

Mayke Franklin da Cruz Santos
Edwar Saliba Júnior

Manual Básico de Git / GitHub

Paracatu
Agosto de 2022

Sumário

O que é Git?.....	1
Como instalar?.....	1
GitHub.....	1
Criando uma conta no GitHub.....	2
Hospedando um código-fonte no GitHub.....	2
Criando mecanismos de acesso e/ou para facilitar o acesso.....	11
Configurações locais.....	12
Situações e comandos (passo a passo).....	13
Criando um repositório remoto (resumindo).....	13
Compartilhando projetos / repositórios.....	13
Como visualizar todas as alterações que foram realizadas antes de aplicá-las ao repositório local?.....	14
Fiz alterações em diversos arquivos. Como faço o versionamento?.....	14
Fiz alterações em diversos arquivos. Mas, quero enviar apenas alguns para a <i>stage area</i>	14
Criei diversos <i>branches</i> . Mas, não me lembro de seus nomes.....	14
Preciso criar um novo <i>branch</i> . Como faço?.....	14
Estou em um <i>branch</i> mas preciso mudar para outro.....	15
Preciso remover um <i>branch</i>	15
Foram realizadas alterações em um <i>branch</i> . Como posso verificar a diferença de um <i>branch</i> para outro?.....	15
Como faço para verificar o histórico de versões de um determinado <i>branch</i> ?	15
Gostaria de verificar as mudanças que ocorreram em um determinado <i>commit</i>	15
Alguém pode ter feito modificações no <i>software</i> . Como faço para atualizar a versão do código-fonte que se encontra no meu computador?.....	16
Preciso fazer a fusão/mesclagem (<i>merge</i>) de um <i>branch</i> com o <i>branch</i> principal “main”.....	16
Bibliografia.....	19

O que é Git?

De acordo com Git (2021), é um sistema de controle de versões distribuído, usado principalmente no desenvolvimento de *software*. Mas, pode ser usado para registrar o histórico de edições de qualquer tipo de arquivo. Exemplo: alguns livros digitais são disponibilizados no GitHub e escrito aos poucos publicamente.

O Git foi inicialmente projetado e desenvolvido por Linus Torvalds para o desenvolvimento do kernel¹ GNU/Linux (Linux, para os íntimos!), mas foi adotado por muitos outros projetos.

Cada diretório de trabalho do Git é um repositório com um histórico completo e habilidade total de acompanhamento das revisões, não dependente de acesso a uma rede ou a um servidor central.

Como instalar?

Primeiramente você deverá fazer o *download* do Git em sua página oficial <https://git-scm.com/>. Existem versões para: Linux, macOS e Windows. Sendo que para o Linux, na maioria das distribuições, o *software* já se encontra nos principais repositórios. Bastando para instalá-lo, um simples comando como:

```
sudo apt-get install git (distribuições baseadas em Debian)
```

Mas, que fique claro que o Git é um *software* que opera localmente na máquina que ele foi instalado. Ou seja, todo o controle de versão é feito na sua máquina local. Se por algum motivo você perder o HD da máquina, então, toda a informação contida no Git se perderá também.

Para evitarmos a perda de dados e também para facilitar o trabalho em equipe num mundo totalmente conectado, surgiram diversos tipos de repositórios *on-line* baseados em Git, sendo que alguns destes são gratuitos. Dentre eles podemos citar os mais famosos: GitHub e GitLab.

GitHub

De acordo com GitHub (2021), o GitHub é uma plataforma de hospedagem de códigos-fonte e arquivos com controle de versão usando o Git. Ela permite que programadores, utilitários ou qualquer usuário cadastrado na plataforma contribua em projetos privados e/ou *Open Source* de qualquer lugar do mundo. O GitHub é amplamente utilizado por programadores para divulgação de seus trabalhos ou para que outros programadores contribuam

¹ em computação, o núcleo ou kernel, é o componente central do sistema operacional da maioria dos computadores; ele serve de ponte entre aplicativos e o processamento real de dados feito a nível de *hardware* (NÚCLEO, 2021).

com o projeto, além de promover fácil comunicação através de recursos que relatam problemas ou mesclam repositórios remotos (*issues, pull request*).

Criando uma conta no GitHub

Se você ainda não tem uma conta no GitHub, poderá criá-la acessando o site oficial da plataforma <https://github.com/>.

Após a criação da conta você terá em mãos não só uma ferramenta de controle de versão de última geração, mas também um *back-up on-line* para todos os trabalhos que você hospedar na plataforma.

Hospedando um código-fonte no GitHub

Para fazermos isto, vamos dividir esta tarefa em dois passos que veremos a seguir:

Criando um repositório no GitHub

No GitHub você tem a opção de criar dois tipos de repositório com características um pouco diferentes um do outro. São elas:

- repositório **público**: onde qualquer pessoa poderá ver e alterar o seu código-fonte, ou
- repositório **privado**: onde somente você e os colaboradores do projeto poderão ver códigos-fonte e alterá-los.

