

Desenvolvimento de Sistemas

Repositórios remotos e sistemas de gerenciamento de configuração

Introdução

Quando se pensa em controle de código e versionamento, sabe-se que um projeto, caso se esteja trabalhando sozinho nele, pode estar apenas no repositório local, armazenado na própria máquina do desenvolvedor. No entanto, quando se trabalha com uma equipe de desenvolvedores, torna-se necessário o uso de um repositório remoto que possa ser acessado por toda a equipe e que centralize todos os *branches*, *commits*, *logs* etc.

Veja algumas características de repositórios locais e remotos:

Localização

Quanto à localização, os repositórios locais podem estar armazenados nos computadores dos membros da equipe de desenvolvimento. Já os repositórios remotos estarão armazenados em computadores que são acessados remotamente por toda a equipe, podendo ser hospedados tanto na Internet quanto na própria rede interna da equipe.

Funcionalidades

Quanto às funcionalidades de cada um, falando tecnicamente para a equipe, os repositórios local e remoto funcionam da mesma maneira, com os mesmos comandos e os os mesmos *branches*, *commits* e *logs* (descritos no

decorrer deste conteúdo). No entanto, em um repositório local, haverá uma cópia de todo o trabalho e todas as versões do projeto que podem ser usadas. Já no repositório remoto não há essa pasta com todos os arquivos, há apenas a pasta **.git** do projeto.

Repositórios

Quanto ao uso de repositórios, é importante sempre lembrar que os trabalhos nos projetos sempre ocorrem na máquina local; todas as modificações no código e seus **commits** são feitos localmente. Com isso, e após isso, as modificações podem ser “subidas” para um repositório remoto com o intuito de compartilhar com todo o time de desenvolvimento. Nunca se deve esquecer de que repositórios remotos são uma maneira de compartilhar e dividir código entre desenvolvedores, e não uma maneira de armazenar e trabalhar com arquivos.

Criação de repositórios

Quanto à criação de repositórios, há duas opções: usar um repositório local, que pode ser criado do zero ou clonado de um repositório local existente, ou então criar um repositório remoto, que pode ser feito de duas maneiras (caso você já tenha um repositório local, basta você cloná-lo com a **tag --bare**; e em caso de criar um novo repositório remoto em branco, basta usar o comando **git init** também com a **tag --bare**).

Fluxo de trabalho

Quanto ao fluxo de trabalho em repositórios remotos e locais, ao usar o Git para controle de seu repositório, há apenas alguns comandos para interagir com repositórios remotos, pois a grande maioria das operações que são executadas se dão no repositório local. A menos que você tenha clonado

um projeto remoto, estará trabalhando exclusivamente em um repositório local que nunca deixará a própria máquina. Com isso, para trabalhar com um repositório Git local, nem mesmo a Internet é necessária, pois todo o trabalho pode ser feito de maneira *off-line*.

A seguir, veja uma figura que representa o fluxo de trabalho de um repositório Git que é tanto local quanto remoto.

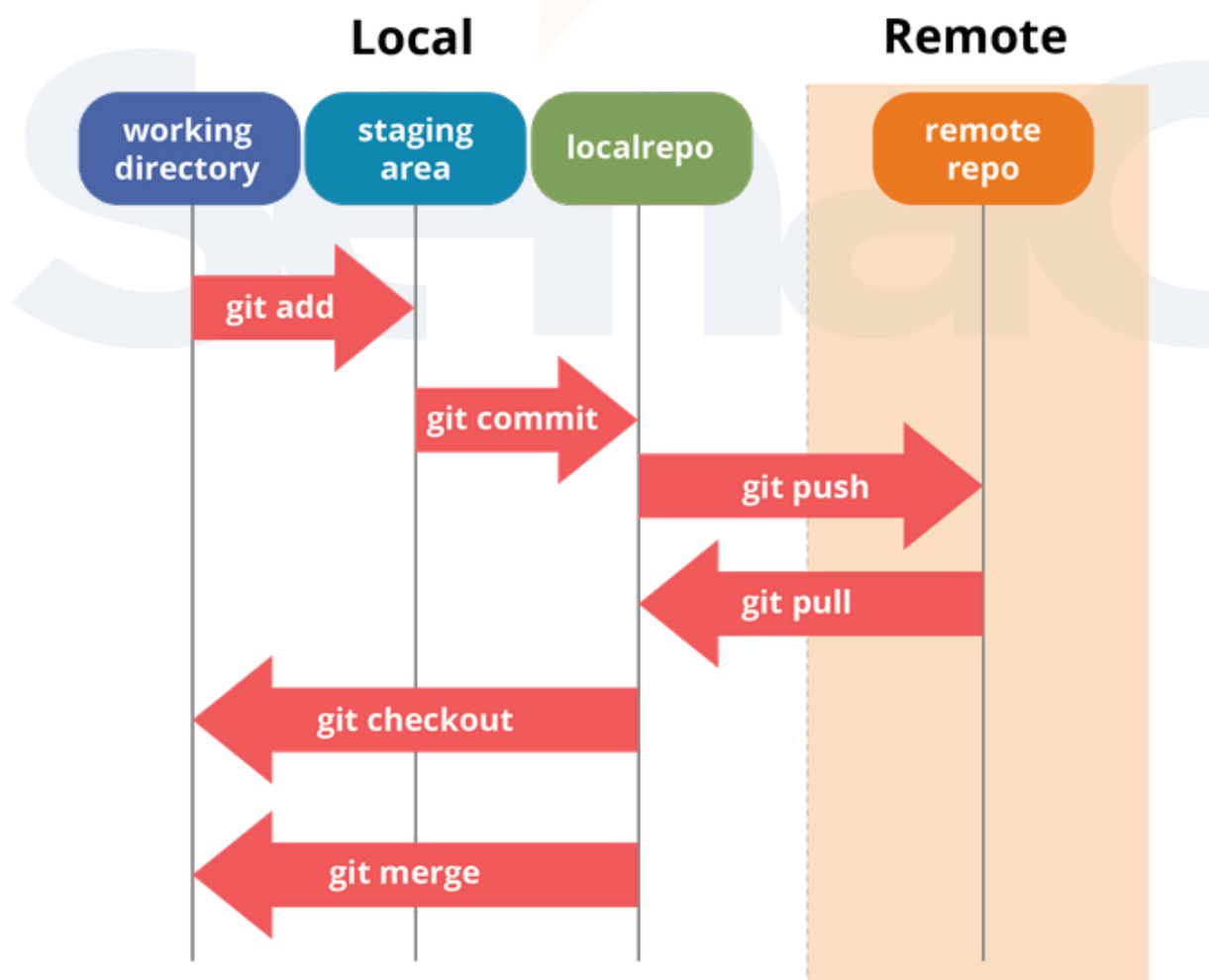


Figura 1 – Visualizando como funciona um repositório local e remoto

Fonte: Senac EAD (2023)

Armazenamento em nuvem

O armazenamento em nuvem refere-se a algo armazenado em um computador que não está localmente com o usuário, e sim em algum lugar remoto.

Pode-se pensar em diversos serviços de armazenamento na nuvem disponíveis: iCloud, OneDrive, Google Drive etc. No entanto, para versionamento de código, não basta ter um lugar na nuvem para armazenamento, pois, assim, haverá apenas os seus arquivos *on-line*, sem uma organização específica ou um controle de envios e atualizações.

Apenas armazenar arquivos em nuvem não difere muito (nem deve diferir) de guardar arquivos em uma pasta local do computador. Assim, caso sejam feitas alterações localmente nos arquivos, tornaria-se muito inconveniente manter a nuvem e o armazenamento local sincronizados. O problema ficaria ainda maior caso houvesse uma equipe de desenvolvimento, pois várias pessoas mantendo vários arquivos em um armazenamento *on-line* seria muito caótico e antiprodutivo.

Armazenar códigos na nuvem refere-se a serviços *on-line* que têm um servidor Git rodando e ao qual se tem acesso para fazer a manutenção dos códigos.

Em meados de 2005, o criador do Kernel Linux, Linus Torvalds, já não conseguia manter um código tão grande e complexo e contava com a participação de milhares de colaboradores do projeto Linux. Ele então pensou em como criar uma ferramenta que pudesse ter velocidade, fosse simples, pudesse manter milhares de ramificações de código, fosse completamente distribuída e *on-line* e conseguisse lidar com grandes projetos com eficiência. Ele criou então, no ano de 2005 o Git, um sistema de gerenciamento de configuração que ajuda você a acompanhar todas as alterações feitas no código do seu projeto. Você pode salvar cópias de seus trabalhos em momentos diferentes, para que possa voltar e ver o que escreveu anteriormente, por exemplo. Você também pode compartilhar seus códigos com outras pessoas e trabalhar neles juntos.

O Git funciona criando uma pasta especial em seu computador (repositório local), na qual você guarda todos os seus arquivos e códigos. Quando você quiser fazer uma alteração em um arquivo, você executa um comando **git** (no caso, o comando **commit**) para salvar uma cópia do código. Você também pode adicionar uma mensagem para explicar o que mudou.

O Git controla todas as cópias de seus arquivos e as mensagens que você adicionou. Dessa forma, você pode ver um histórico de todas as alterações feitas e o que estava pensando quando as fez.

Para usar o Git, você precisará instalá-lo em seu computador. Depois de instalado, você pode criar um novo projeto Git criando uma nova pasta e aplicar alguns comandos, como **git init**, para inicializar o repositório, **git add**, para informar alterações em um arquivo para o controle de versão, e **git commit**, para salvar a cópia atualizada do arquivo no controle de versão, informando uma observação que explica o que foi alterado.

Você também pode usar o Git para compartilhar seus arquivos com outras pessoas. Para fazer isso, você pode usar um serviço de nuvem como o GitHub (você verá como fazer uso do GitHub na sequência do conteúdo), que é um *site* que permite armazenar seus repositórios Git *on-line*. Você pode usar o Git para enviar suas alterações para o repositório no GitHub, e outras pessoas podem usar o Git para enviar essas alterações para os próprios computadores.

Há muito mais coisas que você pode fazer com o Git, mas essas são as etapas básicas para começar.

Veja, a seguir, alguns comandos Git comuns e uma breve explicação do que eles fazem.

git init

Inicializa um novo repositório Git.

git clone

Faz uma cópia de um repositório Git existente.

git add

Adiciona um arquivo à área de alterações de arquivos, que é uma lista de alterações que serão incluídas no próximo **commit**.

git commit

Salva um “retrato atual” das alterações na área de alterações de arquivos e adiciona uma mensagem descrevendo-as.

git push

Envia as alterações confirmadas para um repositório remoto, como um repositório no GitHub.

git pull

Baixa as alterações de um repositório remoto, mesclando-as no repositório local.

git diff

Mostra as diferenças entre duas versões de um arquivo.

git log

Mostra um histórico de **commits** para um repositório.

Veja, a seguir, um exemplo de como você pode usar alguns desses comandos.

- ◆ Crie uma nova pasta e navegue até ela no terminal.
- ◆ Execute **git init** para inicializar a pasta como um repositório Git.
- ◆ Crie alguns novos arquivos ou faça alterações nos arquivos existentes no repositório.
- ◆ Execute **git add** seguido pelos nomes dos arquivos que você deseja adicionar à área de teste. Por exemplo, **git add arquivo1.txt arquivo2.txt**.
- ◆ Execute **git commit** para salvar um instantâneo das alterações na área de preparação e adicione uma mensagem descrevendo as alterações.
- ◆ Se você deseja compartilhar suas alterações com outras pessoas, pode executar **git push** para enviar as alterações confirmadas para um repositório remoto.

Pensando nesse modelo de colaboração em códigos de programação, há vários serviços gratuitos de armazenamento de códigos, por exemplo, GitHub, GitLab, BitBucket, entre outros. Porém não se pode esquecer de que é possível

criar o próprio servidor na nuvem e, com ele, configurar e instalar um servidor Git para que seja utilizado, então, para o armazenamento de seus códigos.

