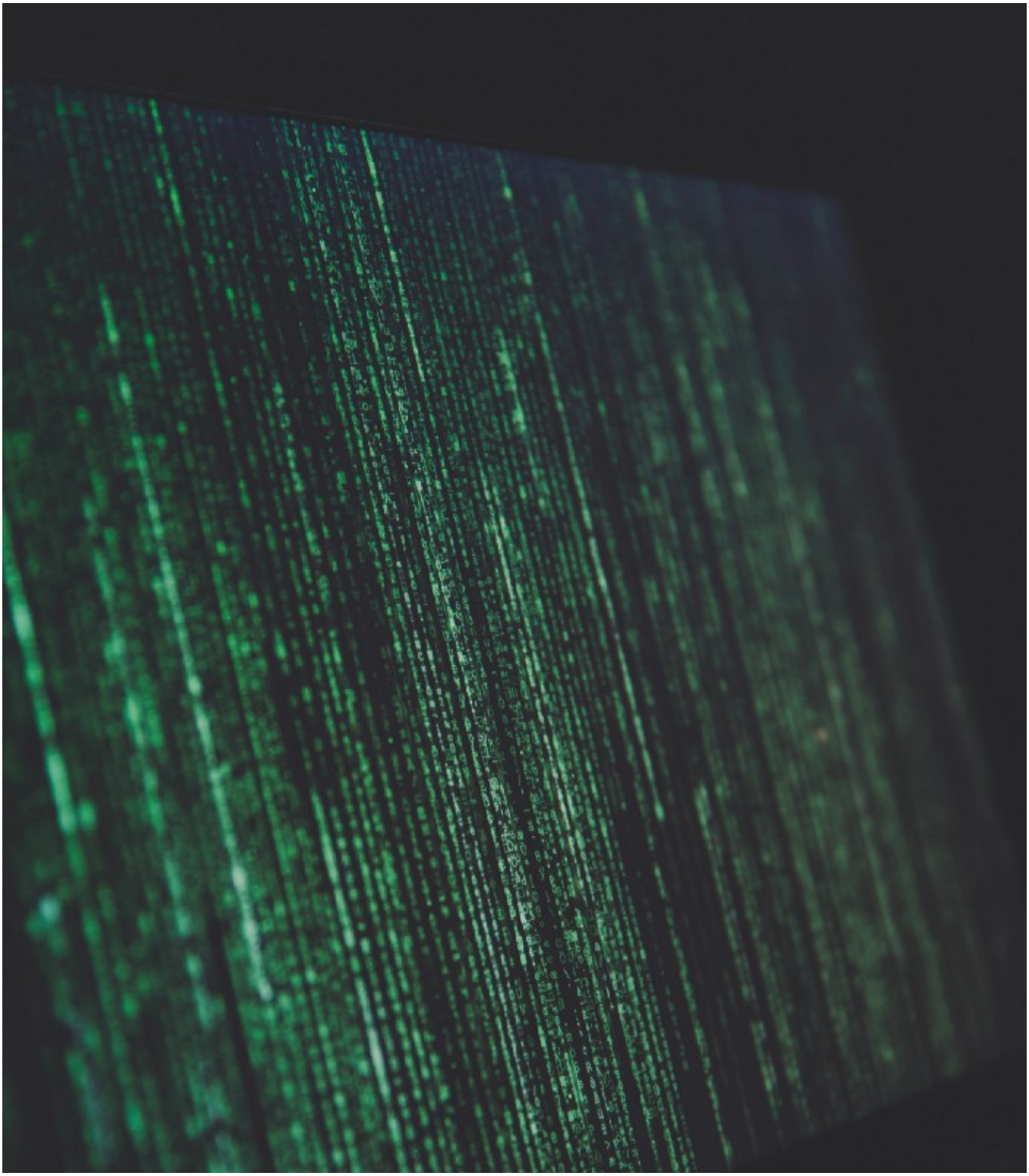
Uma introdução ao Git: O que é e como usar

Tacio S. S.
Jan 11, 2019 · 7 min read



by Markus Spiske on Unsplash

*** Este texto e uma tradução/adaptação completamente livre deste post do freeCodeCamp*

Git é um Sistema de controle de versão distribuido Open Source. Isto são muitas palavras pra definir o Git.

Vamos por partes pra entender bem isto:

- Sistema de Controle: Isto basicamente significa que o Git e um rastreador de conteúdo. Ou seja, o Git pode ser usado para armazenar conteúdo, geralmente código já que foi criado para isto, mas pode ser qualquer tipo de conteúdo.
- Sistema de Controle de Versão: O código armazenado no Git continua sendo alterado conforme mais código e adicionado. Acontece que vários desenvolvedores podem adicionar código em paralelo. Então o Sistema de Controle de Versão ajuda a lidar com isso mantendo um histórico de todas as alterações que aconteceram. O Git também prove algumas funcionalidades como branches e merges, sobre os quais vamos falar mais a frente.
- Sistema de controle de versão distribuido: O Git possui um repositório remoto que é armazenado em um servidor e um repositório local que é armazenado no computador de cada um dos desenvolvedores. Isto significa que o código não esta armazenado apenas em um servidor central, mas uma copia completa do código está também salva nos computadores de todos os programadores. Vamos falar do cenceito de repositório remoto e repositório local mais a frente neste artigo.