Para criar um novo repositório remoto basta entrar na sua conta do GitHub. No canto superior direito da tela aparecerá seu avatar. Caso você ainda não tenha alterado a imagem para uma foto ou outro desenho qualquer, então aparecerá uma imagem alternativa.

Do lado esquerdo do avatar aparecerá um botão com o sinal “+” (seta **vermelha** na Figura 1). Clique neste botão e aparecerá o *menu* suspenso mostrado na Figura 1.

Então, neste *menu* suspenso, basta escolher a opção **New repository** (seta **verde** na Figura 1).

Caso você já esteja na tela de repositórios, então, para criar um novo bastará clicar no botão *New* (seta **azul** na Figura 2).

Você poderá criar um novo repositório no GitHub, também, apenas acessando o link <https://github.com/new> pelo navegador.

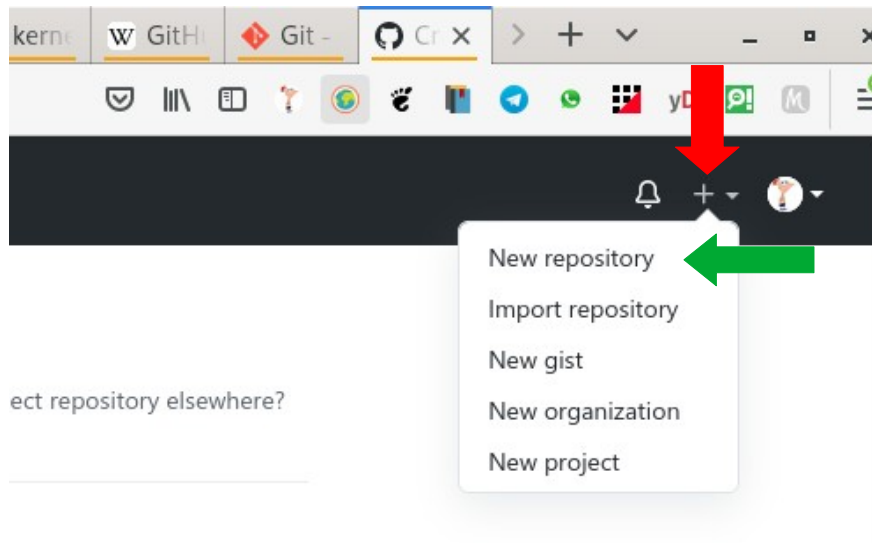


Figura 1: Acessando seus repositórios.

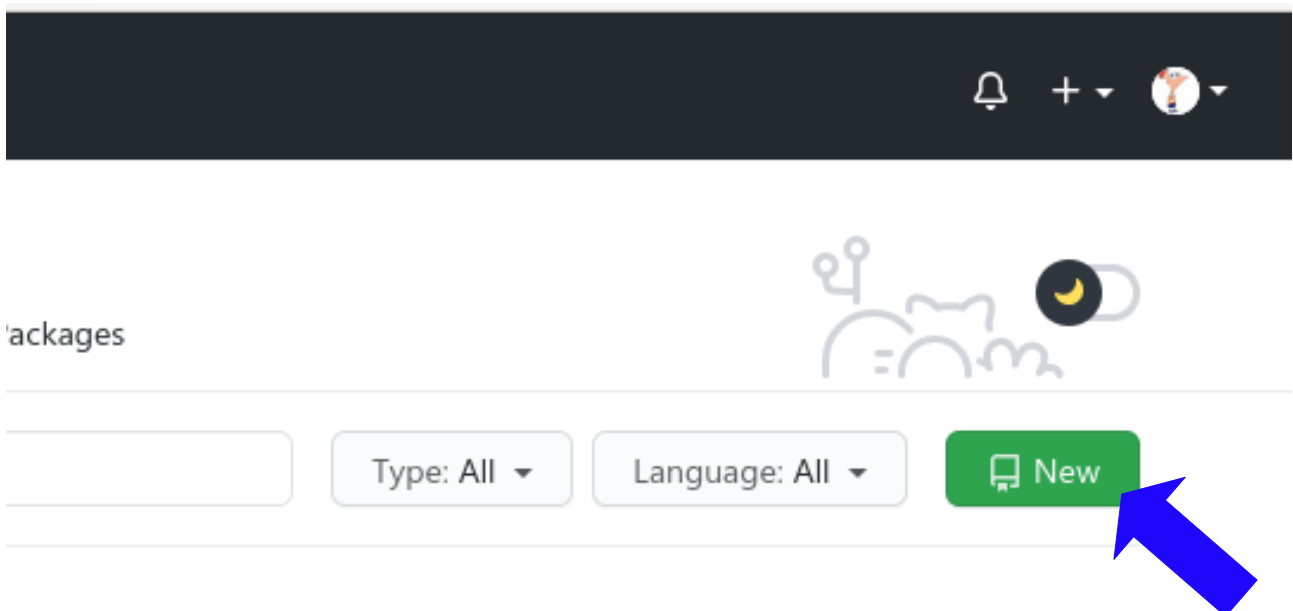


Figura 2: Botão New.