Esse tipo de serviço é muito utilizado por empresas de desenvolvimento que não têm um servidor local. Então, algumas utilizam um servidor privado virtual (VPS) para fazer a manutenção de seus códigos na nuvem. Algumas empresas que alugam servidores virtuais privados são: Vultr, Digital Ocean, Amazon Aws, Google Cloud, entre outras.

Serviços disponíveis

Para armazenar seus códigos, hoje, existem diversos serviços gratuitos e *on-line*, que servem inclusive para usar como portfólio. Afinal de contas, as empresas, na hora de contratar um novo desenvolvedor, sempre dão uma conferida no seu GitHub, GitLab ou semelhante. Inclusive, é sempre bom colocar em seu currículo o *link* para os seus códigos armazenados nesses serviços *on-line*.

Aqui neste conteúdo, você verá como utilizar o GitHub, por ser o mais popular entre esses serviços. Contudo, tenha em mente que, em sua maioria, eles funcionam de maneira muito semelhante (e com pequenas alterações você será capaz de utilizar o que melhor lhe servir).

Para criar uma conta no GitHub, siga as etapas a seguir.

1

Acesse o *síte* do GitHub.

2

Clique no botão **Inscreva-se** (ou **Sign up**) no canto superior direito da página.

3

Digite seu endereço de *e-mail*, escolha um nome de usuário e crie uma senha.

4

Clique no botão **Continuar**.

5

Siga as instruções para concluir o processo de inscrição. Isso pode incluir a verificação do seu endereço de *e-mail* ou o preenchimento de um *captcha*.

6

Ao final, você pode escolher quantas pessoas podem colaborar em seu time e de quais ferramentas você necessita (marque ao menos **código colaborativo**).

Depois de criar sua conta, você pode usar o GitHub para armazenar e gerenciar seus repositórios Git *on-line*. Você pode usar o Git para enviar suas alterações para o repositório no GitHub, e outras pessoas podem usar o Git para enviar essas alterações para os próprios computadores.

Utilização

Um repositório tem a sua URL (*uniform resource locator*, ou localizador uniforme de recursos) remota, que nada mais é do que uma maneira “bonitinha” de dizer “o local onde o seu código está”. Nessa URL está a localização do seu código. No caso do GitHub, existem duas maneiras de subir o seu código utilizando essa URL remota, quais sejam:

- ◆ Uma URL remota de acesso via HTTPS (*hyper text transfer protocol secure*, ou protocolo de transferência de hipertexto seguro) – `https://github.com/user/repo.git`.
- ◆ Uma URL remota de acesso via SSH (*secure shell*) – `git@github.com:user/repo.git`.

Para criar um repositório remoto, utiliza-se o comando **git remote add**.

```
git remote add origin <REMOTE_URL>
```

O GitHub associa a URL remota a um nome, e, por padrão, usa-se o nome “origin”.

Caso seja necessário alterar essa URL, é possível usar o comando **git remote set-url** para fazer tal alteração.

Você saberá mais sobre esse tipo de operação na próxima parte do conteúdo, a qual abordará os programas utilizados para a configuração e a manutenção desses repositórios.

As URLs “https://” do GitHub estão disponíveis para todos os repositórios da plataforma. No entanto, quando você for executar um comando **git clone**, **git fetch**, **git pull** ou **git push**, será pedida uma senha de acesso e serão exibidas novas

políticas de segurança da plataforma. Não se pode usar uma simples senha para executar esses comandos. Contudo, em vez da senha, você usará um *token* pessoal de acesso ao repositório.

Tokens de acesso pessoal aos repositórios são uma alternativa às senhas e devem ser usados para o acesso remoto a repositórios armazenados na plataforma GitHub.

Existem dois tipos de *token* de acesso aos repositórios, os *fine-grained personal access tokens* e os *personal access tokens (classic)*. A plataforma sugere que seja utilizado o primeiro deles, em razão das vantagens que ele apresenta. Inclusive, com o *fine-grained*, as organizações podem liberar ou restringir acesso a certos repositórios, o que é interessante para empresas com um grande número de desenvolvedores.

As vantagens do *fine-grained personal access token* são:

- ◆ Cada *token* só pode acessar recursos pertencentes a um único usuário ou uma única organização.
- ◆ Cada *token* só pode acessar repositórios específicos.
- ◆ Cada *token* recebe permissões específicas, que oferecem mais controle do que os escopos concedidos aos *tokens* de acesso pessoal (clássico).
- ◆ Cada *token* deve ter uma data de expiração.
- ◆ Os proprietários da organização podem exigir aprovação para qualquer *token* de acesso pessoal refinado que possa acessar recursos na organização.

Para efeito de estudo, neste conteúdo, será usado o *token* clássico, pois você trabalhará com repositórios pessoais de projetos de estudo.

Para criar esse *token* pessoal de acesso clássico, siga os seguintes passos:

1. Verifique seu endereço de *e-mail*, caso ainda não tenha sido verificado.
2. No canto superior direito de qualquer página dentro do GitHub, clique na foto do seu perfil e clique em **Settings** (configurações).

The logo for Senac, featuring a stylized star above the word "Senac" in a light blue, sans-serif font. The star is composed of several overlapping triangles in shades of blue and yellow.



Signed in as **octocat**



Set status

Your profile

Your repositories

Your organizations

Your projects

Your stars

Your gists

Feature preview

Help

Settings

Sign out

Figura 2 – Clicando no ícone de conta do usuário

Fonte: Senac EAD (2023)

3. Na barra lateral esquerda, clique em **Developer Settings** (configurações do desenvolvedor), último item do menu à esquerda.

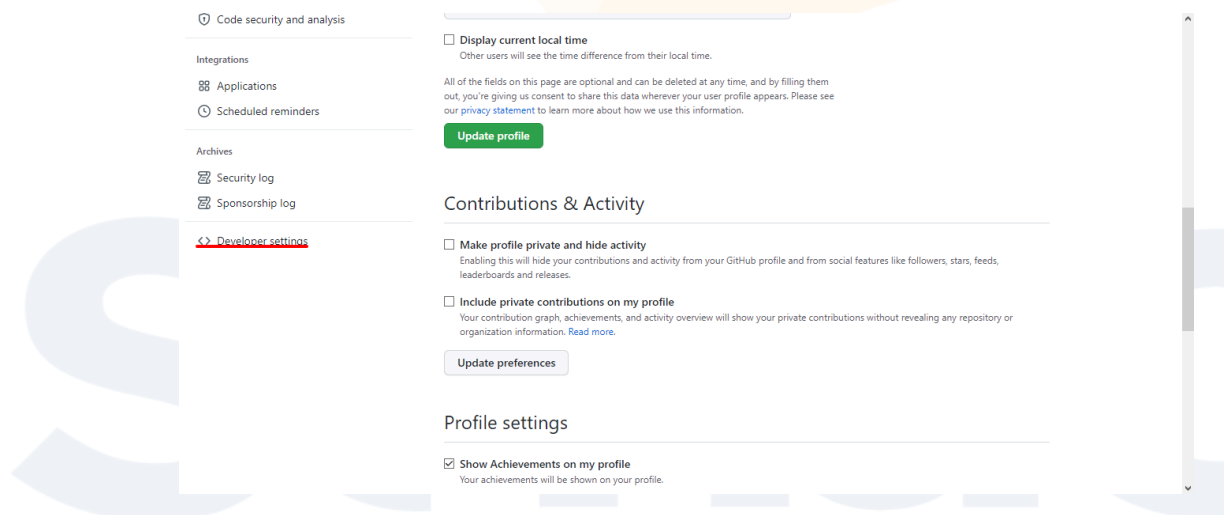


Figura 3 – Item **Developer Settings** na página de configurações da conta

Fonte: GitHub (2023)

4. Na barra lateral esquerda, em **Personal Access Tokens** (*tokens de acesso pessoal*), clique em **Tokens (classic)**.
5. Selecione **Generate new token** (gerar novo *token*) e clique em **Generate new token (classic)**.

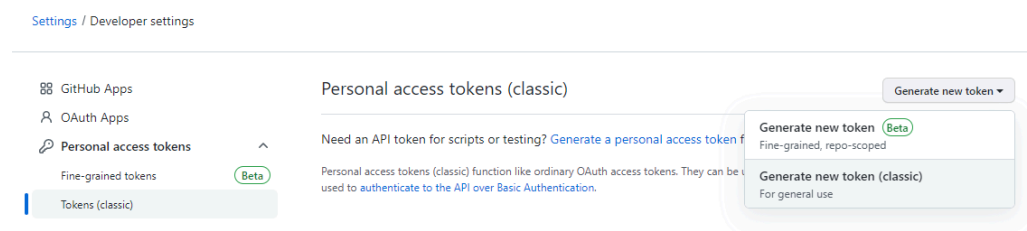


Figura 4 – Opção **Generate new token (classic)**

Fonte: GitHub (2023)

6. No campo **Note**, dê um nome descritivo ao seu *token*.

Note

My bash script

What's this token for?

Figura 5 – Nomeando o *token* a ser gerado

Fonte: Senac EAD (2023)

7. Para dar uma data de expiração ao seu *token*, selecione o menu suspenso **Expiration** e clique em um padrão (semanas ou meses) ou então use o seletor de calendário.

Expiration

7 days



The token will expire on Friday, Feb 8 2008

Figura 6 – Campo de data de expiração do *token*

Fonte: Senac EAD (2023)

8. Selecione os escopos que você gostaria de conceder a esse *token*. Para usar seu *token* para acesso a repositórios por meio da linha de comando, selecione **repo** (veja no *gif* a seguir). Um *token* sem escopos atribuídos só pode acessar informações públicas.
9. Clique em **Generate token** (gerar *token*).

☐ write:gpg_key

Write user gpg keys

☐ read:gpg_key

Read user gpg keys

Generate token

Cancel

Figura 7 – Botão **Generate token**

Fonte: Senac EAD (2023)

O *token*, então, é gerado e pode ser usado como senha quando for questionado para executar as ações via linha de comando no repositório. É importante certificar-se de copiar e salvar esse *token* em algum arquivo, pois ele será exibido apenas uma vez pelo GitHub.

Tokens you have generated that can be used to access the [GitHub API](#).

Make sure to copy your new personal access token now. You won't be able to see it again!

✓ ghp_IqIMN0ZH6z0wIEB4T9A2g4EHMy8Ji42q4HA5 

Enable SSO ▾

Delete

Figura 8 – *Token* gerado e pronto para ser copiado e usado

Fonte: Senac EAD (2023)

No Git, quando for necessário utilizar o *token*, é necessário informá-lo via linha de comando ou por meio da interface gráfica sendo utilizada. A seguir, veja um exemplo de como seria o uso do *token* em uma linha de comando.

```
$ git clone https://GitHub.com/USERNAME/REPO.git
Username: YOUR_USERNAME
Password: YOUR_TOKEN
```

Então, utilizam-se o seu nome de usuário do GitHub e, no campo de *password*, o *token* gerado para acesso ao repositório.

Sistema de gerenciamento de configuração

É claro que nem sempre será preciso recorrer a todos os comandos Git para executar seu trabalho diário em códigos e projetos. Para facilitar todas essas operações, existem diversos clientes Git que podem ser instalados, e suas ferramentas podem ser utilizadas para controlar e configurar seus repositórios.

Os mais populares entre esses clientes Git são:



Neste conteúdo, você verá como usar o GitHub Desktop, por ser o mais popular entre os clientes Git do mercado. Lembre-se de que qualquer um deles tem similaridades com o GitHub Desktop.

Instalação

Ao acessar a página do GitHub Desktop (busque por “*desktop* GitHub” na Internet), você verá o botão, ao centro, com o *link* para o *download* do instalador para a versão Windows. Caso você seja usuário de macOS, logo abaixo, à direita, há o *link* para a versão do instalador.

