

VERSIONAMENTO

SENAI

SUMÁRIO

PARA COMEÇAR	3
SITUAÇÃO-PROBLEMA	4
DESAFIO 1.....	5
VERSÕES DE UM ARQUIVO	5
VERSIONAMENTO DE CÓDIGO	6
GIT.....	8
INSTALAÇÃO DO GIT NO WINDOWS	9
CONFIGURAÇÕES INICIAIS	9
NESTE DESAFIO.....	11
DESAFIO 2 - PARTE 1	12
ESTRUTURA DIGITAL	13
TERMOS E COMANDOS.....	13
criando a estrutura inicial	15
criando um repositório git local	16
comandos iniciais.....	18
criando um repositório online	26
DESAFIO 2 - PARTE 2	30
PUBLICANDO NO REPOSITÓRIO ONLINE	30
FLUXO DE TRABALHO	36
NESTE DESAFIO.....	44
DESAFIO 3.....	45
TRABALHO DE EQUIPE	46
TRUNK E BRANCH.....	46
CLONANDO UM REPOSITÓRIO	47
criando uma branch	50
criando outras branchs.....	55
UNIFICANDO OS RAMOS.....	59
RESOLVENDO CONFLITOS	60
DEMARCANDO PONTOS NO CÓDIGO: TAGS.....	64
NESTE DESAFIO.....	68
PARA CONCLUIR.....	68
REFERÊNCIAS	69
CRÉDITOS.....	71

PARA COMEÇAR

Esta etapa aborda conceitos relacionados às Metodologias para Controle de Versões em Projetos Web.

No decorrer de seus estudos, esperamos que você desenvolva as seguintes capacidades:

- Visão sistêmica de compartilhamento e evolução de códigos.
- Reconhecimento do sequencial de versões de códigos.
- Utilização de ferramentas de versionamento em projeto de TI.

Para desenvolver tais capacidades, você deverá estudar os seguintes temas:

- Metodologias de versionamento.
- Ferramentas disponíveis para gerenciamento de versões: instalação, configuração, repositórios, versionamento, alterações, correção de erros, *Branchs* e *tags*, arquivos, repositório remoto e boas práticas de versionamento.

Esse estudo será necessário para que você resolva a situação-problema a seguir. Então, avance para conhecê-la.

SITUAÇÃO-PROBLEMA

Você foi contratado como programador por uma empresa de desenvolvimento de software, chamada Hroad, que está realizando alterações no fluxo de trabalho de seus programadores, a fim de manter a integridade do código-fonte de seus softwares/solução.

Um dos atuais problemas é a necessidade de alteração de um código-fonte por diversos programadores. Quando duas ou mais pessoas precisam alterar o mesmo arquivo, muitos códigos são perdidos e sem a identificação das alterações que foram feitas por seus desenvolvedores. Além disso, quando uma alteração é feita por um desenvolvedor(a), ela fica difícil de ser rastreada.

Dado o contexto acima, seu coordenador lhe propôs a busca de soluções em que a equipe possa fazer o trabalho simultâneo no mesmo código-fonte sem quaisquer perdas ou intervenções. E, se possível, que o histórico das alterações seja facilmente rastreado.

Para que não haja atraso em nenhuma etapa, seu superior lhe atribuiu os seguintes desafios:

Desafio 1

Instalar um software que tem a finalidade de gerenciar diferentes versões — histórico e desenvolvimento — dos códigos-fontes.

Desafio 2

Realizar todas as configurações e demais ações necessárias à prática de monitoramento e gerenciamento de alterações no código.

Desafio 3

Realizar a conciliação de alterações em um mesmo repositório.

DESAFIO 1

Nesta etapa, você deverá resolver o desafio 1:

- Instalar um software que tem a finalidade de gerenciar diferentes versões — histórico e desenvolvimento — dos códigos-fontes.

Para isso, você estudará os seguintes conteúdos:

- Metodologias de versionamento.



VERSÕES DE UM ARQUIVO

Uma das vantagens de se trabalhar com arquivos digitais é a facilidade de alterá-los. Para manter o histórico dessas atualizações, é possível criar versões do arquivo original. Mas como manter o controle dessas versões ao longo do desenvolvimento do projeto?

VERSIONAMENTO DE CÓDIGO

Quando um novo projeto baseado em programação é criado, seja um software, plataforma ou site, é comum, mesmo após a entrega do projeto, a atualização de versões, com correção de *bugs*¹, adição de novas ferramentas, entre outros.

O versionamento de código é realizado por um sistema de controle de versões.

Um sistema de controle de versão armazena um conjunto de arquivos ou, como é comum dizer no mundo do desenvolvimento, o *código-fonte*² do projeto (independentemente da linguagem de programação utilizada).

Ele oferece uma maneira eficiente e concisa de gerenciar as alterações de um mesmo arquivo. Cada membro pode visualizar o histórico de todas as modificações feitas pela equipe, incluindo a data de alteração e qual ponto do arquivo foi editado. O time pode atuar de maneira colaborativa, sem conflitos e sem sobreposição das alterações.

E se, porventura, você estiver desenvolvendo um determinado requisito de seu projeto e ele falhar, você pode voltar os seus arquivos a um ponto anterior em que seu projeto esteja estável.

Sendo assim, não é necessário criar múltiplas versões do mesmo arquivo e todos podem usar um único, simplificando o trabalho.

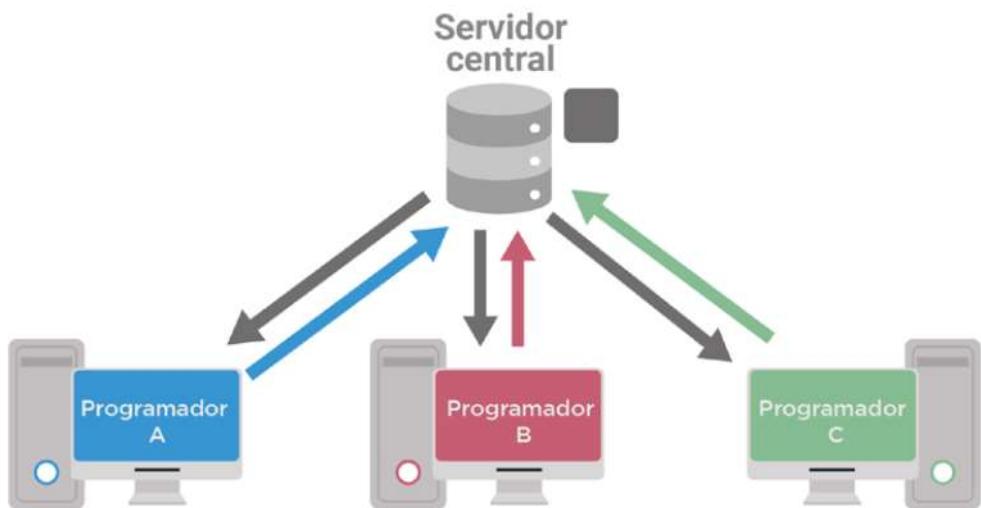
Podemos classificar os sistemas de controle em centralizados e distribuídos.

No **Sistema de Controle de Versão Centralizado (CVCS)**, apenas um único local armazena o histórico completo de alterações e, portanto, as estações de trabalho devem passar primeiro pelo servidor central para comunicarem-se.

¹ Falhas no software ou hardware.

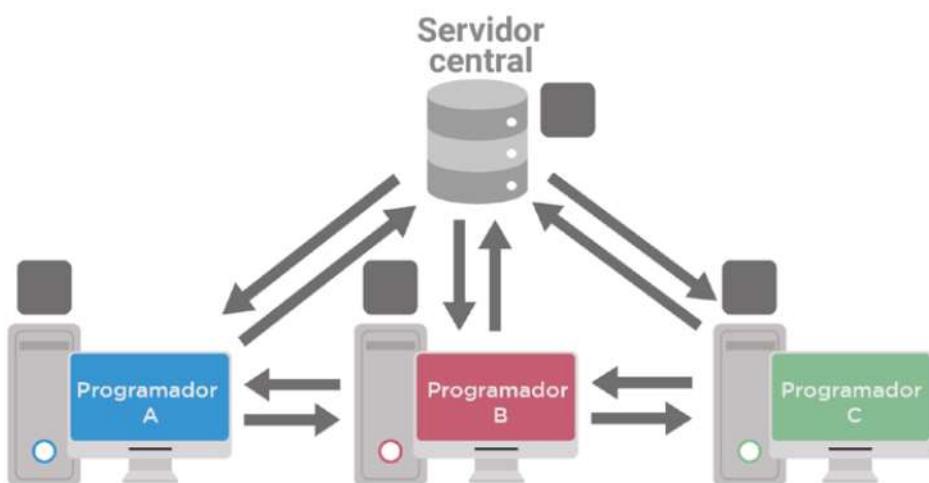
² Linhas de programação desenvolvidas em um software em sua forma original. São as instruções que geram o resultado para o usuário, como uma receita de bolo.

$$\text{Blue Box} + \text{Red Box} + \text{Green Box} = \text{Grey Box}$$



No **Sistema de Controle de Versão Distribuído (DVCS)** há uma cópia de todo o trabalho desenvolvido com o histórico completo de todas as alterações feitas na máquina de cada desenvolvedor. Sua vantagem é que, caso ocorra alguma perda na parte central, basta realizar o backup de uma máquina de algum desenvolvedor ou membro da equipe.

$$\text{Blue Box} + \text{Red Box} + \text{Green Box} = \text{Grey Box}$$



GIT

O Git é um Sistema de Controle de Versão Distribuído, portanto todos os programadores que estão utilizando-o têm uma cópia de todo o trabalho em suas máquinas locais. A cada alteração realizada pelo programador, seu histórico é mantido em sua máquina local e também no servidor central.

Além de ser um DVCS, o Git é também um **open source** (código aberto), ou seja, é um software que qualquer pessoa pode baixar, usar e modificar o código.

O sistema Git possibilita:

- o registro de todo o histórico de alterações realizadas no código de um projeto e de quem realizou a mudança;
- o retorno para versões anteriores e acesso ao código como estava antes de ser alterado;
- o trabalho de vários programadores simultaneamente, no mesmo arquivo;
- a restauração de um código removido ou modificado.

Você sabia?

O Git e o GitHub foram desenvolvidos por Linus Torvalds, o criador do sistema Linux.



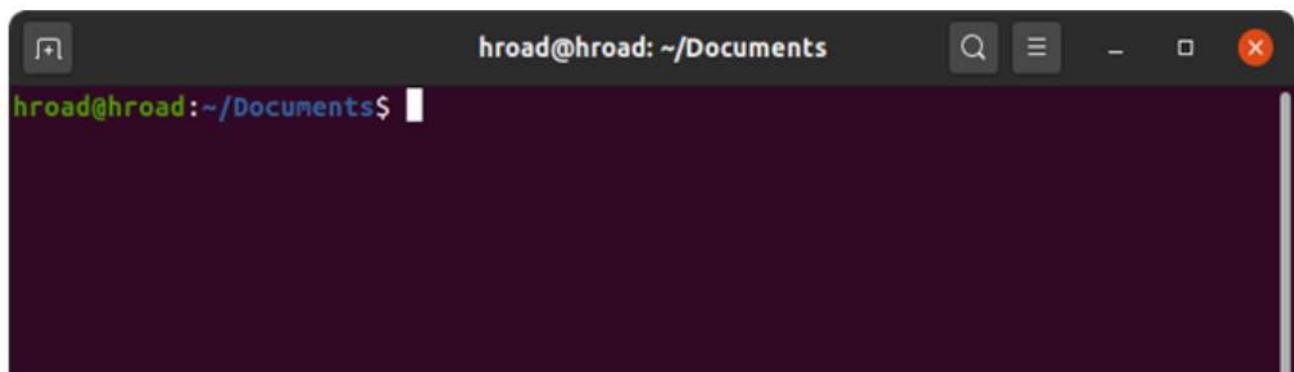
Saiba mais

Estude mais sobre sistemas centralizados e distribuídos acessando o link: <https://www.infoescola.com/informatica/sistema-de-informacao-centralizado/>

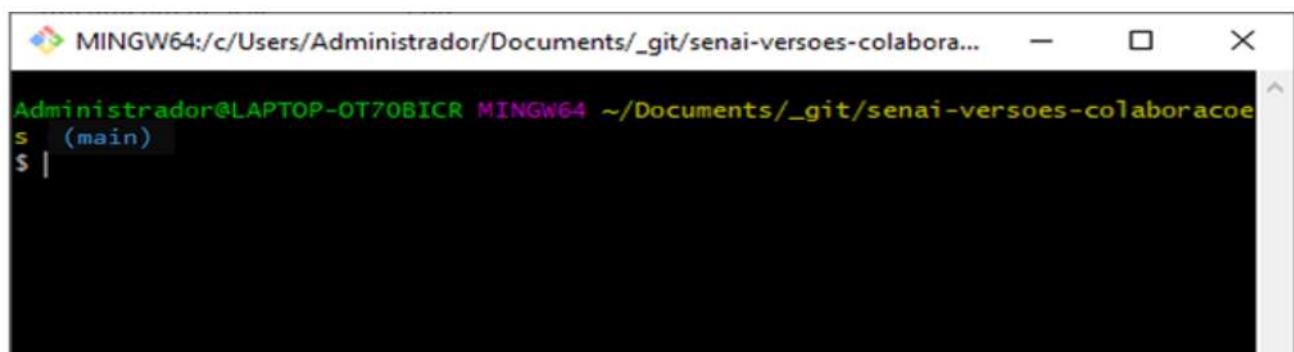


INSTALAÇÃO DO GIT NO WINDOWS

O Git foi projetado para ambientes estilo Unix, como os sistemas operacionais Linux e macOS. Para o sistema Microsoft Windows, é necessário instalar o Git Bash que *emula*³ o terminal (ou *prompt*⁴) do Git.



Git no Linux



Git no Windows

CONFIGURAÇÕES INICIAIS

Depois de instalado o Git, é hora de realizar as configurações iniciais.

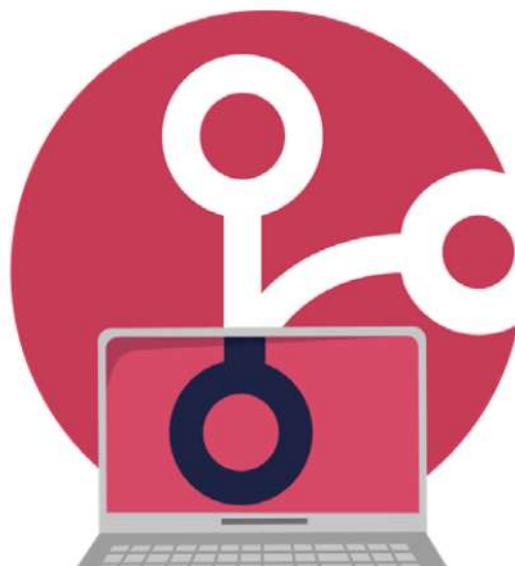
³ Na computação, um emulador é um software que reproduz as funções de um determinado ambiente.

⁴ Janela na qual são digitadas as linhas de comando.

A primeira configuração é identificar o nome de usuário e um endereço de e-mail. Assim, todas as atualizações (ou *commits*) feitas poderão ser identificadas.

Essa configuração pode ser global (válida para todo o sistema) ou local (para um projeto específico).

Fazendo a configuração global, não é necessário configurar cada projeto. Se a configuração local precisa ser alterada para um projeto específico, você pode rodar o comando sem a configuração global neste projeto.



Para esta configuração, utilizaremos o comando **git config**.

- Abra um terminal (ou *prompt* de comando) em sua máquina. Digite **git config** e depois digite seu nome de usuário e seu e-mail, como na imagem a seguir, e tecle Enter.

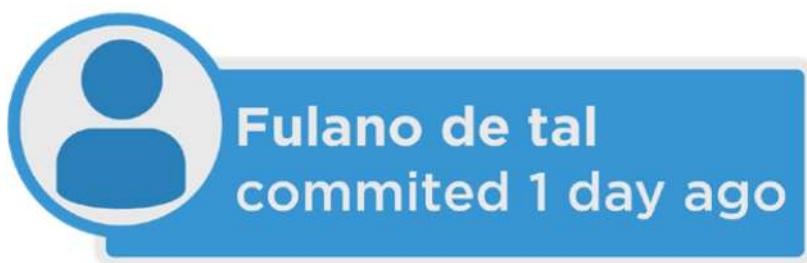
```
MINGW64:/c/Users/Administrador/Documents/_git/senai-versoes-colaboracoes (main)
$ git config --global user.name "fulano de tal"
Administrator@LAPTOP-OT70BICR MINGW64 ~/Documents/_git/senai-versoes-colaboracoes (main)
$ git config --global user.email "fulano@email.com"
Administrator@LAPTOP-OT70BICR MINGW64 ~/Documents/_git/senai-versoes-colaboracoes (main)
$ |
```

Importante!

Esses dois passos atualizarão suas informações de usuário e cada alteração feita manterá sua identidade como autor.



Posteriormente, seu resultado será algo semelhante a isto.



NESTE DESAFIO...

VERSIONAMENTO | DESAFIO 1



Você estudou que o processo de programação de qualquer sistema pode gerar uma grande quantidade de códigos e milhares de arquivos. Por isso, é imprescindível a utilização de uma ferramenta de versionamento que garanta o controle de cada versão do desenvolvimento de códigos, de modo que vários programadores possam realizar alterações simultaneamente e o

autor da alteração também seja identificado.

Uma dessas ferramentas é o Git, um sistema gratuito para controle de versão de código, que registra todo o histórico de alterações e seus respectivos autores, permitindo o acesso a versões anteriores, o trabalho simultâneo em um mesmo arquivo, além da restauração de um código removido ou modificado.

NO PRÓXIMO DESAFIO...

...você conhecerá como realizar todas as configurações e demais ações necessárias para a prática de monitoramento e gerenciamento de alterações no código.

DESAFIO 2 - PARTE 1

No desafio 1, você viu que o Git é um sistema de controle de versão, fez a sua instalação e a configuração inicial necessária para o seu funcionamento. Daremos início agora à jornada para utilizá-lo na prática.

Nesta etapa, você desenvolverá o desafio 2:

- Realizará todas as configurações e demais ações necessárias, para a prática de monitoramento e gerenciamento de alterações no código.

Para facilitar o seu entendimento, dividiremos os conhecimentos a seguir em duas partes:

Na primeira parte, você estudará:

- A estrutura e organização de arquivos;
- Termos e comandos básicos para criar arquivos, rastrear alterações, adicionar versões, verificar e salvar alterações;
- Criação de repositórios locais e remotos (online).

Na segunda parte, você conhecerá:

- Publicação de repositório online;
- Criação de ramificações (*branches*);
- Como ignorar alterações.



ESTRUTURA DIGITAL



Antes de mais nada, é necessário organizar a estrutura digital para o projeto.

TERMOS E COMANDOS

Agora vamos abordar outros termos típicos da área para que você comece a se familiarizar.

Prompt

É a janelinha na qual digitamos os comandos, também chamada de terminal.

Repositório local

Pasta ou diretório na máquina do programador, na qual são gravadas todas as versões do código.

Repositório remoto

Pasta ou diretório na nuvem, na qual é gravado o projeto completo, com todas as versões do código.

Staging

Local intermediário entre a máquina do desenvolvedor e o repositório remoto, para revisão de alterações antes de salvar no histórico.

Commit

Ato de registrar ou salvar as alterações.

Ramo principal

Sequência principal do código-fonte.

Branch

É uma ramificação do ramo principal, para testes em geral.

README

Arquivo que é uma mescla de cartão de visitas e documentação do projeto.

.gitignore

Arquivo de texto que guarda os arquivos ou pastas que devem ser ignorados, ou seja, os arquivos nomeados no .gitignore não serão monitorados pelo Git.

Vamos abordar também alguns comandos, como:

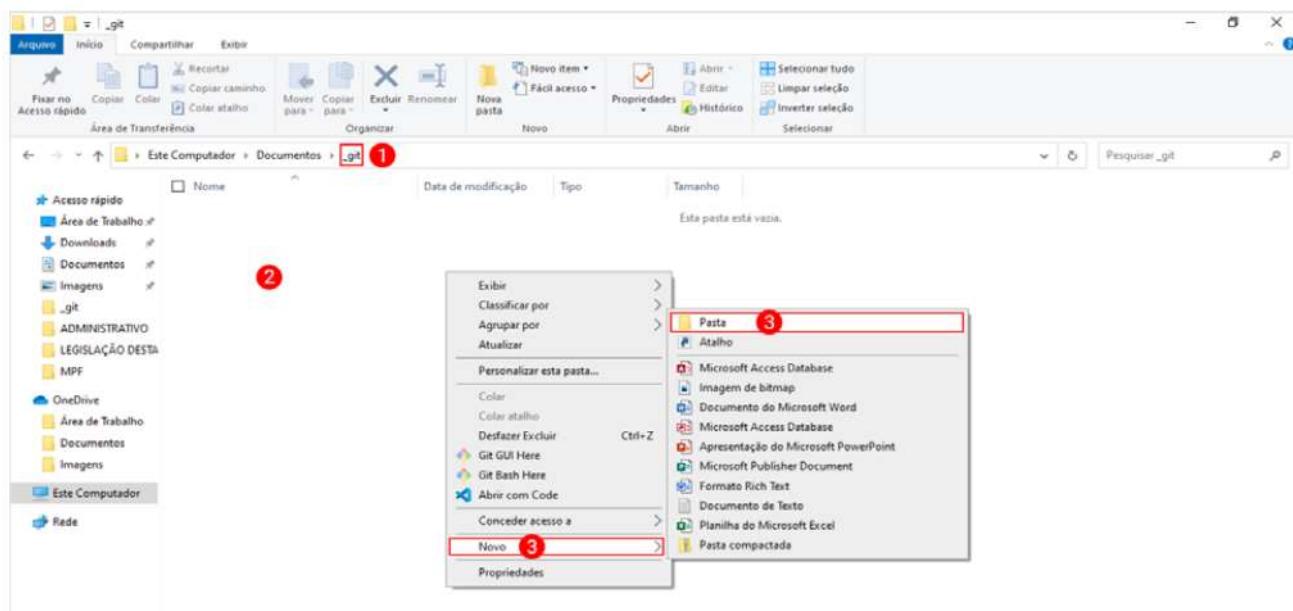
git init	Cria a estrutura inicial do repositório Git no computador local.
git status	Verifica o status das alterações realizadas no repositório.
git add	Adiciona arquivos ao histórico do projeto, na <i>staging</i> .
git commit	Registra/salva a alteração no repositório.
git log	Permite visualizar as alterações feitas.
git show número-do-commit	Permite visualizar informações sobre qualquer commit.
git remote add origin "destino"	Informa a pasta remota.
git remote -v	Permite visualizar o repositório remoto.
git push -u origin master	Publica as alterações no repositório remoto.
git pull	Baixa as alterações no repositório remoto.

