

Depois de ter criado a conta na plataforma e instalado o Git na sua máquina, você já pode hospedar o seu primeiro projeto no GitHub.com. Para isso:

1. Abra o terminal de comando do seu sistema operacional;
2. Dentro da pasta do projeto, digite: `git init`
O comando acima irá criar toda a estrutura básica do repositório;
3. Para adicionar todos os arquivos alterados à fila de atualizações do repositório, execute o comando: `git add .`
4. Antes de sincronizar as alterações, configure seu usuário do GitHub com os comandos:
`git config --global user.name "seu nome"`
`git config --global "email no GitHub"`
5. Confirme as alterações com o comando: `git commit -m "mensagem"`
no qual “mensagem” geralmente é um resumo das alterações.
6. Adicione o *remote*, ou seja, o link para o servidor do seu projeto no GitHub:
`git remote add origin usuário no GitHub>/<nome do repositório>.git`
7. Por fim, envie as alterações com o comando:
`git push remote origin`
8. Se tudo deu certo, será exibido uma mensagem confirmando o envio. Você também pode checar na página do repositório do projeto no GitHub.com.

Existem vários outros recursos importantes para conhecer e utilizar no GitHub, como *branches*, que são versões paralelas do seu código que facilitam a criação de novas funcionalidades ou correções de bugs sem prejudicar o código-fonte original, e *pull requests*, que são “chamados” à comunidade para correção ou implementação de novos recursos em projetos.

A versatilidade da plataforma também permite que ela seja utilizada de outras formas, como hospedagem de blogs e portfólios pessoais com o [GitHub Pages](#), fóruns de discussão, páginas de divulgação de vagas, publicação de wikis e até de livros completos. Se você já é um profissional ou tem interesse pela área de tecnologia, saiba mais sobre como usar o GitHub lendo a [documentação oficial](#).

Por que precisamos do GitHub?

Entenda que o mundo da tecnologia de informação está cada vez mais concorrido, novas tecnologias vêm surgindo aos montes e em uma velocidade impressionante.

Pessoas, para desenvolverem aplicativos e programas capazes de lidar e controlar essas tecnologias, estão sendo cada vez mais procuradas.

Com toda esta concorrência e diversidade, como fazer para se destacar neste meio?

Em algumas profissões, é comum os profissionais terem um portfólio para divulgação de seu trabalho.

Ilustradores podem expor seus desenhos no DeviantArt, músicos e cantores divulgam suas canções no SoundCloud ou YouTube.

Mas e os programadores? Onde eles poderiam exibir seu programas e códigos? É aí onde entra o GitHub!

Porém, não tem como entendermos sobre GitHub sem falarmos de Git, por isso, para entender de onde surgiu o Git, de GitHub, vamos falar brevemente sobre o assunto.

O que é Git

O Git é um repositório de controle de versão de arquivos, utilizado para controlar versões de códigos-fonte e desenvolvimento colaborativo.

Foi criado em 2005 pelo Linus Torvalds, criador do kernel do Linux, para hospedar o código-fonte do Linux e permitir a colaboração em seu desenvolvimento, já que a antiga empresa que hospedava seu código, a BitKeeper, queria passar a cobrar pelos seus serviços.

Para você entender melhor sobre a importância do Git, na segunda edição da GeekStream, Felipe Fialho confidenciou que dominar os [comandos do Git](#) é uma das coisas mais importantes na carreira de desenvolvedor.

Como instalar o Git

O git pode ser instalado em plataformas Linux/Debian por meio do comando:

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

Em outras plataformas como Windows ou Mac, você pode baixar por meio destes links:

- [Download Git para Windows](#)
- [Download Git para Mac.](#)

Como transformar o [VSCode como editor padrão do Git](#)

A importância do GitHub

Já o GitHub, lançado em 2008, é uma versão para web do Git.

Os projetos no GitHub são em sua grande maioria open source, mas é possível tornar seu código código privado, disponível somente para algumas pessoas ou criando times.

O GitHub já se tornou um dos, senão o maior armazém de códigos na nuvem.

Git, SVN e CVS — [comparação dos principais VCS](#)

Para que serve o Github?

Nunca se esqueça: **Git e GitHub são coisas diferentes.**

Você pode ter o Git para controle de versão somente em seu hd.

Já o GitHub serve caso você queira compartilhar seu código ou colocá-lo na nuvem.

O GitHub possui todas as funcionalidades do Git, e conta com várias outras funcionalidades extras, listadas a seguir:

- Documentação;
- Rastreamento de problemas;
- Wikis;
- *Pull requests* com comentário e revisão de código;
- Histórico de *commits*;
- Notificações por email e emojis;
- Arquivos do Photoshop podem ser vistos e comparados com versões anteriores;
- GitHub Pages: pequenos websites podem ser feitos.

Esta última funcionalidade que iremos explorar para a construção do seu portfólio, mais para a frente.

Como usar o GitHub

Vamos ao nosso GitHub tutorial sobre **como utilizar o GitHub** com os principais passos na ferramenta.

Antes, você precisa criar sua conta pessoal ou uma conta de equipe na [página oficial do GitHub](#).

É totalmente gratuito para repositórios pessoais e para times, como alguns planos pagos mais avançados para times e empresas.

Depois de criar sua conta, utilize o Hello World Guide da plataforma para saber como criar o seu primeiro repositório, fazer um branch ou abrir um pull request.

O que é um repositório

Um repositório é um espaço em que todos os arquivos de um projeto específico são armazenados.

Cada projeto tem seu próprio repositório e você pode acessá-lo com um URL exclusivo.

O que é um fork

“Forking” é quando você cria um novo projeto baseado em outro projeto que já existe, como a bifurcação de um caminho.

Esse é um recurso que incentiva amplamente o desenvolvimento de programas e projetos entre a comunidade ou entre equipes, de forma geral.

Digamos que, ao você encontrar um projeto no GitHub com o qual gostaria de contribuir, poderá dividir o repositório, fazer as alterações desejadas e liberar o projeto revisado como um novo repositório.

Se o repositório original que você criou para criar seu novo projeto for atualizado, você poderá facilmente adicionar essas atualizações ao seu **fork** atual.

Pull Request: o que é?

Imagine que você bifurcou (fez um **fork**) um repositório, fez uma boa revisão no projeto e deseja que ele seja reconhecido pelos desenvolvedores originais.

Você pode fazer isso criando uma solicitação de recebimento, o famoso **Pull Request**.

Os autores do repositório original podem ver seu trabalho e, em seguida, optar por aceitá-lo ou não no projeto oficial.

Sempre que você faz um **pull request**, o GitHub fornece um meio perfeito para você e o mantenedor do projeto principal se comunicarem entre si.

GitHub também é uma rede social

O GitHub também permite que os projetos cresçam de forma exponencial com o uso social da ferramenta.

Cada usuário no GitHub tem seu próprio perfil como em qualquer rede social, ideal para mostrar seus trabalhos e contribuições em outros projetos.

As revisões do projeto podem ser discutidas publicamente, favorecendo o debate público e compartilhamento de conhecimento.

O GitHub favorece o *networking* entre desenvolvedores e pode alavancar a sua carreira. Nada chama mais a atenção de um recrutador que um GitHub popular e ativo com a comunidade.

GitHub Pages: montando seu portfólio de códigos

Além de ser um repositório de códigos e rede social, o GitHub ainda oferece a possibilidade de hospedagem de mini-sites estáticos, quantos você quiser!

Isso é uma ótima maneira para os desenvolvedores criarem e divulgarem seus códigos diretamente.

Há duas maneiras de construir suas páginas no GitHub: com o GitHub Theme Chooser, ou escrevendo o código do zero.