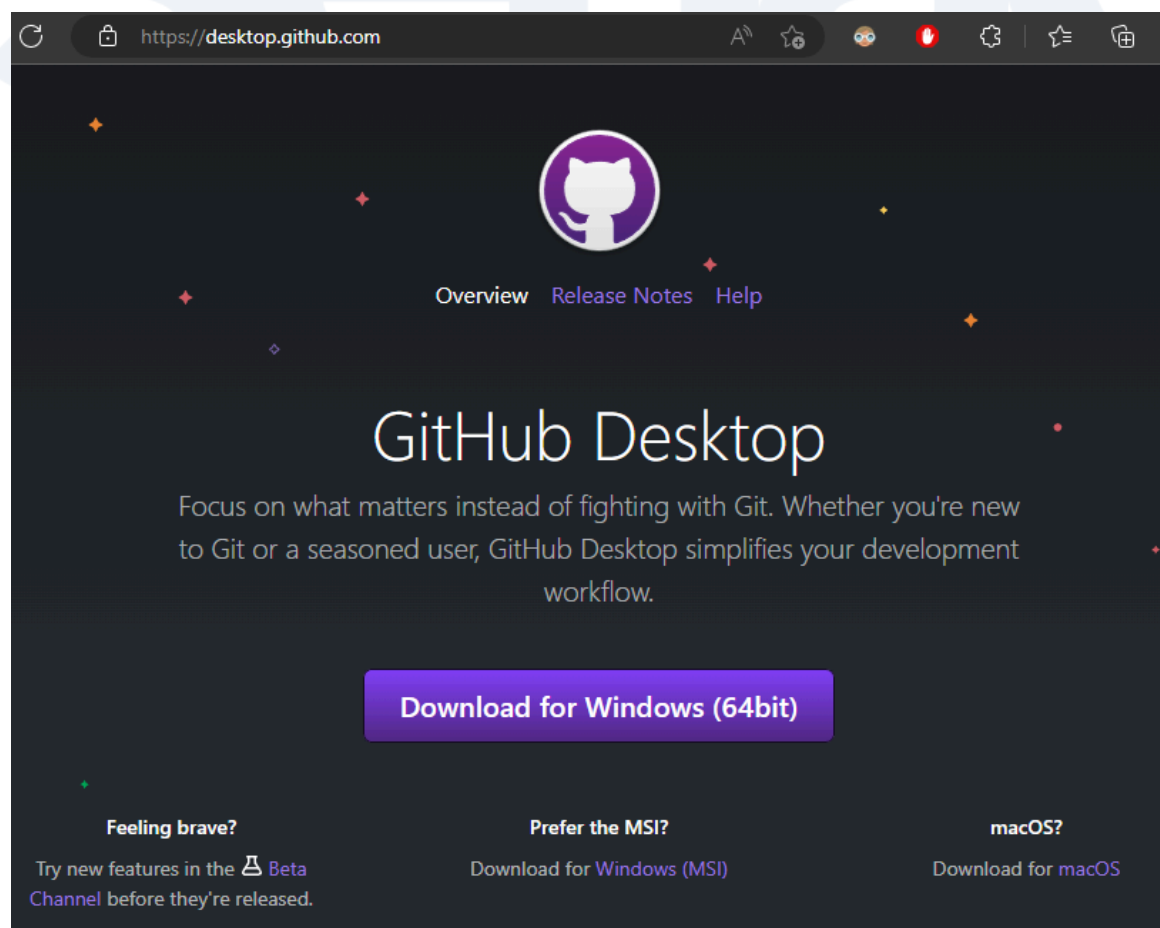


Figura 9 – Página inicial para o *download* do *app* do GitHub Desktop

Fonte: GitHub Desktop (2023)

Configuração

Ao executar o instalador, você é levado à primeira tela, que faz a sugestão de logar com sua conta do GitHub, o que facilitará muito o processo de configuração. Faça, então, o *login* com o seu usuário e a sua senha do GitHub.

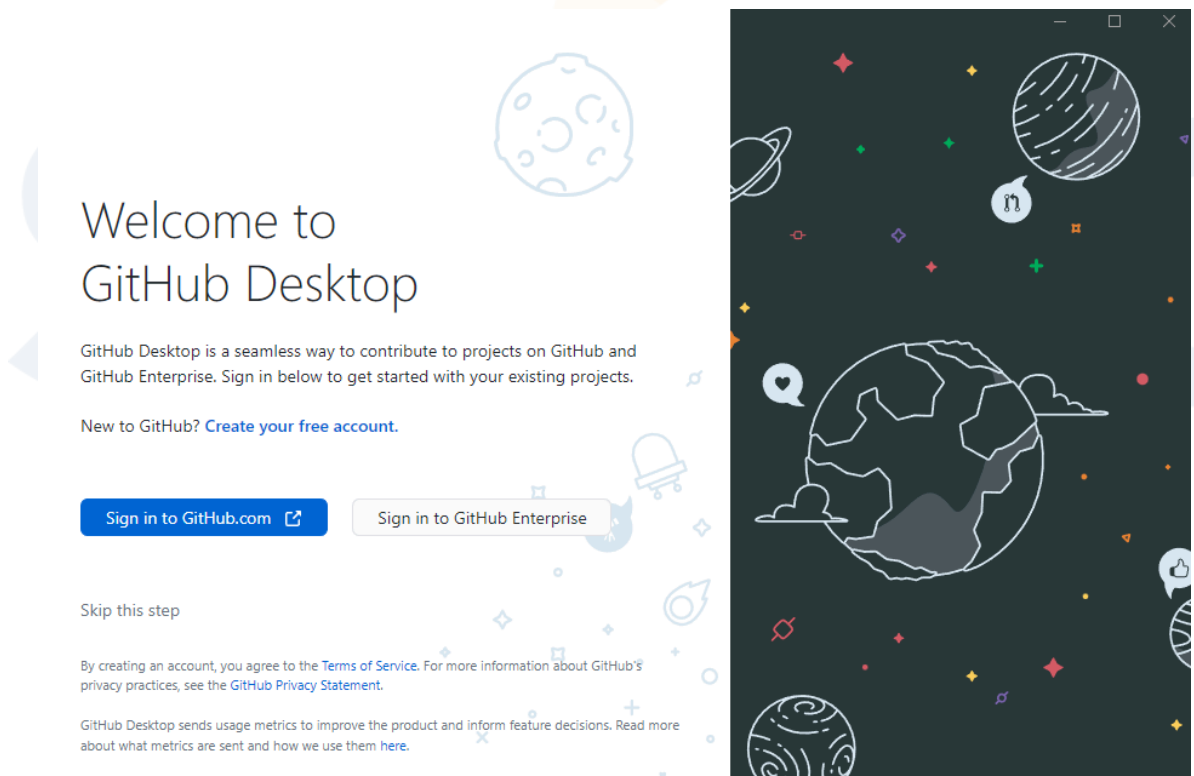




Figura 10 – Tela inicial de instalação do GitHub Desktop

Fonte: GitHub Desktop (2023)

Ao clicar em **Sign in to GitHub.com** (botão azul), você é levado à página de *login* do site do GitHub.

Sign in to GitHub
to continue to GitHub Desktop

Username or email address

Password [Forgot password?](#)

[Sign in](#)

New to GitHub? [Create an account.](#)

[Terms](#) [Privacy](#) [Security](#) [Contact GitHub](#)

Figura 11 – Entrando com usuário e senha para logar no GitHub
Fonte: GitHub (2023)

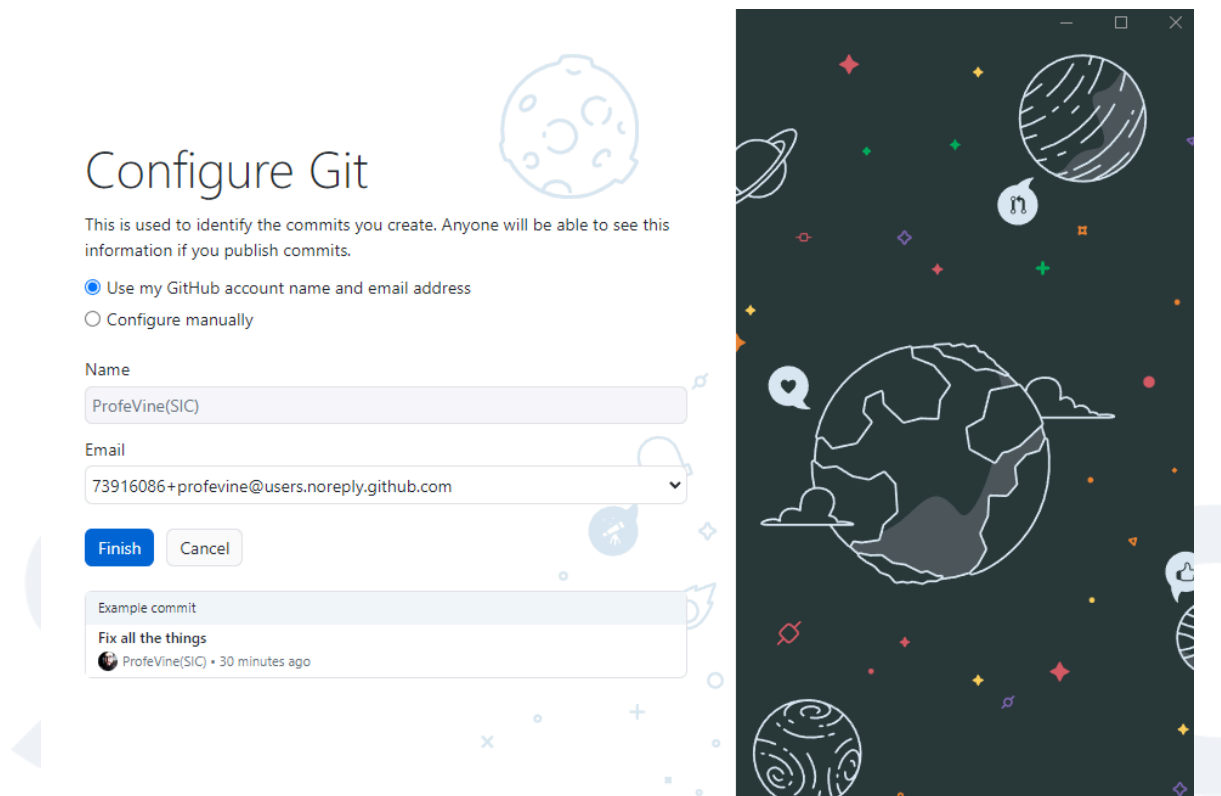


Figura 12 – Tela de configuração inicial do GitHub Desktop

Fonte: GitHub Desktop (2023)

É possível perceber que, agora, os dados da sua conta já foram “puxados” e você está pronto para concluir a instalação, clicando no botão azul **Finish**.

Após essa etapa, você já é levado à tela inicial do aplicativo, onde pode escolher a opção **Create a tutorial repository** (criar um repositório de tutorial) para aprender como criar um novo repositório no GitHub por meio do *app*.