CRIANDO A ESTRUTURA INICIAL

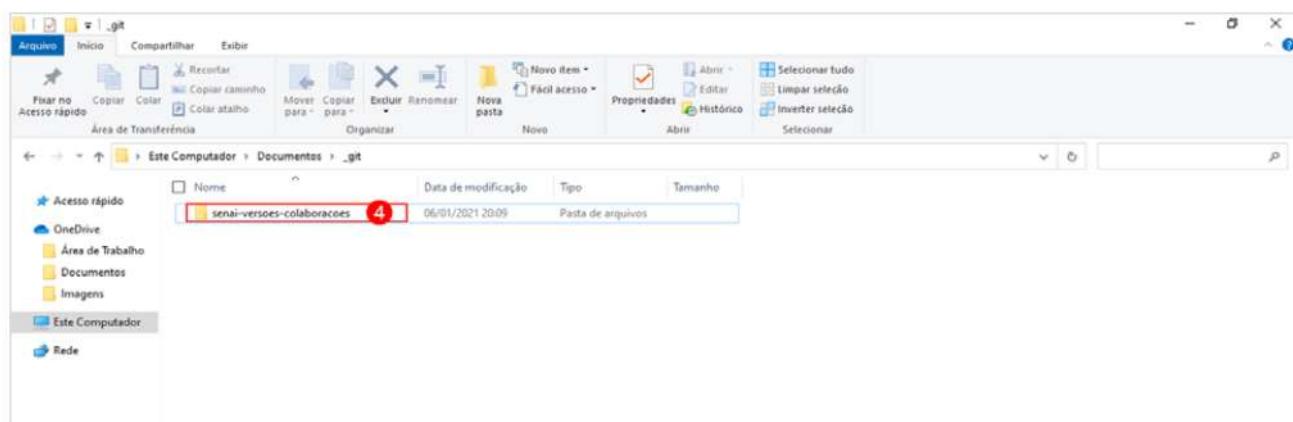
Para resolver o desafio proposto (realizar as configurações para o monitoramento e gerenciamento de alterações no código) usando o Git, o primeiro passo é criar o *repositório*⁵ local.

Vamos criar uma pasta chamada **senai-versoes-colaboracoes**.

1. Acesse uma pasta de sua preferência em seu computador (no exemplo, é a "**_git**");
2. Clique na área em branco dentro da pasta "**_git**" com o botão direito do mouse;
3. Com o botão esquerdo do mouse, clique em "**Novo**" / "**Pasta**".

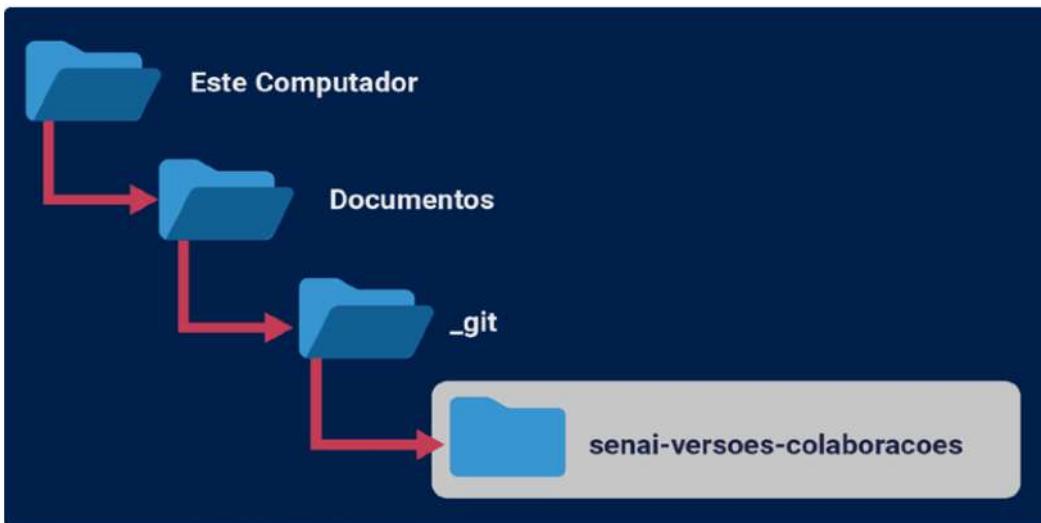


4. Nomeie como "**senai-versoes-colaboracoes**".



⁵ Local de armazenamento virtual, também conhecido por pasta ou diretório, para projetos.

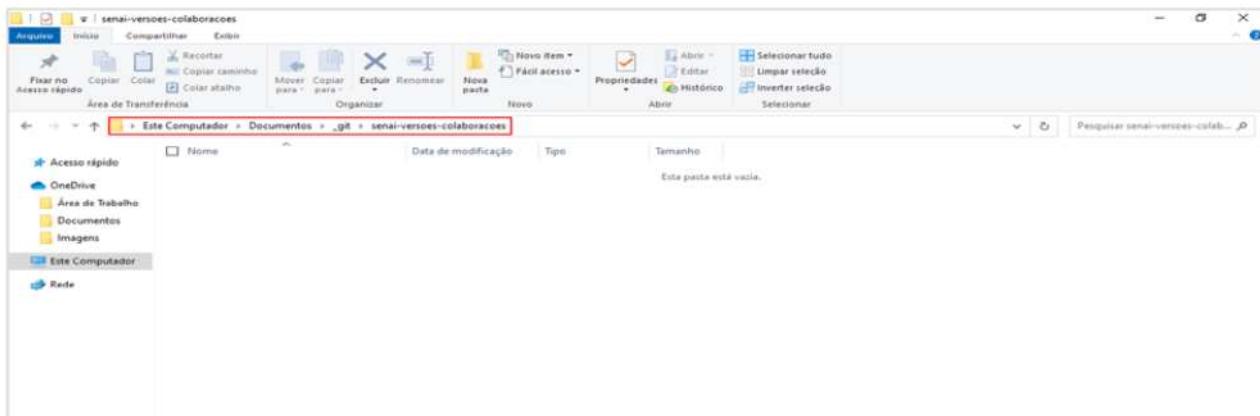
5. Nesta pasta "**senai-versoes-colaboracoes**", serão feitos o rastreamento de suas atividades, a criação dos arquivos e também a solução desse desafio.



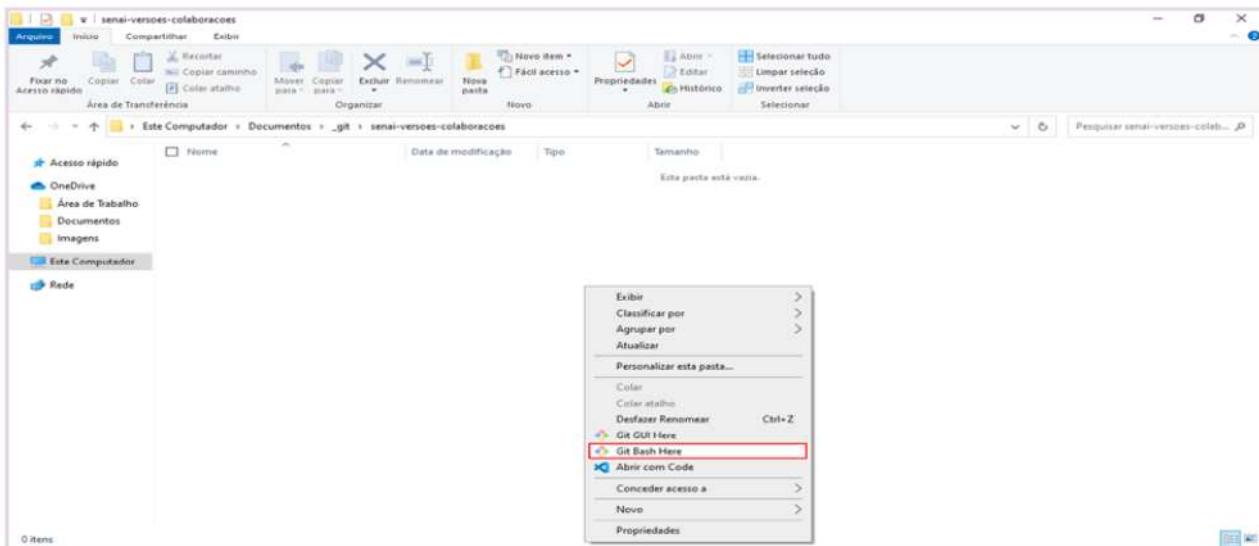
CRIANDO UM REPOSITÓRIO GIT LOCAL

Um repositório Git é um armazenamento virtual para os projetos, no qual serão gravadas as versões do código e que você poderá sempre acessar quando necessário. O repositório local é a pasta/diretório da sua máquina com o seu projeto.

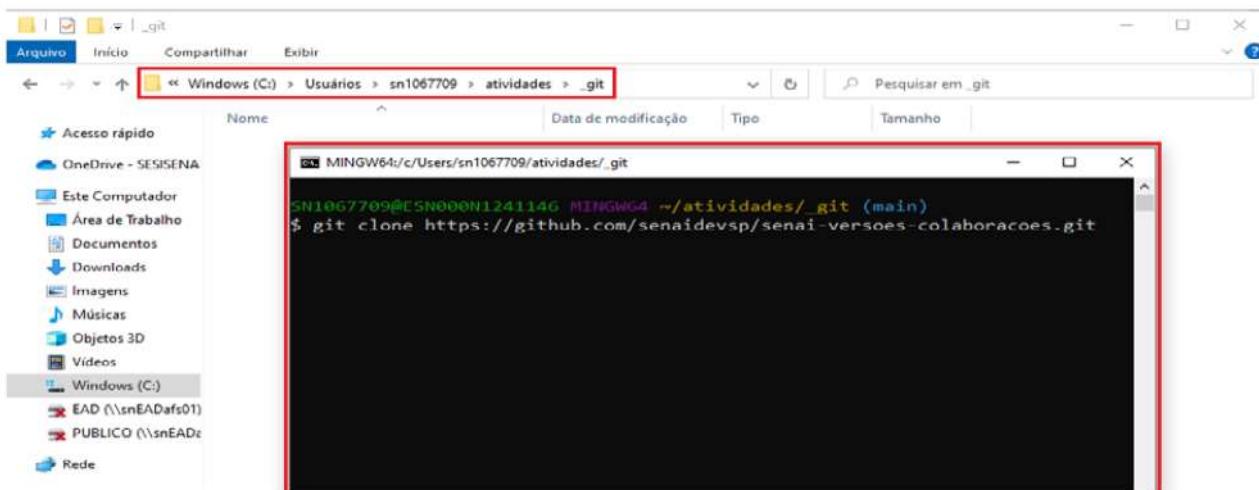
- Para criar o repositório Git local, dê um duplo clique na pasta "**senai-versoes-colaboracoes**" criada anteriormente para acessá-la.



- Clique na área em branco dentro da pasta "**senai-versoes-colaboracoes**" com o botão direito do mouse e inicialize o Git Bash, clicando em "**Git Bash Here**".



- O *prompt*⁶ abrirá e nele você vai digitar todos os comandos.



Você pode usar o *prompt* de comando do seu sistema operacional para digitar os comandos Git. Por estarmos usando o Windows como exemplo de sistema operacional, usaremos o Git Bash, que emula o terminal do Git, e digitaremos todos os comandos no terminal do Git Bash.

⁶ Também chamado de "terminal" ou popularmente de "janela", é o local onde são digitados os comandos.

COMANDOS INICIAIS

Aqui abordaremos alguns comandos básicos para começar a trabalhar com o Git.



Os próximos passos serão:

- iniciar o repositório - comando **git init**;
- criar um arquivo de texto dentro do repositório;
- rastrear alterações no repositório - comando **git status**;
- adicionar versão no repositório - comando **git add**;
- verificar alterações antes de salvar - comando **git status**;

• salvar alterações - comando **git commit**; e

• visualizar log da alteração - comando **git log**.

INICIAR O REPOSITÓRIO: COMANDO *git init*

O primeiro comando que executaremos é o **git init**, que vai inicializar o repositório local em nossa pasta `senai-versoes-colaboracoes`.

Ao ser executado, esse comando cria **uma pasta chamada .git, com subdiretórios (objects, refs/heads, refs/tags)**, e transforma o diretório atual em um repositório do Git. Na imagem a seguir, o terminal informa que o repositório está vazio, pois nenhum arquivo foi incluído.

```
MINGW64:/c/Users/sn1067709/senai-versoes-colaboracoes
SN1067709@ESN000N1241146 MINGW64 ~/senai-versoes-colaboracoes (main)
$ git init
Initialized empty Git repository in C:/Users/sn1067709/senai-versoes-colaboracoes/.git/
SN1067709@ESN000N1241146 MINGW64 ~/senai-versoes-colaboracoes (main)
$ -
```

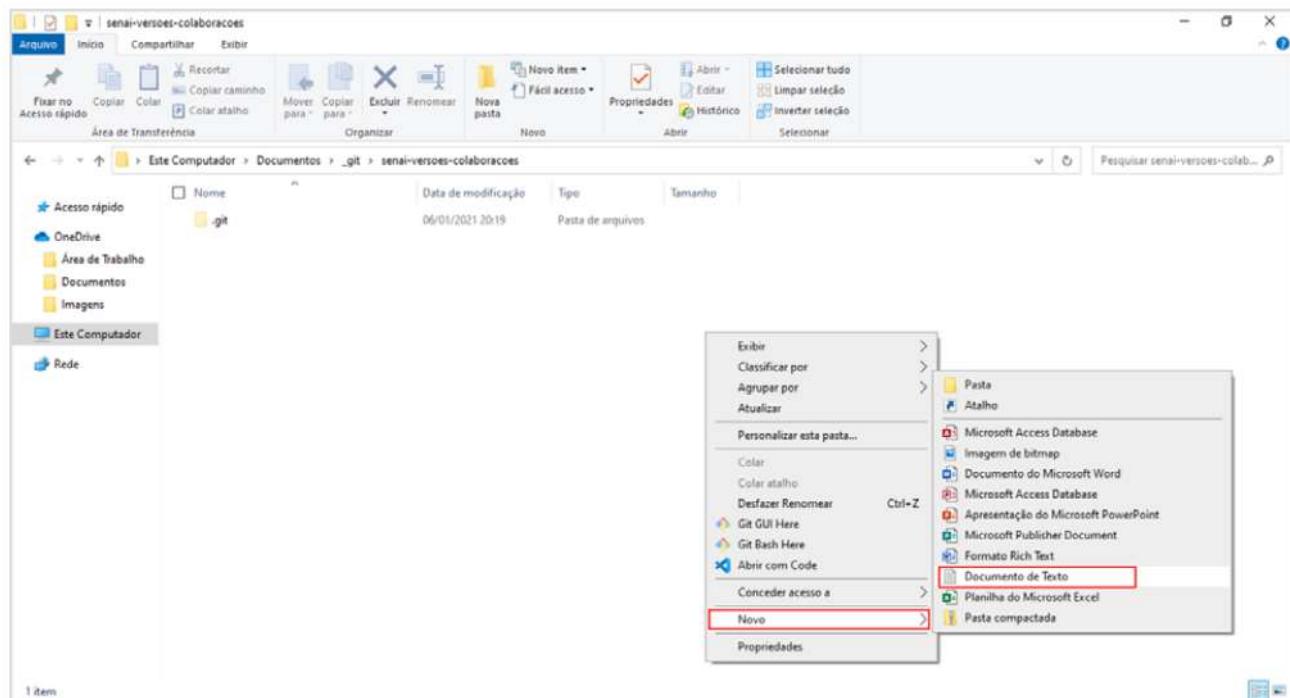
Agora, a pasta `senai-versoes-colaboracoes` é o repositório local e, dentro dela, há outra pasta chamada `.git`, criada com o comando **git init**.



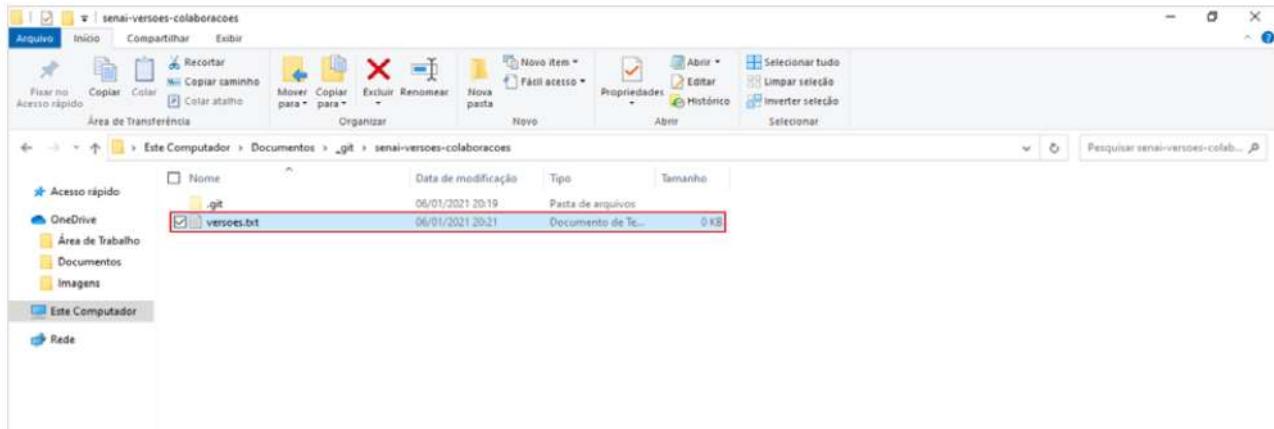
CRIAR UM ARQUIVO DE TEXTO DENTRO DO REPOSITÓRIO

Antes do próximo comando (rastrear alterações), devemos criar um arquivo para ser alterado e rastreado.

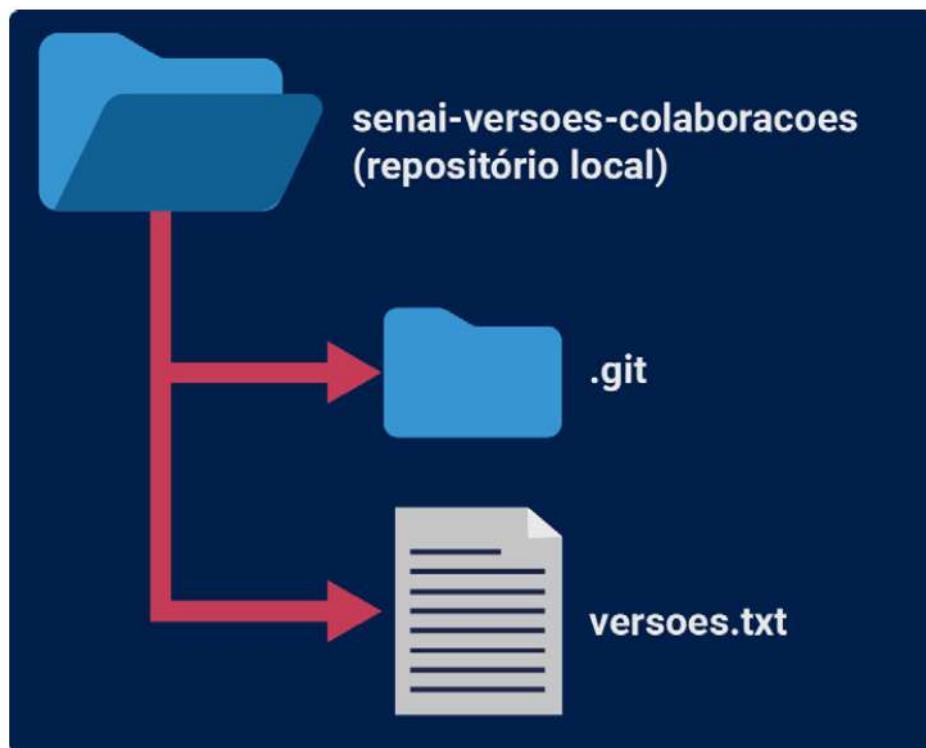
- Dentro da pasta "**senai-versoes-colaboracoes**", crie um arquivo de texto, chamado "**versoes**", com a extensão `".txt"`. Para criar esse arquivo, clique na área em branco da pasta "**senai-versoes-colaboracoes**" com o botão direito do mouse e, então, clique em "**Novo**" / "**Documento de texto**".



- Nomeie o arquivo como "**versoes.txt**".



- Você possui uma pasta chamada "**senai-versoes-colaboracoes**" com uma subpasta do .git e um arquivo de texto criado.



RASTREAR ALTERAÇÕES NO REPOSITÓRIO: COMANDO *git status*

Você criou um arquivo, mas ele ainda não está sendo rastreado pelo Git e, consequentemente, as informações não foram salvas em seu repositório. Isso significa que o Git ainda não está fazendo o monitoramento de alterações de seu arquivo.