Porque eu preciso de um sistema de controle de versão como o Git?

Projetos na vida real geralmente tem vários desenvolvedores trabalhando em paralelo. Então um sistema de controle de versões como o Git e necessário para resolver os

conflitos de código entre os desenvolvedores.

Além disso, os requisitos de um projeto podem mudar bastante, e um sistema de controle de versão permite aos desenvolvedores voltar o projeto para uma versão antiga do código.

E por último, algumas vezes vários projetos que estão em paralelo involvem a mesma base de código. Neste caso, o conceito de Branchs do Git é muito importante.

Começando com o Git

Agora que ja mencionamos todos os conceitos, vamos explicar estes conceitos do git ateavés de um exemplo simples de seguir.

Instalando o Git

Este link traz detalhes de como instalar o git em diversos sistemas operacionais: https://git-scm.com/book/en/v2/Getting-Started-Installing-Git

Verifique se o git esta instalado com o seguinte comando no terminal:

```
git --version
```

Criando seu repositório local

No seu computador, crie uma pasta para o seu projeto. Vamos chamá-la de `simple-git-demo`.

Entre nesta pasta e adicione um repositório local ao projeto usando os seguintes comandos:

```
cd simples-git-demo
git init
```

O comanto `git init` adiciona um repositório local ao projeto.

Adicionando um pouquinho de código

Crie um arquivo chamado `demo.txt` na pasta do projeto com o seguinte texto:

```
Conteúdo Inicial
```

Aqui nos vamos brincar apenas com texto porque o foco deste artigo e o Git e não uma linguagem de programação específica.

Preparando e “Commitando” o código

O commit e o processo no qual o codigo e adicionado ao repositório local. Antes de commitar o código, ele deve estar na area de staging. A área de staging e onde são mantidas as alterações que ainda não foram commitadas.

Qualquer arquivo que não tenha sido adicionado a área de stagind não será commitado. Isto permite ao desenvolvedor controlar qual arquivo sera commitado.

Peparando

Use o seguinte comando para preparar o arquivo para ser commitado, ou seja mandá-lo para a área de staging.

```
git add demo.txt
```

No caso de querer adicionar vários arquivo você pode usar:

```
git add file1 file2 file3
```

Se você deseja adicionar todos os aquivos dentro da pasta do projeto, use o seguinte comando:

```
git add .
```

Use este último comando com cuidado para não adicionar arquivos indesejados na área de staging.

Commitando

Use o seguinte comando para commitar o arquivo:

```
git commit -m "Meu primeiro Commit"
```

“Meu primeiro Commit” será a mensagem do commit. Procure usar mensagems de commit que sejam relevantes e que indiquem o que as alterações do código fazem.

Git Status e Git Log

Agora modifique o arquivo `demo.txt` e adicione o seguinte texto:

```
Conteúdo Inicial
Add mais conteúdo
```

Status

Use o comando `git status` para encontrar informações sobre quais arquivos foram modificados e quais estão na ara de staging, e portanto prontos para receberem um commit. Outras informações também vao aparecer mas vamos ignora-las por enquanto.

Use o seguinte comando:

```
git status
```

O status mostra que o `demo.txt` foi modificado e não esta na área de staging.

Agora vamos adicionar o `demo.txt` para a área de staging e realizar o commit com os seguintes comandos:

```
git add demo.txt
git commit -m "demo.txt file is modified"
```

Log

Use `git log` para exibir todos os commits realizados ate agora.

O log mostra o autor de cada commit, a data do commit, e a mensagem do commit.