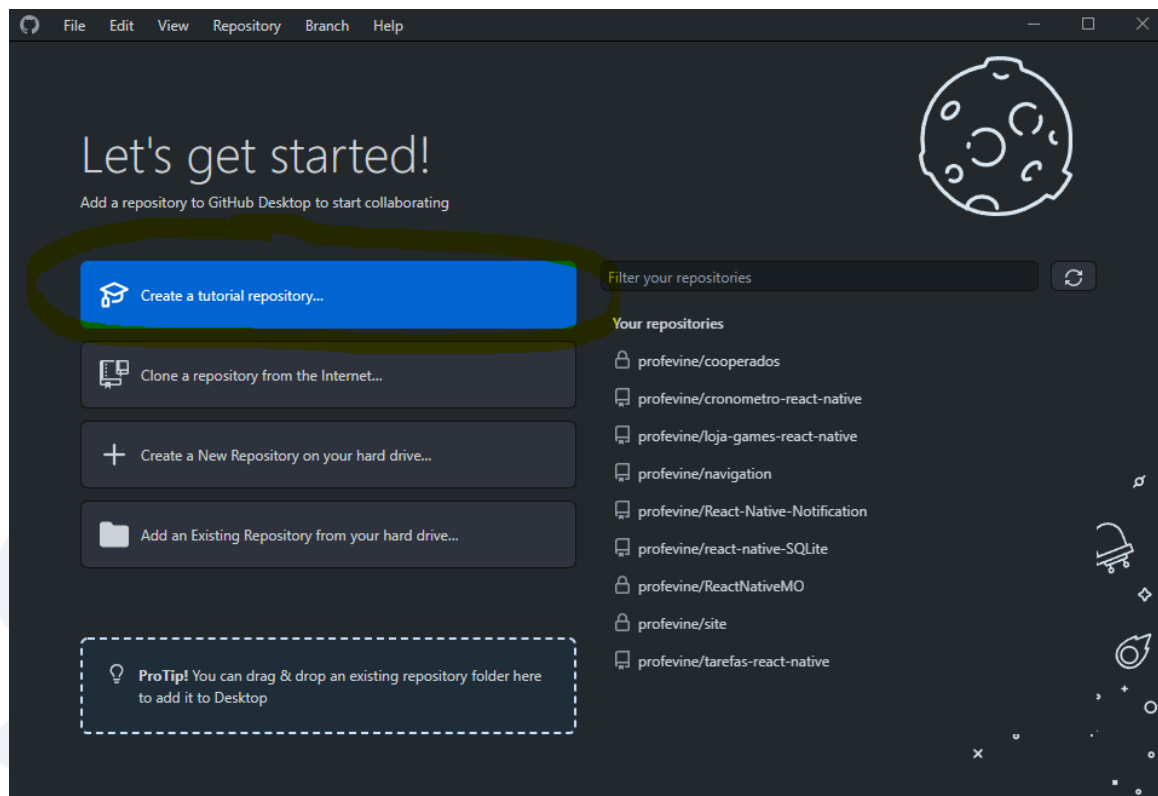


Figura 13 – Tela inicial do GitHub Desktop (criando um repositório de tutorial)

Fonte: GitHub Desktop (2023)

A seguir, veja o botão azul que inicia a criação de um repositório e faz um tutorial de como criá-lo. É possível ver que, na lista à direita, em destaque amarelo, estão os repositórios que você já tem em sua conta do GitHub.

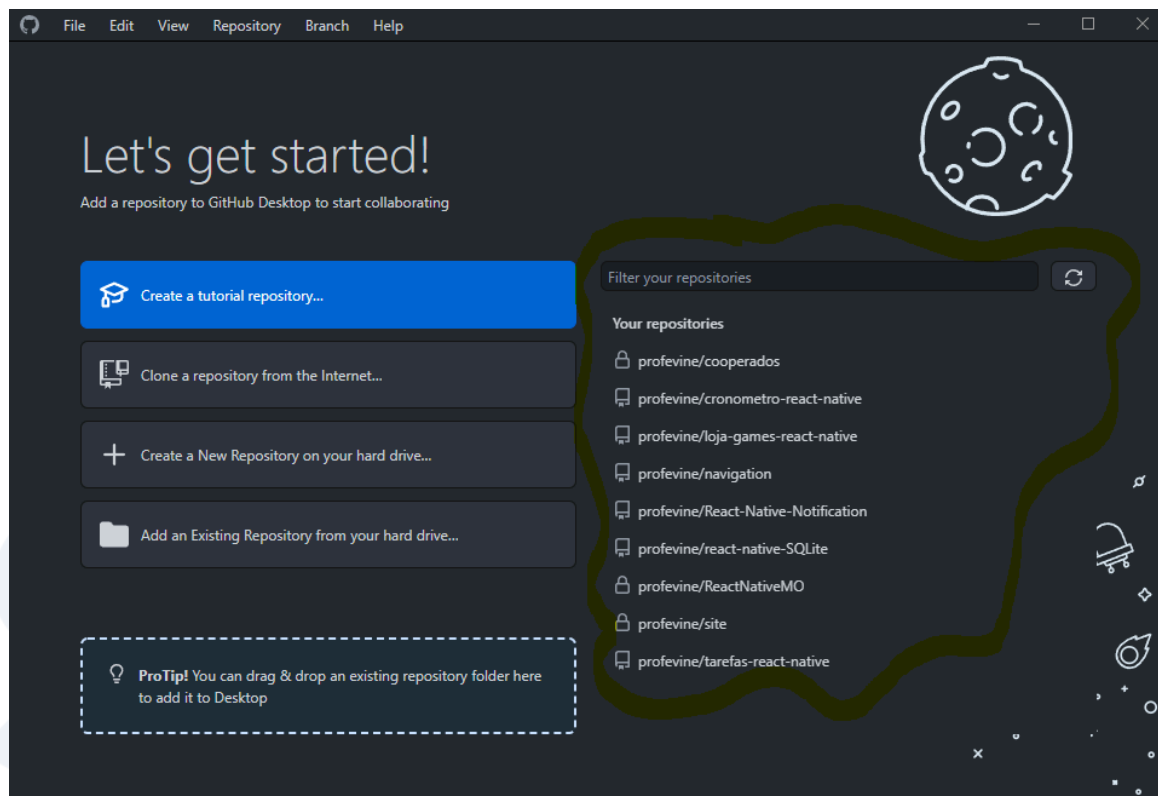


Figura 14 – Repositórios remotos encontrados na conta logada em questão

Fonte: Senac EAD (2023)

Fazendo o tutorial de criação de repositório

Ao clicar no botão de criação de repositório, tutorial, você receberá um alerta:

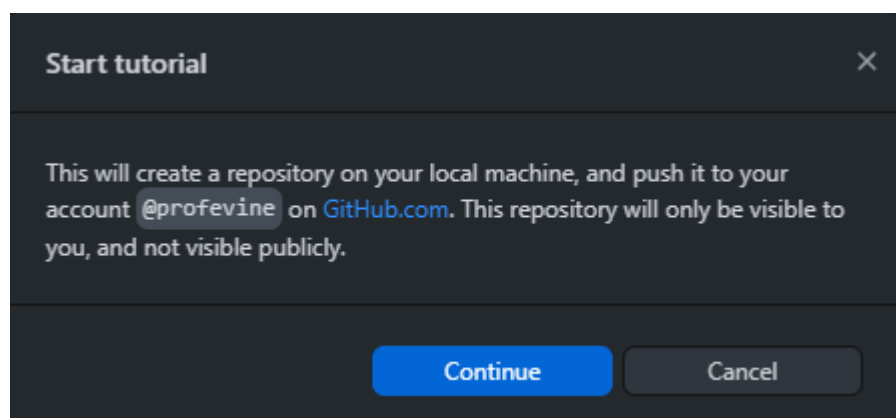


Figura 15 – Caixa de alerta de início do tutorial

Fonte: GitHub (2023)

Nesse aviso, está escrito que o tutorial criará um repositório na sua máquina local com o nome da sua conta no GitHub e que esse repositório será visível apenas para você que o criou. É possível clicar, então, no botão azul **Continue**.

A criação do repositório começará com a barra de progresso em branco sendo preenchida. Assim que o GitHub Desktop for fazer o *upload* do repositório para o *site* do GitHub, você será questionado sobre o *token* de conexão que criou anteriormente, mas, como você está no ambiente *desktop*, pode fazer novamente o *login* pelo *site*, clicando no botão azul **Sign in with your Browser**. Caso queira logar com o *token* criado, basta clicar no botão cinza **Sign in with a code**.

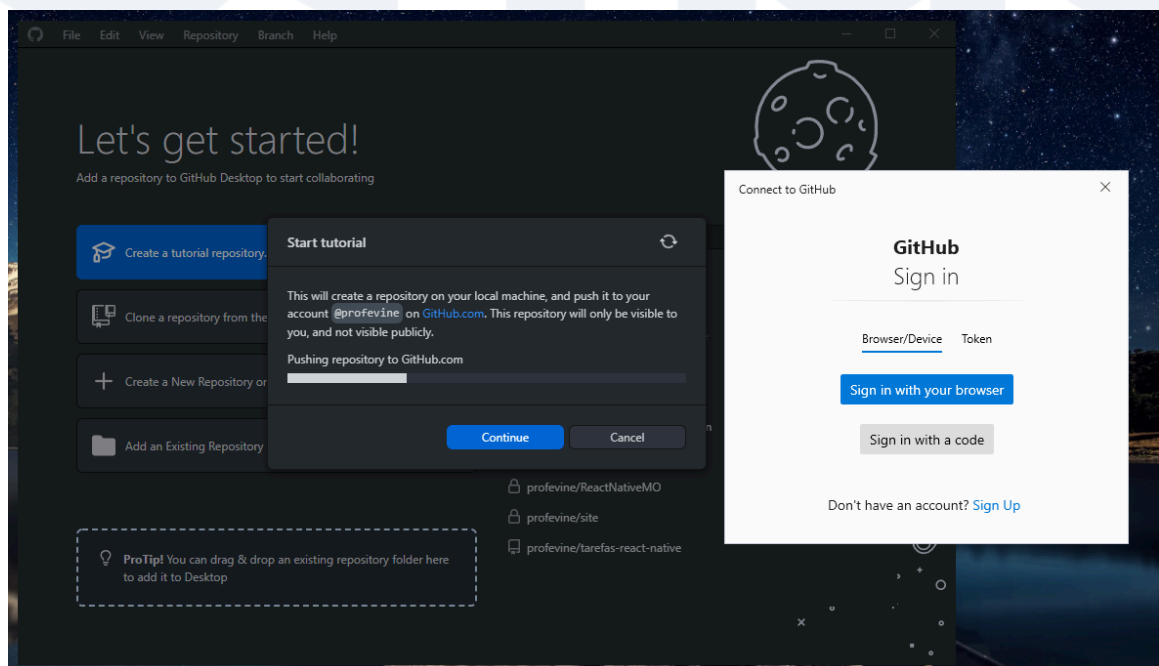


Figura 16 – Logando com o navegador ou com o *token*

Fonte: GitHub (2023)

Clicando em **Sign in with your Browser**, você será redirecionado mais uma vez ao navegador, mas, como já fez o *login* antes, desta vez será apenas direcionado ao aplicativo e receberá no navegador a tela de conexão bem-sucedida.

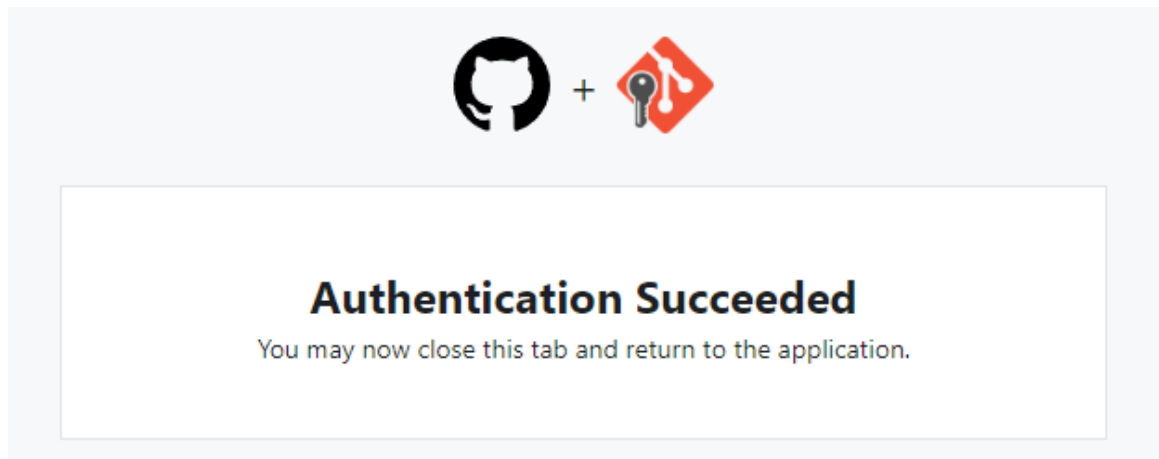


Figura 17 – Alerta de *login* com sucesso

Fonte: GitHub (2023)

Feito isso, a configuração do seu primeiro repositório local e remoto está pronta.