Para verificarmos o status das alterações que foram realizadas dentro de um repositório, utiliza-se o comando **git status**. Esse comando nos ajuda a verificar ao longo do tempo de desenvolvimento de seu projeto, quais arquivos estão sendo alterados por você em sua máquina dentro do repositório.

```
MINGW64:/c/Users/sn1067709/senai-versoes-colaboracoes
SN1067709@ESN000N1241146 MINGW64 ~/senai-versoes-colaboracoes (main)
$ git status
On branch main

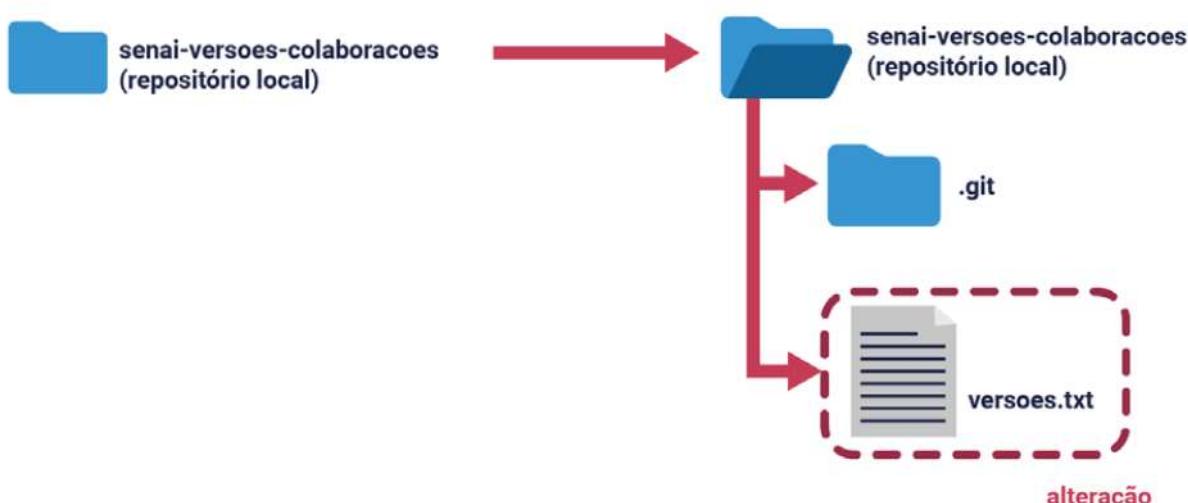
No commits yet

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

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

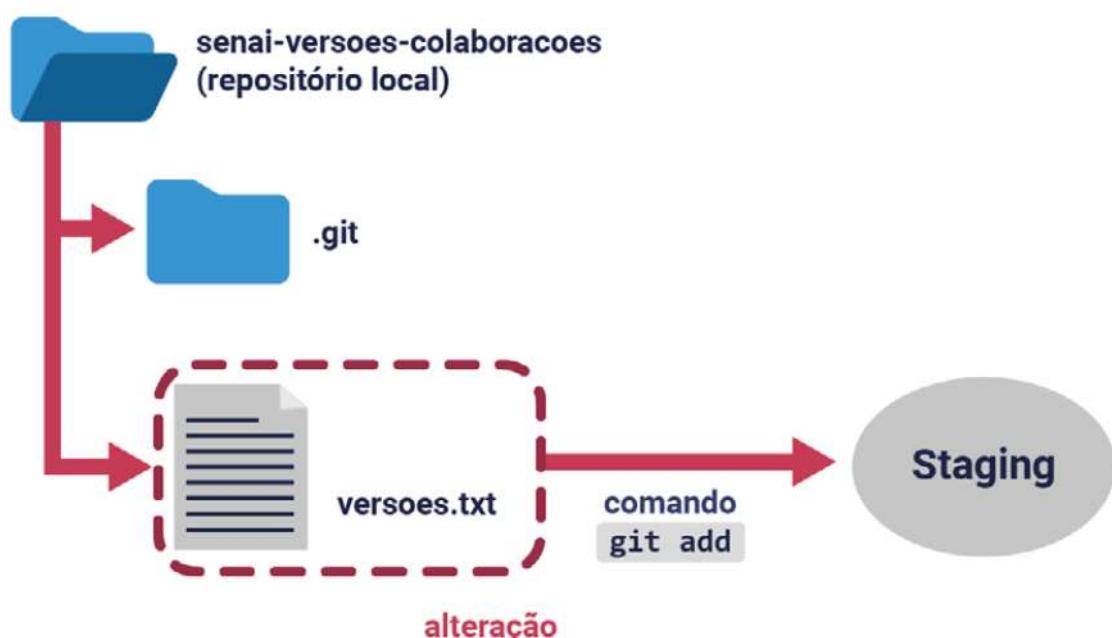
SN1067709@ESN000N1241146 MINGW64 ~/senai-versoes-colaboracoes (main)
$
```

Então, de um repositório Git vazio do início, agora temos o diretório `senai-versoes-colaboracoes` com uma pasta `.git`, **que não conta como uma alteração** e que contém informações relacionadas a nosso versionamento e informações do nosso repositório, e um arquivo de texto chamado "versoes.txt". Ao adicionar um arquivo, fizemos uma alteração.



ADICIONAR VERSÃO NO REPOSITÓRIO: COMANDO `git add`

O arquivo "versoes.txt" ainda não foi adicionado ao histórico do projeto. Para isso, primeiro devemos registrar a alteração com o comando **git add**, depois salvá-la no nosso repositório com o comando **git commit** e, aí sim, podemos considerar a alteração adicionada ao histórico. Depois do comando **git add**, as alterações vão para um ponto seguro, onde é possível verificar as alterações antes de salvá-las, chamado *Staging*.



O comando **git add [nome-do- arquivo]** adiciona os arquivos modificados na área de *staging*, que é um ponto intermediário entre a máquina do desenvolvedor e o repositório de trabalho. A *staging* é uma área que você pode fazer uma revisão do que foi alterado antes de ser salvo no histórico de alterações.

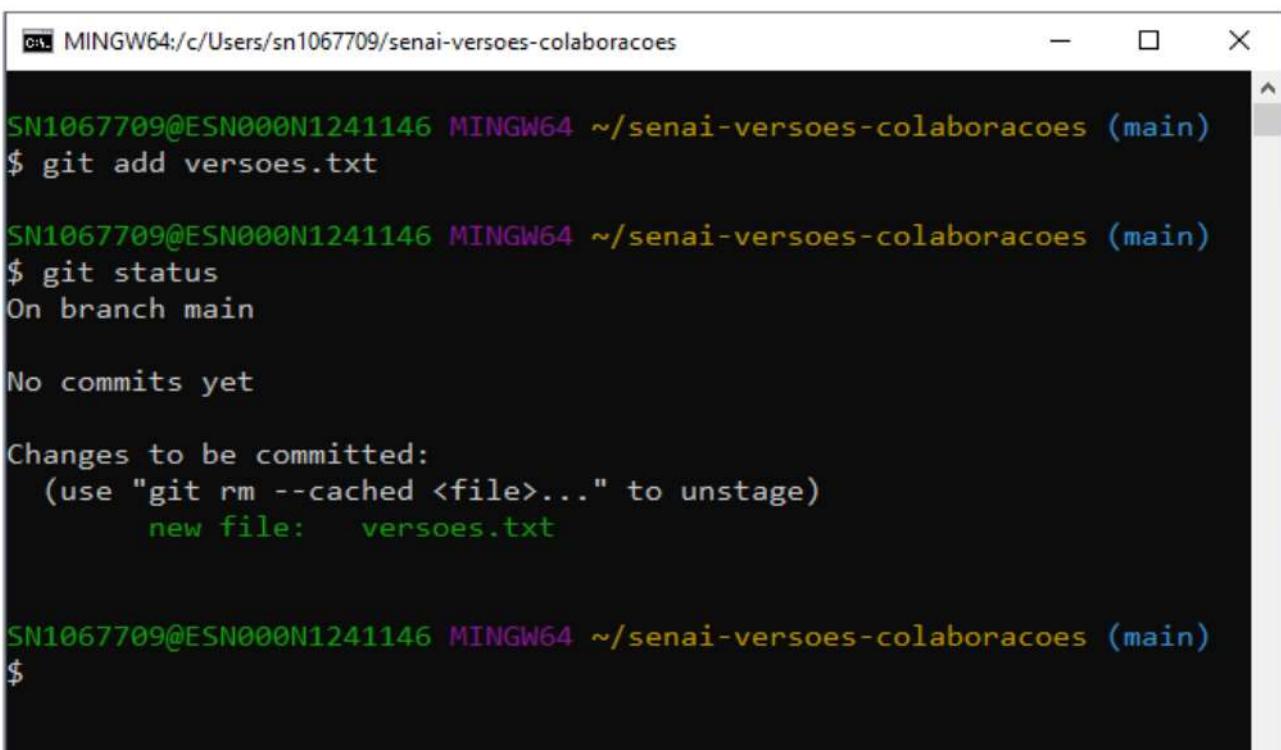
```
MINGW64:/c/Users/sn1067709/senai-versoes-colaboracoes
SN1067709@ESN000N1241146 MINGW64 ~/senai-versoes-colaboracoes (main)
$ git add versoes.txt

SN1067709@ESN000N1241146 MINGW64 ~/senai-versoes-colaboracoes (main)
$ -
```

Caso você tenha mais de um arquivo, você pode executar o comando **git add .** para adicionar todos os arquivos que foram modificados.

VERIFICAR ALTERAÇÕES ANTES DE SALVAR: COMANDO **git status**

Para verificar essas alterações antes de confirmar, você pode utilizar o comando **git status**



```
MINGW64:/c/Users/sn1067709/senai-versoes-colaboracoes
SN1067709@ESN000N1241146 MINGW64 ~/senai-versoes-colaboracoes (main)
$ git add versoes.txt

SN1067709@ESN000N1241146 MINGW64 ~/senai-versoes-colaboracoes (main)
$ git status
On branch main

No commits yet

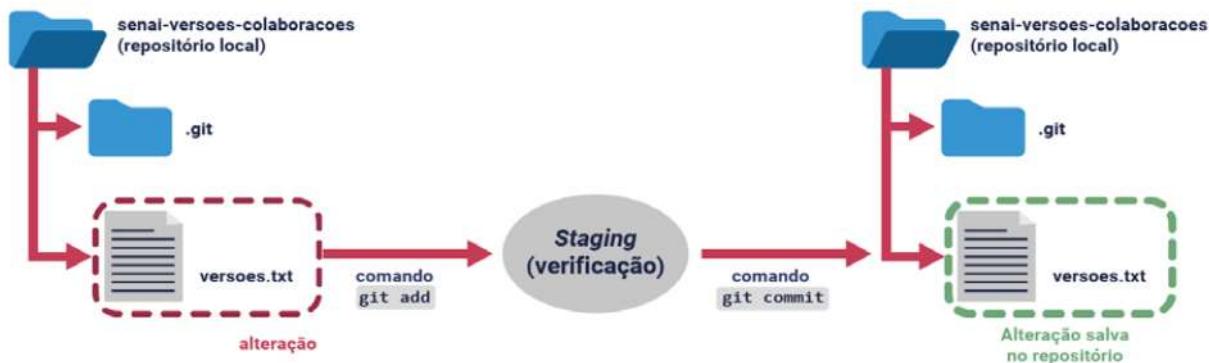
Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
    new file:   versoes.txt

SN1067709@ESN000N1241146 MINGW64 ~/senai-versoes-colaboracoes (main)
$
```

As alterações no Git, diferentemente de outros tipos de versionamento de código, podem ser feitas localmente e depois publicadas em um repositório remoto (disponível online e para outras pessoas).

SALVAR ALTERAÇÕES: COMANDO **git commit**

Um dos passos mais importantes é confirmar as alterações do fluxo de trabalho em versões no histórico do repositório. O Git irá confirmar a captura do diretório de *staging* (isto é, a captura de todas as alterações adicionadas no **git add**) e salvar no histórico de confirmação de repositórios, completando o *commit*.



Veja como digitar o comando completo **git commit -m "meu primeiro commit"** na imagem a seguir.

```
git MINGW64:/c/Users/sn1067709/senai-versoes-colaboracoes
SN1067709@ESN000N1241146 MINGW64 ~/senai-versoes-colaboracoes (main)
$ git commit -m "meu primeiro commit"
[main (root-commit) 5bf39a9] meu primeiro commit
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 versoes.txt

SN1067709@ESN000N1241146 MINGW64 ~/senai-versoes-colaboracoes (main)
$
```

Importante!

Por padrão, todas as alterações serão feitas no ramo principal do repositório, chamada `main`. Você pode criar ramificações da `main` para testar parte do código, chamadas de *branch*. Por enquanto, mantenha em mente que todas as alterações estão sendo feitas na linha de trabalho principal, na `main`¹.

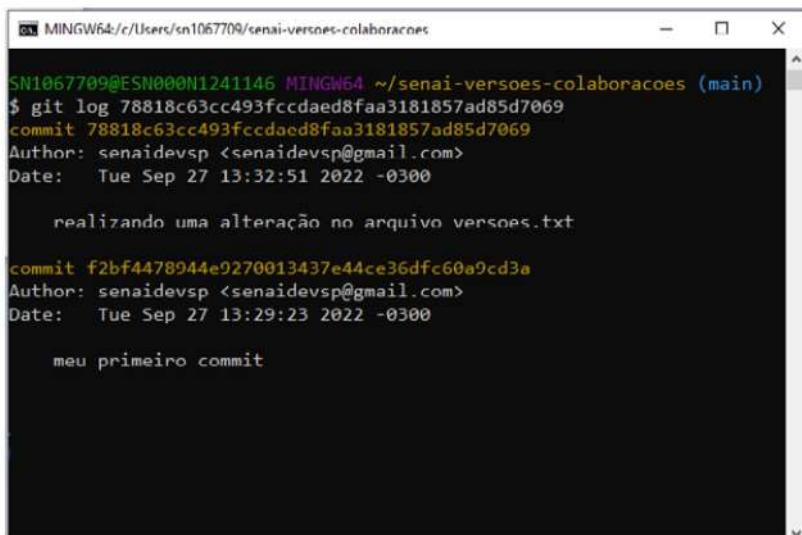


```
git MINGW64:/c/Users/sn1067709/senai-versoes-colaboracoes
SN1067709@ESN000N1241146 MINGW64 ~/senai-versoes-colaboracoes (main)
$ git commit -m "meu primeiro commit"
[main (root-commit) 5bf39a9] meu primeiro commit
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 versoes.txt

SN1067709@ESN000N1241146 MINGW64 ~/senai-versoes-colaboracoes (main)
$
```

VISUALIZAR LOG DA ALTERAÇÃO: COMANDO *git log*

Para visualizar as alterações feitas no seu repositório de trabalho, você pode executar o **git log** no terminal.



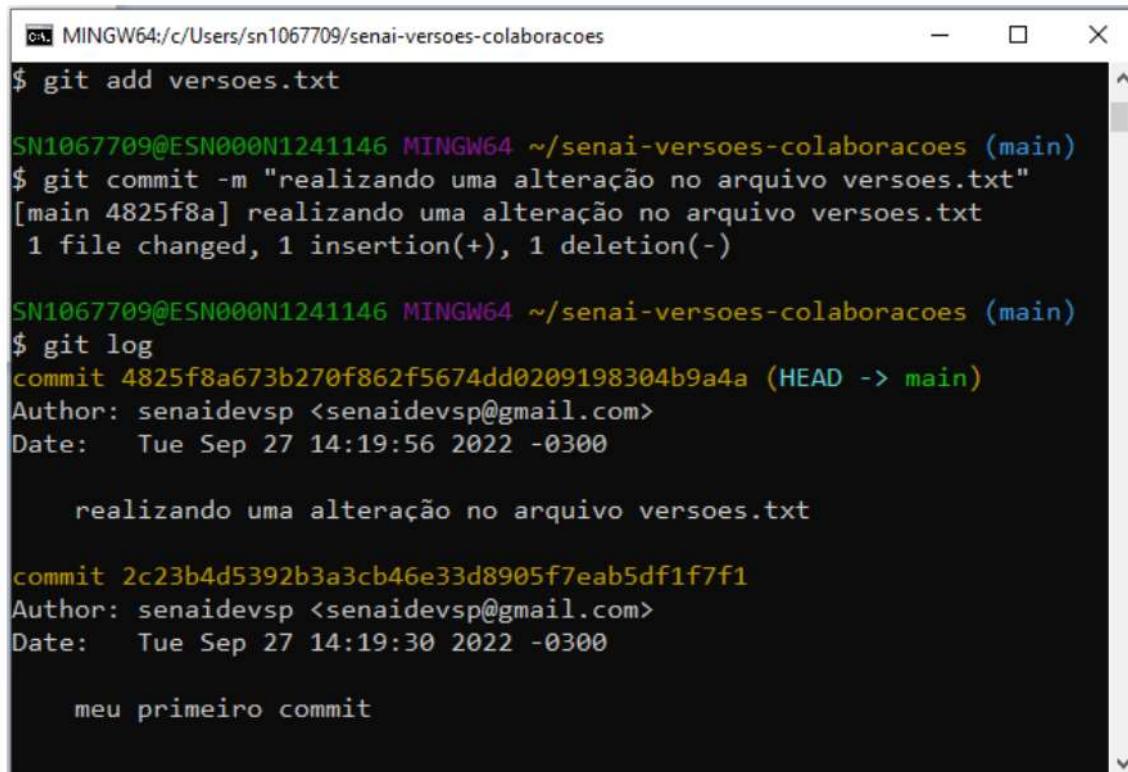
```
SN1067709@ESN000N1241146 MINGW64 ~/senai-versoes-colaboracoes (main)
$ git log 78818c63cc493fccdaed8faa3181857ad85d7069
commit 78818c63cc493fccdaed8faa3181857ad85d7069
Author: senaidevsp <senaidevsp@gmail.com>
Date:   Tue Sep 27 13:32:51 2022 -0300

    realizando uma alteração no arquivo versoes.txt

commit f2bf4478944e9270013437e44ce36dfc60a9cd3a
Author: senaidevsp <senaidevsp@gmail.com>
Date:   Tue Sep 27 13:29:23 2022 -0300

    meu primeiro commit
```

Uma das vantagens do Git é visualizar as informações que foram alteradas em um determinado ponto do desenvolvimento. As alterações que foram confirmadas podem ser visualizadas pelo terminal ao executar o comando **git log**.



```
SN1067709@ESN000N1241146 MINGW64 ~/senai-versoes-colaboracoes (main)
$ git add versoes.txt

$ git commit -m "realizando uma alteração no arquivo versoes.txt"
[main 4825f8a] realizando uma alteração no arquivo versoes.txt
 1 file changed, 1 insertion(+), 1 deletion(-)

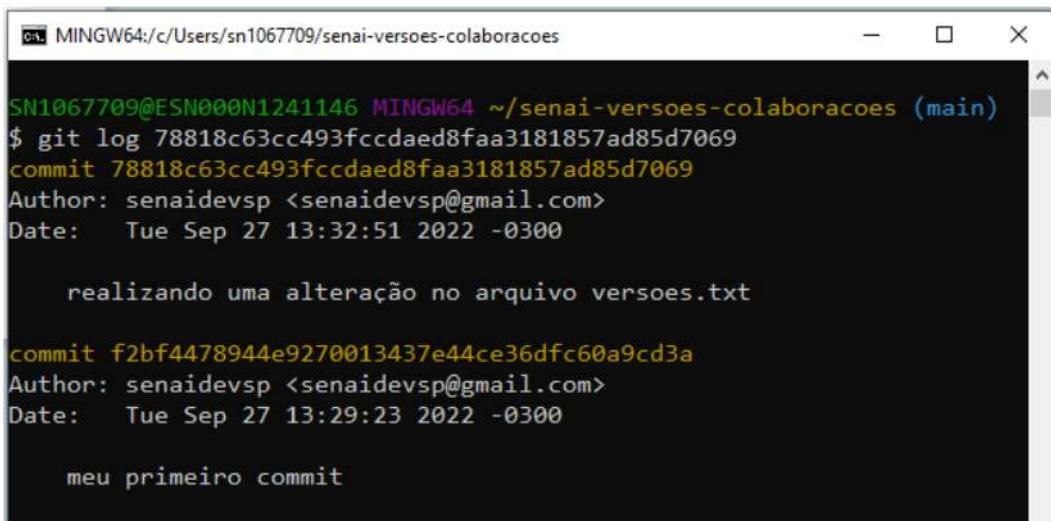
SN1067709@ESN000N1241146 MINGW64 ~/senai-versoes-colaboracoes (main)
$ git log
commit 4825f8a673b270f862f5674dd0209198304b9a4a (HEAD -> main)
Author: senaidevsp <senaidevsp@gmail.com>
Date:   Tue Sep 27 14:19:56 2022 -0300

    realizando uma alteração no arquivo versoes.txt

commit 2c23b4d5392b3a3cb46e33d8905f7eab5df1f7f1
Author: senaidevsp <senaidevsp@gmail.com>
Date:   Tue Sep 27 14:19:30 2022 -0300

    meu primeiro commit
```

Para visualizar as alterações ao longo do projeto, usamos o comando **git show número-do-commit**.



```
SN1067709@ESN000N1241146 MINGW64 ~/senai-versoes-colaboracoes (main)
$ git log 78818c63cc493fccdaed8faa3181857ad85d7069
commit 78818c63cc493fccdaed8faa3181857ad85d7069
Author: senaidevsp <senaidevsp@gmail.com>
Date:   Tue Sep 27 13:32:51 2022 -0300

    realizando uma alteração no arquivo versoes.txt

commit f2bf4478944e9270013437e44ce36dfc60a9cd3a
Author: senaidevsp <senaidevsp@gmail.com>
Date:   Tue Sep 27 13:29:23 2022 -0300

    meu primeiro commit
```

CRIANDO UM REPOSITÓRIO ONLINE

Para disponibilizar o repositório local, temos que utilizar um repositório online e, para isso, vamos utilizar a plataforma GitHub.



Importante!

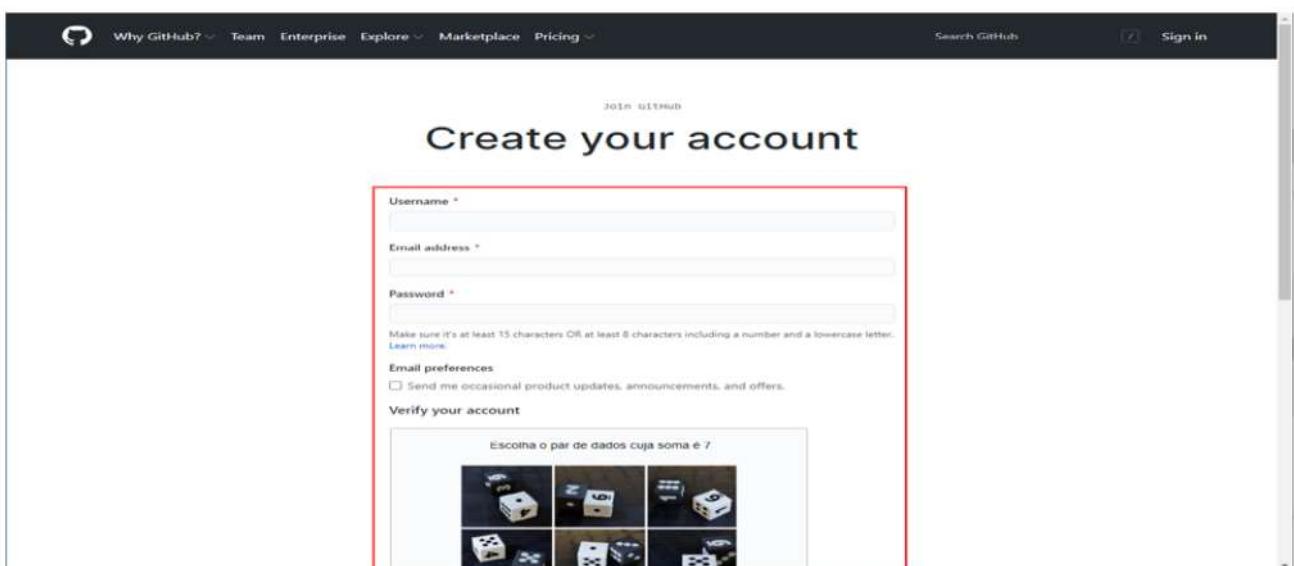
O usuário do GitHub aparece em alguns comandos no repositório local. Lembre-se, portanto, de **trocar** o do **exemplo** (hstrada) pelo seu usuário quando for praticar.