Feito isto será apresentada uma tela para você configurar seu novo repositório (Figura 3). Faça o seguinte:

1. dê um nome ao seu repositório (seta **vermelha** na Figura 3). Geralmente damos o mesmo nome da aplicação que foi criada em nosso computador. Exemplo: `GestaoDeVeiculos` ou `gestao-de-veiculos`). **Não é recomendado o uso de acentos e/ou caracteres especiais;**
2. você poderá informar uma descrição para o seu projeto (seta **azul** na Figura 3), sendo que o preenchimento deste campo é facultativo;
3. você deverá escolher se seu projeto será “público” ou “privado” (seta **verde** na Figura 3);

4. o GitHub ainda te dá a opção de inicializar seu novo projeto com três tipos de arquivos (seta **laranja** na Figura 3):
 1. arquivo `README.md` - (leia-me em Português) é geralmente um arquivo explicativo sobre o projeto e o que a pessoa que tem interesse deve saber sobre o mesmo,
 2. arquivo `.gitignore` - este arquivo especifica arquivos e/ou diretórios que deverão ser ignorados pelo GitHub, ou seja, arquivo e/ou diretórios que não estarão no projeto no repositório remoto e
 3. arquivo de licença - que especifica uma licença de uso do *software*, geralmente restringindo o uso do *software* para alguns fins;
5. botão *Create repository* é o último passo para a criação do novo repositório (seta **rosa** na Figura 3).

The screenshot shows the GitHub repository creation interface. At the top, there is a 'Repository template' section with a 'No template' dropdown. Below that, the 'Owner' is set to 'eddiesaliba' and the 'Repository name' is 'CadastroDeAlunos', with a red arrow pointing to a green checkmark next to the name. A blue arrow points to the 'Description' field containing 'Apenas um teste para criação de uma apostila sobre Git / GitHub.'. A green arrow points to the 'Public' radio button, which is selected. An orange arrow points to the 'Initialize this repository with:' section, which includes checkboxes for 'Add a README file', 'Add .gitignore', and 'Choose a license'. At the bottom, a pink arrow points to the 'Create repository' button.

Figura 3 Configuração do novo repositório.

Sugere-se criar manualmente, no repositório local, os arquivos: `README.md`, `.gitignore` e o de licença (se tiverem que existir). Isto, para evitar problemas na hora de vincular o repositório local ao remoto.

Criaremos o repositório com a configuração descrita na Figura 3 para servir de exemplo para esta apostila.

Ao apertarmos o botão *Create repository* (seta rosa na Figura 3) o GitHub criará o novo repositório e apresentará uma tela semelhante à mostrada na Figura 4.

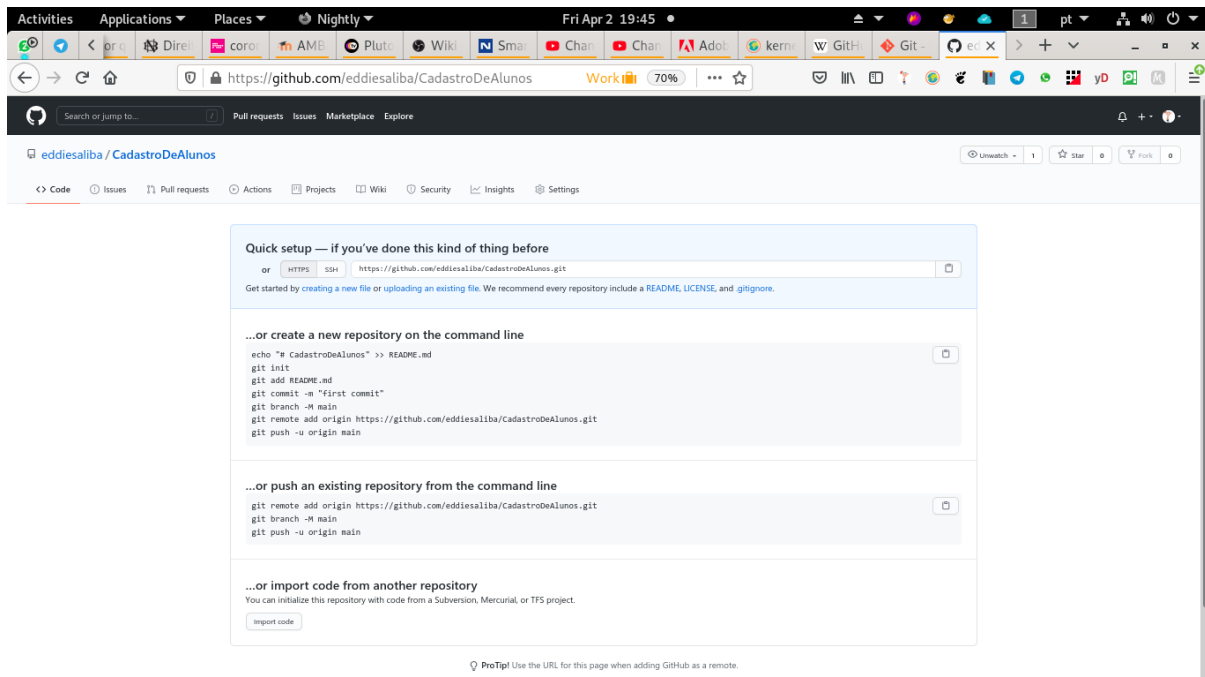


Figura 4: Tela informativa de repositório criado e instruções de acesso ao mesmo.