O repositório local será criado em uma pasta semelhante a esta: **C:\Users\usuario\Documents\GitHub\desktop-tutorial**. Assim, você estará pronto para fazer seus **commits** e **pushes** à vontade.

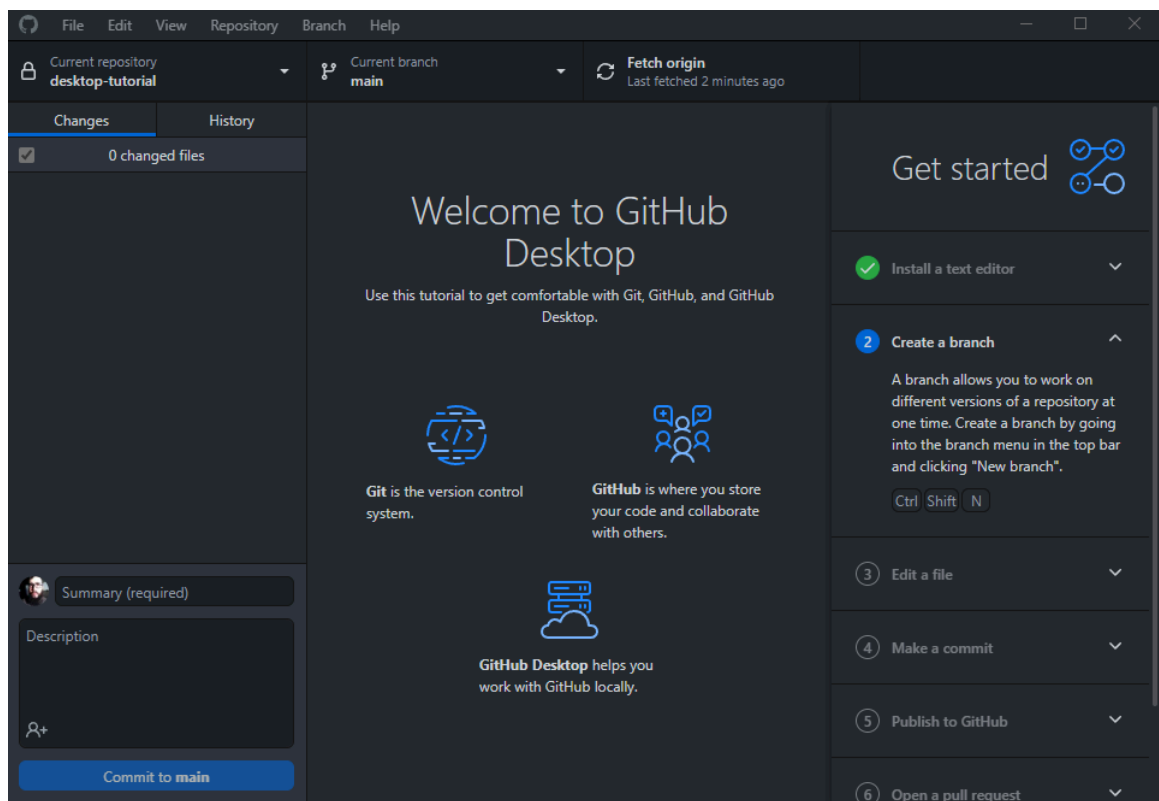


Figura 18 – Repositório de tutorial criado com sucesso

Fonte: GitHub Desktop (2023)

Gerenciamento

Após a criação do repositório, você pode, então, começar propriamente o tutorial à direita, onde é possível ver uma lista com alguns passos a serem seguidos para aprender o básico sobre o gerenciamento do repositório criado.

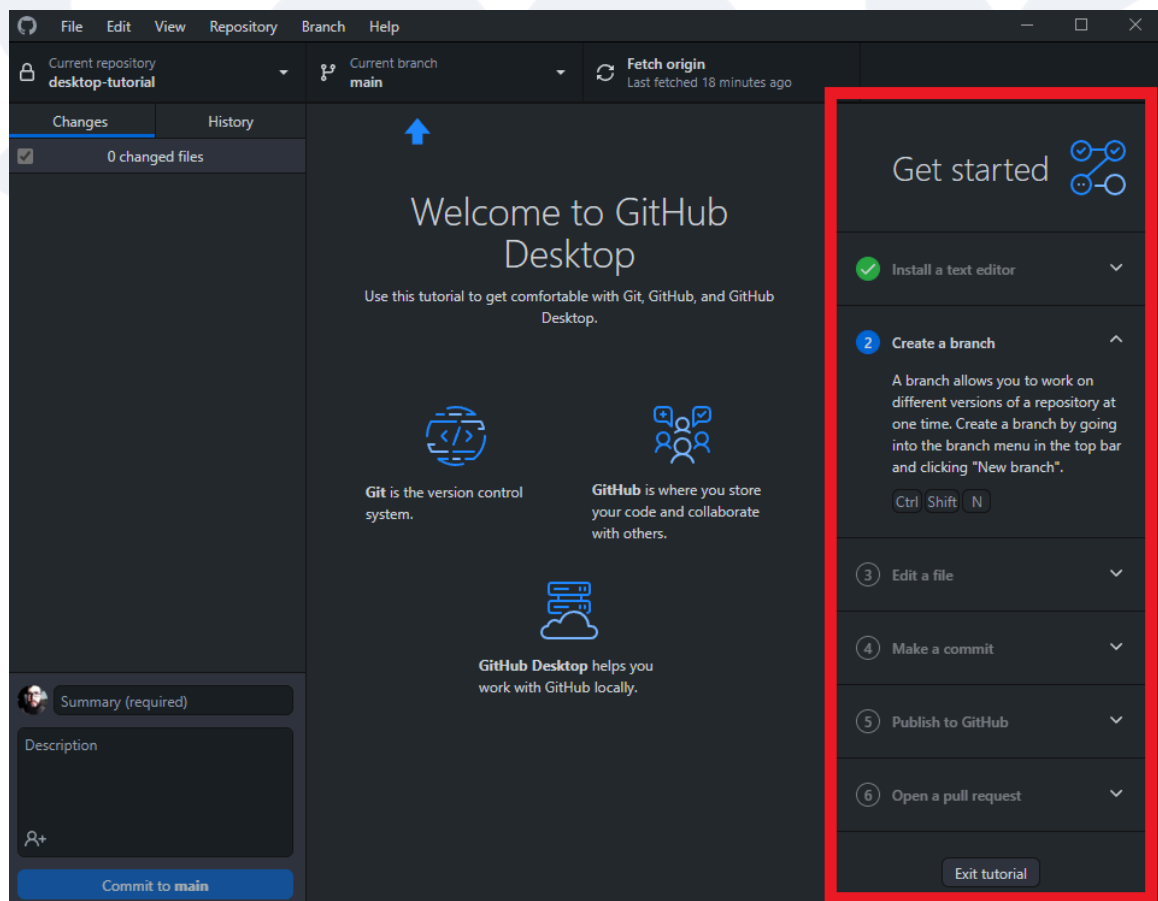


Figura 19 – Barra com o passo a passo do tutorial

Fonte: GitHub Desktop (2023)

Pode-se ver, neste caso, que o primeiro passo, **Install a text editor** (instalar um editor de texto), já está cumprido, pois, na máquina, você já tem um editor de texto instalado. Na sua máquina, você pode utilizar o editor de código ou texto de

sua preferência, NetBeans, VsCode, Sublime etc.

O passo número dois do tutorial nos ensina a como criar uma nova *branch* (ramificação) para o nosso repositório. Essas ramificações podem sofrer alterações sem prejudicar nem alterar o código da *branch* principal. Esse novo ramo criado só entrará para o ramo principal caso ele seja *merged* (mesclado) com o principal do projeto.

Há duas maneiras de fazer isso: clicando no seletor de *branches*, que, no momento, está em **branch main**, como aponta a seta azul, ou então apertando as teclas **Ctrl + Shift + N**.

Ao apertar o atalho **Ctrl + Shift + N**, a janela de criação de nova *branch* se abre e pede que você escolha um nome para ela. Neste caso, será criada a *branch* “teste”.

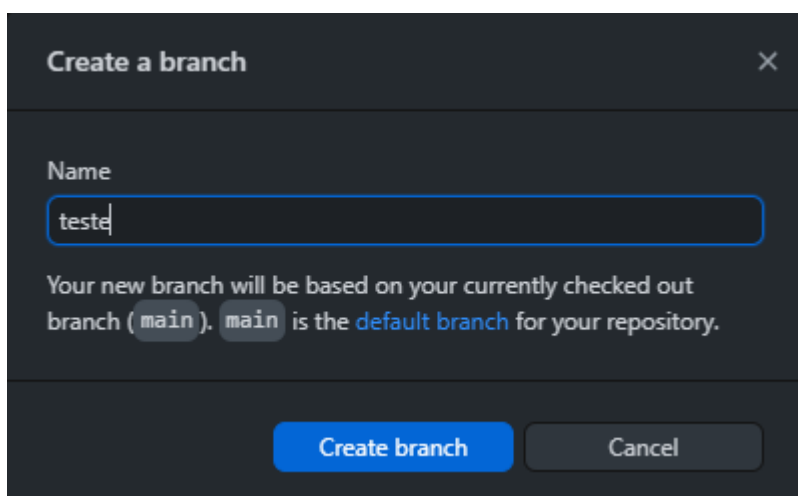


Figura 20 – Criando uma nova *branch* no projeto do tutorial

Fonte: GitHub Desktop (2023)

Após clicar no botão **Create branch**, você verá que a *branch* foi criada e que é atualmente a selecionada.

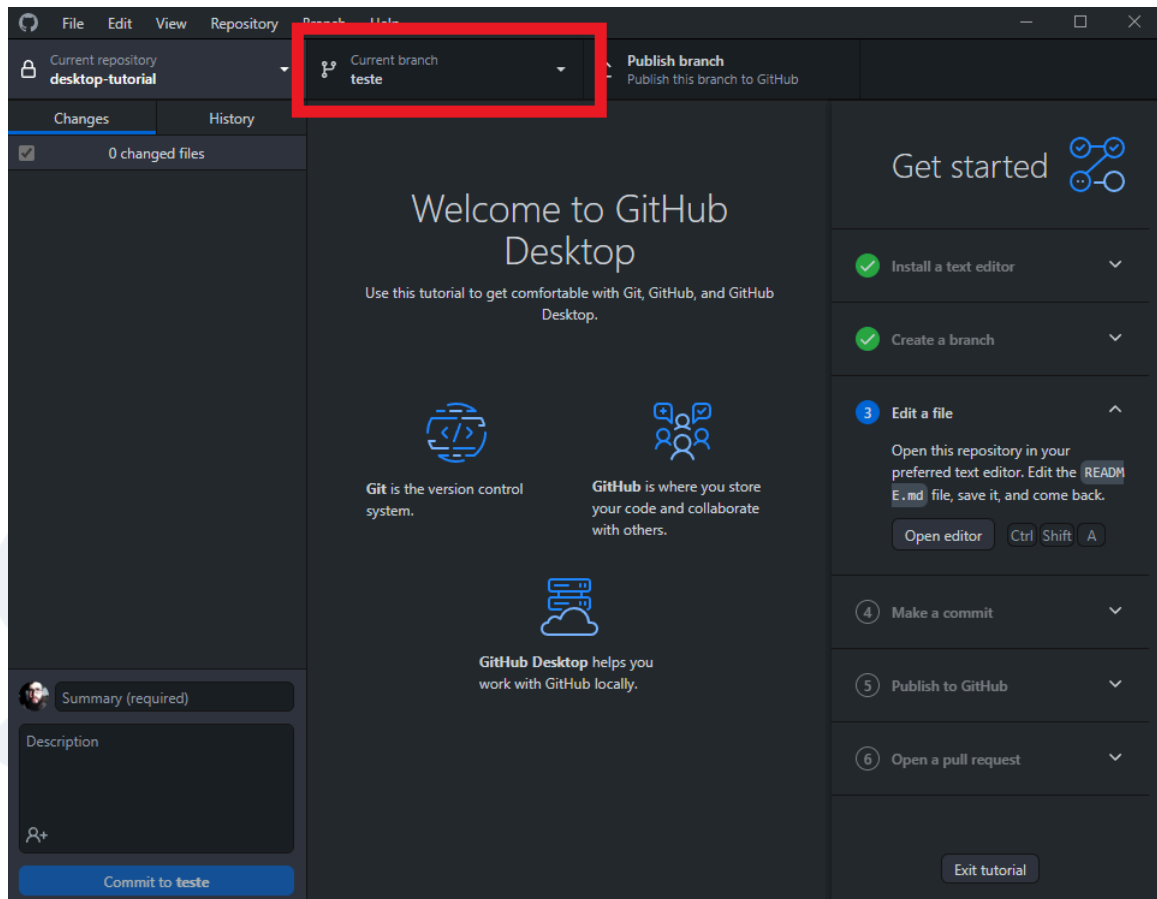


Figura 21 – *Branch* criada e em uso com sucesso

Fonte: GitHub Desktop (2023)

O próximo passo é editar o arquivo **README.md**.