- Acesse o site <https://github.com> e clique em "**sign up**" para fazer o seu cadastro.

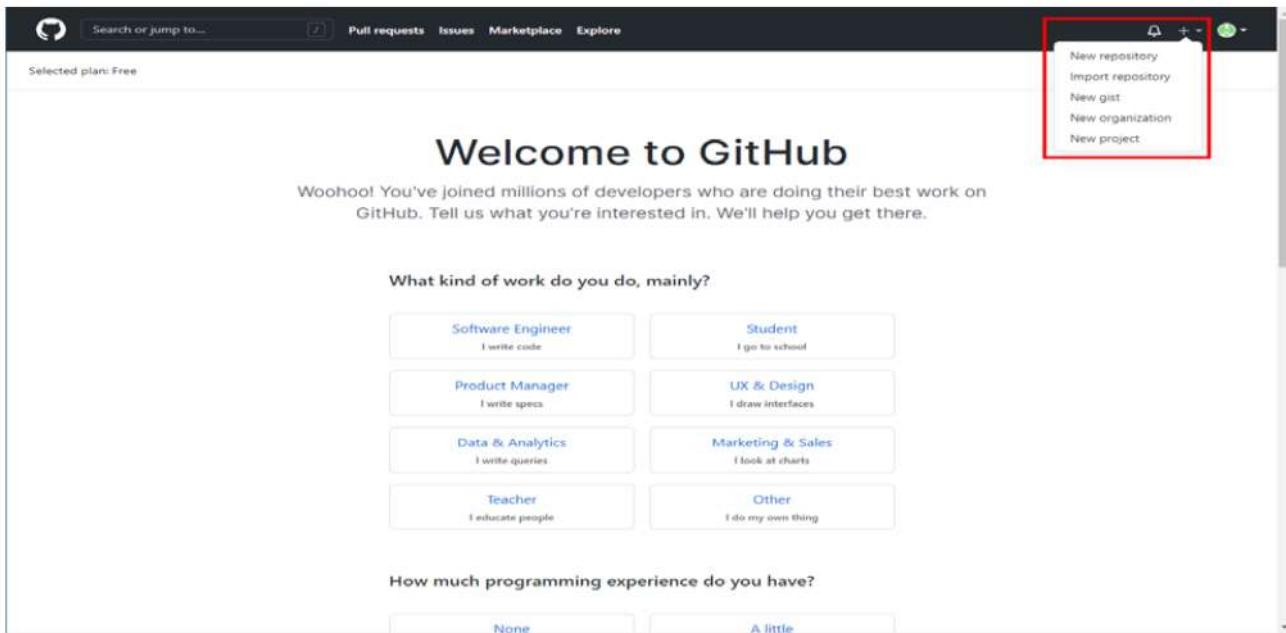


- Insira seus dados e faça o teste pedido pelo site. Depois de confirmar seu e-mail, retorne ao site e logue clicando em "sign in".



Para criar um repositório remoto, clique, no canto superior direito da tela, no botão + e depois clique em "**New repository**" (ou novo repositório, em português).

Versionamento - Desafio 2



- Na tela a seguir, nomeie o repositório como **senai-versoes-colaboracoes**, inclusive a estrutura que você criou no seu computador. Você pode também inserir uma breve descrição.

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? Import a repository.

Owner * Repository name *

senalidevsp / senai-versoes-colaboracoes ✓

Great repository names are short and memorable. Need inspiration? How about potential-fortnight?

Description (optional)

Repositório de versões e colaborações.

Public Anyone on the internet can see this repository. You choose who can commit.

Private You choose who can see and commit to this repository.

Initialize this repository with:

Skip this step if you're importing an existing repository.

Add a README file This is where you can write a long description for your project. Learn more.

Add .gitignore Choose which files not to track from a list of templates. Learn more.

.gitignore template: None

Choose a license A license tells others what they can and can't do with your code. Learn more.

- Configure seu repositório como **público ou privado**.

Description (optional)

Repositório de versões e colaborações.

Public Anyone on the internet can see this repository. You choose who can commit.

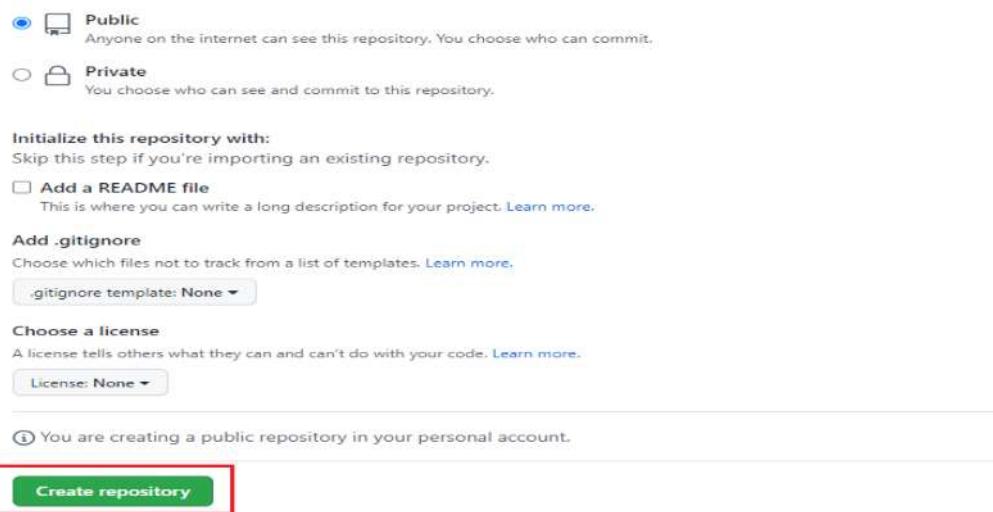
Private You choose who can see and commit to this repository.

Initialize this repository with:

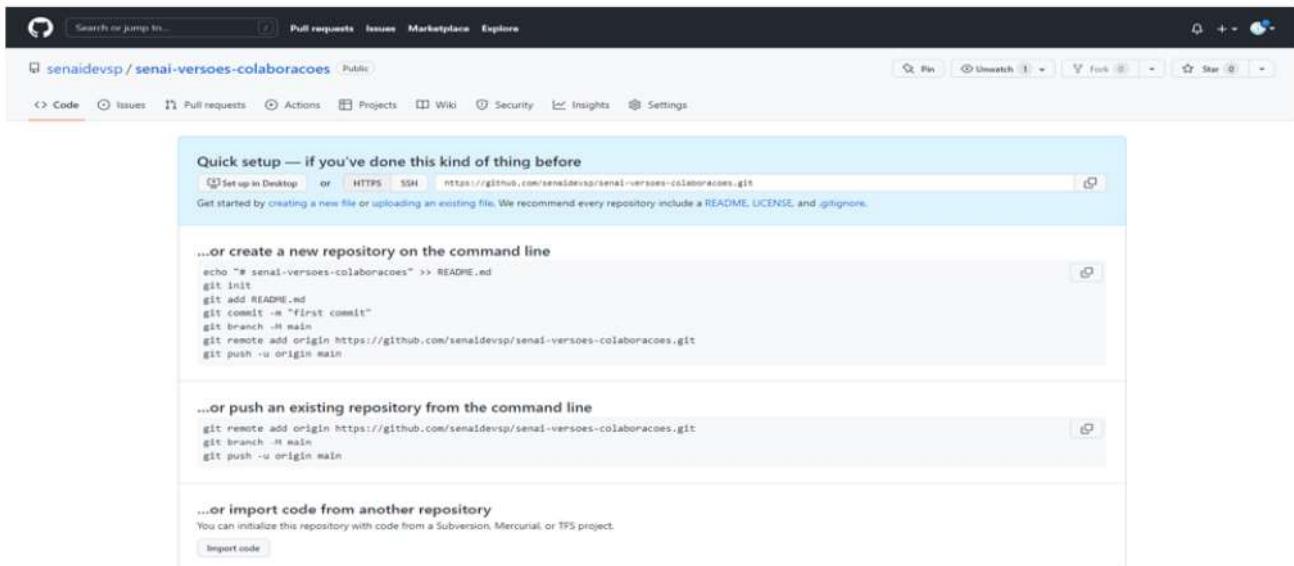
Skip this step if you're importing an existing repository.

Add a README file This is where you can write a long description for your project. Learn more.

- Reppositórios públicos podem ser acessados por qualquer usuário, através do link do seu repositório. Como as informações contidas ali ficam acessíveis, os repositórios públicos são criados para disponibilizar projetos particulares e projetos de código aberto.
- Repositórios privados são usualmente criados por empresas e utilizados quando contêm informações sensíveis e informações de código-fonte das empresas que não podem ser compartilhadas, por questões de segurança.
- Clique em "**Create repository**".



- Seu repositório online está criado.



DESAFIO 2 - PARTE 2

Agora que você já sabe como organizar arquivos no Git, os comandos básicos para criar arquivos, rastrear alterações, adicionar versões, verificar e salvar alterações e criar repositórios locais e remotos, siga em frente para a segunda parte para aprender como:

- Publicar repositório online;
- Criar ramificações (*branches*);
- Ignorar alterações.

PUBLICANDO NO REPOSITÓRIO ONLINE



Você criou um repositório online, mas não especificou que é neste repositório que gostaria de publicar seu trabalho.

Depois de criar o repositório remoto, precisamos fazer a ligação com o repositório local. Para isso, no terminal local, vamos:

- informar a pasta remota: **git remote add origin git@github.com:hstrada/senai-versoes-colaboracoes.git** (lembre-se de **trocar o usuário** no comando);
- visualizar o repositório remoto: **git remote -v**; e
- publicar as alterações no repositório remoto: **git push -u origin main**.

No repositório remoto, vamos:

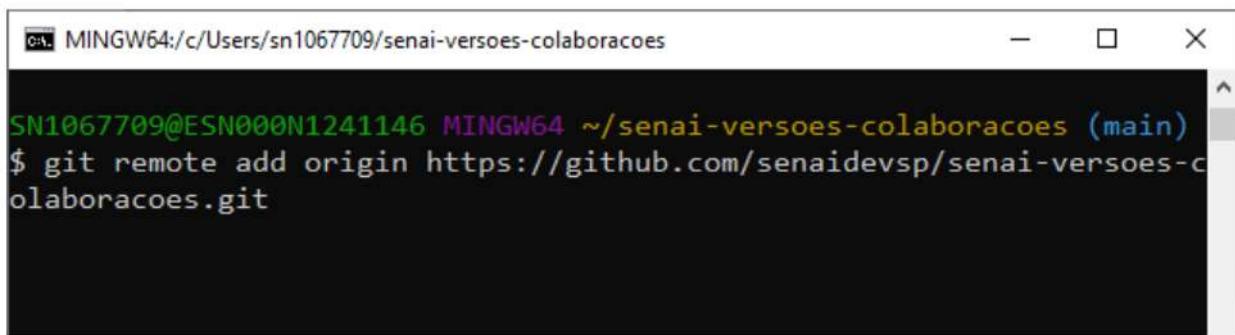
- autorizar o usuário.

Depois, vamos verificar se todas as etapas foram completadas corretamente, então, devemos:

- no repositório local: visualizar a autenticação de usuário feita no repositório remoto;
- no repositório remoto: visualizar a publicação feita.

INFORMAR NO REPOSITÓRIO LOCAL A PASTA REMOTA: *git remote add origin git@github.com:hstrada/senai-versoes-colaboracoes.git*

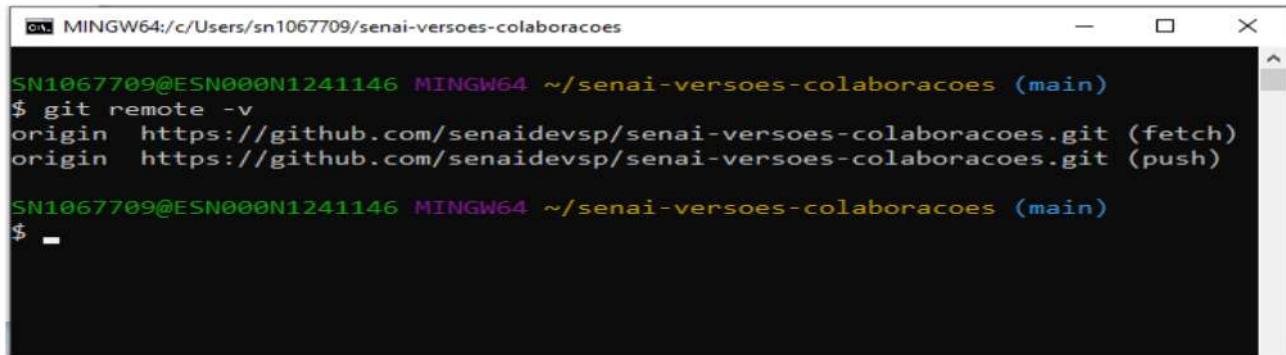
Para realizar a ligação dos repositórios, primeiro é necessário informar no repositório local que o repositório remoto que você deseja trabalhar é aquele criado no GitHub, através do comando **git remote add origin git@github.com:hstrada/senai-versoes-colaboracoes.git**.



```
MINGW64:/c/Users/sn1067709/senai-versoes-colaboracoes
SN1067709@ESN000N1241146 MINGW64 ~/senai-versoes-colaboracoes (main)
$ git remote add origin https://github.com/senaidevsp/senai-versoes-colaboracoes.git
```

VISUALIZAR O REPOSITÓRIO REMOTO: *git remote -v*

Para visualizar o repositório remoto informado e visualizar as informações, digite no terminal **git remote -v**.



```
MINGW64:/c/Users/sn1067709/senai-versoes-colaboracoes
SN1067709@ESN000N1241146 MINGW64 ~/senai-versoes-colaboracoes (main)
$ git remote -v
origin  https://github.com/senaidevsp/senai-versoes-colaboracoes.git (fetch)
origin  https://github.com/senaidevsp/senai-versoes-colaboracoes.git (push)

SN1067709@ESN000N1241146 MINGW64 ~/senai-versoes-colaboracoes (main)
$ -
```

PUBLICAR AS ALTERAÇÕES NO REPOSITÓRIO REMOTO: *git push -u origin main*

Para publicar as alterações no repositório remoto, execute o comando **git push -u origin main**.

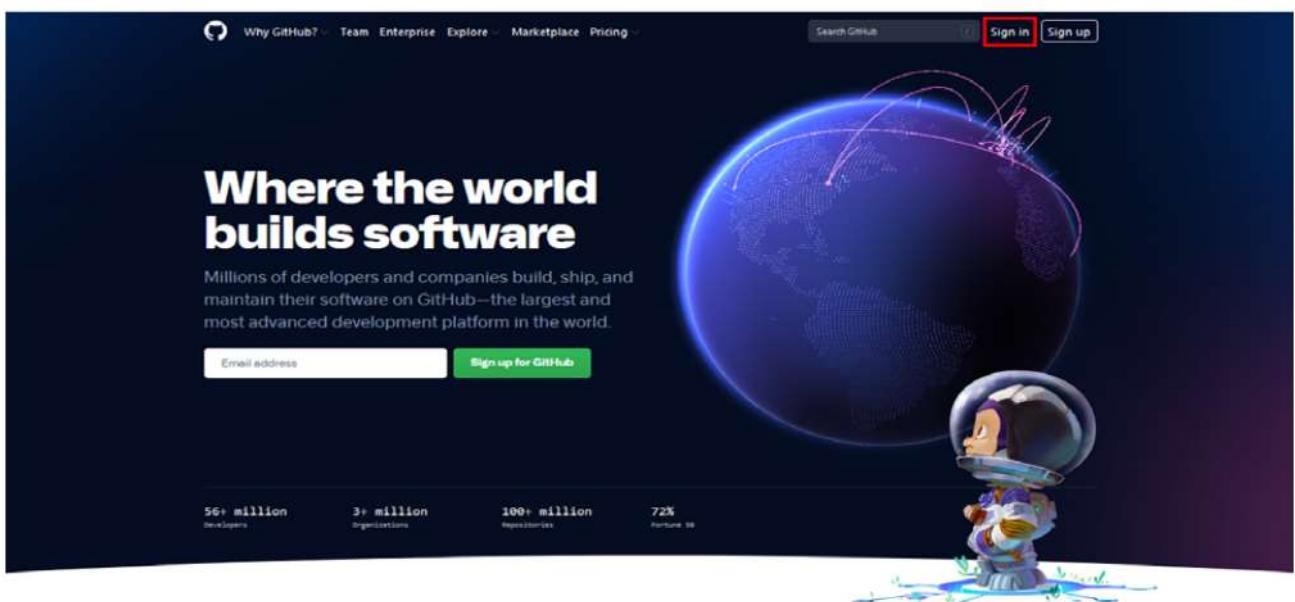
```
MINGW64:/c/Users/sn1067709/senai-versoes-colaboracoes
SN1067709@ESN000N1241146 MINGW64 ~/senai-versoes-colaboracoes (main)
$ git remote -v
origin  https://github.com/senaidevsp/senai-versoes-colaboracoes.git (fetch)
origin  https://github.com/senaidevsp/senai-versoes-colaboracoes.git (push)

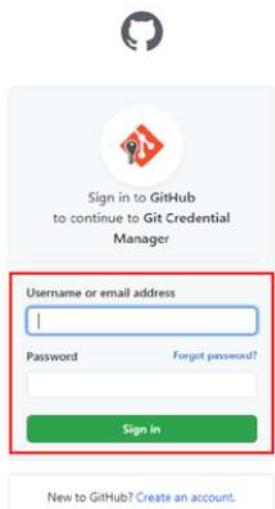
SN1067709@ESN000N1241146 MINGW64 ~/senai-versoes-colaboracoes (main)
$ git push -u origin main
```

AUTORIZAR USUÁRIO NO REPOSITÓRIO REMOTO

O repositório criado no GitHub está atrelado à sua conta e, portanto, seu user deve ser autorizado a se comunicar com seu repositório local.

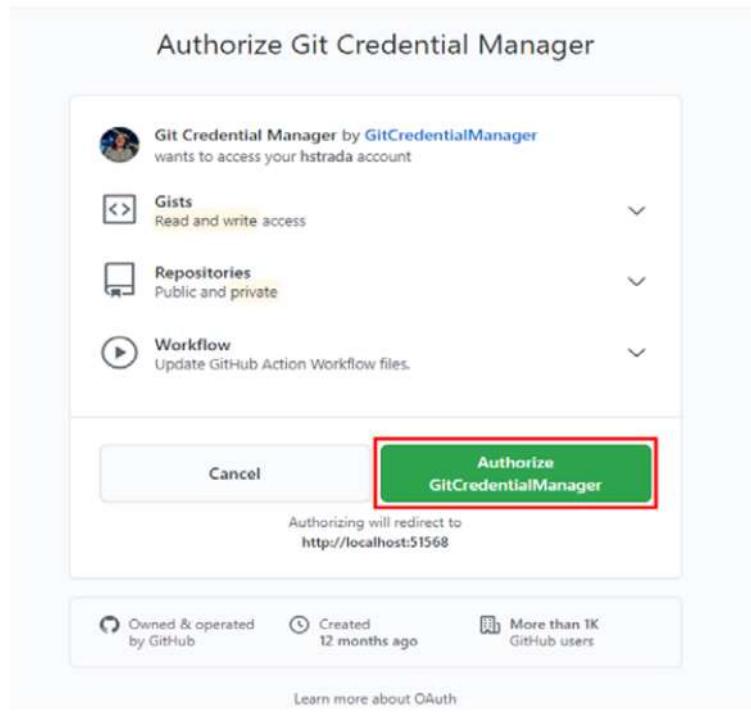
- Acesse o site <https://github.com> e clique em "sign in" para fazer o seu cadastro.



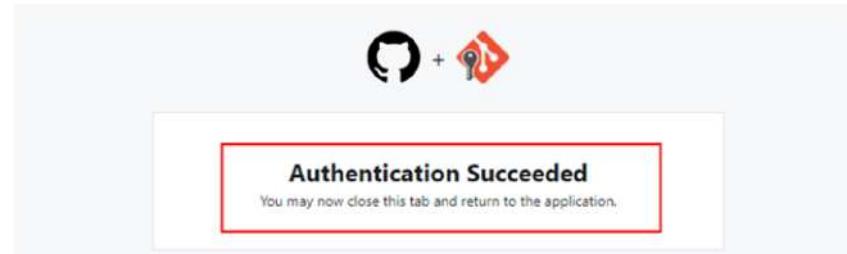


- Digite seus dados e clique em "**sign in**".

- Clique em "**Authorize GitCredentialManager**" para autorizar o seu usuário.