criação do repositório. Inclusive, ele nos sugere a maneira como deveremos agir para criarmos nosso repositório local usando o Git.

...or create a new repository on the command line

```
echo "# CadastroDeAlunos" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin https://github.com/eddiessaliba/CadastroDeAlunos.git
git push -u origin main
```

Arquivos: README.md, .gitignore e de licença

Uma vez criado o repositório remoto, então, devemos abrir o Terminal e executar os comandos acima. Mas, antes de o fazermos, podemos verificar que o GitHub nos forneceu o comando de como criar o arquivo `README.md`, mas não nos forneceu o comando de como criar o arquivo `.gitignore` (que é essencial na maioria dos projetos).

Então, vamos ter que criar o arquivo `.gitignore` usando os comandos do nosso sistema operacional. Mas, este não é o maior problema. O maior problema que temos é:

- o que exatamente colocar dentro do arquivo `.gitignore` para que o GitHub não guarde arquivos desnecessários?

Para responder esta pergunta fizemos uma consulta na Internet e descobrimos que, segundo Mota (2021), o *site*:

<http://gitignore.io> (Figura 5)

cria o arquivo `.gitignore` automaticamente para nós. Bastando apenas informarmos ao *site* qual ou quais IDE's utilizamos em nosso projeto.

Para manipular nosso projeto exemplo, que é feito em Java, nós usamos a IDE Eclipse ou a IDE Netbeans. E ambas geram arquivos que não devem ser "guardados" no GitHub. Sejam os arquivos compilados (`.class`) ou mesmo os arquivos de controle da própria IDE.

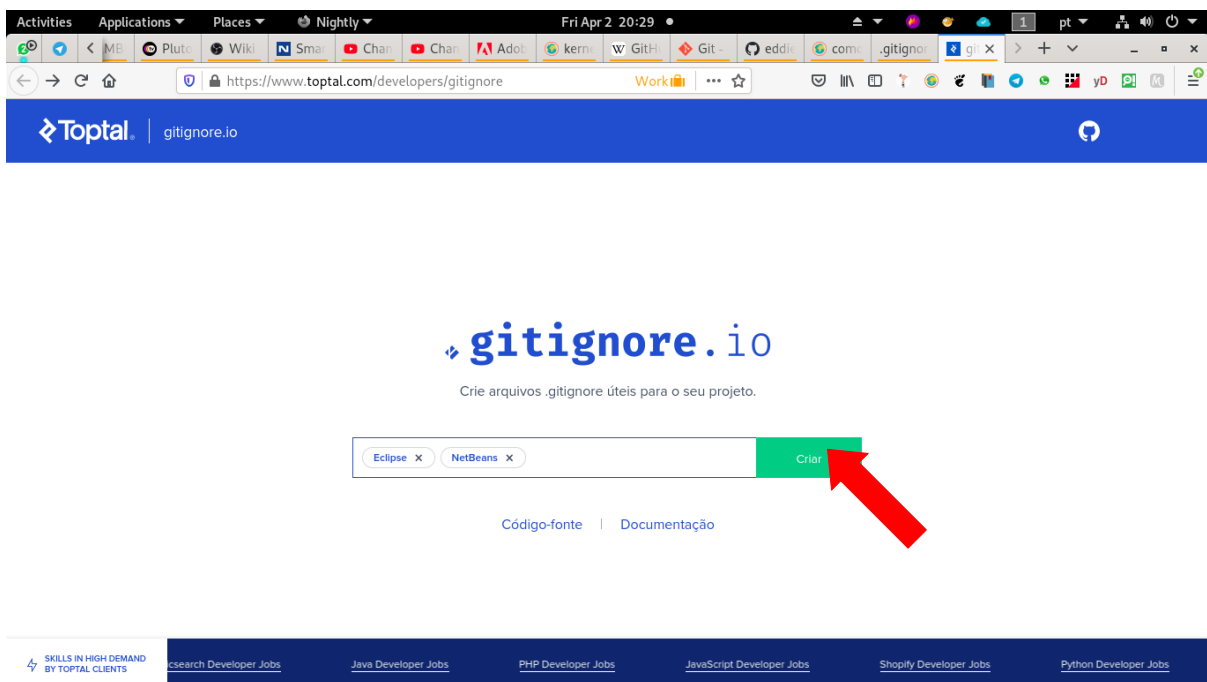


Figura 5: Imagem do site *gitignore.io*

"Criar" (seta vermelha na Figura 5) e o *site* gera o conteúdo que deverá ser colocado no arquivo `.gitignore`.