Antes de começar, é preciso preparar seu GitHub para hospedar seu site. Aqui vai um passo a passo resumido para isso:

1. Crie um novo repositório chamado ***username.github.io***, trocando o *username* pelo seu nome de usuário no GitHub. Este será o repositório do seu site e o GitHub automaticamente o reconhecerá como a sua home page
2. Crie o arquivo ***index.html*** com o que você quiser que apareça na sua página. Um “Hello world” para teste é suficiente
3. Faça um add, commit e um push para visualizar as mudanças

Você também pode baixar templates para seu portfólio.

Há também a opção do [GitHub Résumé](#), que gera um currículo a partir das informações do seu GitHub.

Um resumo com as linguagens que utiliza, projetos que participa e seus repositórios mais populares são organizados automaticamente.

O que colocar no GitHub?

Depois da sua página pronta, chega a hora de colocar seus projetos para o acesso de quem estiver interessado. E aí vem a dúvida: o que colocar no portfólio?

Resumindo em uma palavra: **TUDO!**

Aquele seu projeto pessoal ou da faculdade que você acha que não serve para nada, pode conter algum conceito ou técnica de programação que alguma empresa procura.

Fez um curso complementar de programação onde teve que desenvolver algum projeto? Coloque-o no GitHub. Tem um projeto pessoal sendo desenvolvido, coloque-o também.

Além do seu código ficar visível para o mundo, pessoas ainda podem ajudá-lo a desenvolvê-lo.

Se você já é programador e desenvolve para uma empresa, muito provavelmente não poderá divulgar seu código. Neste caso crie projetos simples com a tecnologia e metodologia que está usando e coloque no GitHub, descrevendo tudo o que fez.

Como enviar um projeto para o GitHub

Confira este vídeo tutorial sobre **como enviar um projeto para o Github** sem enrolação:

O README do GitHub

Um elemento importante para os seus projetos mas que nem sempre recebe a devida atenção é o arquivo README.

O README é a primeira impressão do seu trabalho. Sempre tenha um na raiz do projeto.

Um bom README é aquele que contém uma boa descrição do objetivo, onde e como usar, lista de funcionalidades, dependências para funcionar. Exemplos de uso será um plus.

Lembre-se que o README é para ser um resumo, não uma documentação do projeto. Para a documentação, você pode utilizar a Wiki do próprio GitHub.

Uso de licenças

Não esqueça de definir uma licença para o seu projeto. Você pode perder oportunidades caso não tenha uma licença definida. O site [Choose a License](#) pode ajudá-lo a definir qual licença adotar.

Para aplicar uma licença ao seu projeto basta criar um arquivo com o nome LICENSE e colar exatamente o conteúdo da licença nele. Altere somente se a licença tiver espaços para serem preenchidos com seu dados.

O GitHub automaticamente identificará a licença e adicionará um pequeno ícone dela no topo do seu projeto. Caso isso não ocorra, revise o arquivo.

Como o recrutador irá avaliar seu portfólio?

É difícil afirmar com certeza como o recrutador irá avaliar seu portfólio, pois cada empresa possui um processo de seleção diferente.

Mesmo assim, vamos falar aqui de algumas possibilidades e métodos de avaliação pelo GitHub.

A primeira delas é olhando diretamente a página do seu portfólio.

É importante produzir uma página bem elaborada mesmo que você não seja um desenvolvedor front-end, pois isso irá mostrar um cuidado com a entrega do seu trabalho. Se achar melhor, use um dos vários templates disponíveis.

Apesar de ser uma opção, ter uma *home page* no GitHub não é extremamente necessário para a divulgação do seu portfólio. Um link para o seu repositório já é o suficiente.

Bons desenvolvedores têm bons projetos

Não é comum recrutadores analisarem todo o GitHub de um candidato, mas algum projeto específico pode chamar atenção.

Você também pode mostrar um projeto específico que possa ser relevante para uma determinada vaga.

Repositórios para testes

Outra maneira de se usar o GitHub em um processo seletivo é por meio de testes.

O recrutador cria um repositório, pede que o candidato faça um *fork* deste repositório e depois dê um *commit* com alguma modificação, que será a resolução do teste.

Este tipo de teste serve tanto para avaliar a capacidade de programação do candidato quanto sua familiaridade com o Git. É o tipo de avaliação mais comum.

Depois de todas essas dicas, dificilmente você passará despercebido no mundo do desenvolvimento e ao olhos dos recrutadores.

Instalando git


O git é um programa que pode ser instalado [neste link](#) para Windows, [neste](#) para Mac, ou então através do comando `sudo apt-get install git` para plataformas Linux/Debian, como o Ubuntu. Se você usa uma VM na nuvem, como o [cloud9](#) ou [koding](#), o git já estará disponível em sua linha de comando.

Criando a conta no GitHub

O github não possui instalação, ele é um serviço, e caso você não tenha uma conta, chegou a hora de criá-la, [neste link](#). Após criar a conta, você verá um botão verde **+New Repository** na qual poderá criar um repositório de acordo com a tela a seguir.

Owner

Repository name

 danieltableless ▾


 /

site ✓


Great repository names are short and memorable. Need inspiration? How about **irksome-succotash**.

Description (optional)