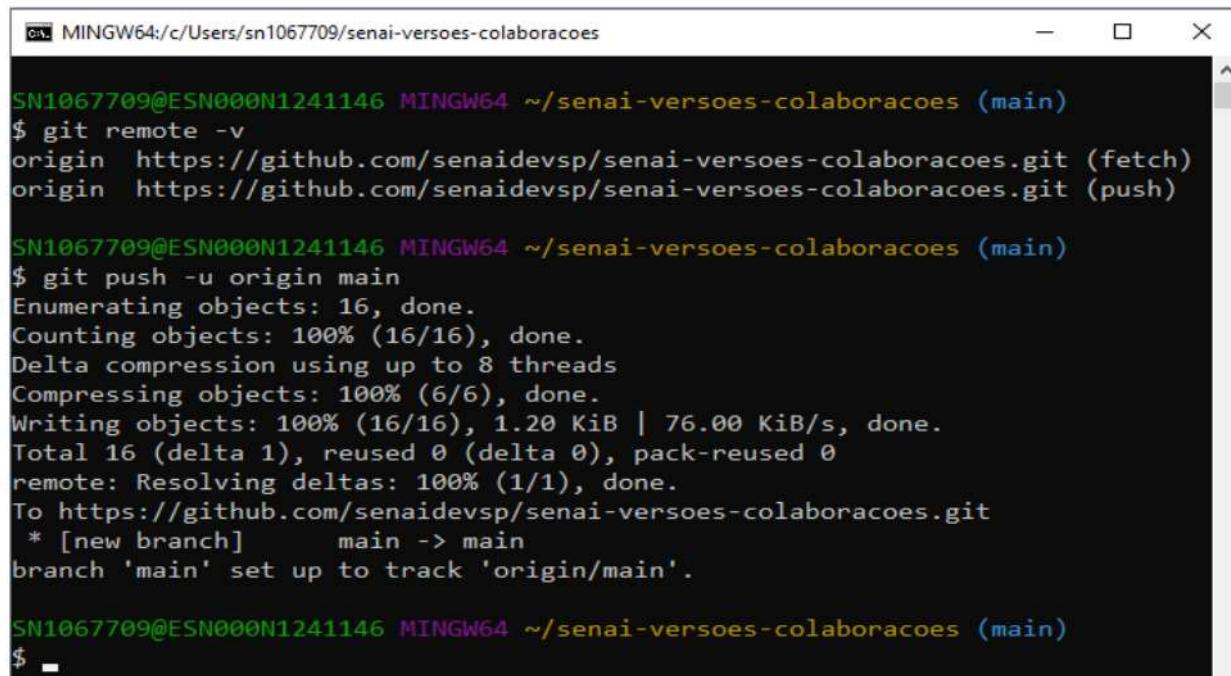
- A autenticação será feita com sucesso. Feche o navegador e retorne ao terminal local para visualizar.



VERIFICAÇÕES

Depois de autorizar seu usuário no GitHub, vamos fazer as verificações local e remota.

Seu terminal local deve mostrar as informações como a imagem a seguir.

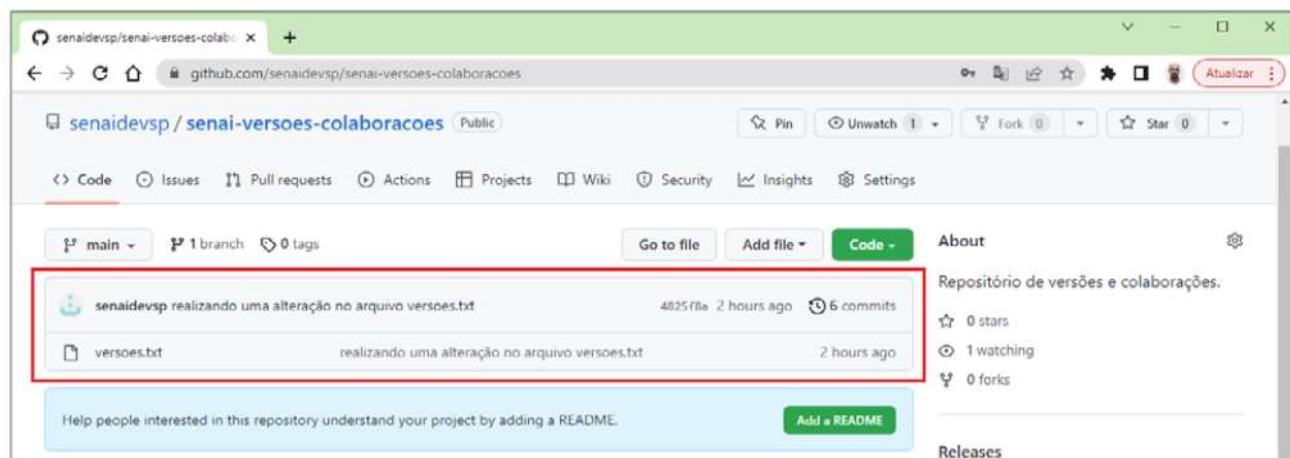


```
MINGW64:/c/Users/sn1067709/senai-versoes-colaboracoes
SN1067709@ESN000N1241146 MINGW64 ~/senai-versoes-colaboracoes (main)
$ git remote -v
origin https://github.com/senaidevsp/senai-versoes-colaboracoes.git (fetch)
origin https://github.com/senaidevsp/senai-versoes-colaboracoes.git (push)

SN1067709@ESN000N1241146 MINGW64 ~/senai-versoes-colaboracoes (main)
$ git push -u origin main
Enumerating objects: 16, done.
Counting objects: 100% (16/16), done.
Delta compression using up to 8 threads
Compressing objects: 100% (6/6), done.
Writing objects: 100% (16/16), 1.20 KiB | 76.00 KiB/s, done.
Total 16 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), done.
To https://github.com/senaidevsp/senai-versoes-colaboracoes.git
 * [new branch]      main -> main
branch 'main' set up to track 'origin/main'.

SN1067709@ESN000N1241146 MINGW64 ~/senai-versoes-colaboracoes (main)
$
```

- Acesse o navegador e atualize a página para visualizar as informações publicadas.



- Para visualizar as alterações, clique nos "**commits**".
- É o equivalente a executar o git show no terminal local.

Versionamento - Desafio 2

The screenshot shows a GitHub repository page for 'senaidesp / senai-versoes-colaboracoes'. The 'Code' tab is selected. At the top, there are buttons for 'Pin', 'Unwatch', 'Fork', 'Star', and 'Settings'. Below the header, it says 'main' (branch), '1 branch', '0 tags', 'Go to file', 'Add file', and 'Code'. A red box highlights the 'Code' button. To the right, there's an 'About' section with the repository description 'Repositório de versões e colaborações.', star count (0 stars), watch count (1 watching), fork count (0 forks), and release information ('No releases published' and 'Create a new release'). Below the repository info, there's a 'Help people interested in this repository understand your project by adding a README.' section with a 'Add a README' button. The main content area shows a list of commits:

- senaidevsp realizando uma alteração no arquivo versoes.txt (commit 4825f8a, 2 hours ago)
- versoes.txt (commit 2 hours ago)

Below the commits, there's a note: 'Help people interested in this repository understand your project by adding a README.' with a 'Add a README' button.

- Podemos visualizar as alterações de cada "**commit**" ou estado atual do projeto.
- Clique no "**commit**" mais antigo.

The screenshot shows the same GitHub repository page, but now the commit list is expanded. A red box highlights the first commit: 'realizando uma alteração no arquivo versoes.txt' (commit 4825f8a, 2 hours ago). Below it, other commits are listed:

- meu primeiro commit (commit 2e223b4d, 2 hours ago)
- realizando uma alteração no arquivo versoes.txt (commit 602500f, 2 hours ago)
- realizando uma alteração no arquivo versoes.txt (commit 78818c6, 3 hours ago)
- meu primeiro commit (commit f2bf447, 3 hours ago)

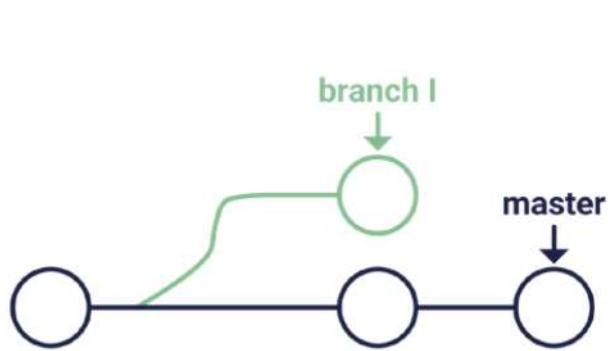
- Todas as informações sobre as alterações realizadas no arquivo serão exibidas.

The screenshot shows the detailed view of the commit 'meu primeiro commit' (commit f2bf447, 3 hours ago). The commit message is 'realizando uma alteração no arquivo versoes.txt'. The commit details show '1 parent' and '1 commit'. It indicates 'Showing 1 changed file with 1 addition and 0 deletions.' A red box highlights the file 'versoes.txt' which has been modified from '00 -0,0' to '00 +0,0'. Below the file view, it says '0 comments on commit f2bf447'.

A imagem a seguir mostra o fluxo atual de trabalho com as alterações feitas.



FLUXO DE TRABALHO



Neste primeiro cenário de fluxo de trabalho, todas as alterações estão sendo feitas no nosso ramo principal, que chamamos de **master**.

Muitas vezes, durante o desenvolvimento do projeto, há a necessidade de testar ou corrigir algum trecho do código-fonte ou adicionar um novo recurso, para isso, criamos ramificações do **master**, chamados **branches**.

Esse tipo de abordagem facilita o processo de desenvolvimento entre diferentes pessoas, possibilitando manter o código oficial como base, descentralizando o desenvolvimento.

PENSE NISSO

Assim como você, outros programadores da equipe também enviam suas respectivas alterações para o repositório online. Portanto, cada programador tem em sua máquina apenas as suas alterações e somente o repositório remoto está atualizado e completo. Acompanhe, a seguir, como baixar as alterações do repositório online para a sua máquina.

BAIXANDO AS ALTERAÇÕES DE UM REPOSITÓRIO REMOTO

O nosso repositório local deve ser uma cópia do repositório remoto. Isso significa que devemos enviar nossas alterações para o repositório remoto e também baixar as alterações de lá para nossa máquina.

Então, para aprender como sincronizar e baixar as alterações quando o repositório remoto for atualizado, vamos:

- fazer uma alteração diretamente no repositório remoto; e
- baixar a alteração no repositório remoto: **git pull**.

Vamos também abordar uma técnica muito útil para:

- impedir que uma alteração seja monitorada: arquivo **.gitignore**.

FAZER UMA ALTERAÇÃO DIRETAMENTE NO REPOSITÓRIO REMOTO

Vamos criar um arquivo README no repositório remoto para simular uma alteração feita por outro programador.

Dica!



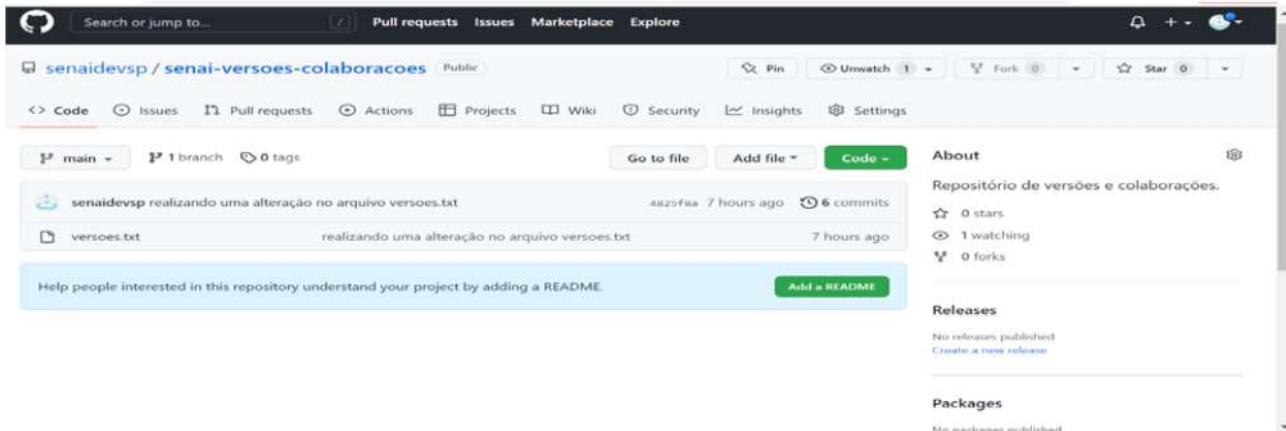
O README.md é um arquivo de texto utilizado para descrever, documentar ou exemplificar seu projeto. Apesar de não ser obrigatório (seu código vai funcionar perfeitamente sem ele), o README é essencial, pois mescla o cartão de visitas e a ementa do projeto, sendo o responsável por cativar interesse em seu trabalho ou relembrá-lo dos aspectos gerais e pontos relevantes daquele desenvolvimento.

A extensão .md representa um arquivo Markdown, uma linguagem de marcação com a qual você pode exibir diversas formatações de texto (bold, itálico, cabeçalhos, entre outros).

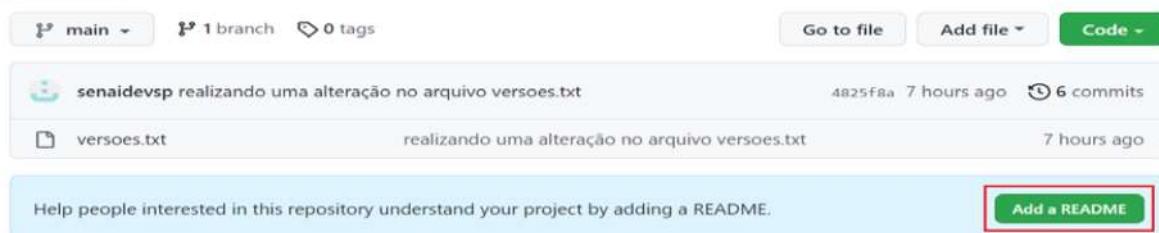
Por padrão, o README.md sempre vai estar disponível, acessível e visível nos repositórios remotos.

Versionamento - Desafio 2

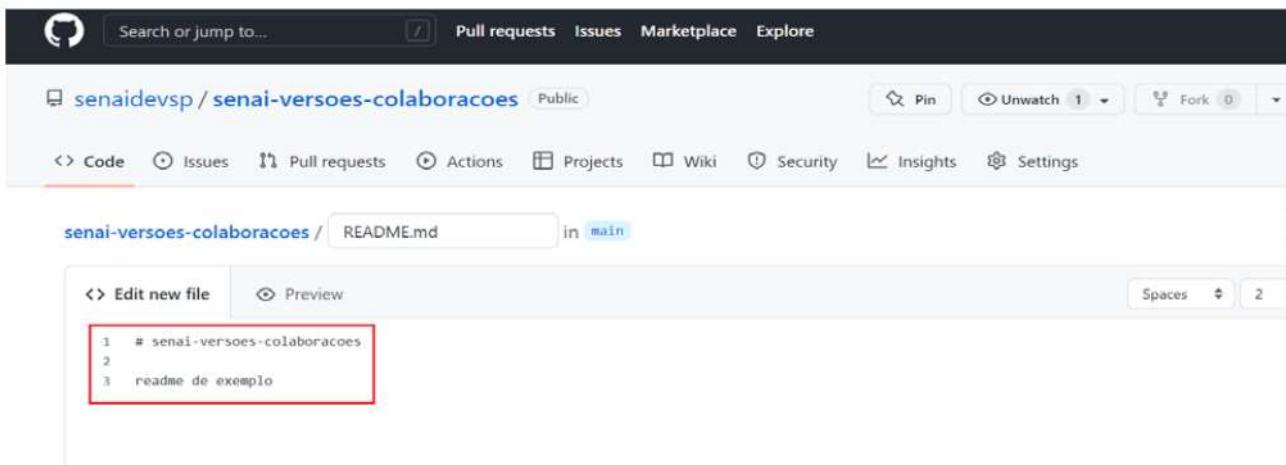
- Acesse o repositório remoto através do link "**github/seu-usuário/nome-repositório**". No nosso exemplo: "**https://github.com/hstrada/senai-versoes-colaborações**".



- Clique em "**Add a README**".



- Adicione um comentário.



- Clique em "**Commit new file**".



- Você criou o "**README**" no repositório remoto, agora é necessário baixar as alterações para o repositório local.

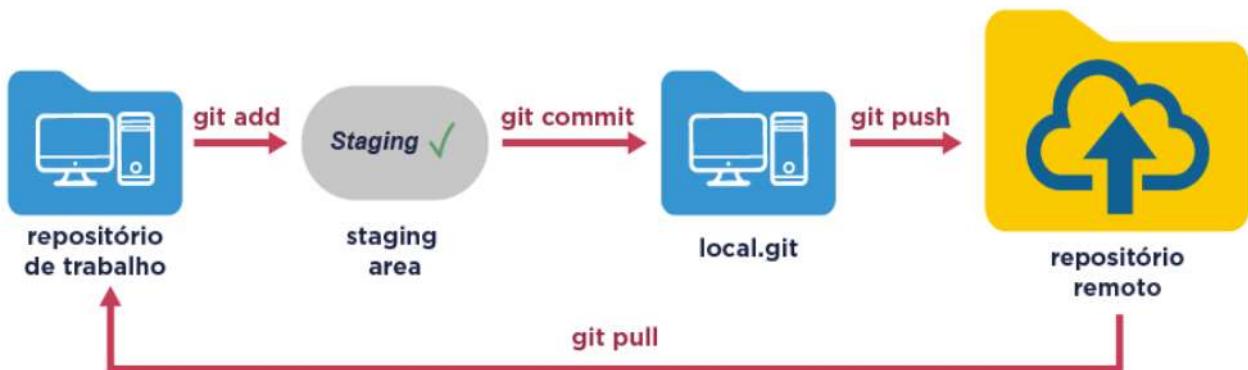
BAIXAR A ALTERAÇÃO NO REPOSITÓRIO REMOTO: **git pull**

Para baixar a atualização de seu repositório remoto para seu repositório local, execute no terminal local o comando **git pull**.

Agora é possível visualizar a atualização baixada no diretório local.

```
SN1067709@ESN000N1241146 MINGW64 ~/senai-versoes-colaboracoes (main)
$ git pull
remote: Enumerating objects: 4, done.
remote: Counting objects: 100% (4/4), 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), 703 bytes | 15.00 KiB/s, done.
From https://github.com/senaidevsp/senai-versoes-colaboracoes
 4825f8a..57c6c8e main      -> origin/main
Updating 4825f8a..57c6c8e
Fast-forward
 README.md | 3 +++
 1 file changed, 3 insertions(+)
 create mode 100644 README.md

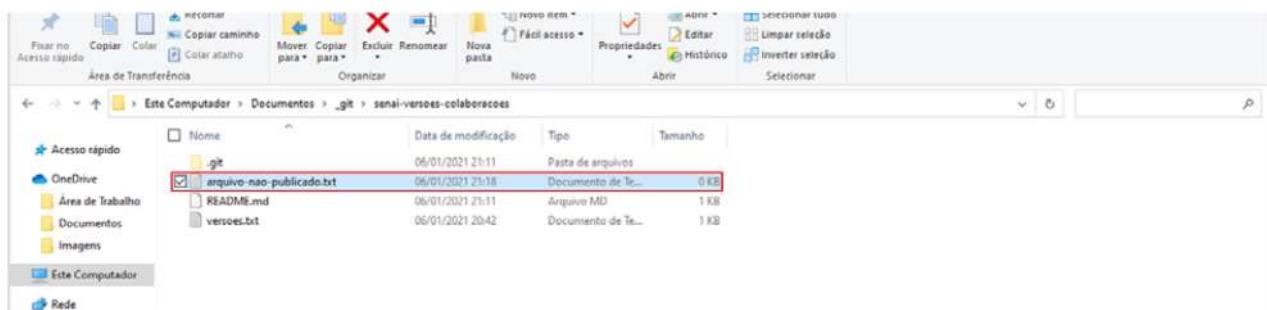
SN1067709@ESN000N1241146 MINGW64 ~/senai-versoes-colaboracoes (main)
$
```



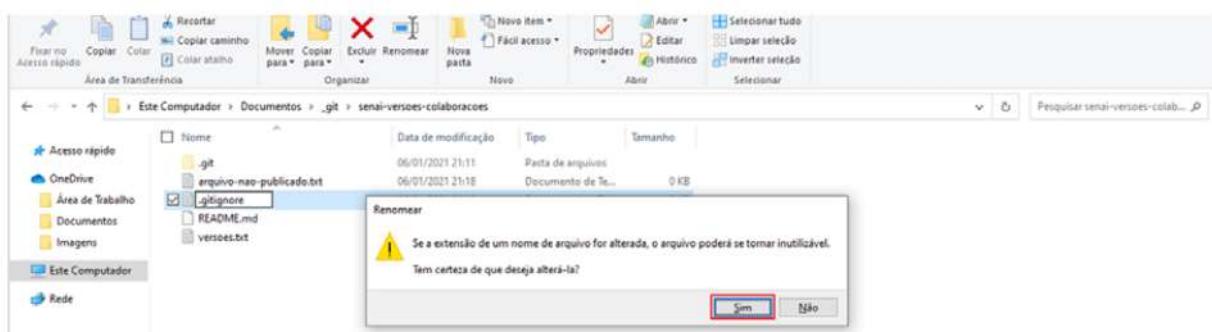
IMPEDIR QUE UMA ALTERAÇÃO SEJA MONITORADA: ARQUIVO `.gitignore`

Imagine que você estava testando um trecho de código que não deu certo e, portanto, você não quer publicá-lo. Com o Git, é possível gerenciar arquivos que não desejamos publicar. Para isso, você deve criar um arquivo **.gitignore** dentro da pasta do projeto e informar quais arquivos você deseja ignorar. Podemos adicionar pastas, arquivos ou qualquer outro item que não desejamos publicar e/ou adicionare em nosso repositório.

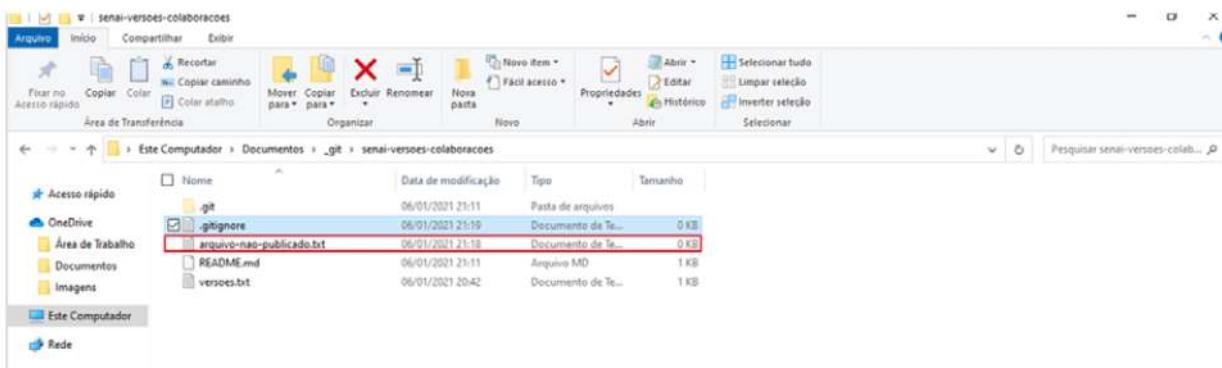
- Crie um arquivo de texto chamado "**arquivo-nao-publicado.txt**" em nossa pasta de trabalho "**senai-versoes-colaboracoes**".