IMPORTANTE: antes de executar qualquer um dos comandos mostrados na Figura 4, deve-se primeiramente criar e configurar o arquivo `.gitignore` no projeto que será versionado pelo GitHub. Isto evitará muitas dores de cabeça. Caso você já tenha adicionado o seu projeto ao GitHub sem ter criado o arquivo `.gitignore` anteriormente, então, os arquivos que deveriam ser ignorados foram copiados para o seu repositório no GitHub. Neste caso, além de criar o arquivo `.gitignore` você deverá fazer outros passos para tirar do GitHub os arquivos que estão lá indevidamente. Mas, estes passos não serão explicados neste tutorial, pois, estão muito bem explicados em Mota (2021), cuja a referência para o blog poderá ser encontrada na seção Bibliografia, no final deste tutorial.

Por fim, geramos o conteúdo do nosso arquivo `.gitignore` no *site* que indicamos anteriormente e o resultado foi:


```

# Created by https://www.toptal.com/developers/gitignore/api/eclipse,netbeans
# Edit at https://www.toptal.com/developers/gitignore?templates=eclipse,netbeans

### Eclipse ###
.metadata
bin/
tmp/
*.tmp
*.bak
*.swp
*~.nib
local.properties
.settings/
.loadpath
.recommenders

# External tool builders
.externalToolBuilders/

# Locally stored "Eclipse launch configurations"
*.launch

# PyDev specific (Python IDE for Eclipse)
*.pydevproject

# CDT-specific (C/C++ Development Tooling)
.cproject

# CDT- autotools
.autotools

# Java annotation processor (APT)
.factorypath

# PDT-specific (PHP Development Tools)
.buildpath

# sbteclipse plugin
.target

# Tern plugin
.tern-project

# TeXlipse plugin
.texlipse

# STS (Spring Tool Suite)
.springBeans

# Code Recommenders
.recommenders/

# Annotation Processing
.appt_generated/
.appt_generated_test/

# Scala IDE specific (Scala & Java development for Eclipse)
.cache-main
.scala_dependencies
.worksheet

# Uncomment this line if you wish to ignore the project description file.
# Typically, this file would be tracked if it contains build/dependency configurations:
#.project

### Eclipse Patch ###
# Spring Boot Tooling
.sts4-cache/

### NetBeans ###
**/nbproject/private/
**/nbproject/Makefile-*.mk
**/nbproject/Package-*.bash
build/
nbbuild/
dist/
nbdist/
.nb-gradle/

# End of https://www.toptal.com/developers/gitignore/api/eclipse,netbeans

```

Então copiamos o texto gerado, criamos nosso arquivo `.gitignore` dentro da pasta do projeto (neste caso “CadastroDeAlunos”) e colamos o texto dentro dele.

Observação: o nome do arquivo começa com “.” (ponto) mesmo. Isto, em Linux, significa que é um arquivo oculto.

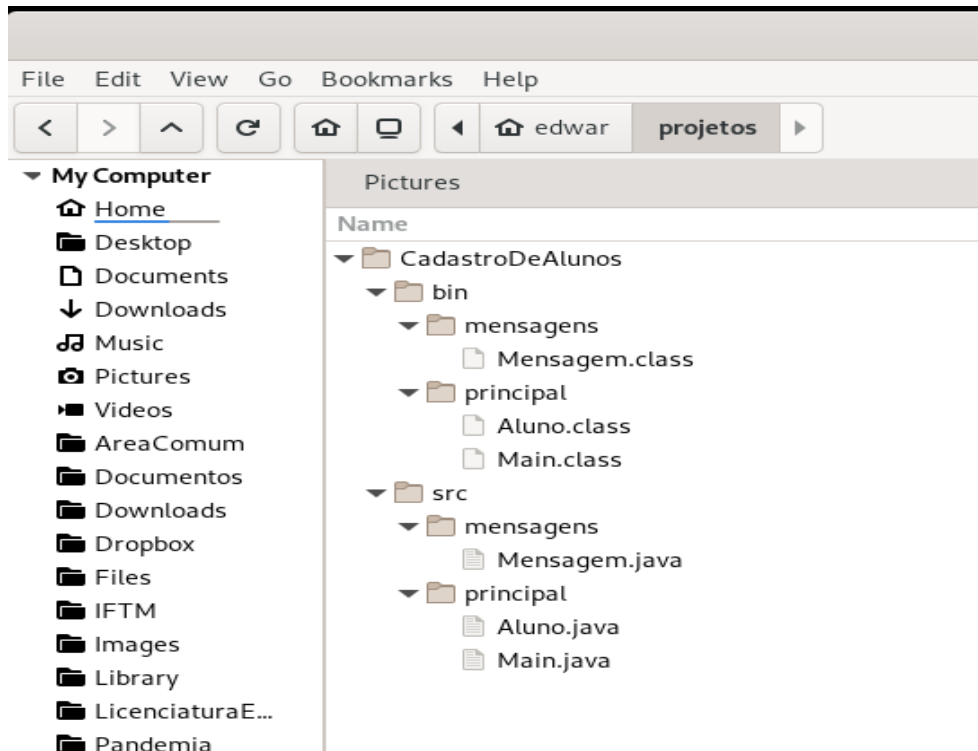


Figura 6: Imagem dos arquivos existentes nos diretórios do projeto.