A importância do README.md

O **README.md** é a “pagina” principal do seu repositório e deve conter informações sobre o repositório para auxiliar quem for usar o código ou contribuir com ele. Principalmente se o código for compartilhado com uma equipe, é no **README.md** que você colocará as instruções para a equipe.

É preciso levar em consideração alguns fatores ao criar o **README.md**.

O nome do arquivo deve sempre ser **README.md**, e esse arquivo contém (não obrigatoriamente) o seguinte:

1. O que é esse *software* e para que serve
2. Quem é o responsável pelo código
3. O *status* atual do projeto (se é livre, deprecado, ainda não lançado etc.)
4. Mais informações sobre o projeto

Exemplo de um **README.md**:

```
# Repositório de teste para conteúdo do curso TDS EAD Senac - Criado por...

Este é apenas um repositório de teste para configuração e funcionalidades do aplicativo para desktop GitHub Desktop

Livre para uso pessoal irrestrito.
```

Voltando ao tutorial, siga com o próximo passo.

Usando o atalho **Ctrl + Shift + A**, o seu editor configurado como padrão em sua máquina já abrirá o arquivo **README.md** do seu repositório local.

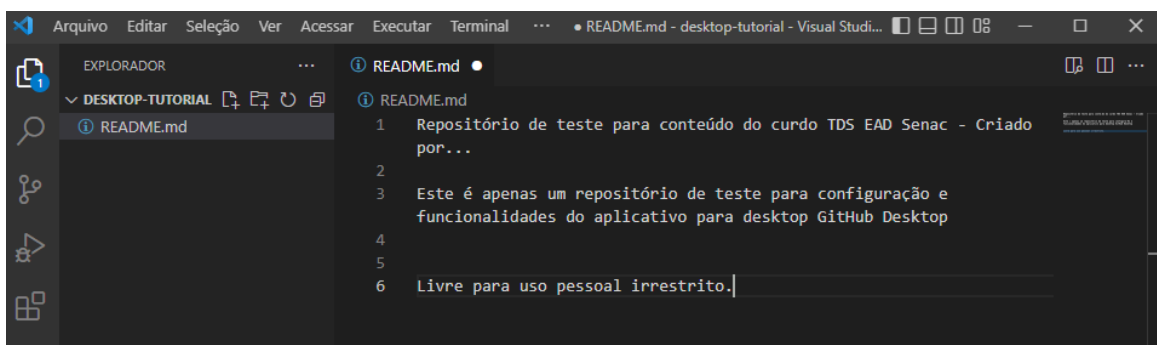


Figura 22 – Tela do editor de texto criando uma alteração no **README.md**

Fonte: Senac EAD (2023)

Após inserir o texto do **README.md**, é possível salvar o arquivo e voltar para o GitHub Desktop.

Ao voltar, você verá que as alterações feitas no **README.md** já foram apontadas e as diferenças já foram mostradas.

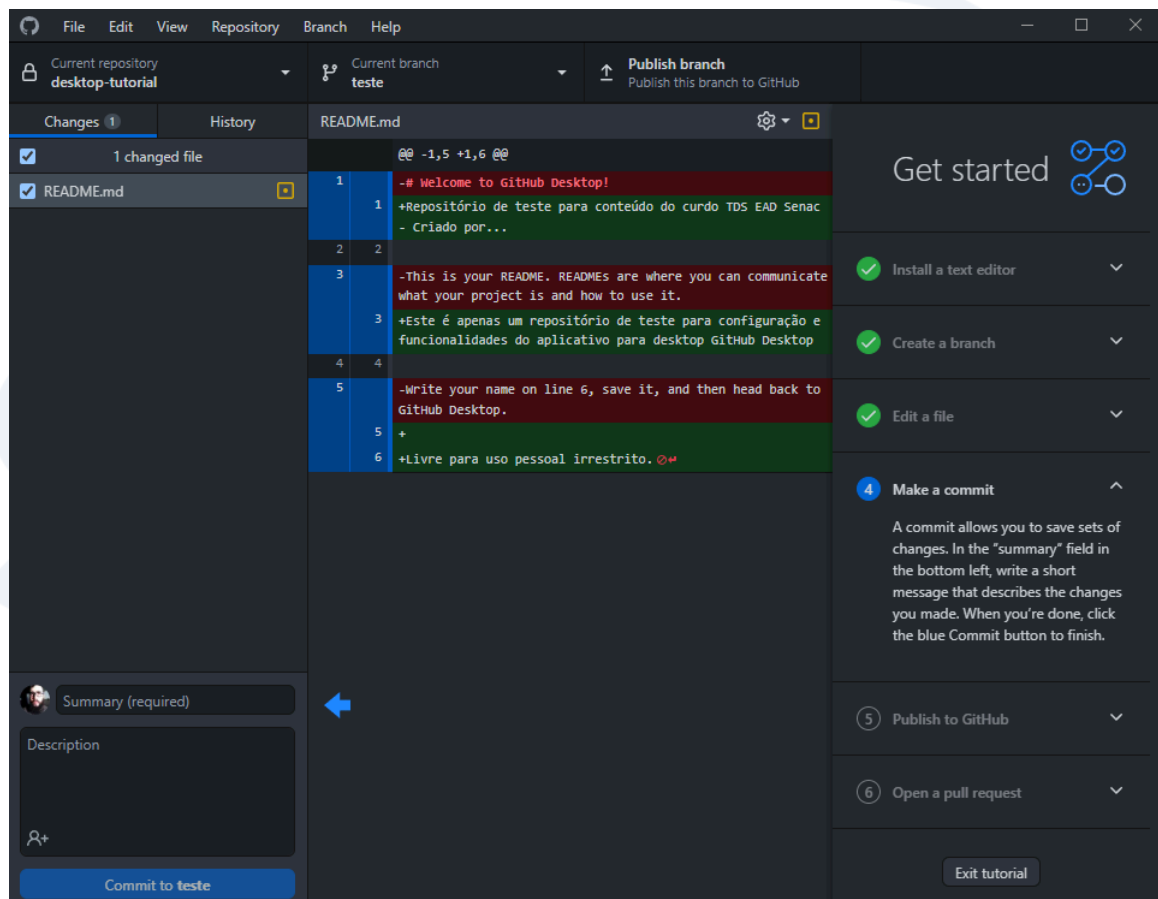


Figura 23 – Diferenças a serem aplicadas com o **commit**, mostrando as linhas que foram apagadas, em vermelho, e as que serão adicionadas, em verde
Fonte: GitHub Desktop (2023)

Como indicado, à esquerda, veja o arquivo que foi alterado, no caso o **README.md**. No centro, as linhas de código foram alteradas, no caso, foi apagado o **readme** padrão e o texto novo foi inserido.

Você está pronto para fazer o seu primeiro **commit**.

Para efetuar o **commit**, você deve escrever um resumo (*summary*) do que foi feito de alterações, incluir uma descrição e, daí sim, clicar em **Commit to teste**, em que “teste” é a *branch* que você criou e está selecionada.

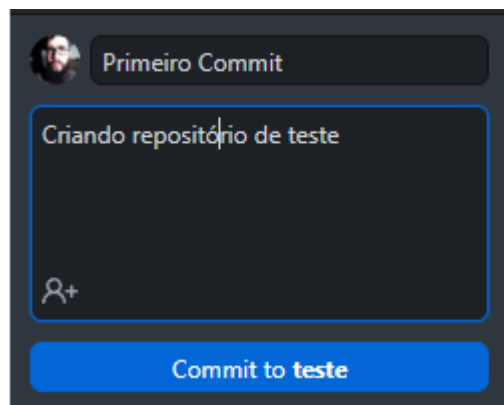


Figura 24 – Primeiro **commit**

Fonte: Senac EAD (2023)

Após isso, o primeiro **commit** foi feito e falta apenas o **push** para que seja publicado (*upload*) para a *branch* teste no repositório remoto do GitHub.

Para fazer o **push**, é possível usar o atalho **Ctrl + P** ou clicar onde indica a seta azul.

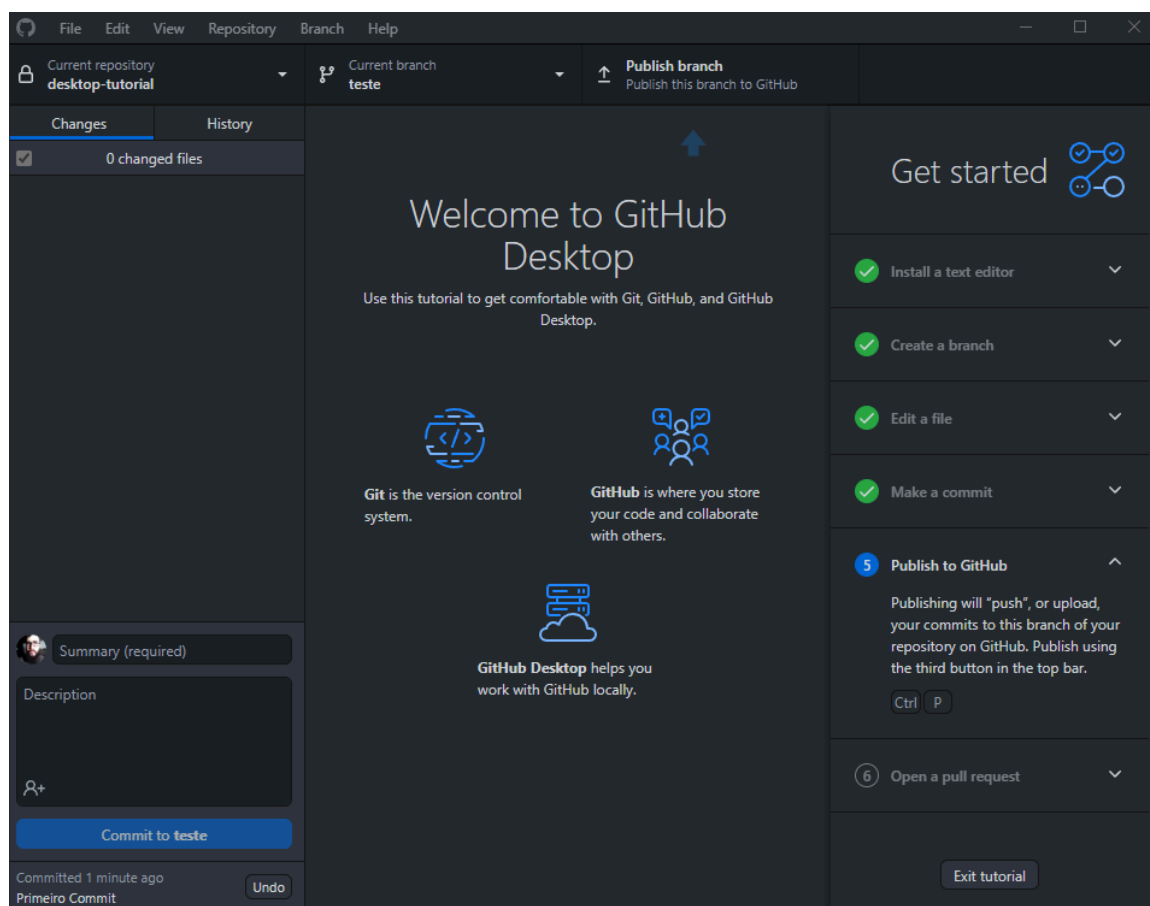


Figura 25 – Fazendo o primeiro **push**

Fonte: GitHub Desktop (2023)

Após isso, só resta solicitar um *pull request*, que significa que se está solicitando que a nossa *branch* seja mesclada com a *branch* principal.

