto no GitHub, qualquer um pode acessar o código e ver o histórico, mesmo sem uma conta. Se a pessoa tiver cadastrada no GitHub, será possível baixar o código.

Vamos simular isso, baixando o código em outro diretório do seu computador.

Na sua pasta pessoal, crie um diretório chamado `projetos_git`. Então, o acesse pelo terminal:

```
$ cd ~/projetos_git
```

Para obter o código do projeto lá do GitHub, execute o comando `git clone` conforme o seguinte:

```
git clone https://github.com/fulanodasilva/citacoes.git
```

Não esqueça de **alterar** `fulanodasilva` para o **seu usuário do GitHub**.

Deverá aparecer algo como:

```
Cloning into 'citacoes'...
remote: Counting objects: 6, done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 6 (delta 1), reused 6 (delta 1)
Unpacking objects: 100% (6/6), done.
Checking connectivity... done
```

Observe no diretório `projetos_git` que foi criado um subdiretório chamado `citacoes`. Dentro desse subdiretório há o arquivo `filmes.txt`, com exatamente o mesmo conteúdo lá do GitHub.

Há também um diretório oculto chamado `.git`, revelando que temos uma cópia do repositório original.

Vá até o subdiretório `citacoes`, executando o comando:

```
$ cd citaco
```

Podemos executar o comando `git log` nesse novo repositório:

```
$ git log
```

Teremos a mesma saída de antes:

```
commit 7878787000000000000000000000000000  
Author: Fulano da Silva <fulanodasilva.git@gmail.com>  
Date: Fri Apr 11 21:21:31 2014 -0300
```

Inserindo nova citacao

```
commit 8666880000000000000000000000000000000000000000000000000000000000
Author: Fulano da Silva <fulanodasilva.git@gmail.com>
Date:   Fri Apr 11 21:21:31 2014 -0300
```

Arquivo inicial de citacoes

Se você tiver outro computador, faça o `clone` nele. Não se esqueça de instalar o Git, seguindo os passos do início deste capítulo.

GIT E GITHUB SÃO A MESMA COISA?

Não. Git é o sistema de controle de versões, com o qual interagimos na linha de comando. Já o GitHub é uma rede social para programadores que disponibiliza repositórios Git acessíveis remotamente. O GitHub é muito utilizado para projetos open source que possuem vários colaboradores do mundo todo. Mais adiante aprofundaremos em ambos.