Na Figura 6 pode-se ver a árvore de diretórios que compõe o projeto “CadastroDeAlunos”. E pode-se observar também que existem arquivos com a extensão “.class”, que não deverão ser armazenados no GitHub.

Iniciando um repositório localmente e fazendo o *upload* para o repositório remoto

Agora que já se tem uma conta no GitHub e já criou-se o primeiro repositório remoto, então, precisa-se iniciar o monitoramento do Git no diretório do projeto no computador localmente e, depois, vincular este diretório local do projeto ao repositório criado no GitHub.

Para tanto, no **Terminal** ou **Prompt de Comando**, acesse o diretório do projeto e comece a executar os comandos que o GitHub nos orientou a executar (Figura 4). Lembrando que já foi criado o arquivo `.gitignore`.

```
$ cd /home/edwar/projetos/CadastroDeAlunos
```

```
$ echo "# CadastroDeAlunos" >> README.md
$ git init
```

Assim que o comando `git init` for executado, uma mensagem como esta "Initialized empty Git repository in [...]" (Figura 7) será mostrada.

A terminal window titled 'edwar@alpha: ~/projetos/CadastroDeAlunos' showing the execution of 'git init'. The output includes several hints about the default branch name and how to configure it, followed by the message 'Initialized empty Git repository in /home/edwar/projetos/CadastroDeAlunos/.git/'.

```
edwar@alpha:~/projetos/CadastroDeAlunos$ git init
hint: Using 'master' as the name for the initial branch. This default branch name
hint: is subject to change. To configure the initial branch name to use in all
hint: of your new repositories, which will suppress this warning, call:
hint:
hint:   git config --global init.defaultBranch <name>
hint:
hint: Names commonly chosen instead of 'master' are 'main', 'trunk' and
hint: 'development'. The just-created branch can be renamed via this command:
hint:
hint:   git branch -m <name>
Initialized empty Git repository in /home/edwar/projetos/CadastroDeAlunos/.git/
edwar@alpha:~/projetos/CadastroDeAlunos$
```

Figura 7: Inicializando monitoramento Git em diretório de projeto.

Então podemos executar os outros comandos:

```
$ git add README.md
$ git add .gitignore
$ git add .
$ git commit -m "Primeiro commit."
$ git branch -M main
$ git remote add origin https://github.com/eddiesaliba/CadastroDeAlunos.git
$ git push -u origin main
```

Atenção! Se ao executar o comando `git commit -m "Primeiro commit."` você tiver recebido uma mensagem como mostrado na Figura 8, então você deverá executar os passos descritos na mensagem para cadastrar localmente os dados do seu usuário. Estes mesmos passos descritos na mensagem da Figura 8 estão sendo explicados em Configurações locais. Ah! E se isto aconteceu, então o comando `git commit -m "Primeiro commit."` não foi executado. E, assim sendo, após a configuração do usuário local você deverá executar este comando novamente.

```
edwar@delta: ~/eclipse-workspace/TesteGithub
hint: git config --global init.defaultBranch <name>
hint:
hint: Names commonly chosen instead of 'master' are 'main', 'trunk' and
hint: 'development'. The just-created branch can be renamed via this command:
hint:
hint: git branch -m <name>
Initialized empty Git repository in /home/edwar/eclipse-workspace/TesteGithub/.git/
edwar@delta:~/eclipse-workspace/TesteGithub$ git add .
edwar@delta:~/eclipse-workspace/TesteGithub$ git commit -m "Primeiro commit."
Author identity unknown

*** Please tell me who you are.

Run

  git config --global user.email "you@example.com"
  git config --global user.name "Your Name"

to set your account's default identity.
Omit --global to set the identity only in this repository.

fatal: unable to auto-detect email address (got 'edwar@delta.(none)')
edwar@delta:~/eclipse-workspace/TesteGithub$
```

Figura 8: Autor não identificado.