Site de teste

☒  **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** ▾

 |

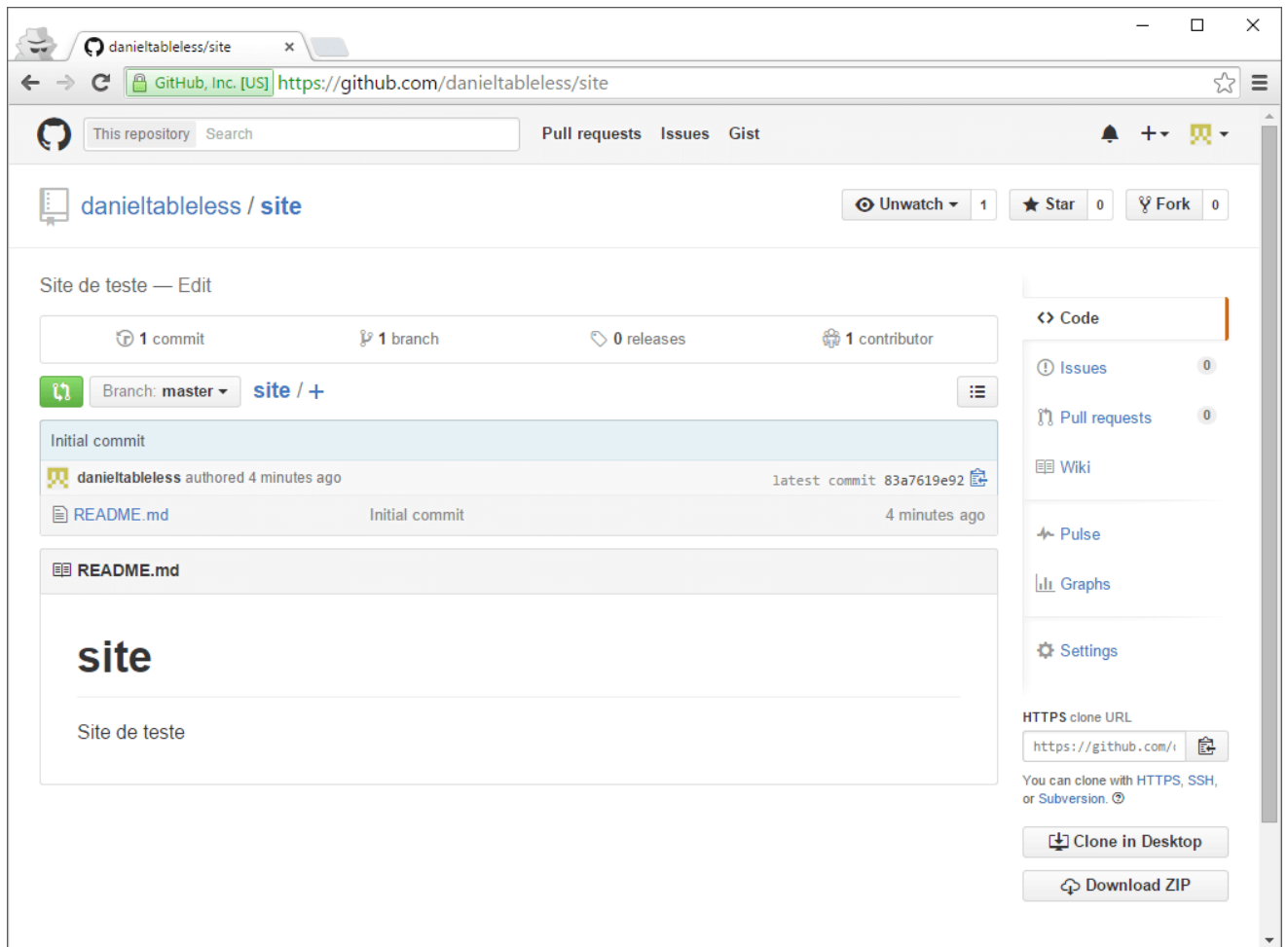
Add a license: **None** ▾



Create repository

Nesta imagem estamos criando um repositório cujo nome é `site`, de domínio público (podem ser criados reps privados pagando uma mensalidade), e com o arquivo `README.md` embutido, que contém uma descrição do seu projeto. Para que possamos começar a entender como o git funciona, é fundamental criar um rep como este para os nossos testes.

Após a criação do repositório, ele estará disponível no endereço <https://github.com/<username>/site>, onde `username` é o login que você usou para se cadastrar. Acessando esta url temos a seguinte resposta:



Temos muitas informações nesta tela, pois ela é a tela principal do seu projeto. Explicaremos algumas informações ao longo deste artigo, por enquanto repare apenas no botão **HTTPS Clone Url** na parte inferior à direita. Esta URL será necessária para que possamos “clonar” este projeto em nosso ambiente de estudo (sua máquina windows, mac, linux ou a vm). Clique no botão de copiar URL e perceba que a seguinte URL está na área de transferência: <https://github.com/<username>/site.git>

Configurando o git

Existem 2 pequenos passos para configurar o seu GIT para ter um acesso mais simplificado ao github. Aqui estaremos estabelecendo que, sempre que necessitar, você irá fornecer o seu login e senha ao GitHub. Existem meios para salvar a senha em local seguro, mas vamos pular

esta etapa. Para abrir um terminal GIT no Windows, basta criar uma pasta no seu sistema e, nela, clicar com o botão direito do mouse e escolher **Git Bash Here**. Em sistemas mac/linux você já está acostumado a usar o terminal/console, o git estará lá disponível. Neste artigo estaremos utilizando a máquina virtual cloud9, que você pode aprender a usá-la neste [artigo](#).

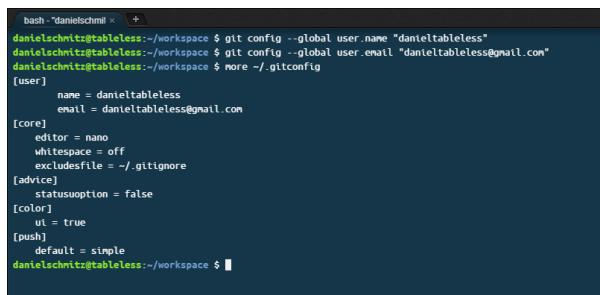
Então, com o seu terminal git aberto, vamos digitar:

```
$ git config --global user.name "YOUR NAME"
```

```
$ git config --global user.email "YOUR EMAIL ADDRESS"
```

Estas configurações ficam alocadas no arquivo `~/.gitconfig`, onde o `~` é o seu diretório home.

No Windows, ele fica em `c:\Usuarios\<username>\.gitconfig`. Veja a figura a seguir com a minha configuração no cloud9.



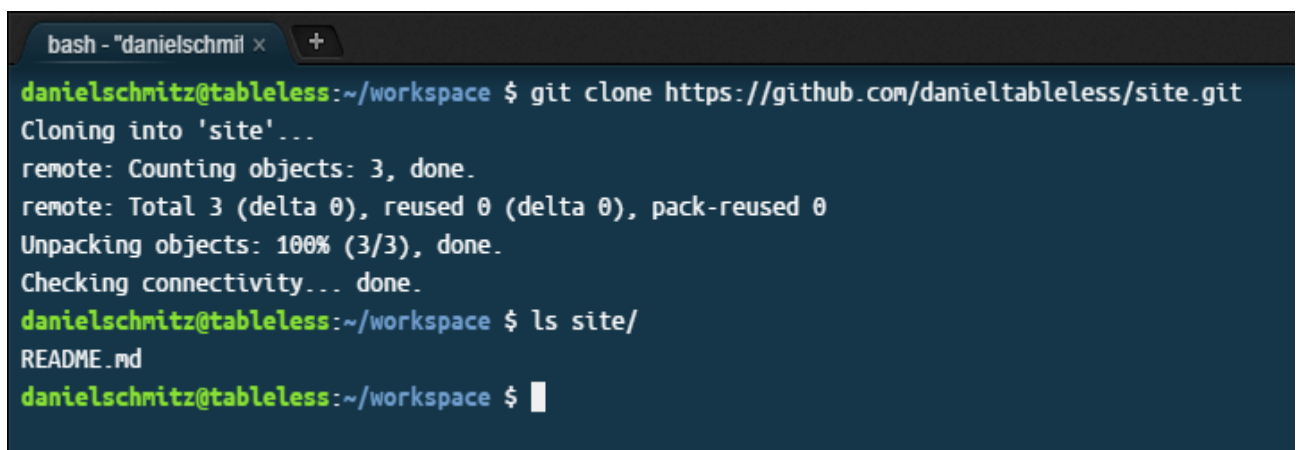
```
bash - "danielschmil" x
danielschmitz@tableless:~/workspace $ git config --global user.name "danieltableless"
danielschmitz@tableless:~/workspace $ git config --global user.email "danieltableless@gmail.com"
danielschmitz@tableless:~/workspace $ more ~/.gitconfig
[user]
  name = danieltableless
  email = danieltableless@gmail.com
[core]
  editor = nano
  whitespace = off
  excludesfile = ~/.gitignore
[advice]
  statusuoption = false
[color]
  ui = true
[push]
  default = simple
danielschmitz@tableless:~/workspace $
```

Vamos clonar!

Então o que temos até agora é o git configurado para utilizar o github e o projeto no github criado. Precisamos trazer este projeto para o nosso git, e este processo se chama **clonar**.

Então, quando você quiser começar um projeto utilizando git, você cria ele no github e clona na sua máquina. O comando para clonar o projeto é **git clone "url"**, veja:

```
git clone https://github.com/<username>/site.git
```



```
bash - "danielschmil" x
danielschmitz@tableless:~/workspace $ git clone https://github.com/danieltableless/site.git
Cloning into 'site'...
remote: Counting objects: 3, done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), done.
Checking connectivity... done.
danielschmitz@tableless:~/workspace $ ls site/
README.md
danielschmitz@tableless:~/workspace $
```

Perceba que, ao fazer o git clone, o projeto é baixado para a sua máquina, e uma pasta com o nome do projeto é criada.

Quer dizer que qualquer pessoa pode baixar o meu projeto? Sim, isso é natural, já que o seu repositório está público. Qualquer um pode clonar ele para si, mas eles não podem alterar os seus arquivos, isso não vai acontecer, exceto que você permita.