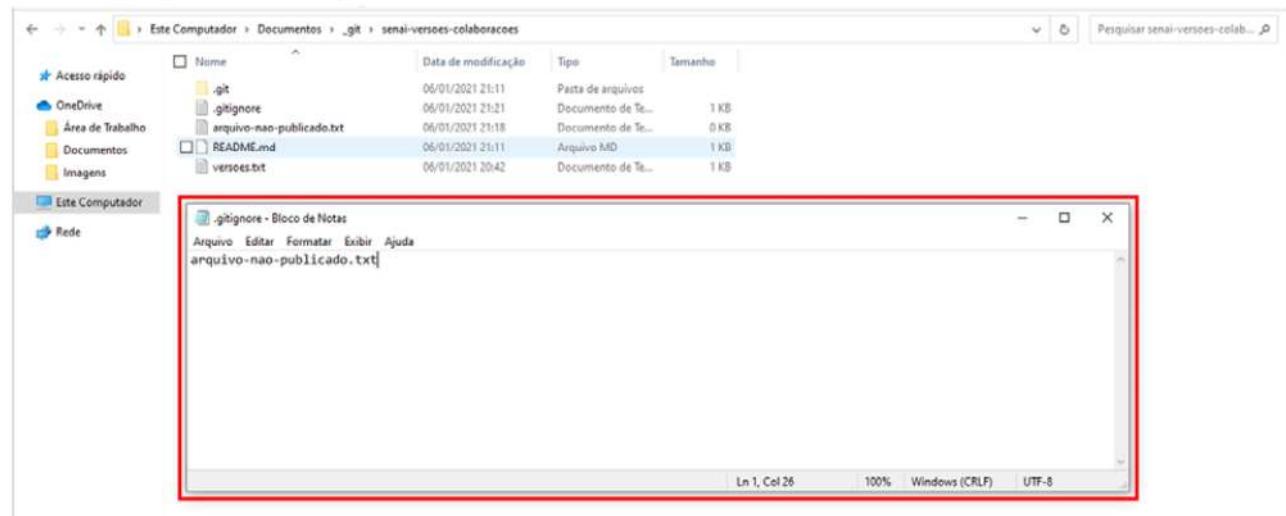
- Crie um arquivo de texto e o nomeie como "**.gitignore**".
- Clique em "Sim" para aceitar.



- Agora vamos inserir o arquivo que deve ser ignorado, o "**arquivo-não-publicado.txt**".



- Adicione o nome do arquivo "**arquivo-nao-publicado.txt**" que você não deseja que o Git faça o rastreio.



- Vamos verificar os status das modificações locais. No Git Bash, execute o comando **git status**.
- Note que o "**arquivo-nao-publicado.txt**" não foi reconhecido como modificado/inserido/editado.

```
SN1067709@ESN000N1241146 MINGW64 ~/senai-versoes-colaboracoes (main)
$ git status
On branch main
Your branch is up to date with 'origin/main'.

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

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

SN1067709@ESN000N1241146 MINGW64 ~/senai-versoes-colaboracoes (main)
$
```

- Adicione as alterações com o **git add**, confirme com o **git commit** e publique no repositório remoto com o **git push**.

```

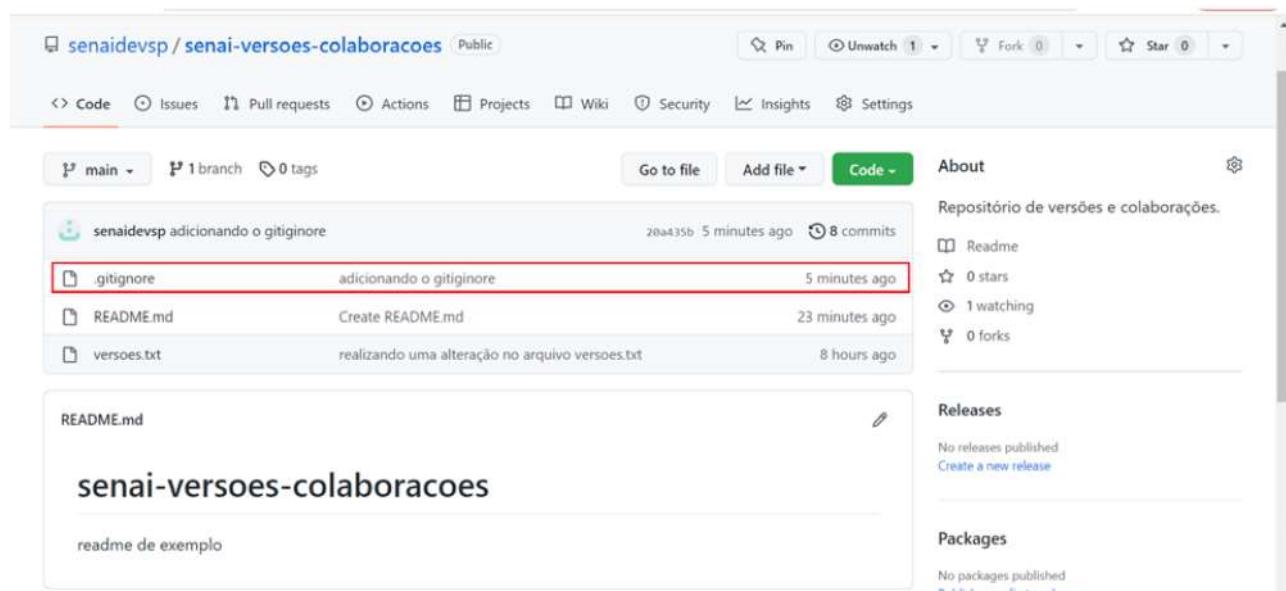
MINGW64:/c/Users/sn1067709/senai-versoes-colaboracoes
$ git add .

SN1067709@ESN000N1241146 MINGW64 ~/senai-versoes-colaboracoes (main)
$ git commit -m "adicionando o gitignore"
[main 20a435b] adicionando o gitignore
 1 file changed, 1 insertion(+)
 create mode 100644 .gitignore

SN1067709@ESN000N1241146 MINGW64 ~/senai-versoes-colaboracoes (main)
$ git push -u origin main
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 8 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 342 bytes | 68.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/senaidevsp/senai-versoes-colaboracoes.git
  57c6c8e..20a435b main -> main
branch 'main' set up to track 'origin/main'.

SN1067709@ESN000N1241146 MINGW64 ~/senai-versoes-colaboracoes (main)
$ 
```

- Retorne ao GitHub e atualize a página. Observe que o arquivo que adicionamos ao `.gitignore` não sofreu nenhuma ação ao realizarmos um commit (isso funciona para os arquivos que ainda não foram rastreados).



The screenshot shows a GitHub repository page for 'senaidevsp / senai-versoes-colaboracoes'. The repository is public. The commit history is displayed, showing the following entries:

- senaidevsp adicionando o gitignore** (20a435b, 5 minutes ago) - This commit is highlighted with a red box.
- .gitignore** (5 minutes ago) - This commit is also highlighted with a red box.
- README.md** (Create README.md, 23 minutes ago)
- versoes.txt** (realizando uma alteração no arquivo versoes.txt, 8 hours ago)

The right sidebar shows repository statistics: 8 commits, 0 stars, 1 watching, and 0 forks. It also includes sections for Releases (No releases published, Create a new release) and Packages (No packages published, Publish your first package).

Dica!

Você pode ter reparado que, em alguns momentos durante os *commits*, adicionávamos comentários semânticos, ou seja, direcionados a quem estava lendo aquela alteração. No exemplo a seguir, o comentário está entre aspas ("adicionando o gitignore").

```
MINGW64:/c/Users/sn1067709/senai-versoes-colaboracoes
SN1067709@ESN000N1241146 MINGW64 ~/senai-versoes-colaboracoes (main)
$ git commit -m "adicionando o gitignore"
[main 20a435b] adicionando o gitignore
 1 file changed, 1 insertion(+)
 create mode 100644 .gitignore
```

Esse comentário serve como lembrete do que foi feito no commit. Assim, ao puxar o histórico, você saberá o que foi feito nesse commit sem precisar abri-lo.

Tenha cuidado ao escrever o comentário. Se no exemplo o comentário fosse apenas "gitignore", você não teria como saber qual a alteração feita no arquivo gitignore (o gitignore foi apagado ou adicionado?) e você teria que abrir o commit para descobrir o que significa o comentário.

É uma boa prática de documentação, que deve estar presente em todos os seus projetos e que facilita o desenvolvimento e a visualização das alterações entre a equipe.

NESTE DESAFIO...

VERSÕES E COLABORAÇÕES | DESAFIO 2



Você conheceu melhor os termos *trunk* e *branch*. Estudou como criar novas *branches* e como unificá-las quando necessário, além de praticar a criação de *tags*.

Os desenvolvimentos colaborativo e paralelo são itens muito importantes durante a sua carreira como desenvolvedor, permitindo atuar em diferentes tarefas, aumentar a capacidade de entrega,

formalizar o processo de desenvolvimento, além de centralizar as informações que são necessárias no projeto.

NO PRÓXIMO DESAFIO...

...você aprenderá a atuar em diferentes ramos de trabalho.

DESAFIO 3

Depois de instalar o Git, você praticou os comandos básicos, como publicar no repositório remoto e ignorar arquivos.

Nesta etapa, para desenvolver o desafio 3, você deverá:

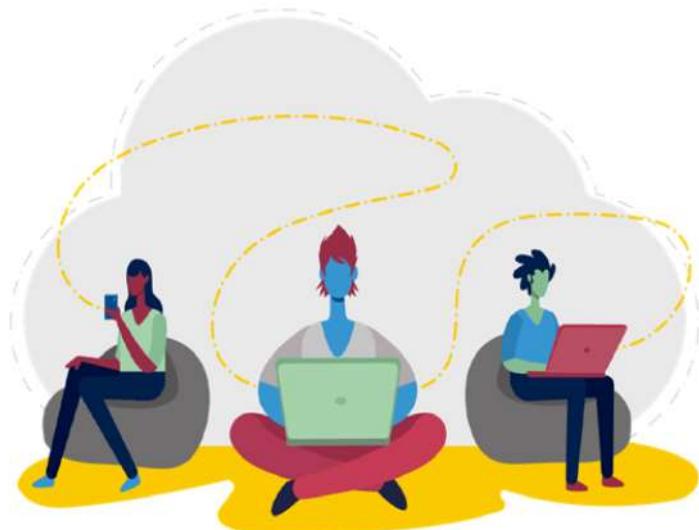
- Realizar a conciliação de alterações em um mesmo repositório.

Para isso, você estudará os seguintes conteúdos:

- *Branchs e tags*
- Correção de erros
- Boas práticas



TRABALHO DE EQUIPE



Um dos pontos importantes do Git é a sua flexibilidade de trabalho, seja para realizar um trabalho individual, em equipes pequenas ou grandes. Por meio do versionamento e rastreamento, é possível trabalharmos de maneira paralela em tarefas distintas e entregarmos itens de modo mais rápido durante o processo. Para trabalhar várias tarefas e trechos de códigos simultaneamente, usamos *branch* e *trunk*.

TRUNK E BRANCH

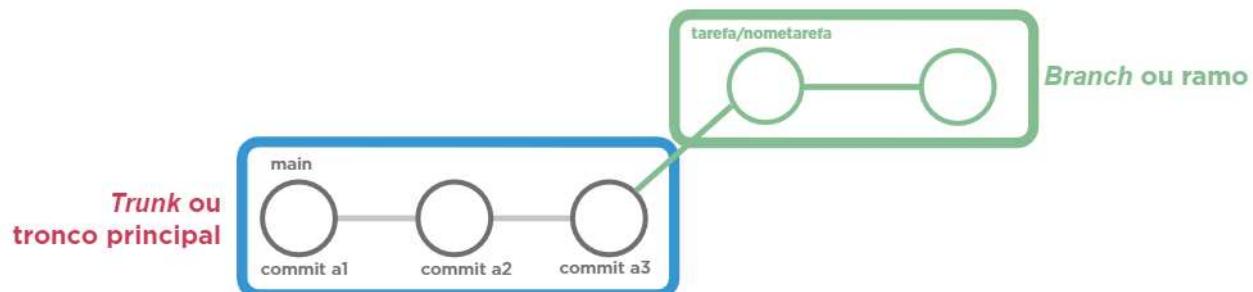
No primeiro e no segundo desafio, todas as alterações estavam sendo feitas no mesmo ramo de trabalho. Agora, depois de conhecer os termos *trunk* e *branch*, você conhecerá mais a fundo o que acontece quando vários programadores editam o mesmo arquivo.

Considerando um determinado ponto do código-fonte, nós podemos criar uma cópia desse trabalho e atuarmos nas tarefas de maneira paralela.



O QUE ISSO SIGNIFICA NA PRÁTICA?

Na prática, para cada atividade que formos executar, atuaremos em um ramo distinto e, posteriormente, uniremos o trabalho dos ramos com o tronco principal.



CLONANDO UM REPOSITÓRIO

No desafio 2, você criou uma pasta chamada senai-versoes-colaboracoes. Nessa pasta, o repositório foi criado e todo o versionamento de código foi feito.

Para darmos continuidade ao nosso processo, devemos garantir que todas as informações estão atualizadas com o repositório remoto. Para isso, podemos usar dois comandos:

- **git pull:** para baixar as atualizações do repositório remoto; ou
- **git clone:** para clonar a pasta do repositório remoto.

git pull: para baixar as atualizações do repositório remoto

Você já praticou esse comando antes, basta repetir os mesmos passos.

Abra a pasta do repositório local (no nosso exemplo, senai-versoes-colaboracoes), clique com o botão direito do mouse na área em branco dentro da pasta e inicialize o Git Bash clicando em Git Bash Here. O terminal abrirá e você pode digitar o comando **git pull**.

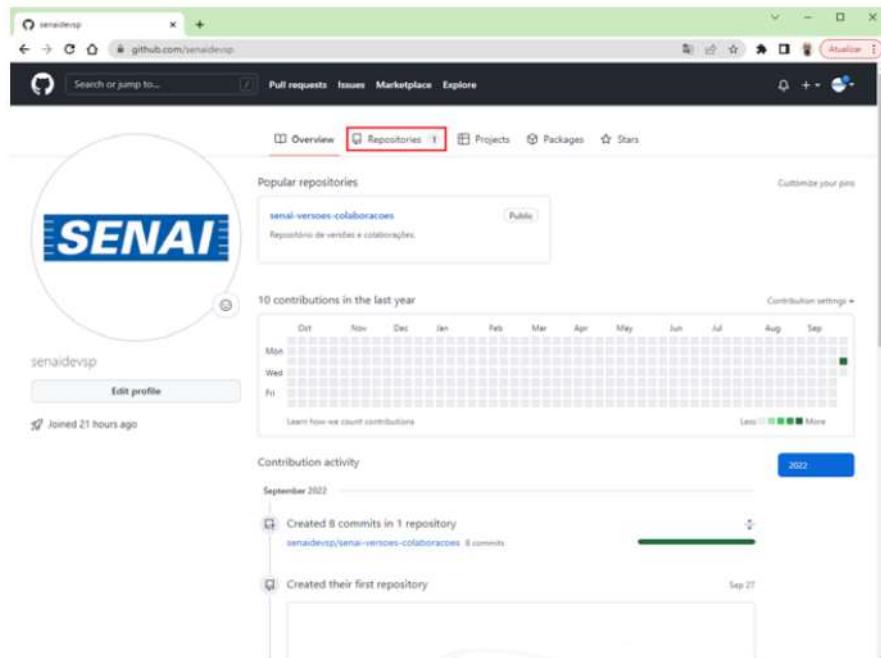
```
MINGW64:/c/Users/sn1067709/senai-versoes-colaboracoes
$ git pull
remote: Enumerating objects: 4, done.
remote: Counting objects: 100% (4/4), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), 703 bytes | 15.00 KiB/s, done.
From https://github.com/senaidevsp/senai-versoes-colaboracoes
  4825f8a..57c6c8e main      -> origin/main
Updating 4825f8a..57c6c8e
Fast-forward
 README.md |  3 +++
 1 file changed, 3 insertions(+)
 create mode 100644 README.md

SN1067709@ESN000N1241146 MINGW64 ~/senai-versoes-colaboracoes (main)
$ -
```

git clone: para clonar a pasta do repositório remoto

Caso você não tenha o repositório/pasta em seu computador, você pode clonar (copiar) o conteúdo do repositório remoto em sua máquina local.

- Acesse o repositório remoto através do link "github.com/seu-usuário". No nosso exemplo, o link é "<http://github.com/hstrada>"
Clique em "**repositories**".



Versionamento - Desafio 3

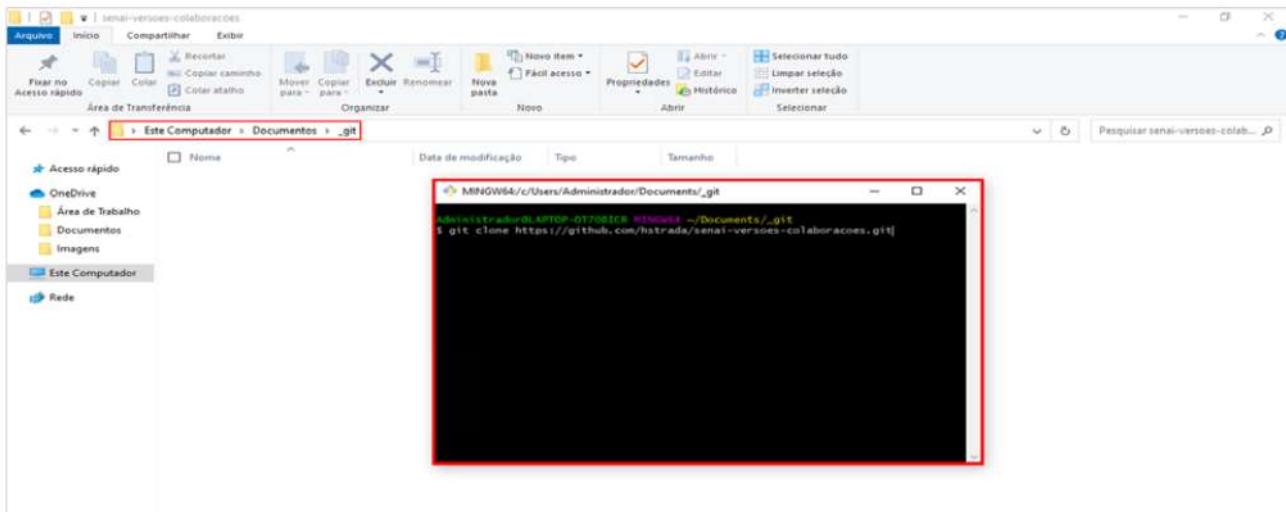
- Clique no repositório do desafio 2. No nosso exemplo, "**senai-versoes-colaboracoes**".

A screenshot of a web browser displaying a GitHub repository page. The URL in the address bar is `github.com/senaidevsp?tab=repositories`. The page shows a list of repositories under the user `senaidevsp`. One repository, `senai-versoes-colaboracoes`, is highlighted with a red box. The repository details include the name, description ("Repositório de versões e colaborações."), and the last update time ("Updated 11 hours ago").

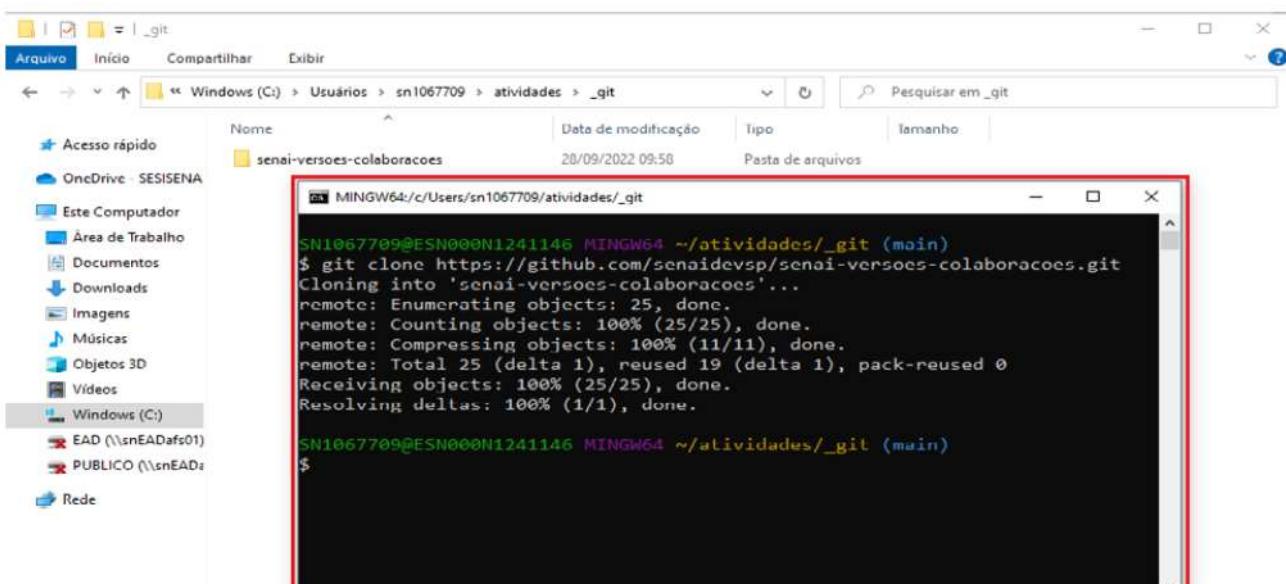
- Clique em "**code**" e copie a URL do repositório.

A screenshot of a web browser displaying the same GitHub repository page for `senai-versoes-colaboracoes`. This time, the "Code" button in the top navigation bar is highlighted with a red box. Below it, the "Clone" section shows a copy link: `https://github.com/senaidevsp/senai-versoes-colaboracoes`, which is also highlighted with a red box.

- Em sua pasta local .git, abra o Git bash, clicando com o botão direito do mouse.
- No terminal, digite o comando `git clone "https://github.com/hstrada/senai-versoes-colaboracoes.git"`



- No desafio 2, você "empurrou" **git pull** as informações da sua máquina local para o repositório remoto. Aqui você está copiando as informações do repositório remoto para a sua máquina.