Explicando os comandos de: acesso, inicialização, registro, nomeação de *branch*, vinculação de repositórios e registro no repositório remoto

Vamos as explicações sobre os comandos que foram executados.
Comando:

- `cd /home/edwar/projetos/CadastroDeAlunos` - vai para a raiz do diretório do projeto;
- `echo "# CadastroDeAlunos" >> README.md` - cria um arquivo com o nome de `README.md` e coloca os dizeres `"# CadastroDeAlunos"` (sem as aspas) dentro do arquivo;
- `git init` - inicializa o repositório local, ou seja, o monitoramento dos arquivos e diretórios contidos no diretório do projeto;
- `git add README.md` - adiciona o arquivo `README.md` a *stage area*²;
- `git add .gitignore` - adiciona o arquivo `.gitignore` a *stage area*;
- `git add .` - adiciona todos os arquivos do diretório a *stage area*. Nem este comando e tampouco o anterior estavam na lista de comandos fornecidos pelo GitHub. Contudo, ambos precisaram ser executados para

2 *Stage Area* é um local que existe entre um comando *add* e um comando *commit* no Git. Ou seja, quando você usa o comando *add* você envia os arquivos para a *stage area* e só depois que você executa o comando *commit*, é que estes arquivos vão para o repositório local com as devidas modificações que foram realizadas.

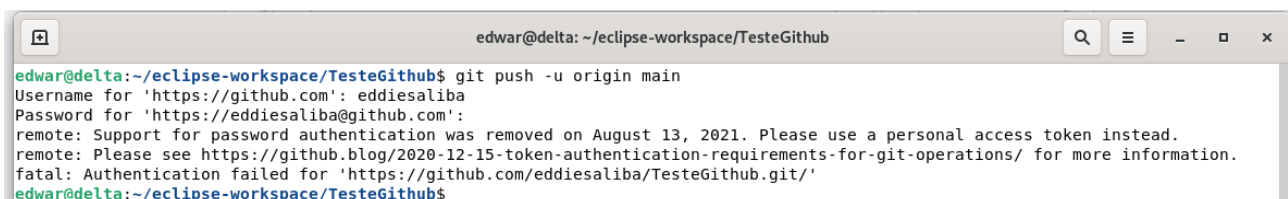
que todos os arquivos fossem adicionados a *stage area* e posteriormente fossem enviados ao GitHub. Sendo mais preciso, estes três comandos `git add` poderiam ser substituídos só pelo último, ou seja, “`git add .`”, pois, este último comando coloca todos os arquivos na *stage area*;

- `git commit -m "Primeiro commit."` - este comando registra todas as modificações que foram feitas nos arquivos que se encontravam na *stage area*, no repositório local;
- `git branch -M main` - muda o nome do *branch*, no repositório local, para “`main`”;
- `git remote add origin https://github.com/eddiesaliba/CadastroDeAlunos.git` - cria o vínculo do repositório local com o repositório no GitHub;
- `git push -u origin main` OU `git push --set-upstream origin main` - faz o *upload* de todas as modificações, que foram realizadas no repositório local, para o repositório remoto (GitHub). A opção `-u` ou então `--set-upstream` tem que ser usada nesta situação, pois, o *branch* “`main`” não está criado no repositório remoto.

Por padrão, quando a aplicação tem apenas um *branch*, dá-se o nome de *master* ou *main* a este *branch*. Mas, uma aplicação pode ter quantos *branches* forem necessários. Em geral os *branches* são criados da seguinte forma: um para cada pessoa que estiver fazendo alterações no projeto.

Se você seguiu os passos descritos até aqui, deve ter reparado que ao executar o comando `git push -u origin main` foi solicitado o usuário e senha para acesso ao GitHub. Isto é necessário para que terceiros não façam postagens em sua conta sem sua autorização.

Então você digitou seu usuário e senha, com plena certeza de que estes dados estão certos. Mas, ainda assim recebeu uma mensagem de erro como mostrado na Figura 9.



```
edwar@delta: ~/eclipse-workspace/TesteGithub
edwar@delta:~/eclipse-workspace/TesteGithub$ git push -u origin main
Username for 'https://github.com': eddiesaliba
Password for 'https://eddiesaliba@github.com':
remote: Support for password authentication was removed on August 13, 2021. Please use a personal access token instead.
remote: Please see https://github.blog/2020-12-15-token-authentication-requirements-for-git-operations/ for more information.
fatal: Authentication failed for 'https://github.com/eddiesaliba/TesteGithub.git/'
edwar@delta:~/eclipse-workspace/TesteGithub$
```

Figura 9: Solicitação de Personal Access Token.

Criando mecanismos de acesso e/ou para facilitar o acesso

Personal Access Token

Desde o dia 13 de Agosto de 2021, o GitHub deixou de aceitar a senha de usuário e passou a exigir um *Personal Access Token* (PAT).

O uso de tokens tem algumas vantagens, pois, você pode configurar quais são os direitos que determinado *token* tem, como por exemplo autorizações para: incluir, excluir, alterar e etc. e também o tempo que o *token* será válido. Além de que, se necessário, você pode invalidá-lo e criar outro.

Para aprender a criar e usar um *token* ou PAT, vamos indicar-lhe um tutorial na Internet que deixa este assunto bem claro. E o mesmo pode ser encontrado na URL:

<https://www.alura.com.br/artigos/nova-exigencia-do-git-de-autenticacao-por-token-o-que-e-o-que-devo-fazer>

Neste tutorial você encontrará tudo que precisa saber sobre como criar e configurar o PAT no sistema operacional que você estiver utilizando.