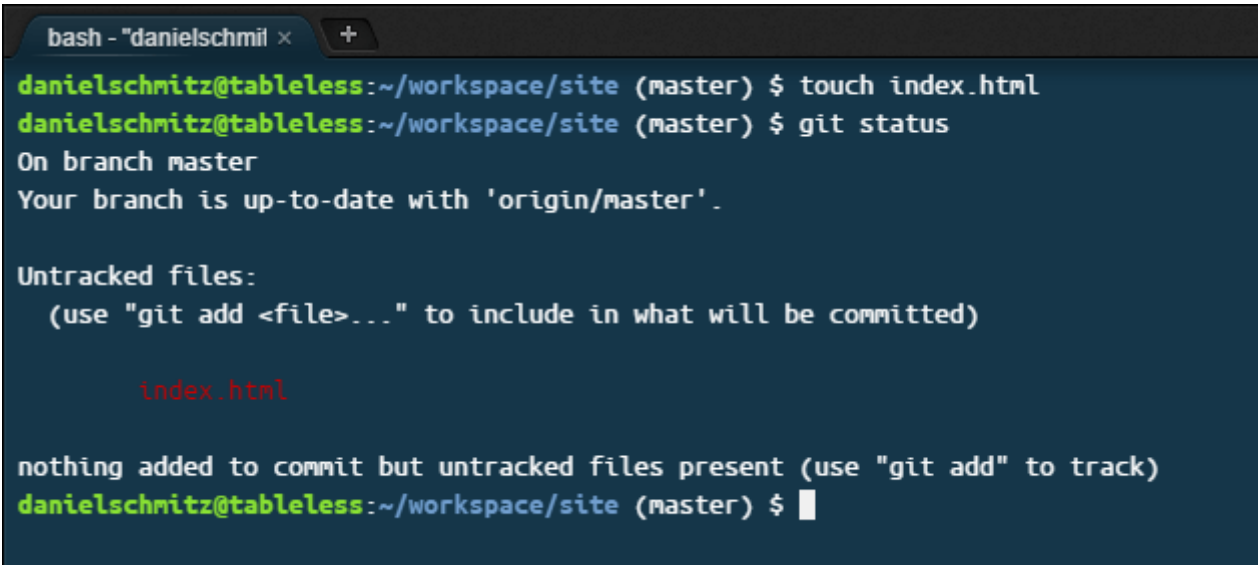
Comandos iniciais do git

Com o repositório na sua máquina, vamos aprender 4 comandos iniciais que farão parte da sua vida a partir de agora:

- `git add <arquivos...>` Este comando adiciona o(s) arquivo(s) em um lugar que chamamos de INDEX, que funciona como uma área do git no qual os arquivos possam ser enviados ao Github. É importante saber que ADD não está adicionando um arquivo novo ao repositório, mas sim dizendo que o arquivo (sendo novo ou não) está sendo preparado para entrar na próxima revisão do repositório.
- `git commit -m "comentário qualquer"` Este comando realiza o que chamamos de "commit", que significa pegar todos os arquivos que estão naquele lugar INDEX que o comando add adicionou e criar uma revisão com um número e um comentário, que será vista por todos.
- `git push` Push (empurrar) é usado para publicar todos os seus commits para o github. Neste momento, será pedido a sua senha.
- `git status` Exibe o status do seu repositório atual

Vamos praticar!

Chegou o momento de praticar um pouco o que vimos até agora, e com bastante calma para que você possa entender cada passo. Após clonar o seu projeto, crie o arquivo `index.html` na pasta site que é o seu repositório git. Após criar o arquivo, execute o comando `git status`. A resposta é semelhante a figura a seguir:

A terminal window with a dark background and light green text. The window title is 'bash - "danielschmitz" x'. The prompt is 'danielschmitz@tableless:~/workspace/site (master)'. The user has entered 'touch index.html' and 'git status'. The output shows the branch is up-to-date and lists 'index.html' as an untracked file.

```
bash - "danielschmitz" x
danielschmitz@tableless:~/workspace/site (master) $ touch index.html
danielschmitz@tableless:~/workspace/site (master) $ git status
On branch master
Your branch is up-to-date with 'origin/master'.

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

        index.html

nothing added to commit but untracked files present (use "git add" to track)
danielschmitz@tableless:~/workspace/site (master) $
```

Ou seja, o comando `git status` nos trouxe várias informações, que iremos ignorar a princípio, exceto pelo `Untracked files`, dizendo que existe um arquivo que não está sendo “mapeado” pelo git. Para preparar este arquivo para o seu versionamento, usamos o comando `git add`, veja:

```
bash - "danielschmil x +
danielschmitz@tableless:~/workspace/site (master) $ git add index.html
danielschmitz@tableless:~/workspace/site (master) $ git status
On branch master
Your branch is up-to-date with 'origin/master'.

Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        new file:   index.html

danielschmitz@tableless:~/workspace/site (master) $
```

Agora temos o nosso arquivo `index.html` no INDEX do repositório, ou se você quiser pensar: “preparado para um commit”. Para commitar este arquivo, usamos:


```
bash - "danielschmil x +
danielschmitz@tableless:~/workspace/site (master) $ git commit -m "Criação do arquivo index.html"
[master 7bd652b] Criação do arquivo index.html
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 index.html
danielschmitz@tableless:~/workspace/site (master) $ git status
On branch master
Your branch is ahead of 'origin/master' by 1 commit.
  (use "git push" to publish your local commits)

nothing to commit, working directory clean
danielschmitz@tableless:~/workspace/site (master) $
```

Após “commitar” o arquivo, ele já está presente no nosso repositório local, tanto que realizamos o comando `git status` novamente e ele retornou que não havia nada de novo no projeto. Perceba agora que, mesmo recarregando o projeto no github, nada muda. Ou seja, estas mudanças até agora foram locais, você pode realizar várias operações antes de publicá-las no github. Para publicar, usamos o comando `git push`:

```
bash - "danielschmil" x +
danielschmitz@tableless:~/workspace/site (master) $ git push
Username for 'https://github.com': danieltableless
Password for 'https://danieltableless@github.com':
Counting objects: 4, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 298 bytes | 0 bytes/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To https://github.com/danieltableless/site.git
83a7619..7bd652b master -> master
danielschmitz@tableless:~/workspace/site (master) $
```

Após realizar o git push podemos ver no site github as mudanças realizadas no projeto:


 **danieltableless / site** Unwatch 1

Site de teste — Edit

2 commits1 branch0 releases1 contributor

Branch: master site / +

Criação do arquivo index.html

 danieltableless authored 8 minutes ago

latest commit 7bd652bd8a

README.md

Initial commit

an hour ago

index.html

Criação do arquivo index.html

8 minutes ago

README.md

site

Site de teste

Desta forma, aprendemos os 4 comandos mais básicos do git, e com ele podemos começar a compreender como funciona o processo de versionamento de arquivos com git e github.

Errei a mensagem do commit, como arrumo?

Imagine que você tenha errado a mensagem que escreveu no commit ou simplesmente queira melhorar a descrição do seu trabalho. Você já comitou a mensagem mas ainda não fez o push das suas modificações para o servidor. Nesse caso você usa a flag `--amend`. Fica assim:

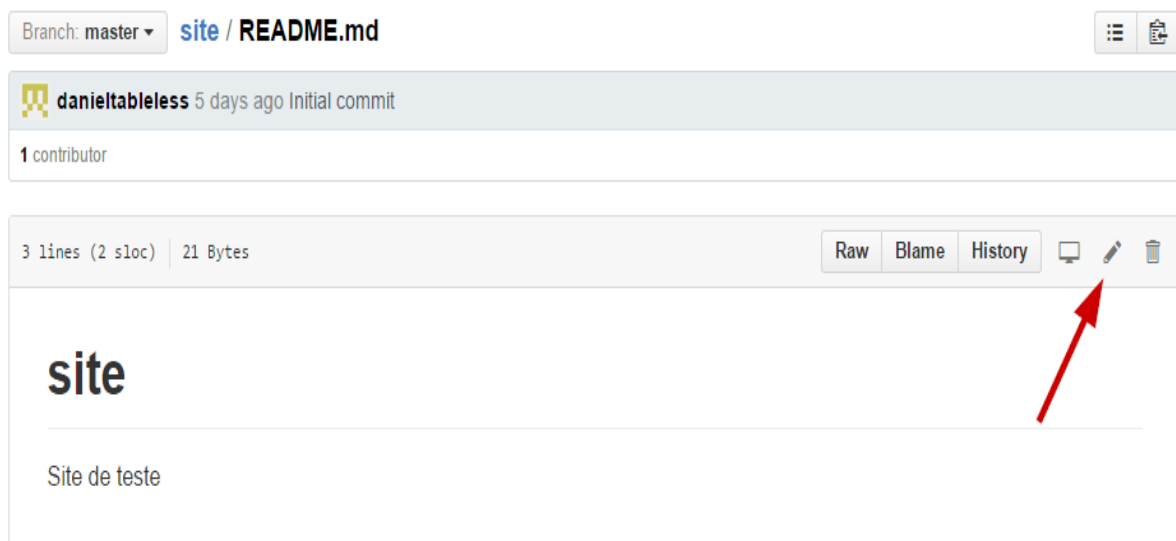
```
$ git commit --amend
```

O `git commit --amend` modifica a mensagem do commit mais recente, ou seja, o último commit feito por você no projeto. Além de você mudar a mensagem do commit, você consegue adicionar arquivos que você se esqueceu ou retirar arquivos comitados por engano. O git cria um commit totalmente novo e corrigido.