CRIANDO UMA BRANCH

Com o repositório atualizado e com as informações do tronco principal também atualizadas, inicie a criação do primeiro ramo de trabalho.

Considerando um determinado ponto do código-fonte, nós podemos criar uma cópia desse trabalho e atuarmos nas tarefas de maneira paralela.

Para isso, os passos serão:

- criar uma *branch* com **git checkout -b tarefa/minha-primeira-branch**;
- fazer uma alteração nessa *branch*;
- verificar no repositório remoto.

git checkout -b tarefa/minha-primeira-branch: criar uma branch



Para realizar esta ação, dentro do repositório local senai-versoes-colaboracoes, digite o comando **git checkout -b tarefa/minha-primeira-branch** no terminal.

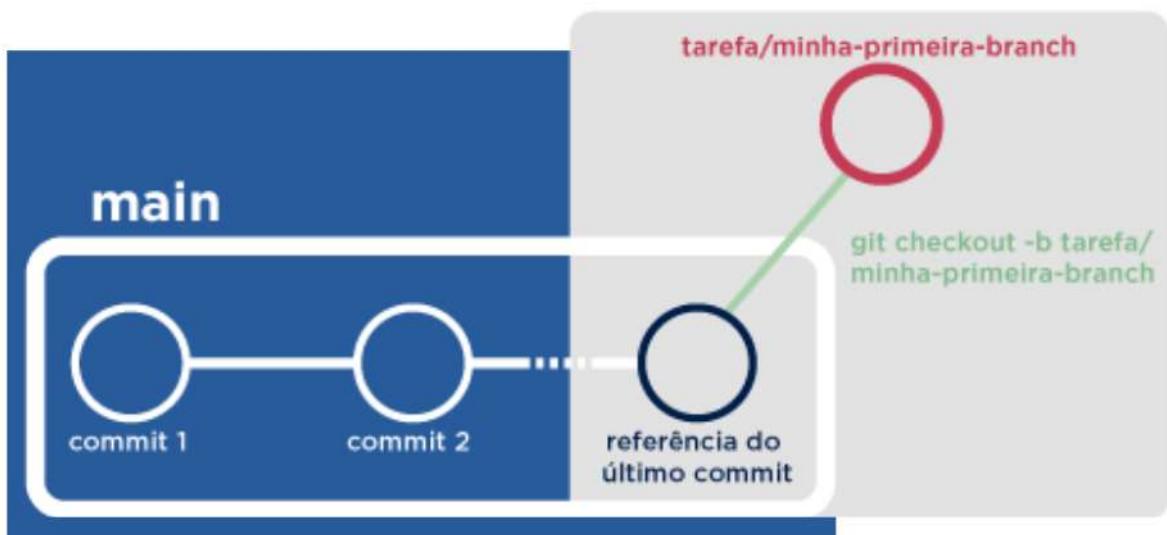
O parâmetro "**-b**" é utilizado para informar que desejamos criar uma *branch* e que esse ramo ainda não existe.

Note que o ramo de trabalho mudou, passando do principal *main* para a *branch* criada, chamado "tarefa/minha-primeira-branch"

```
MINGW64:/c/Users/sn1067709/atividades/_git/senai-versoes-colaboracoes
SN1067709@ESN000N1241146 MINGW64 ~/atividades/_git/senai-versoes-colaboracoes
(main)
$ git checkout -b tarefa/minha-primeira-branch
Switched to a new branch 'tarefa/minha-primeira-branch'

SN1067709@ESN000N1241146 MINGW64 ~/atividades/_git/senai-versoes-colaboracoes
(tarefa/minha-primeira-branch)
$ -
```

O esquema a seguir resume a criação da nova *branch*.



Importante!

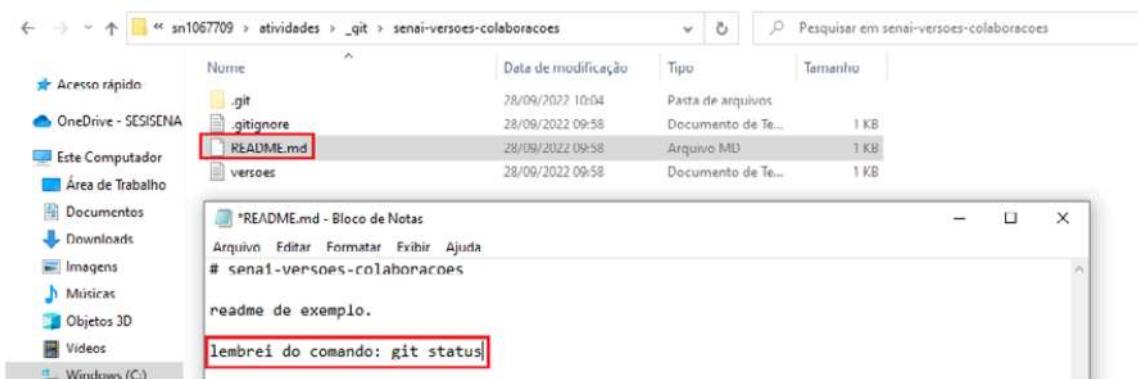
A partir deste momento, todas as alterações realizadas estarão na *branch*, ou seja, no ramo criado, e não mais no tronco (trunk) principal e não impactarão diretamente no ponto principal do nosso código-fonte.



FAZER UMA ALTERAÇÃO NESSA BRANCH

Estamos na *branch* criada. Isso significa que as alterações feitas irão acontecer nesse ramo. Vamos praticar alterando o arquivo README.md e depois verificando no repositório remoto.

- Na pasta senai-versoes-colaboracoes, abra o README.md com o bloco de notas. Adicione o comentário: "**lembrei do comando: git status**" e salve-o.



- Agora abra o terminal do Git Bash, clicando com o botão direito dentro da pasta "**senai-versoes-colaboracoes**".

Veja a alteração com o **git status**, adicione esse arquivo na área de *staging* com **git add .** e faça o commit com o **git commit -m "comentário"**.

```
SN1067709@ESN000N1241146 MINGW64 ~/atividades/_git/senai-versoes-colaboracoes
(tarefa/minha-primeira-branch)
$ git status
On branch tarefa/minha-primeira-branch
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
    modified:   README.md

no changes added to commit (use "git add" and/or "git commit -a")

SN1067709@ESN000N1241146 MINGW64 ~/atividades/_git/senai-versoes-colaboracoes
(tarefa/minha-primeira-branch)
$ git add .

SN1067709@ESN000N1241146 MINGW64 ~/atividades/_git/senai-versoes-colaboracoes
(tarefa/minha-primeira-branch)
$ git commit -m "adicionando comando git status"
[tarefa/minha-primeira-branch fa64126] adicionando comando git status
 1 file changed, 2 insertions(+)

SN1067709@ESN000N1241146 MINGW64 ~/atividades/_git/senai-versoes-colaboracoes
(tarefa/minha-primeira-branch)
$
```

- Agora "empurre" a alteração para o repositório remoto com o **git push**. Como não estamos na main, lembre-se de digitar o comando completo **git push origin tarefa/minha-primeira-branch**.

As alterações estão sendo feitas no ramo criado, não impactando o ramo principal, como veremos a seguir.

```
SN1067709@ESN000N1241146 MINGW64 ~/atividades/_git/senai-versoes-colaboracoes
(tarefa/minha-primeira-branch)
$ git push origin tarefa/minha-primeira-branch
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 8 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 389 bytes | 77.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
remote:
remote: Create a pull request for 'tarefa/minha-primeira-branch' on GitHub by
remote: visiting:
remote:     https://github.com/senaidevsp/senai-versoes-colaboracoes/pull/ne
w/tarefa/minha-primeira-branch
remote:
To https://github.com/senaidevsp/senai-versoes-colaboracoes.git
 * [new branch]      tarefa/minha-primeira-branch -> tarefa/minha-primeira-br
anch

SN1067709@ESN000N1241146 MINGW64 ~/atividades/_git/senai-versoes-colaboracoes
(tarefa/minha-primeira-branch)
$
```

VERIFICAR REPOSITÓRIO REMOTO

Agora vamos verificar no repositório remoto a alteração. Você vai perceber que as alterações no ramo principal de trabalho, que mantém nosso código-fonte, não foram alteradas.

- Em seu navegador, acesse o repositório remoto através do link "<https://github.com/seu-usuário/repositorio-remoto>". No nosso exemplo, "<https://github.com/hstrada/senai-versoes-colaboracoes>".

Note que não há alterações na main.

- Clique no menu ao lado de "main" para visualizar os ramos.

Importante!

Nenhuma alteração feita na *branch* criada impactou diretamente a base do projeto (a *master*). 

Dentro de um projeto, para cada tarefa, geralmente é criada uma *branch*. Assim, o andamento de cada tarefa não impacta o desenvolvimento do ramo principal nem outros arquivo da equipe.

CRIANDO OUTRAS BRANCHS

Como há outras pessoas na equipe trabalhando com o mesmo código-fonte da *main*, há a necessidade de criar uma *branch* para cada tarefa. Assim, cada um pode adicionar suas informações no mesmo arquivo sem afetar o desenvolvimento das tarefas de outros colegas.

Novas *branches* também devem ser criadas, caso você queira testar duas soluções diferentes para a mesma tarefa ou caso você tenha mais de uma tarefa para solucionar.

Para criar novas *branches*, vamos:

- voltar para a *main* com **git checkout main**;
- criar uma nova *branch* com **git checkout -b exemplo-branch**;
- fazer uma alteração nessa **branch**;
- verificar no repositório remoto.

VOLTAR PARA A MAIN COM *git checkout main*

O primeiro passo para criar uma nova *branch* é voltar para a *master* com o comando **git checkout main**.

```
MINGW64:/c/Users/sn1067709/atividades/_git/senai-versoes-colaboracoes
SN1067709@ESN000N1241146 MINGW64 ~/atividades/_git/senai-versoes-colaboracoes
(tarefa/minha-primeira-branch)
$ git checkout main
Switched to branch 'main'
Your branch is up to date with 'origin/main'.

MINGW64:/c/Users/sn1067709/atividades/_git/senai-versoes-colaboracoes
(main)
$
```

Dica!

O comando para criar uma nova *branch* é `git checkout -b "nome da branch"`, lembra-se? E o parâmetro `-b` indica a criação de uma nova *branch*.



Quando voltamos para a trunk (chamada master), usamos também o comando `git checkout`, mas não é necessário informar o parâmetro `-b`, pois a trunk já existe.

CRIAR UMA NOVA BRANCH COM `git checkout -b exemplo-branch`

Vamos retomar o comando para criar uma nova *branch*, mudando apenas o nome. Portanto, o comando que devemos digitar no terminal é **git checkout -b exemplo-branch**.

```
MINGW64:/c/Users/sn1067709/atividades/_git/senai-versoes-colaboracoes
SN1067709@ESN000N1241146 MINGW64 ~/atividades/_git/senai-versoes-colaboracoes
(tarefa/minha-primeira-branch)
$ git checkout main
Switched to branch 'main'
Your branch is up to date with 'origin/main'.

MINGW64:/c/Users/sn1067709/atividades/_git/senai-versoes-colaboracoes
(main)
$ git checkout -b exemplo-branch
Switched to a new branch 'exemplo-branch'

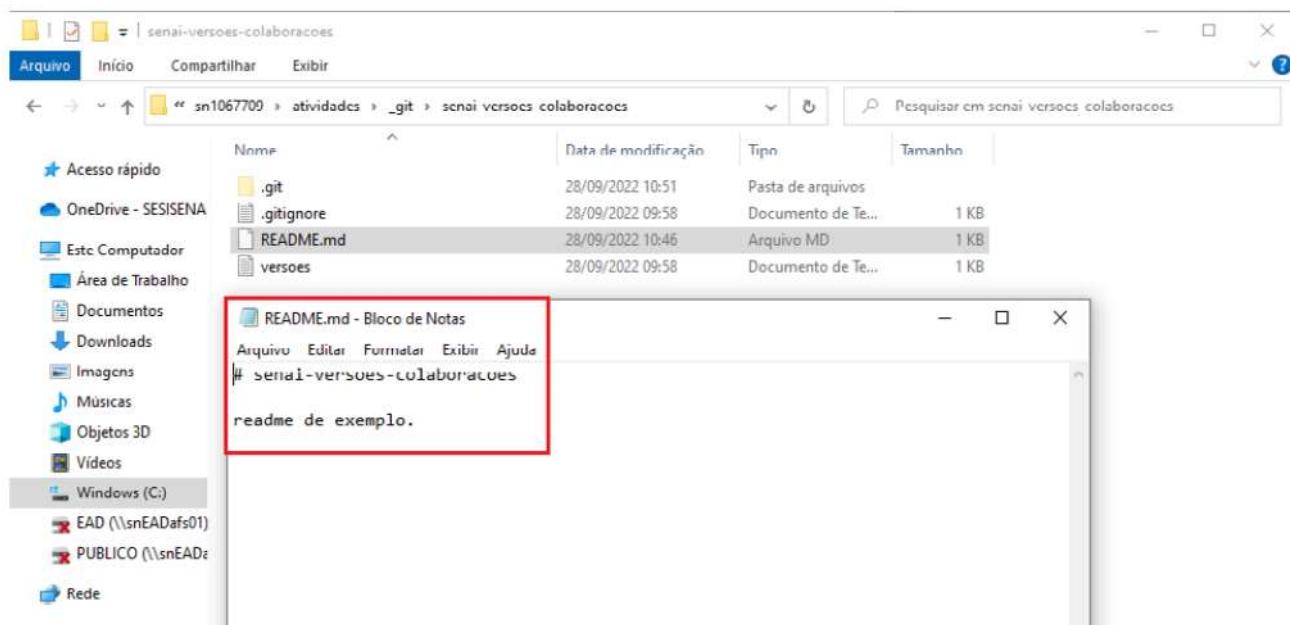
MINGW64:/c/Users/sn1067709/atividades/_git/senai-versoes-colaboracoes
(exemplo-branch)
```

FAZER UMA ALTERAÇÃO NESSA BRANCH

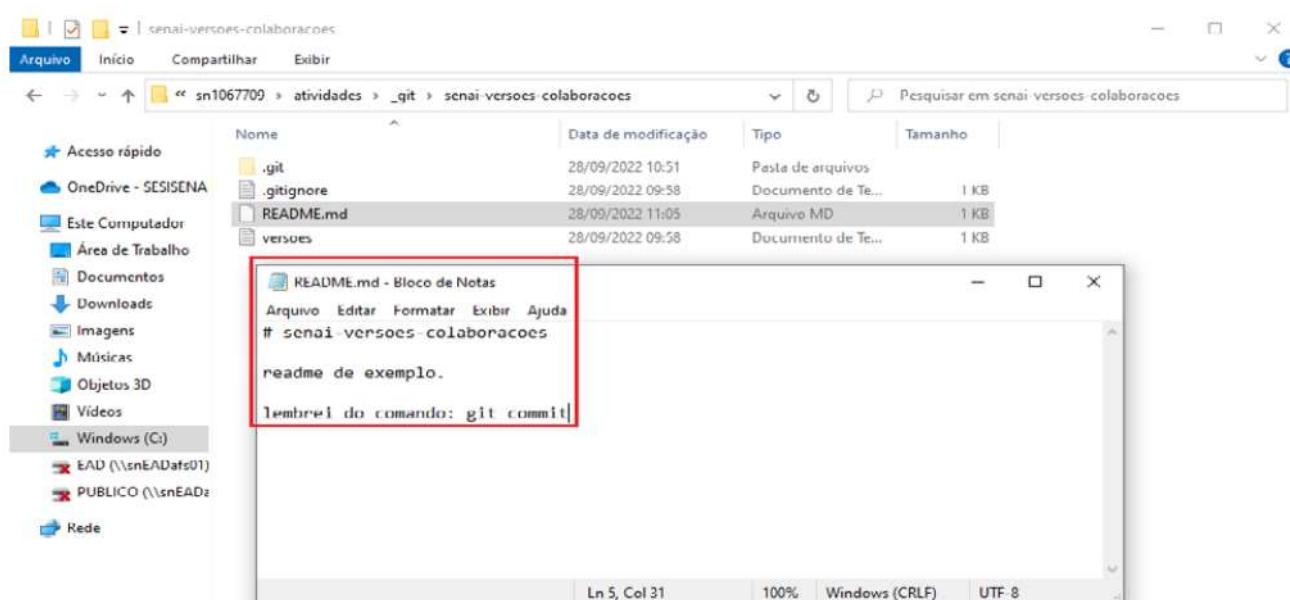
Agora estamos na *branch* exemplo-branch e, portanto, as alterações feitas aqui não afetarão nem a main nem a tarefa/minha-primeira-branch.

- Abra o arquivo "README.md".

Note que as alterações feitas na *branch* tarefa não estão presentes aqui, pois criamos a *branch* exemplo a partir da main. As alterações da *branch* tarefa não afetam a main.



- Edite o "README.md" com o bloco de notas, escrevendo "**lembrei do comando: git commit**" e salve-o.



- Abra o terminal do Git Bash (botão direito dentro da pasta) e execute:
- **git add .** (adiciona ao *staging*);
- **git commit -m "adicionando o comando git commit"** (salva no histórico);
- **git push origin exemplo-branch** (envia para o repositório remoto).

```
MINGW64:/c/Users/sn1067709/atividades/_git/senai-versoes-colaboracoes
SN1067709@ESN000N1241146 MINGW64 ~/atividades/_git/senai-versoes-colaboracoes
(exemplo-branch)
$ git add .

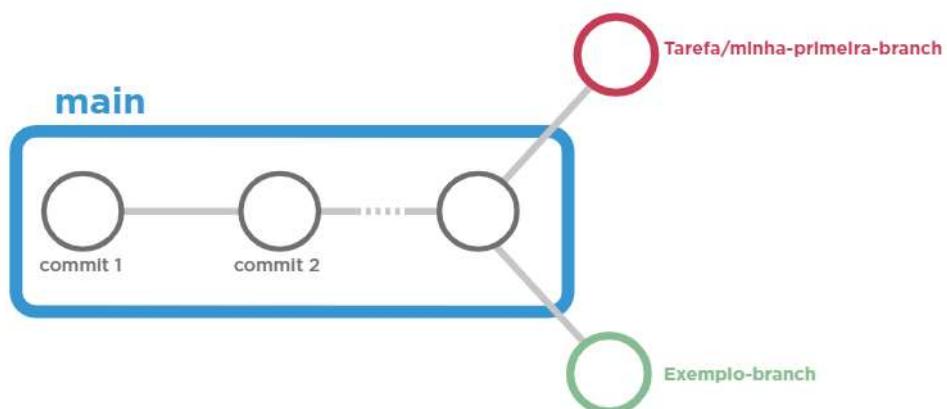
SN1067709@ESN000N1241146 MINGW64 ~/atividades/_git/senai-versoes-colaboracoes
(exemplo-branch)
$ git commit -m "adicionando o comando git commit"
[exemplo-branch dbf6ec0] adicionando o comando git commit
 1 file changed, 2 insertions(+)

SN1067709@ESN000N1241146 MINGW64 ~/atividades/_git/senai-versoes-colaboracoes
(exemplo-branch)
$ git push origin exemplo-branch
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 8 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 389 bytes | 129.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
remote:
remote: Create a pull request for 'exemplo-branch' on GitHub by visiting:
remote:     https://github.com/senaidevsp/senai-versoes-colaboracoes/pull/new/exemplo-branch
remote:
To https://github.com/senaidevsp/senai-versoes-colaboracoes.git
 * [new branch]      exemplo-branch -> exemplo-branch

SN1067709@ESN000N1241146 MINGW64 ~/atividades/_git/senai-versoes-colaboracoes
(exemplo-branch)
```

- O fluxo atual de ramos está como a imagem ao lado.

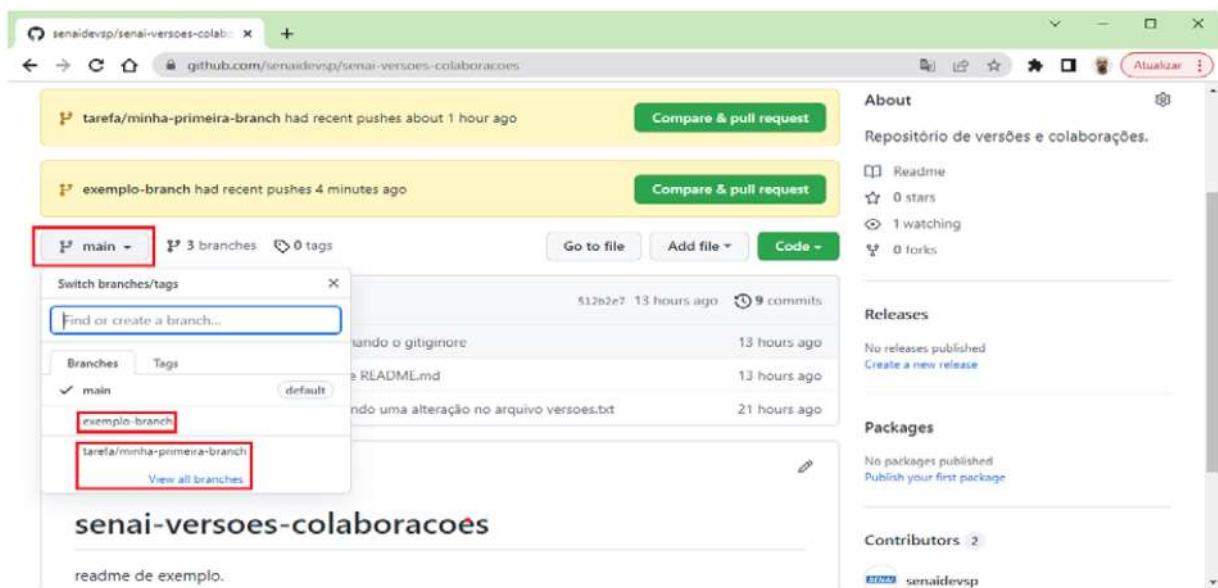
As tarefas feitas separadamente em cada *branch* podem ser unificadas depois.



VERIFICAR NO REPOSITÓRIO REMOTO

Acesse o repositório remoto no seu navegador, com o link "<https://github.com/seu-usuário/repositorio-remoto>" (no nosso exemplo "<https://github.com/hstrada/senai-versoes-colaboracoes>").

Verifique as três *branches* que existem no projeto. A *main* com a base estável do projeto e duas *branches* de “trabalho em desenvolvimento dos colaboradores”.