Pode-se fazer isso clicando no botão **Open a pull request**, para abrir o *pull request* direto no *site* do GitHub.

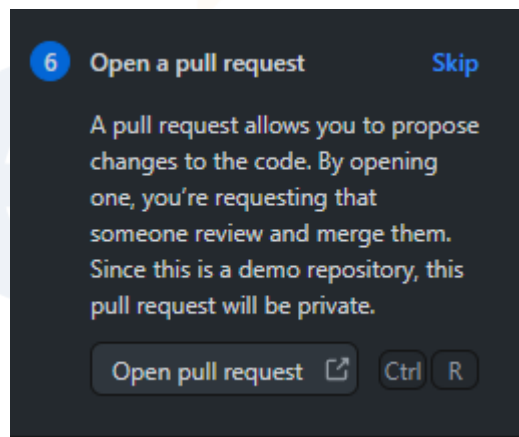


Figura 26 – Botão **Open a pull request**

Fonte: GitHub Desktop (2023)

Clicando no botão, você é levado diretamente para a página de solicitação de *pull request*.

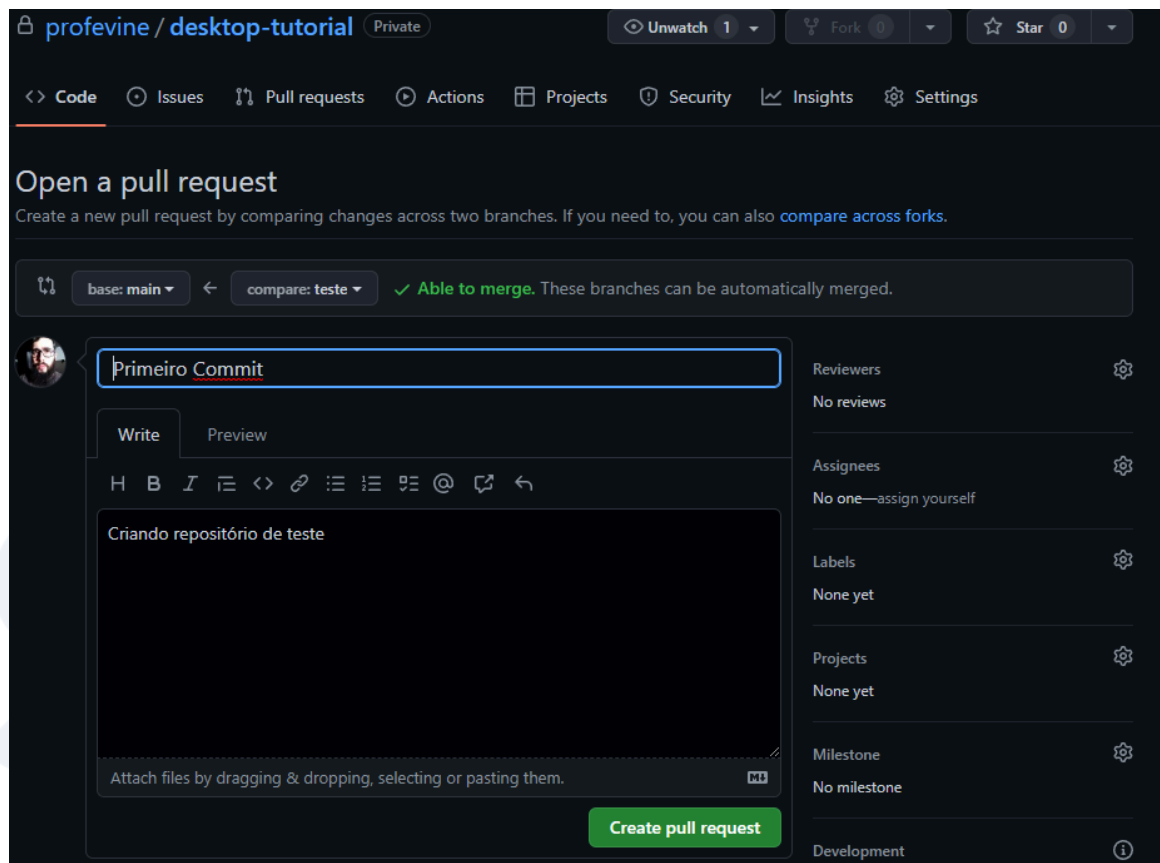


Figura 27 – Página do GitHub de abertura de um *pull request*
Fonte: GitHub (2023)

Ao clicar no botão verde **Create pull request**, veja que você criou uma linha do tempo onde nosso *pull* foi solicitado.

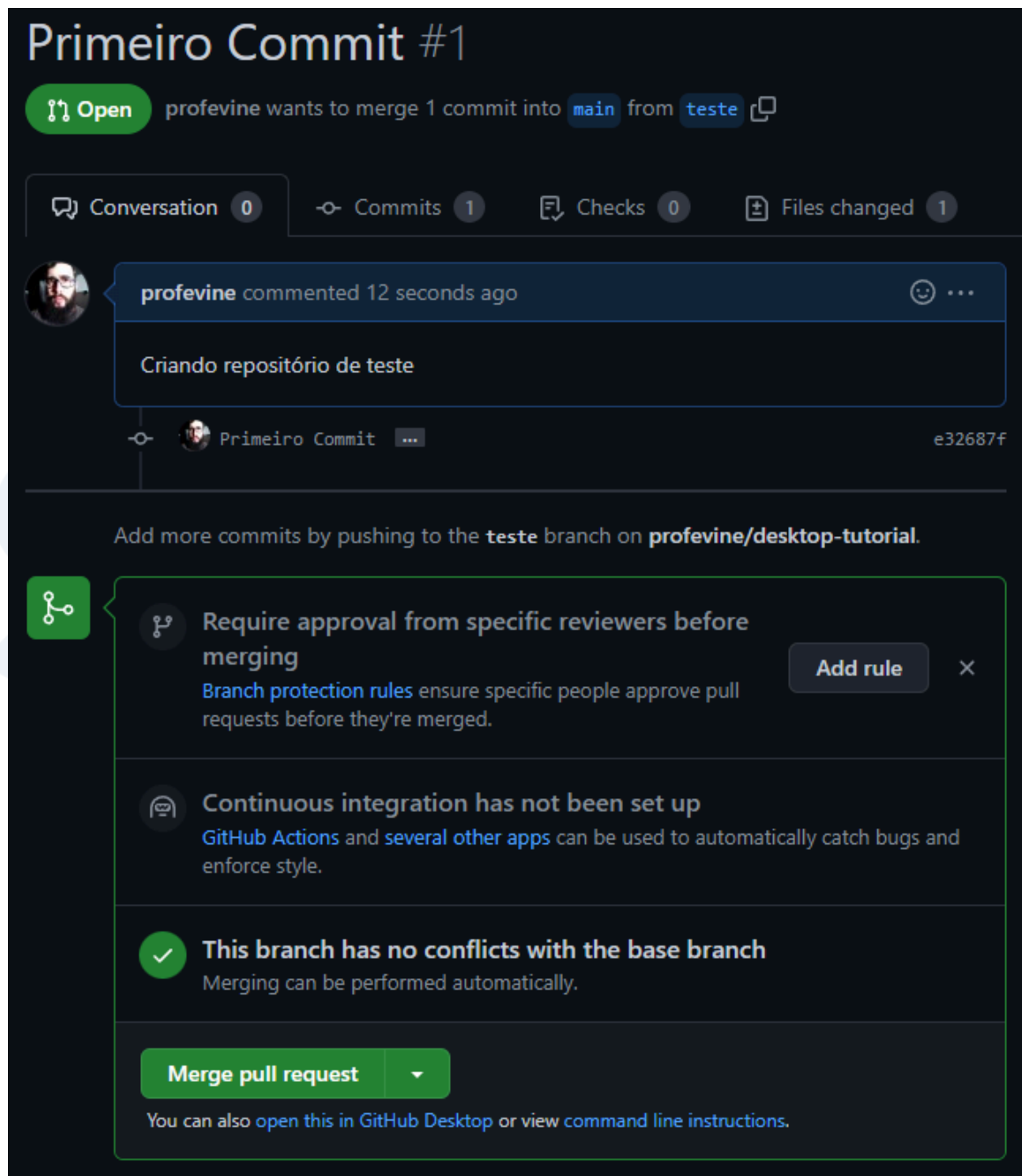


Figura 28 – Agora basta o responsável verificar e aceitar ou não a nossa *request*

Fonte: GitHub (2023)

Alternativa usando a linha de comando com o Git Bash

O GitHub Desktop é um aplicativo de interface gráfica do usuário (GUI) para usar o Git em seu computador, enquanto o Git Bash é um aplicativo de interface de linha de comando (CLI) para usar o Git. Ambos permitem que você use o Git para gerenciar seus repositórios de código, mas funcionam de maneiras diferentes.

O GitHub Desktop foi projetado para ser mais amigável e fácil de usar, principalmente para pessoas que são novas no Git ou que não se sentem confortáveis com o uso da linha de comando. Ele fornece uma interface visual para trabalhar com repositórios, permitindo que você execute tarefas comuns do Git, como confirmar alterações, ramificar e mesclar sem precisar usar a linha de comando.

O Git Bash, por outro lado, é um aplicativo de linha de comando que fornece uma interface *shell* para trabalhar com o Git. Ele permite que você use comandos Git e trabalhe com seus repositórios usando comandos baseados em texto em vez de uma interface gráfica. Isso pode ser mais eficiente para usuários experientes que se sentem confortáveis com a linha de comando, mas pode ser menos intuitivo para quem é novo no Git ou prefere uma interface mais visual.

No geral, a escolha entre o GitHub Desktop e o Git Bash depende das suas preferências pessoais e do nível de conforto com o uso da linha de comando. Ambos podem ser ferramentas úteis para trabalhar com o Git, e você pode usar qualquer um deles, dependendo do que funcione melhor para você.