Cadê o git pull?

Ainda existe um comando importante neste processo, que é o `git pull`. Ele é usado para trazer todas as modificações que estão no github para o seu projeto local. Isso é vital quando existem projetos mantidos por mais de uma pessoa, ou se você possui duas máquinas e precisa manter a sincronia entre elas. Supondo que você possui uma máquina no trabalho e outra em casa. Ambas tem o repositório local ligado ao github. Quando você executar um `git push` em uma das máquinas, terá que realizar um `git pull` na outra.

Para exemplificar, vamos alterar o arquivo README.md diretamente no github. Isso é possível clicando no arquivo e depois clicando no ícone para edição, conforme a imagem a seguir.



Após clicar em edit, adicione algum texto, forneça uma mensagem de commit e clique no botão “Commit Changes”. Com isso, uma nova revisão no seu projeto é criada, mas como ela foi gerada no github, o seu projeto local está desatualizado. Para atualizar o seu projeto, use `git pull`, e perceba que o arquivo README.md é atualizado de acordo com a sua última revisão, semelhante a figura a seguir.


```
bash - "danielschmil" x +
danielschmitz@tableless:~/workspace/site (master) $ git pull
remote: Counting objects: 3, done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), done.
From https://github.com/danieltableless/site
  7bd652b..3af0946  master    -> origin/master
Updating 7bd652b..3af0946
Fast-forward
 README.md | 2 ++
 1 file changed, 2 insertions(+)
danielschmitz@tableless:~/workspace/site (master) $
```

Melhorando o conceito do comando git add

Possivelmente você imaginou que o comando `git add` é usado para novos arquivos, mas isso não é verdade. O comando `add` é usado para adicionar qualquer alteração de arquivo ao INDEX do git, que é uma área especial onde os arquivos estão sendo preparados para o commit. Quando usamos `add`, estamos dizendo que o arquivo estará adicionando ao próximo commit, quando este for realizado. Isso é necessário porque nem sempre queremos que todos os arquivos que alteramos sejam comitados.

Vamos a um exemplo simples, adicionando o seguinte código no arquivo `index.html`:

```
<!DOCTYPE html>

<html>

<head>

<meta charset="UTF-8">

<title> Meu Site </title>

</head>

<body>
```

```
</body>
```

```
</html>
```

Após salvar este modelo html, o comando `git status` irá apresentar:

```
modified: index.html
```

Para adicionar o arquivo e prepará-lo para o commit, usamos `git add index.html`. Desta forma, ele está pronto para usarmos o comando `git commit`, o que não faremos agora. Antes disso, altere novamente o arquivo e adicione algum texto entre as tags `body`, por exemplo:

```
<!DOCTYPE html>
```

```
<html>
```

```
<head>
```

```
<meta charset="UTF-8">
```

```
<title> Meu Site </title>
```

```
</head>
```

```
<body>
```

```
Esse é meu site
```

```
</body>
```

```
</html>
```

Após alterar o arquivo, temos a seguinte situação:

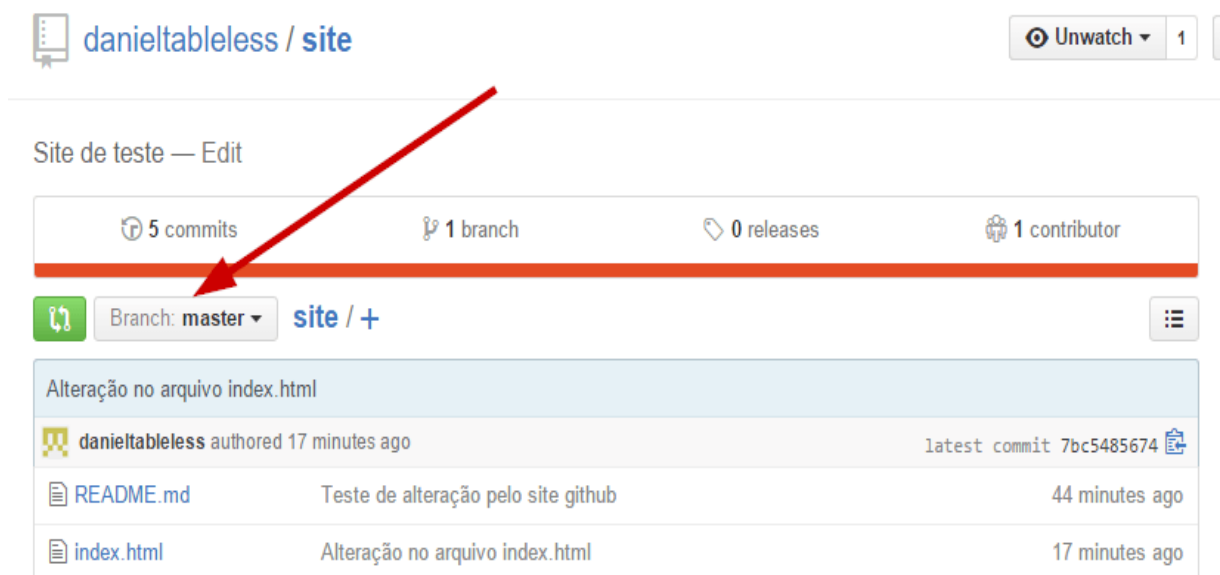
1. Adicionamos o conteúdo html no arquivo `index.html`
2. Realizamos ``git add index.html``
3. Alteramos `index.html` e adicionamos o texto entre as tags `body`

Neste momento, faça: `git commit -m "Alteração no arquivo index.html"`, e após isso, faça: `git push`. Analise agora no github se a sua alteração na tag body está visível. Ela não estará. Mas porque isso aconteceu? Quando usamos o comando `git add`, aquela alteração no body ainda não tinha sido escrita, então ela não estará pronta até que você faça novamente o comando `git add`. Em termos técnicos, a segunda alteração que fez ainda não está na INDEX do repositório. Como tarefa, faça novamente `git add index.html`, `git commit` e `git push`