Branches

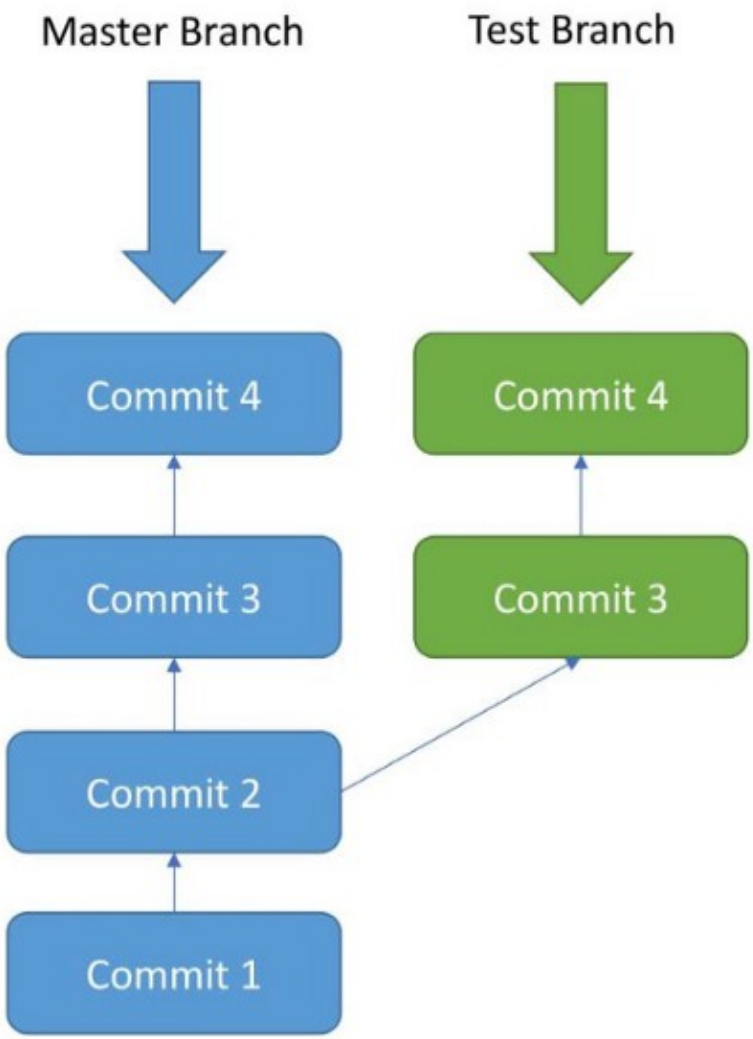
Até agora nos não criamos nenhum branch no Git. Por default, os commits no Git vão para o branch **maste**.

O que é um branch?

Um branch e basicamente um ponteiro para o ultimo commit no repositório Git. Então atualmente nosso branch master e um ponteiro para o segundo commit `"demo.txt file is modified"`.

Porque precisamos de vários Branchs?

Multiplos branchs são necessários para permitir desenvolver em paralelo. Veja na Imagem abaixo como os branches trabalham;



Fonte: https://medium.freecodecamp.org/what-is-git-and-how-to-use-it-c341b049ae61

Inicialmente, os commits 1 e 2 foram feitos no branch master. Depois do commit 2 um novo branch chamado “Test” foi criado, e os commits 3 e 4 foram adicionados ao branch Test.

Ao mesmo tempo, um commit 3 e um commit 4 diferentes foram adicionados ao branch master. Aqui nos podemos ver que depos do commit2, duas features estão sendo desenvolvidas paralelamento em 2 branchs separados.

O Branch Test e o Branch Master tem códigos diferentes que podem ser unidos usando o `git merge`. Vamos falar disso mais a frente.

Criar um novo branch no repositório local

Vamos criar um novo branch chamado **test** usando o seguinte comando:

```
git branch test
```

Este comando cria o branch `test`.

Entretanto nos continuamos no contexto do branch master. Para começarmos a trabalhar no branch **test** use o comando abaixo:

```
git checkout test
```

Agora nos estanos no branch `test`.

Você pode listar todos os branches no seu repositório local com o seguinte comando:

```
git branch
```

Vamos fazer alguns commits no novo branch

Modifique o `demo.txt` adicionando o seguinte conteúdo:

```
Conteúdo Inicial
Add mais conteúdo
Add mais conteúdo no branch Test
```

Agora vamos commitar esta alteração

```
git add demo.txt
git commit -m "Test Branch Commit"
```

Este commit foi realizado no branch test, a agora o branch test está afrente do branch master por 1 commit.

Você pode verificar o historico do Branch Test usando o `git log`.

Merge

Atualmente, o branch test está a frente do master por 1 commit. Vamos dizer que nos queremos que todo o codigo no branch test "volte" para o branch master. Pra isso o `git merge` sera muito útil.

Para realizar o merge do código do branch test para o branch master, siga os seguintes passos:

Primeiro volte para o branch master:

```
git checkout master
```

E execute o commando `merge`:

```
git merge test
```

Após executar estes 2 comandos, o merge deve ter sido concluido com sucesso. Neste exampleo, não houve conflitos;