Etapas para instalar o Git Bash

Para instalar o Git Bash, siga estas etapas:

1. Acesse o *site* do Git (<https://git-scm.com/downloads>) e clique no botão **Download**.

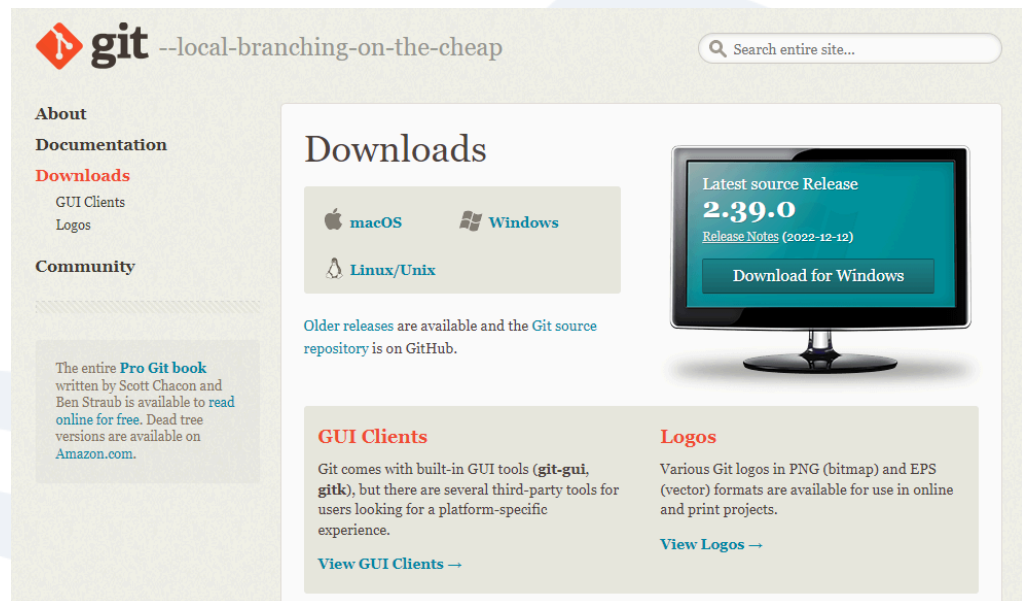


Figura 29 – Página de *downloads* do Git

Fonte: Git (2023)

2. Na página de *download*, clique no botão **Windows** para baixar a versão Windows do Git.
3. Quando o *download* estiver concluído, abra o arquivo de instalação baixado e siga as instruções para instalar o Git Bash no seu computador. Nessa etapa, você pode escolher criar um atalho na área de trabalho após a instalação.

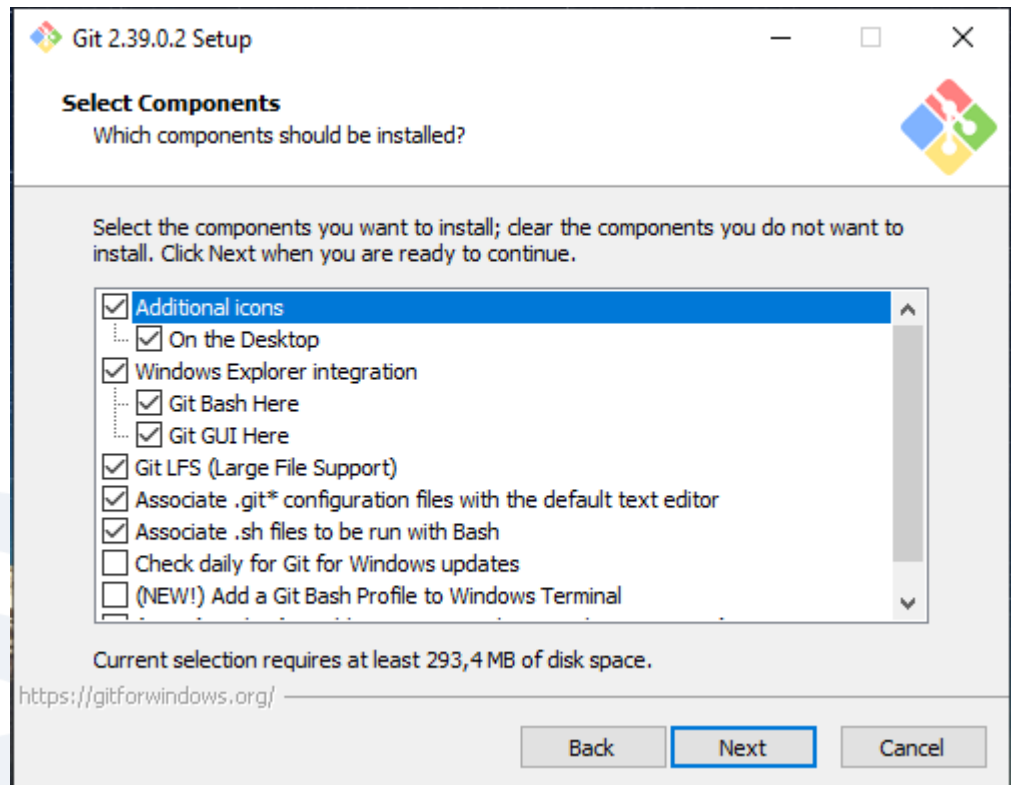


Figura 30 – Instalando o Git Bash e selecionando (em destaque) a criação de ícone de atalho na área de trabalho

Fonte: Git Bash (2023)

4. Nas próximas janelas, basta clicar em **Next** até a instalação ser concluída.

Para usar o Git Bash, você precisará abrir o aplicativo após a instalação. No Windows, você pode fazer isso procurando por “Git Bash” no menu **Iniciar** ou clicando no ícone do Git Bash na área de trabalho (se você escolheu criar um durante a instalação).

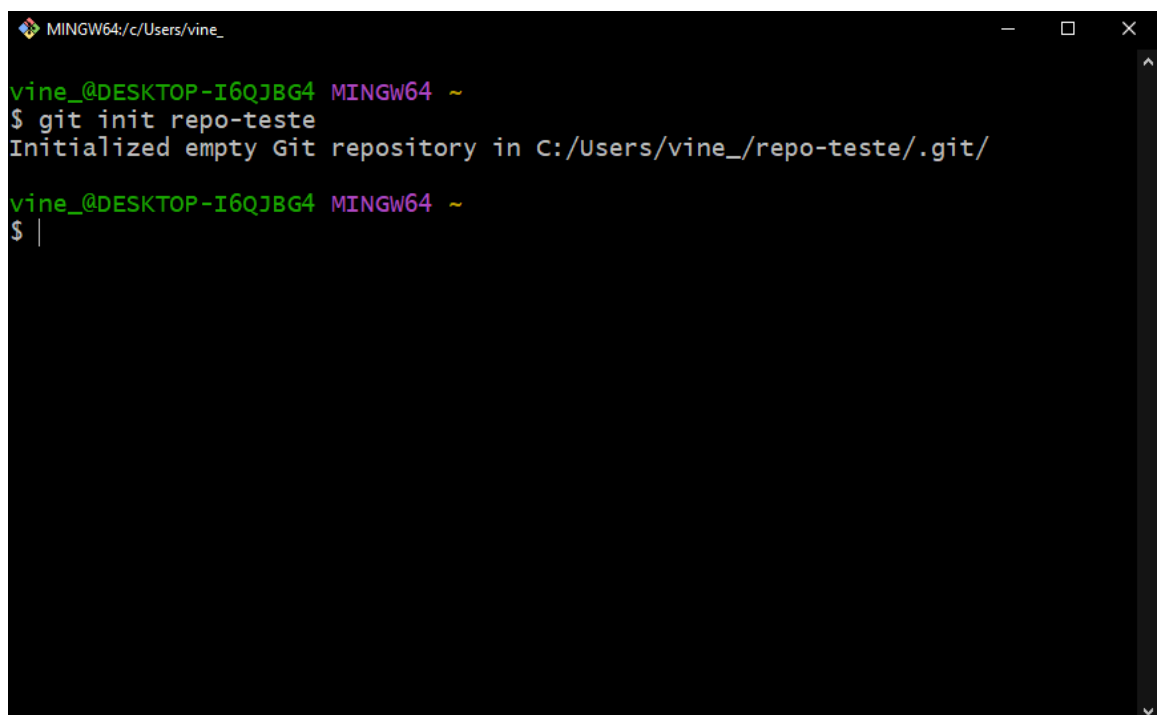
Quando o Git Bash estiver aberto, você verá uma janela de terminal. É aqui que você pode inserir comandos do Git e trabalhar com seus repositórios. Para começar, você pode tentar alguns comandos básicos do Git, como **git clone** para

clonar um repositório de um servidor remoto ou **git init** para criar um novo repositório. Você também pode usar o comando **git help** para obter uma lista de comandos disponíveis e consultar a documentação de ajuda para comandos específicos.

Criando um repositório local usando o Git Bash

Abra o Git Bash e navegue até o diretório onde deseja criar o repositório, usando o comando **cd**. Por exemplo, se você deseja criar o repositório em seu diretório de usuário, pode usar o comando **cd ~** para navegar até o seu diretório inicial.

Execute o comando **git init** para criar um novo repositório. Isso criará um novo diretório **.git** no diretório atual, que será usado para armazenar o histórico do repositório e outros metadados.

A screenshot of a Git Bash terminal window. The title bar at the top reads "MINGW64:/c/Users/vine_". The terminal shows the user "vine_@DESKTOP-I6QJBG4" in a "MINGW64" environment at the "~" (home) directory. The user enters the command "\$ git init repo-teste". The terminal responds with "Initialized empty Git repository in C:/Users/vine_/repo-teste/.git/". The user then enters a new command "\$", and the terminal shows a cursor at the end of the line, ready for input.

```
MINGW64:/c/Users/vine_
vine_@DESKTOP-I6QJBG4 MINGW64 ~
$ git init repo-teste
Initialized empty Git repository in C:/Users/vine_/repo-teste/.git/
vine_@DESKTOP-I6QJBG4 MINGW64 ~
$ |
```

Figura 31 – Criando um repositório local chamado de **repo-teste**

Fonte: Git Bash (2023)

Para criar um novo arquivo no repositório, use um editor de texto ou o comando **echo** (comando usado para escrever texto na tela) para criar um novo arquivo e salve-o no diretório do repositório. Por exemplo, você pode usar o comando **echo "Hello, world!" > hello.txt**. Esse comando criará um novo arquivo chamado **hello.txt** com o texto "Hello, world!"

Para adicionar o arquivo ao repositório, use o comando **git add** seguido do nome do arquivo. Por exemplo, para adicionar o arquivo **hello.txt**, use o comando **git add hello.txt**. Isso preparará o arquivo, o que significa que ele estará pronto para ser enviado ao repositório.

Para confirmar o arquivo no repositório, use o comando **git commit**. Isso abrirá um editor de texto em que você pode inserir uma mensagem de confirmação explicando as alterações feitas. Quando terminar, salve a mensagem e saia do editor de texto. O arquivo será confirmado no repositório e adicionado ao histórico do repositório.

É isso! Você criou um novo repositório e adicionou um arquivo a ele usando o Git Bash.

Mais detalhes quanto ao uso do Git Bash você verá na sequência dos conteúdos desta unidade curricular.

Vantagens e desvantagens

Aqui estão alguns prós e alguns contras do uso do Git:

Clique ou toque para visualizar o conteúdo.

Prós

- ◆ O Git permite que você acompanhe as alterações em seus arquivos ao longo do tempo, para que você possa ver como o seu projeto evoluiu e reverter para versões anteriores, se necessário.
- ◆ O Git facilita a colaboração com outras pessoas no mesmo projeto. Você pode compartilhar seu repositório com outras pessoas e elas podem contribuir com suas próprias alterações e melhorias.
- ◆ O Git é amplamente usado e bem suportado, então você pode encontrar muitos recursos e documentação *on-line* para ajudá-lo a aprender e usá-lo de forma eficaz.

Contras

- ◆ O Git pode ter uma curva de aprendizado íngreme, especialmente se você for novo no controle de versão. Pode levar algum tempo e precisar de prática para se tornar proficiente com o Git.
- ◆ O Git exige que você use a linha de comando ou o terminal, o que pode ser intimidador para alguns usuários.
- ◆ O Git pode ser complexo, com muitos comandos e opções diferentes. Pode ser fácil cometer erros ou fazer as coisas da maneira errada se você não estiver familiarizado com o funcionamento.

Em geral, o Git é uma ferramenta poderosa e útil para gerenciar e colaborar em projetos, mas pode não ser a melhor escolha para todos. É importante pesar os prós e os contras e decidir se é a ferramenta certa para as suas necessidades.

Encerramento

Com a utilização dessa ferramenta, é possível agilizar e controlar códigos enviados por diversas pessoas para um mesmo repositório remoto.

Você viu, neste conteúdo, que pode hospedar os seus repositórios de diversas maneiras, tanto local quanto remotamente.

Além disso, você também aprendeu como instalar, configurar e subir o seu primeiro repositório local para o repositório remoto no GitHub usando o aplicativo oficial GitHub Desktop.



Senac