Trabalhando com branches

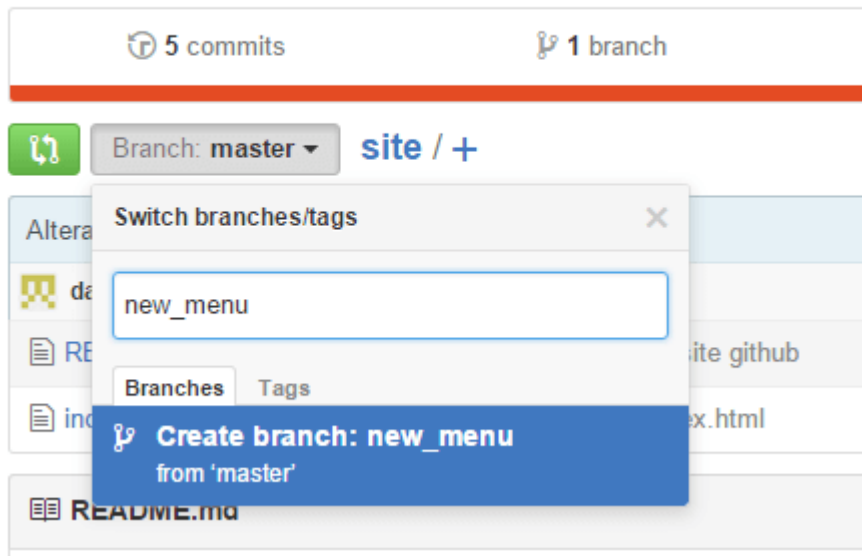
Branches e mergers sempre foram os pesadelos de qualquer gerenciador de versão (ok, do svn...). No git, o conceito de branch tornou-se algo muito simples e fácil de usar. Mas quando que temos que criar um branch? Imagine que o seu site está pronto, tudo funcionando perfeitamente, mas surge a necessidade de alterar algumas partes dele como forma de melhorá-lo. Além disso, você precisa manter estas alterações tanto no computador de casa quanto do trabalho. Com isso temos um problema, se você começa a alterar os arquivos em casa, para na metade da implementação, e precisa terminar no trabalho, como você iria comitar tudo pela metade e deixar o site incompleto?

Para isso existe o conceito de branch, que é justamente ramificar o seu projeto em 2, como se cada um deles fosse um repositório, e depois juntá-lo novamente. Voltando ao github, perceba o detalhe da imagem a seguir.



Sem saber, você já está em um branch, que chamamos de master. Perceba também que, sempre que usávamos `git status`, o nome do branch é exibido, e sempre que comitávamos ou fazíamos o push, o mesmo aparecia. Ou seja, até este momento fizemos todas as alterações no master. Você pode criar um branch no github ou em linha de comando. Inicialmente, vamos pelo github, criando o branch "new_menu".

Site de teste — Edit



Criamos o branch `new_menu`, e para que possamos trabalhar nele, usamos o comando `git checkout new_menu`. No primeiro momento que você cria este branch no github, é necessário realizar o comando `git pull` no seu projeto para que ele possa saber que este branch foi criado. Após realizar `git pull`, pode-se alterar para o novo branch, conforme a imagem a seguir.

```
bash - "danielschmitz" x +
danielschmitz@tableless:~/workspace/site (master) $ git pull
From https://github.com/danieltaleless/site
* [new branch] new_menu -> origin/new_menu
Already up-to-date.
danielschmitz@tableless:~/workspace/site (master) $ git checkout new_menu
Branch new_menu set up to track remote branch new_menu from origin.
Switched to a new branch 'new_menu'
danielschmitz@tableless:~/workspace/site (new_menu) $ git status
On branch new_menu
Your branch is up-to-date with 'origin/new_menu'.

nothing to commit, working directory clean
danielschmitz@tableless:~/workspace/site (new_menu) $
```

Neste momento, estamos no branch `new_menu`, e tudo que fizermos agora será pertencente a ele. Caso haja necessidade de voltar ao branch master, basta realizar o comando `git checkout master`.

Atenção, o comando `checkout` do git não é o mesmo do checkout do svn, caso você o conheça. Ambos tem sentidos totalmente diferentes.

Então, estando no branch `new_menu`, vamos adicionar um simples menu na página:

```
<!DOCTYPE html>
```

```
<html>
```

```
<head>
```

```
<meta charset="UTF-8">
```

```
<title> Meu Site </title>
```

```
</head>
```

```
<body>
```

Meu Site

```
<ul>
```

```
<li><a href="index.html">Home</a></li>
```

```
<li><a href="sobre.html">Sobre</a></li>
```

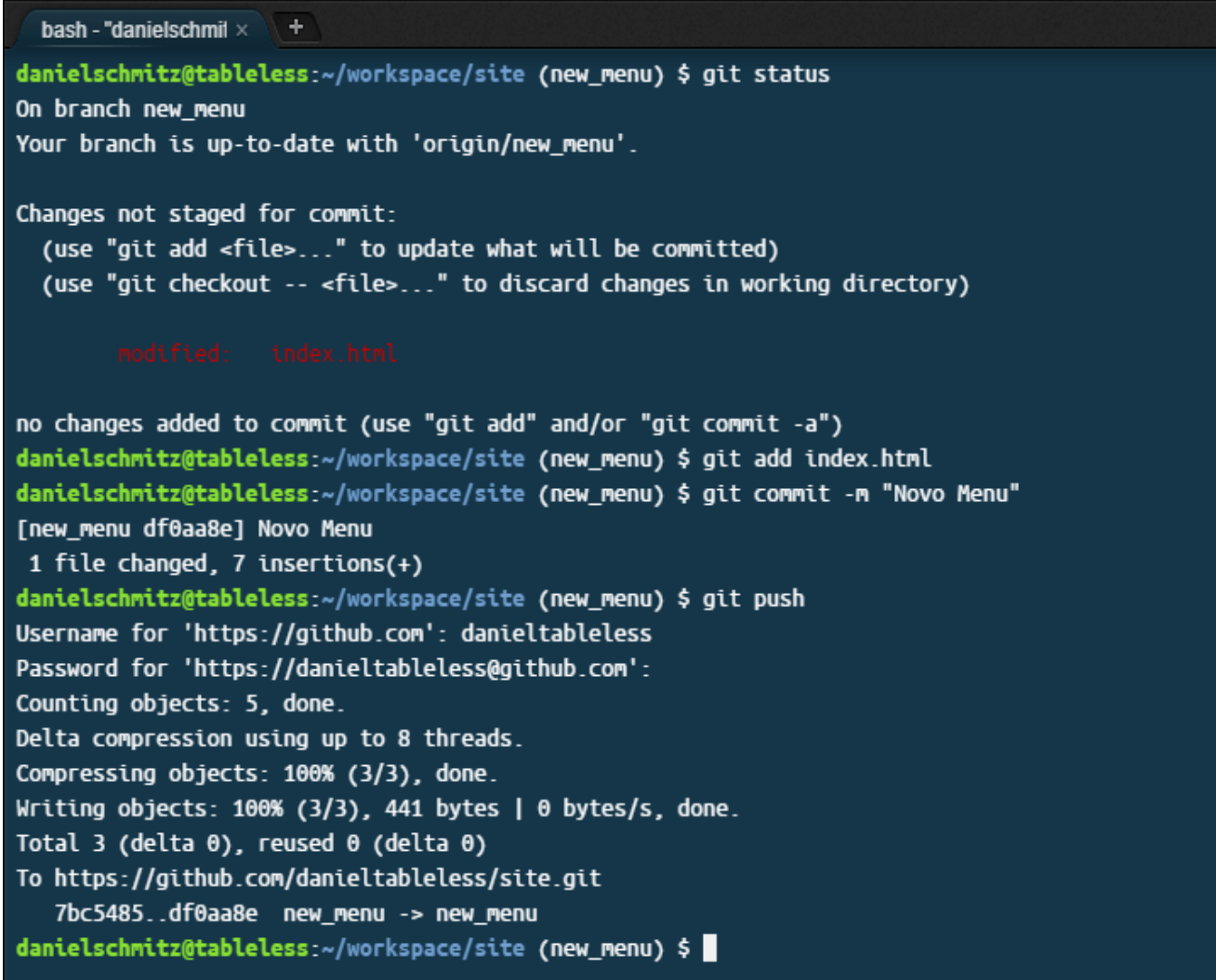
```
<li><a href="contato.html">Contato</a></li>
```

```
</ul>
```

</body>

</html>

Após criar o menu, certifique-se de estar no branch `new_menu` e faça o commit, conforme a figura a seguir.