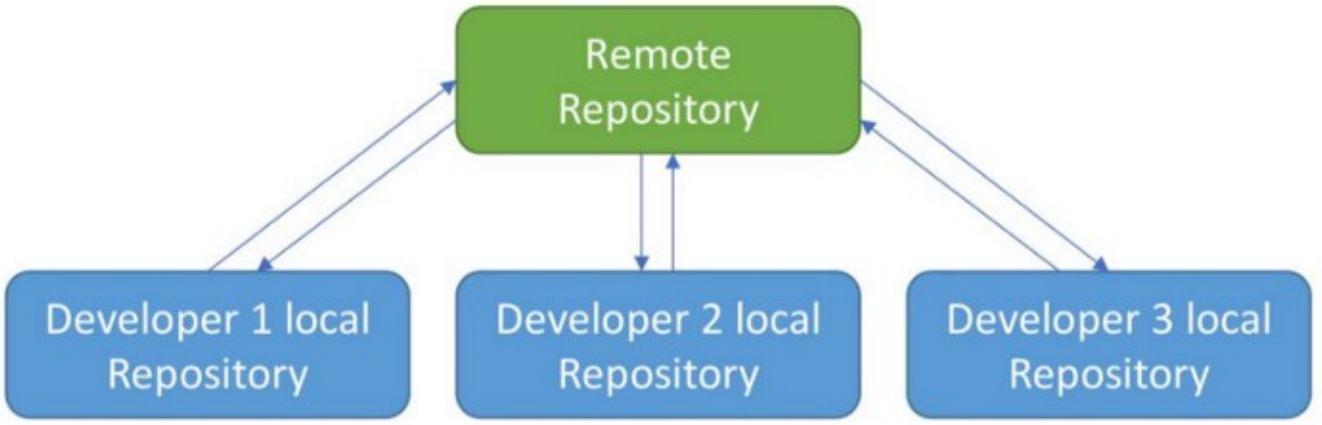
Entretanto em projetos reais, podem aparecer conflitos durante o merge, Resolver os conflitos e algo que vem com a experiência, por isso, ao trabalhar com o Git, vcê serpa

capaz de resolver os conflitos.

Executando o `git log` você irá notar que o master agora possui três commits.

O repositório remoto

Até agora, nos estamos trabalhando apenas no nosso repositório local. Cada desenvolvedor irá trabalhar nos seus repositórios locais mas eventualmente, eles vão "subir" este código para o repositório remoto. Com o codigo no repositório remoto outros desenvolvedores, podem ver e modificar este código.



Showing Remote and Local Repositories — Fonte: https://medium.freecodecamp.org/what-is-git-and-how-to-use-it-c341b049ae61

Github

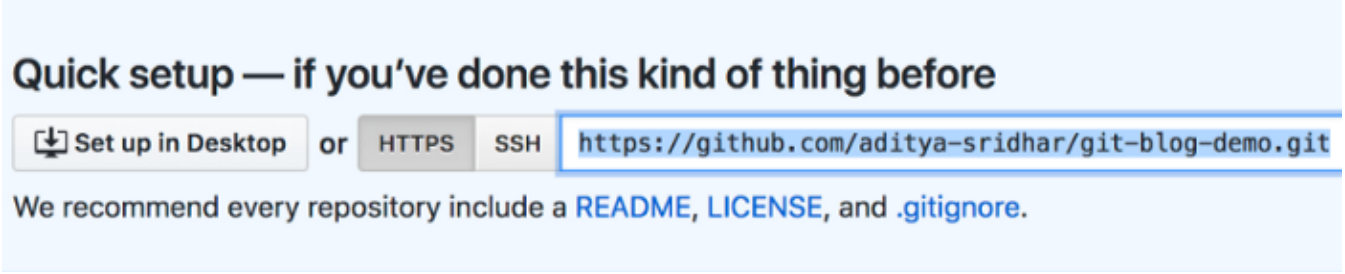
Nos vamos usar o Github como repositório remoto.

Vá para o https://github.com/ e crie uma conta;

Depois de se registrar no Github, click em **Start Project** para criar um novo repositório Git. De um nome para o repositório e clique em "Create Repository"

Vamos criar um repositório com o nome `git-blog-demo` .

Isto vai criar um repositorio remoto no github, e quando você abrir o repositório, uma pagina como a da imagem a baixo ira aparecer:



Fonte: https://medium.freecodecamp.org/what-is-git-and-how-to-use-it-c341b049ae61

O URL do repositorio é a parte destacada da imagem: **https://github.com/aditya-sridhar/git-blog-demo.git**

Para apontar o seu repósitorio local para o repositorio remoto, use o seguinte comando:

```
git remote add origin [repository url]
```

Git Push

Para "subir" o codigo do repositório local para o repositório remoto use o seguinte commando:

```
git push -u origin master
```

Isto ira subir o código do branch master local para o branch master no repositório remoto.

Alguns comandinhos

Git pull

O `git pull` é usado para baixar as versoes mais atuais do repositório remoto. Quando o repositório remoto é constantemente atualizado por outros desenvolvedores o `git pull` é necessário:

```
git pull origin master
```

Git Clone

`git clone` é usado para clonar um repositório remoto ja existente:

```
git clone [repository url]
```