UNIFICANDO OS RAMOS



Você criou duas *branches* com dois conteúdos distintos, pois cada uma tem uma atualização diferente do mesmo arquivo. O Git fornece recursos para unificação desse trabalho.

Agora estamos na *branch* exemplo e vamos unificar o conteúdo com o da *branch* tarefa em um mesmo ramo de trabalho.

Você pode realizar esta ação por meio de dois comandos distintos:

- Realizar o download das informações fornecidas pela outra *branch*, com **git pull origin tarefa/minha-primeira-branch**; ou
- Fazer a mesclagem com a outra branch, com **git merge tarefa/minha-primeira-branch**.

PENSE NISSO...

Lembre-se que fizemos alterações no mesmo arquivo em cada *branch*. O que vai ocorrer?

RESOLVENDO CONFLITOS

Imagine que, como você, os outros membros da equipe estão desenvolvendo suas tarefas em suas respectivas *branches*. Em algum momento, todos os ramos serão unificados para que o trabalho possa ser entregue em um único arquivo. Como ajustar os conflitos nessa unificação?

No nosso exemplo, ajustamos o mesmo ponto do arquivo em duas *branches* separadas e o Git nos informa que existe um conflito entre as alterações.



Você poderá visualizar algo como a imagem a seguir, com:

- comando1.
- execução do comando2.
- aviso de conflito e local de conflito3.
- aviso de falha na execução do comando4.

```

MINGW64:/c/Users/sn1067709/atividades/_git/senai-versoes-colaboracoes
SN1067709@ESN000N1241146 MINGW64 ~/atividades/_git/senai-versoes-colaboracoes
(exemplo-branch)
$ git merge tarefa/minha-primeira-branch 1
Auto-merging README.md 2
CONFLICT (content): Merge conflict in README.md 3
Automatic merge failed; fix conflicts and then commit the result. 4
SN1067709@ESN000N1241146 MINGW64 ~/atividades/_git/senai-versoes-colaboracoes
(exemplo-branch|MERGING)
$ -

```

O que é necessário ser feito é informar qual das atualizações nós desejamos manter em nosso arquivo.

Existem três maneiras básicas de resolver:

1. Incluir somente a sua alteração local.
2. Aceitar somente o que a outra *branch* alterou.
3. Aceitar as duas alterações.

Nesse exemplo, vamos optar pela terceira maneira.

- Abra o arquivo "**README.md**".

Note o aviso marcando o conflito no arquivo.

```

README.md - Bloco de Notas
Arquivo Editar Formatar Exibir Ajuda
# senai-versoes-colaboracoes

readme de exemplo.

<<<<< HEAD
lembrei do comando: git commit
=====
lembrei do comando: git status
>>>>> tarefa/minha-primeira-branch

```

- Remova os conflitos existentes e salve o arquivo.



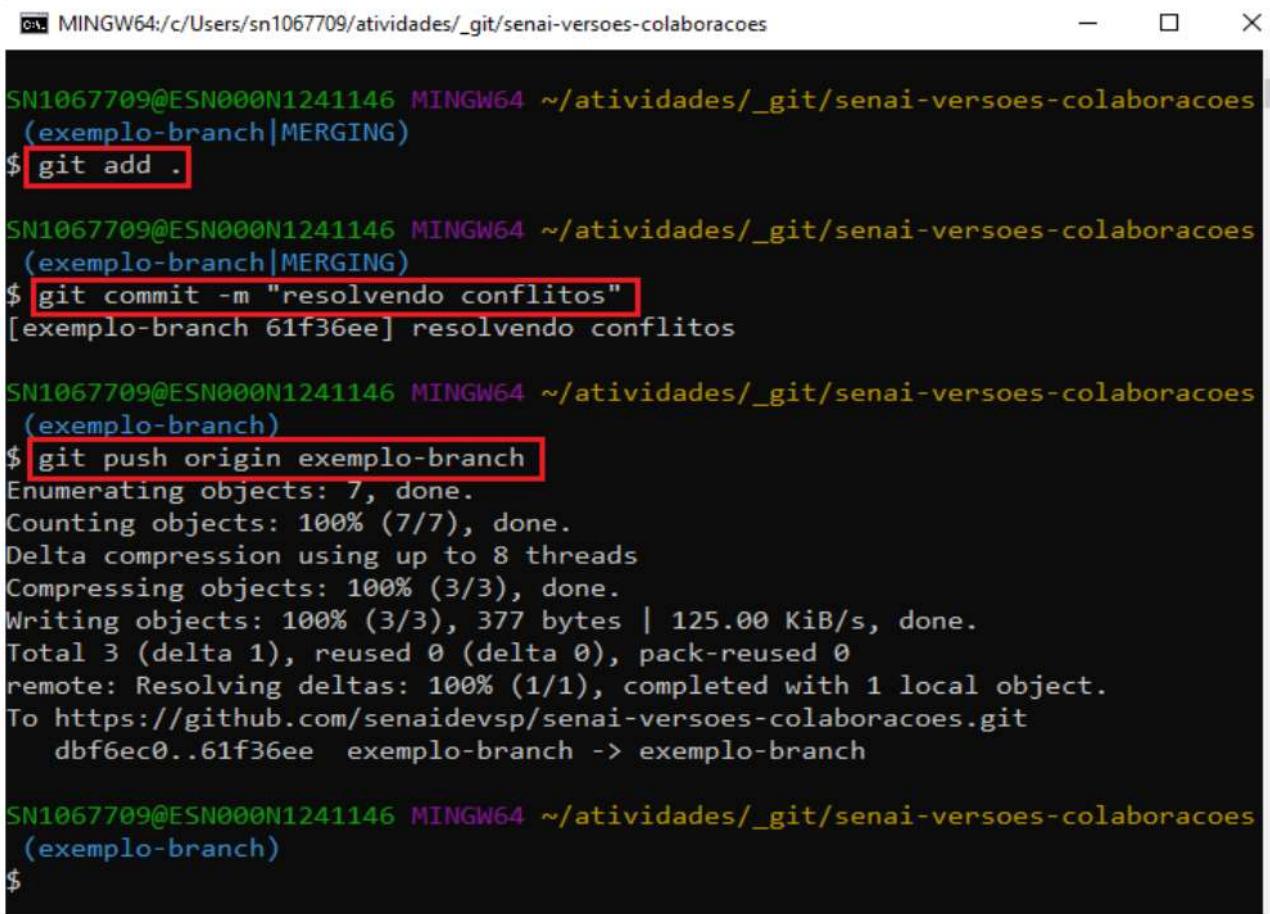
```
README.md - Bloco de Notas
Arquivo Editar Formatar Exibir Ajuda
# senai-versoes-colaboracoes

readme de exemplo.

lembrei do comando: git commit
lembrei do comando: git status
```

- Abra o terminal no seu repositório local e:

- » adicione o arquivo para a área de preparo;
- » faça o commit; e
- » publique no repositório remoto.



```
MINGW64:/c/Users/sn1067709/atividades/_git/senai-versoes-colaboracoes
SN1067709@ESN000N1241146 MINGW64 ~/atividades/_git/senai-versoes-colaboracoes
(exemplo-branch|MERGING)
$ git add .

SN1067709@ESN000N1241146 MINGW64 ~/atividades/_git/senai-versoes-colaboracoes
(exemplo-branch|MERGING)
$ git commit -m "resolvendo conflitos"
[exemplo-branch 61f36ee] resolvendo conflitos

SN1067709@ESN000N1241146 MINGW64 ~/atividades/_git/senai-versoes-colaboracoes
(exemplo-branch)
$ git push origin exemplo-branch
Enumerating objects: 7, done.
Counting objects: 100% (7/7), done.
Delta compression using up to 8 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 377 bytes | 125.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/senaidevsp/senai-versoes-colaboracoes.git
  dbf6ec0..61f36ee  exemplo-branch -> exemplo-branch

SN1067709@ESN000N1241146 MINGW64 ~/atividades/_git/senai-versoes-colaboracoes
(exemplo-branch)
$
```

Para visualizarmos que ambas as alterações de dois ramos distintos foram unidas, vamos abrir o "**github.com**" em nosso repositório e visualizar as modificações em nossa *branch*.

The screenshot shows a GitHub repository page for 'senaidesp/senai-versoes-colaboracoes'. The 'Code' tab is selected. In the top navigation bar, the dropdown menu 'exemplo-branch' is highlighted with a red box. Below it, the repository stats show 3 branches and 0 tags. The commit history lists three commits: 'Mudando o gitignore' (2 hours ago), 'Resolvendo conflitos.' (15 hours ago), and 'Realizando uma alteração no arquivo versoes.txt' (23 hours ago). The README.md file contains the text 'senai-versoes-colaboracoes' and 'readme de exemplo.', with the last line 'lembrei do comando: git commit lembrei do comando: git status' highlighted with a red box. On the right side, there are sections for About, Releases, Packages, and Contributors.

Dica!

É possível criar uma nova *branch* a partir de outra *branch*, basta não voltar para a trunk ao criar um novo ramo. Porém, devemos ter atenção extra para fazer o merge em ramificações distantes da trunk.



Saiba mais

Existem algumas ferramentas que podem nos auxiliar na visualização de conflitos, como o Git GUI com o meld. Para saber mais, acesse o link: <https://gist.github.com/kjlubick/5a49a3ae5f39ae359997> e continue a pesquisar e estudar.



DEMARCANDO PONTOS NO CÓDIGO: TAGS



Quando desenvolvemos projetos maiores e adicionamos novas tarefas, podemos demarcar pontos no decorrer do desenvolvimento do código que representa, por exemplo, a inclusão de uma tarefa ou um item crucial que foi resolvido.

Essa marcação é chamada de *tag* e é importante para delimitar pontos estratégicos, como *commits* que formam o contexto na nossa aplicação.

Importante!

Ao demarcar um ponto com uma *tag* na master, por exemplo, todo o código-fonte até ali é automaticamente compactado num arquivo dentro daquela *tag*. Se a *tag* for criada em uma *branch*, o código compactado será da *branch*, desde a sua criação até o ponto demarcado pela *tag*.



Para exemplificar, vamos criar uma *tag* na master no ponto atual, pelo terminal em seu repositório local.

- No terminal do seu repositório local, crie uma *tag* para marcar o ponto atual com o comando **git tag -a v1.0 -m "minha primeira tag"**.

```
MINGW64:c:/Users/sn1067709/atividades/_git/senai-versoes-colaboracoes
SN1067709@ESN000N1241146 MINGW64 ~/atividades/_git/senai-versoes-colaboracoes
(main)
$ git tag -a v1.0 -m "minha primeira tag"
SN1067709@ESN000N1241146 MINGW64 ~/atividades/_git/senai-versoes-colaboracoes
(main)
$ -
```

- Para ver as tags criadas, execute o comando **git tag**.

```
MINGW64:/c/Users/sn1067709/atividades/_git/senai-versoes-colaboracoes
SN1067709@ESN000N1241146 MINGW64 ~/atividades/_git/senai-versoes-colaboracoes
(main)
$ git tag
v1.0

SN1067709@ESN000N1241146 MINGW64 ~/atividades/_git/senai-versoes-colaboracoes
(main)
$ -
```

- Para ver informações sobre uma tag específica, como a v1.0, digite o comando **git show v1.0**.

```
MINGW64:/c/Users/sn1067709/atividades/_git/senai-versoes-colaboracoes
SN1067709@ESN000N1241146 MINGW64 ~/atividades/_git/senai-versoes-colaboracoes
(main)
$ git show v1.0
tag v1.0
Tagger: senaidevsp <senaidevsp@gmail.com>
Date:   Wed Sep 28 13:27:58 2022 -0300

minha primeira tag

commit 61f36ee09cb20090e9ef95859adfd844ea15b6f1 (HEAD -> main, tag:
v1.0, origin/exemplo-branch)
Merge: dbf6ec0 fa64126
Author: senaidevsp <senaidevsp@gmail.com>
Date:   Wed Sep 28 11:44:32 2022 -0300

    resolvendo conflitos

diff --cc README.md
index 6a6019e,fe9d175..7d888a3
--- a/README.md
+++ b/README.md
@@@ -2,4 -2,4 +2,6 @@@
```

- Vamos "empurrar", ou seja, publicar as tags no repositório remoto com o comando **git push origin --tags**.

```
MINGW64:c/Users/sn1067709/atividades/_git/senai-versoes-colaboracoes
SN1067709@ESN000N1241146 MINGW64 ~/atividades/_git/senai-versoes-colaboracoes
(main)
$ git push origin --tags
Enumerating objects: 1, done.
Counting objects: 100% (1/1), done.
Writing objects: 100% (1/1), 162 bytes | 32.00 KiB/s, done.
Total 1 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/senaidevsp/senai-versoes-colaboracoes.git
 * [new tag]          v1.0 -> v1.0

SN1067709@ESN000N1241146 MINGW64 ~/atividades/_git/senai-versoes-colaboracoes
(main)
$ -
```

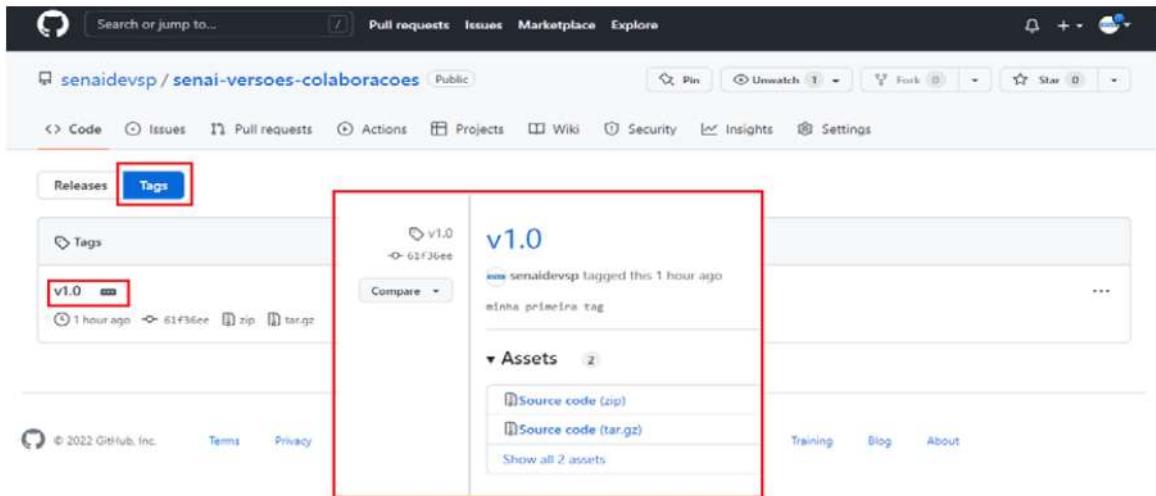
Agora acesse o repositório remoto pelo seu navegador. No github, encontre a tag criada e publicada.

The screenshot shows a GitHub repository page for 'senaidevsp / senai-versoes-colaboracoes'. The 'Code' tab is selected. In the top right, there's a 'Releases' section with a red box around the '1 tags' link. Below it, there's a 'Create a new release' button. The main content area shows a list of commits and files. A commit from 'senaidevsp' titled 'resolvendo conflitos' is visible, along with '.gitignore', 'README.md', and 'versoes.txt' files.

- Clique em releases para ter acesso às informações que foram adicionadas a esse ponto atual de nosso código.

This screenshot is identical to the one above, showing the same GitHub repository page for 'senaidevsp / senai-versoes-colaboracoes'. The 'Releases' section is highlighted with a red box around the '1 tags' link. The main content area shows the same list of commits and files as the previous screenshot.

- Clique em v1.0 para visualizar o que está na *tag*.



Importante!

Como comentamos sobre as tarefas desenvolvidas em todos os *commits* até a *tag* criada de identificação, os arquivos compactados são entregas/lançamentos parciais que irão compor o nosso projeto ao longo do tempo.



NESTE DESAFIO...

VERSIONAMENTO | DESAFIO 3



Você conheceu melhor os termos *trunk* e *branch*. Estudou como criar novas *branches* e como unificá-las quando necessário, além de praticar a criação de *tags*.

Os desenvolvimentos colaborativo e paralelo são itens muito importantes durante a sua carreira como programador, permitindo-lhe atuar em diferentes tarefas, aumentar a capacidade de entrega, formalizar o processo de desenvolvimento, além de centralizar as informações que são necessárias no projeto.

PARA CONCLUIR...

PARABÉNS, VOCÊ CHEGOU AO FIM DO CONTEÚDO DE VERSIONAMENTO!

Durante seus estudos, você percebeu que salvar o progresso do seu trabalho em versões do seu arquivo é muito importante e que fazer o controle manual dessas versões pode ser impossível. Portanto, um sistema de controle de versionamento é essencial para o desenvolvimento de seu trabalho e da sua equipe.

Dentre os diversos sistemas de controle de versionamento e plataformas de publicação online, foram escolhidos o Git (chamado de Git Bash, na versão para os sistemas operacionais Windows) e o GitHub. Baseado nessas escolhas, você conheceu e praticou comandos básicos, como salvar e publicar alterações, além de criar *branches* e *tags*. Estudou também como unificar *branches* e resolver conflitos na mesclagem.

Continue estudando e praticando, sempre!

Bom trabalho e até breve!

REFERÊNCIAS

ATLASSIAN. **Git Bash.** Disponível em: <<https://www.atlassian.com/br/git/tutorials/git-bash>>. Acesso em: 22 jan. 2021.

ATLASSIAN. **O que é controle de versão.** Disponível em: <<https://www.atlassian.com/br/git/tutorials/what-is-version-control>>. Acesso em: 22 jan. 2021.

BURGDORF, Christoph. **The anatomy of a Git commit.** 18 nov. 2014. em: <<https://blog.thoughtram.io/git/2014/11/18/the-anatomy-of-a-git-commit.html>>. Acesso em: 22 jan. 2021.

DEVMEDIA. **O que é controle de versão?** Disponível em: <<https://www.devmedia.com.br/sistemas-de-controle-de-versao/24574#:~:text=Atualmente%2C%20os%20sistemas%20de%20controle,%20baseados%20na%20arquitetura%20cliente%2Dservidor>>. Acesso em: 22 jan. 2021.

ESTEVES, Raul. **Como fazer um README.md bonitão.** 29 ago. 2018. Disponível em: <<https://medium.com/@raullesteves/github-como-fazer-um-readme-md-bonit%C3%A3o-c85c8f154f8>>. Acesso em: 22 jan. 2021.

GIT. **Download.** Disponível em: <<https://git-scm.com/downloads>>. Acesso em: 22 jan. 2021.

GITHUB GIST. **Fórum.** installing Meld. Disponível em: <<https://gist.github.com/kjlubick/5a49a3ae5f39ae359997>>. Acesso em: 22 jan. 2021.

GITHUB. **Site.** Disponível em: <<https://github.com/>>. Acesso em: 22 jan. 2021.

INFOESCOLA. **Sistema de informação centralizado.** Disponível em: <<https://www.infoescola.com/informatica/sistema-de-informacao-centralizado/>>. Acesso em: 22 jan. 2021.

MARINHO, Thiago. **O que é README e porque ele é tão importante.** Disponível em: <<https://blog.rocketseat.com.br/o-que-e-readme-e-porque-e-tao-importante/>>. Acesso em: 22 jan. 2021.

PINHEIRO, Fagner. **Criando um README para seu perfil do GitHub.** 26 de ago. de

2020. Disponível em: <<https://www.treinaweb.com.br/blog/criando-um-readme-para-seu-perfil-do-github/>>. Acesso em: 22 jan. 2021.

ROCKETSEAT. **Como usar Git e Github na prática:** Guia para iniciantes. Mayk Brito. Vídeo, 33'43". 27 ago. 2019. Disponível em: <https://www.youtube.com/watch?t=274&v=2alg7MQ6_sl&feature=youtu.be>. Acesso em: 22 jan. 2021.

TED. **David Keller.** Como construir sua confiança criativa. 11'46". Disponível em: <<https://www.youtube.com/watch?v=16p9YRF0l-g>>. Acesso em: 29 de dez. de 2020.

Créditos

CONFEDERAÇÃO NACIONAL DA INDÚSTRIA -CNI	SENAI - DEPARTAMENTO REGIONAL DE SÃO PAULO
<i>Robson Braga de Andrade</i> Presidente	<i>Ricardo Figueiredo Terra</i> Diretoria Regional
DIRETORIA DE EDUCAÇÃO E TECNOLOGIA - DIRET	<i>Cassia Regina Souza da Cruz</i> Gerência de Educação
<i>Rafael Esmeraldo Lucchesi Ramacciotti</i> Diretor de Educação e Tecnologia	<i>Izabel Rego de Andrade</i> Supervisão do Centro SENAI de Tecnologias Educacionais
SERVIÇO NACIONAL DE APRENDIZAGEM INDUSTRIAL – SENAI Conselho Nacional	<i>Claudia Baroni Savini Ferreira</i> Coordenação do Desenvolvimento do Curso
<i>Robson Braga de Andrade</i> Presidente	<i>Helena Strada Franco de Souza</i> Elaboração de Conteúdo
SENAI - Departamento Nacional <i>Rafael Esmeraldo Lucchesi Ramacciotti</i> Diretor-Geral	<i>Adilson Moreira Damasceno</i> Orientação de Práticas de Educação a Distância
<i>Gustavo Leal Sales Filho</i> Diretor de Operações	<i>Paula Cristina Bataglia Buratini</i> Coordenação da Produção do Curso
SENAI – DEPARTAMENTO NACIONAL UNIDADE DE EDUCAÇÃO PROFISSIONAL E TECNOLÓGICA – UNIEP	<i>Cristina Yurie Takahashi</i> Design Educacional
<i>Felipe Esteves Morgado</i> Gerente Executivo	<i>Luana Dorizo de Melo</i> Diagramação
<i>Luiz Eduardo Leão</i> Gerente de Tecnologias Educacionais	<i>Cleriston Ribeiro de Azevedo</i> <i>Fabiano José de Moura</i> <i>Juliana Rumi Fujishima</i> Ilustrações
<i>Anna Christina Theodora Aun de Azevedo Nascimento</i> <i>Adriana Barufaldi</i> <i>Bianca Starling Rosauro de Almeida</i> <i>Laise Caldeira Pedroso</i> Coordenação Geral de Desenvolvimento dos Recursos Didáticos Nacionais	<i>Camila Ciarini Dias</i> Produção e Edição de Vídeos
	<i>Rafael Santiago Apolinário</i> Programação
	<i>Aldo Toma Junior</i> Web Design