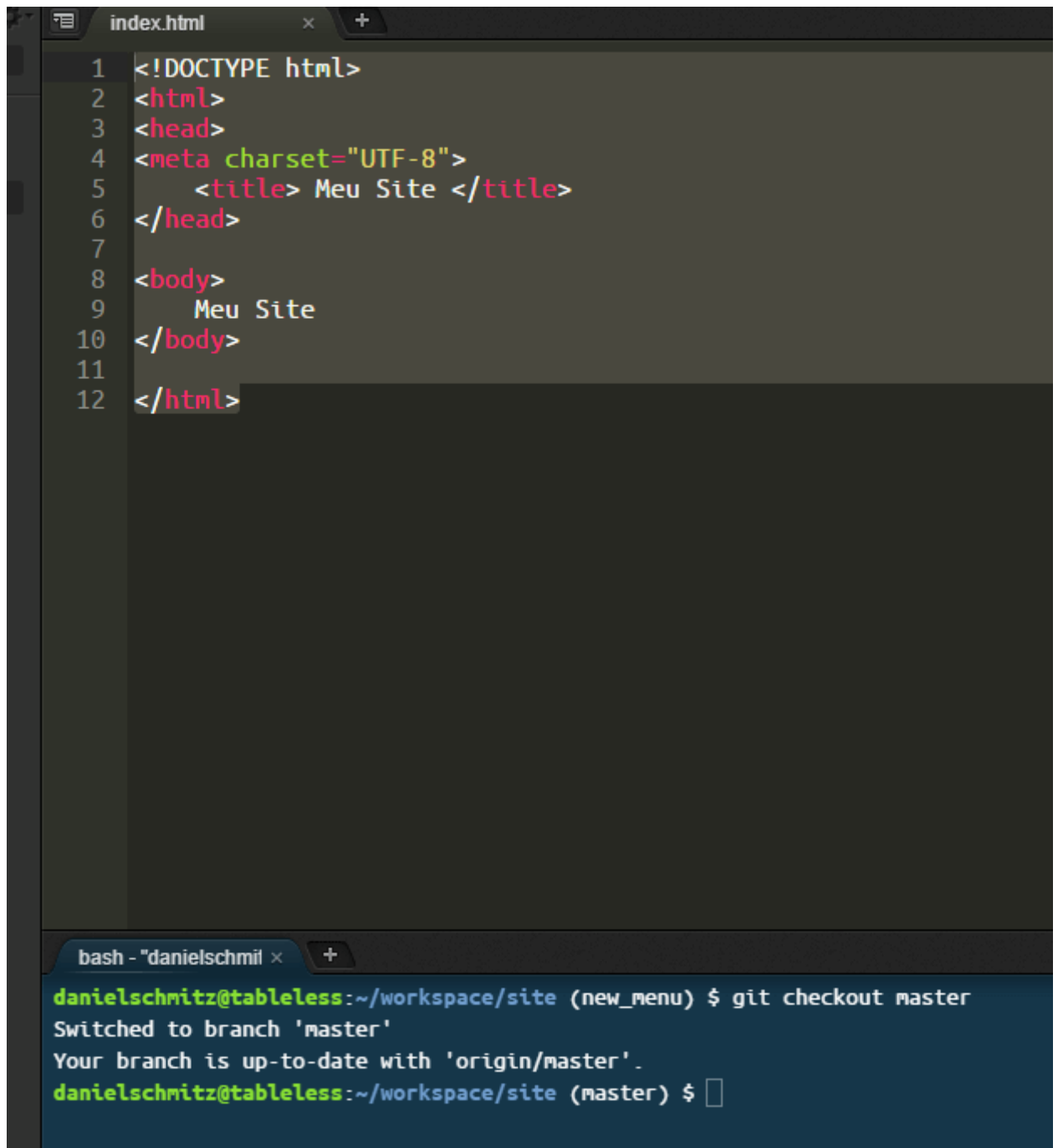
```
bash - "danielschmil x" +
danielschmitz@tableless:~/workspace/site (new_menu) $ git status
On branch new_menu
Your branch is up-to-date with 'origin/new_menu'.

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:   index.html

no changes added to commit (use "git add" and/or "git commit -a")
danielschmitz@tableless:~/workspace/site (new_menu) $ git add index.html
danielschmitz@tableless:~/workspace/site (new_menu) $ git commit -m "Novo Menu"
[new_menu df0aa8e] Novo Menu
 1 file changed, 7 insertions(+)
danielschmitz@tableless:~/workspace/site (new_menu) $ git push
Username for 'https://github.com': danieltableless
Password for 'https://danieltableless@github.com':
Counting objects: 5, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 441 bytes | 0 bytes/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To https://github.com/danieltableless/site.git
   7bc5485..df0aa8e  new_menu -> new_menu
danielschmitz@tableless:~/workspace/site (new_menu) $
```

Agora temos algumas modificações no branch `new_menu`, e podemos trabalhar nesse branch por quanto tempo for necessário, já que o `master` está intacto. Aqui temos uma funcionalidade interessante, que se destaca em relação as outras ferramentas de versionamento. Suponha que, no meio do seu desenvolvimento do menu, surge a necessidade de resolver um bug crítico no `master`, algo como “está faltando o `h1` no título do seu site”.... Ou seja, estamos no branch `new_menu` e precisamos alterar o `master`. Para isso, use o comando `git checkout master`. Ao fazer isso, retornamos ao `master` e aquele menu que criamos não está mais presente, conforme a figura a seguir.



The image shows a code editor window with a file named 'index.html' and a terminal window below it. The code editor contains the following HTML code:

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4 <meta charset="UTF-8">
5   <title> Meu Site </title>
6 </head>
7
8 <body>
9   Meu Site
10 </body>
11
12 </html>
```

The terminal window shows the following commands and output:

```
bash - "danielschmil" × +
danielschmitz@tableless:~/workspace/site (new_menu) $ git checkout master
Switched to branch 'master'
Your branch is up-to-date with 'origin/master'.
danielschmitz@tableless:~/workspace/site (master) $
```

É claro que não perdemos o menu, ele está apenas no branch `new_menu`. Quando retornarmos a ele, voltará. Agora altere o título do site, incluindo o `h1`, veja:

```
<!DOCTYPE html>
```

```
<html>
```

```
<head>
```

```
<meta charset="UTF-8">
```

```
<title> Meu Site </title>
```

```
</head>
```

```
<body>
```

```
<h1>Meu Site</h1>
```

```
</body>
```

```
</html>
```

Após alterar, faça commit e o push! Veja:


```
bash - "danielschmitz" x +
danielschmitz@tableless:~/workspace/site (master) $ git status
On branch master
Your branch is up-to-date with 'origin/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:   index.html

no changes added to commit (use "git add" and/or "git commit -a")
danielschmitz@tableless:~/workspace/site (master) $ git commit -am "Inclusão do título"
[master 9a7fbc9] Inclusão do título
 1 file changed, 1 insertion(+), 1 deletion(-)
danielschmitz@tableless:~/workspace/site (master) $ git push
Username for 'https://github.com': danieltableless
Password for 'https://danieltableless@github.com':
Counting objects: 7, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 346 bytes | 0 bytes/s, done.
Total 3 (delta 1), reused 0 (delta 0)
To https://github.com/danieltableless/site.git
   7bc5485..9a7fbc9  master -> master
danielschmitz@tableless:~/workspace/site (master) $
```

Agora que resolvemos o problema do título, podemos voltar ao new_menu: `git checkout new_menu`. Após realizar este comando, temos o menu de volta no arquivo index.html, mas veja que o título não possui a tag H1. Isso acontece que estamos em outro branch. Tudo que acontece no master, fica no master. Tudo que acontece no new_menu, fica no new_menu

Merge com conflitos

Se desejarmos trazer o título do master para o new_menu, devemos fazer uma operação chamada `merge`, que irá juntar um código no outro. Então, estando no branch new_menu, e querendo trazer uma alteração do master para este branch, precisamos realizar o seguinte comando: `git merge master`. Caso existam alterações nas mesmas linhas entre mesmos arquivos, um conflito será gerado, como no exemplo a seguir:

```
index.html
1 <!DOCTYPE html>
2 <html>
3 <head>
4 <meta charset="UTF-8">
5   <title> Meu Site </title>
6 </head>
7
8 <body>
9 <<<<<< HEAD 2
10   Meu Site
11
12   <ul>
13     <li><a href="index.html">Home</a></li>
14     <li><a href="sobre.html">Sobre</a></li>
15     <li><a href="contato.html">Contato</a></li>
16   </ul>
17
18   =====
19   <h1>Meu Site</h1>
20   >>>>>> master 3
21 </body>
22
23 </html>

bash - "danielschmitz"
danielschmitz@tableless:~/workspace/site (new_menu) $ git merge master
Auto-merging index.html
CONFLICT (content): Merge conflict in index.html 1
Automatic merge failed; fix conflicts and then commit the result.
danielschmitz@tableless:~/workspace/site (new_menu|MERGING) $
```

Este é um exemplo de conflito que pode ocorrer quando realizamos um merge, indicado em 1. Perceba que o código html possui uma definição entre dois blocos, o primeiro, em 2 mostra como é o código do branch new_menu, e o segundo bloco, em 3, mostra como é o código no branch master. Edite o arquivo repassando para a seguinte forma:

```
index.html x +
1 <!DOCTYPE html>
2 <html>
3 <head>
4 <meta charset="UTF-8">
5   <title> Meu Site </title>
6 </head>
7
8 <body>
9   <h1>Meu Site</h1>
10
11   <ul>
12     <li><a href="index.html">Home</a></li>
13     <li><a href="sobre.html">Sobre</a></li>
14     <li><a href="contato.html">Contato</a></li>
15   </ul>
16
17 </body>
18
19 </html>

bash - "danielschmil x +
danielschmitz@tableless:~/workspace/site (new_menu) $ git merge master
Auto-merging index.html
CONFLICT (content): Merge conflict in index.html
Automatic merge failed; fix conflicts and then commit the result.
danielschmitz@tableless:~/workspace/site (new_menu|MERGING) $
```

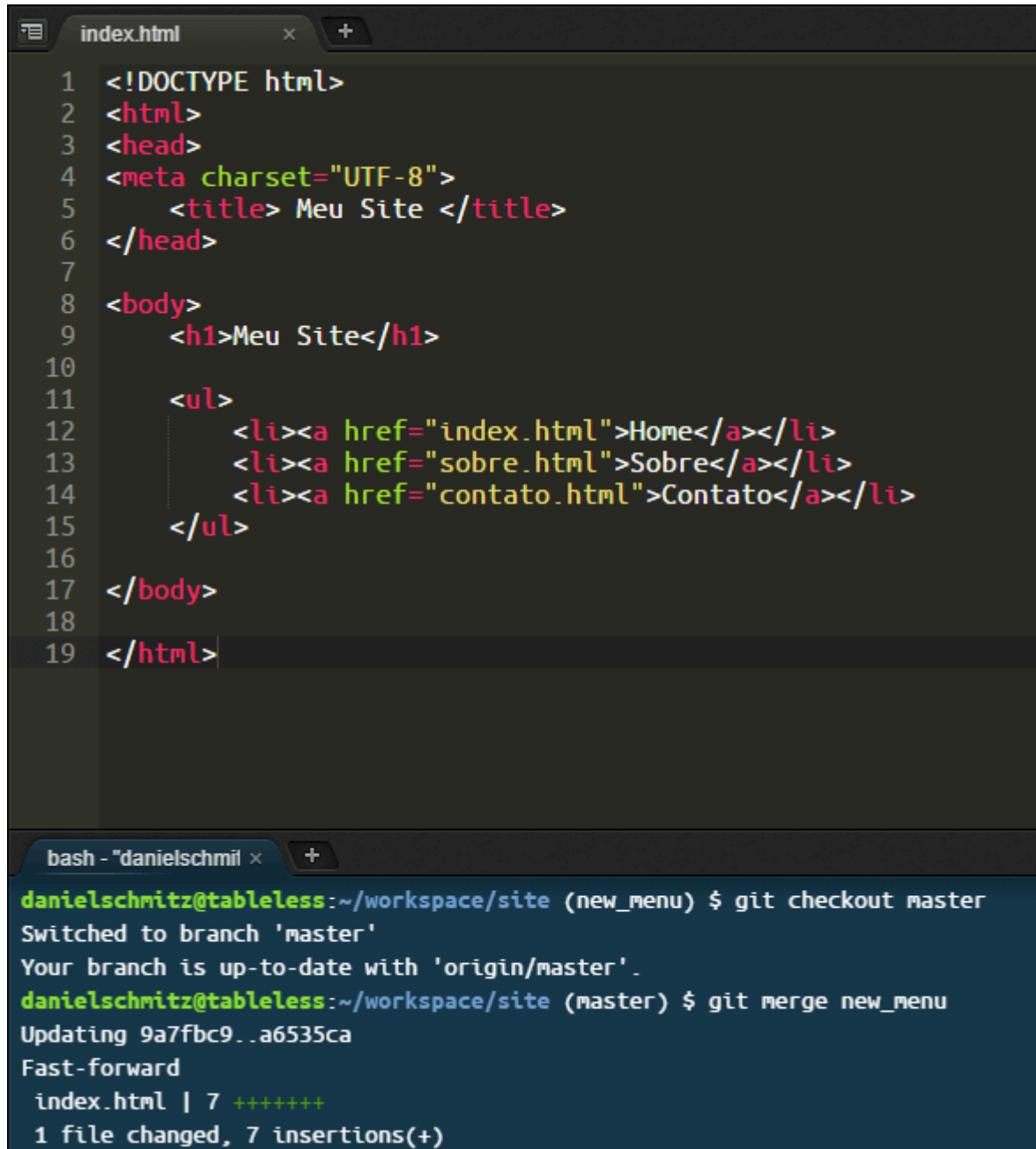
Ou seja, ajustamos os dois blocos, como se fosse um merge manual. Após resolver o conflito, vamos prepará-lo para o commit no branch `new_menu`, com o comando `git add`. Veja:

```
bash - "danielschmil x +
danielschmitz@tableless:~/workspace/site (new_menu|MERGING) $ git add index.html
danielschmitz@tableless:~/workspace/site (new_menu|MERGING) $ git commit -m "Menu com titulo correto"
[new_menu a6535ca] Menu com titulo correto
danielschmitz@tableless:~/workspace/site (new_menu) $ git push
```

Ou seja, resolvemos o conflito “na mão” e depois comitamos normalmente.

Merge sem conflitos

Quando não alteremos a mesma linha de um arquivo em branches diferentes, conseguimos realizar um merge sem ocasionar conflitos. Isso pode ser notado ao trazermos o menu do branch `new_menu` para o `master`, da seguinte forma:



The image shows a code editor window with a file named `index.html` and a terminal window below it. The code editor displays the following HTML code:

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4 <meta charset="UTF-8">
5   <title> Meu Site </title>
6 </head>
7
8 <body>
9   <h1>Meu Site</h1>
10
11   <ul>
12     <li><a href="index.html">Home</a></li>
13     <li><a href="sobre.html">Sobre</a></li>
14     <li><a href="contato.html">Contato</a></li>
15   </ul>
16
17 </body>
18
19 </html>
```

The terminal window shows the following commands and output:

```
bash - "danielschmil x +
danielschmitz@tableless:~/workspace/site (new_menu) $ git checkout master
Switched to branch 'master'
Your branch is up-to-date with 'origin/master'.
danielschmitz@tableless:~/workspace/site (master) $ git merge new_menu
Updating 9a7fbc9..a6535ca
Fast-forward
 index.html | 7 ++++++
 1 file changed, 7 insertions(+)
```

Se não houver conflitos, basta realizar um commit normal para confirmar o merge.