Um pequeno problema com a solução do PAT é o seguinte:

- sempre que você mudar de *branch* ou desligar seu computador, será necessário entrar novamente com o PAT ao tentar fazer um *push* no repositório remoto e
- como o PAT não é algo tão simples de ser lembrado, então, isto pode se tornar um incômodo.

Mas, não se desespere! Sempre há uma solução. Para não termos que digitar mais o PAT ou qualquer outra senha de autenticação para fazermos nossos *push's* no GitHub, basta criarmos uma chave SSH.

Criando uma chave SSH

Para criarmos uma chave SSH no Linux é muito simples. Para tanto, basta seguir os passos no tutorial “Esqueça senhas: Autenticação baseada em chave no GitHub a partir do Linux” de Campos (2020), que poderá ser acessado na URL:

<https://www.youtube.com/watch?v=j7lX9ff9rC8>

Importante!

Atenção! Nunca configure uma chave SSH em um computador público, para este fim é melhor criar um PAT e com tempo de vida limitado ao tempo que você for utilizar neste computador.

Se você for se desfazer de seu computador sem antes formatá-lo, então, não esqueça de apagar a chave SSH que está registrada para ele no seu GitHub.

Configurações locais

Configuração do nome que ficará associado aos registros de alterações (*commits*) locais:

```
$ git config --global user.name "eddiesaliba"
```

E também podemos configurar o *e-mail* que ficará associado aos registros de alterações locais:

```
$ git config --global user.email "eddiesaliba@yahoo.com"
```

E por fim, de acordo com Alves (2022), podemos configurar o Git para armazenar o PAT em *cache*, para que não precisemos digitá-lo sempre que formos fazer o *upload* das alterações para o repositório remoto. Para tanto devemos executar o comando:

```
$ git config --global credential.helper cache
```

e caso haja a necessidade de “descadastrar” o PAT:

```
$ git config --global --unset credential.helper
```

Situações e comandos (passo a passo)

Criando um repositório remoto (recapitulando)

- cria-se um repositório no GitHub com o mesmo nome do projeto no computador local;
- pelo terminal:
 1. entra-se no diretório raiz do projeto local e
 2. cria-se o arquivo `README.md` e `.gitignore`;
- executa-se a sequência de comandos a seguir para vincular e atualizar o repositório remoto:
 3. `git init`
 4. `git add .`
 5. `git commit -m "Primeiro commit."`
 6. `git branch -M master`
 7. `git remote add origin url_do_projeto`
 8. `git push -u origin master`

Compartilhando projetos / repositórios

- para compartilhar um repositório público do GitHub, basta que você forneça ao interessado a URL do repositório. Exemplo: <https://github.com/eddiesaliba/Exemplo001>

- por sua vez, para copiar o projeto para a máquina, o interessado deverá usar o seguinte comando:
1. `git clone https://github.com/eddiesaliba/Exemplo001`

Como visualizar todas as alterações que foram realizadas antes de aplicá-las ao repositório local?

- basta usar o comando:
1. `git status`

Fiz alterações em diversos arquivos. Como faço o versionamento?

- primeiramente você deverá enviar os arquivos para a *stage area* com o comando:
1. `git add .`
- depois registrar as alterações no repositório local com o comando:
2. `git commit -m "mensagem que descreve alterações realizadas"`
- e por fim, deverá enviar as alterações realizadas para o repositório remoto com o comando:
3. `git push -u origin master`

Fiz alterações em diversos arquivos. Mas, quero enviar apenas alguns para a *stage area*

- para enviar todos os arquivos para a *stage area*, usa-se o comando:
1. `git add .`
- para enviar apenas um arquivo para a *stage area*, usa-se o comando:
2. `git add nomeDoArquivo`

Criei diversos *branches*. Mas, não me recordo de seus nomes

- para saber quais *branches* existem num determinado projeto, use o comando:
1. `git branch`

Preciso criar um novo *branch*. Como faço?

- para criar um novo *branch* basta utilizar o comando:


```
1. git branch nomeDoNovoBranch
```

Estou em um *branch* mas preciso mudar para outro

- para mudar de *branchs* utilize o comando:
1. `git checkout nomeDoBranchDestino`

Preciso remover um *branch*

- para remover um *branch* basta utilizar o comando:
1. `git branch -d nomeDoBranch`

Foram realizadas alterações em um *branch*. Como posso verificar a diferença de um *branch* para outro?

- para verificar a diferença entre *branchs* basta utilizar o comando:
1. `git diff nomeDeUmDosBranchs nomeDoOutroBranch`

Como faço para verificar o histórico de versões de um determinado *branch*?

- para verificar o histórico de versões do *branch* inteiro basta usar o comando:
1. `git log`
- para verificar o histórico de versões de um determinado arquivo use o comando:
2. `git log --follow nomeDoArquivo`

Gostaria de verificar as mudanças que ocorreram em um determinado *commit*

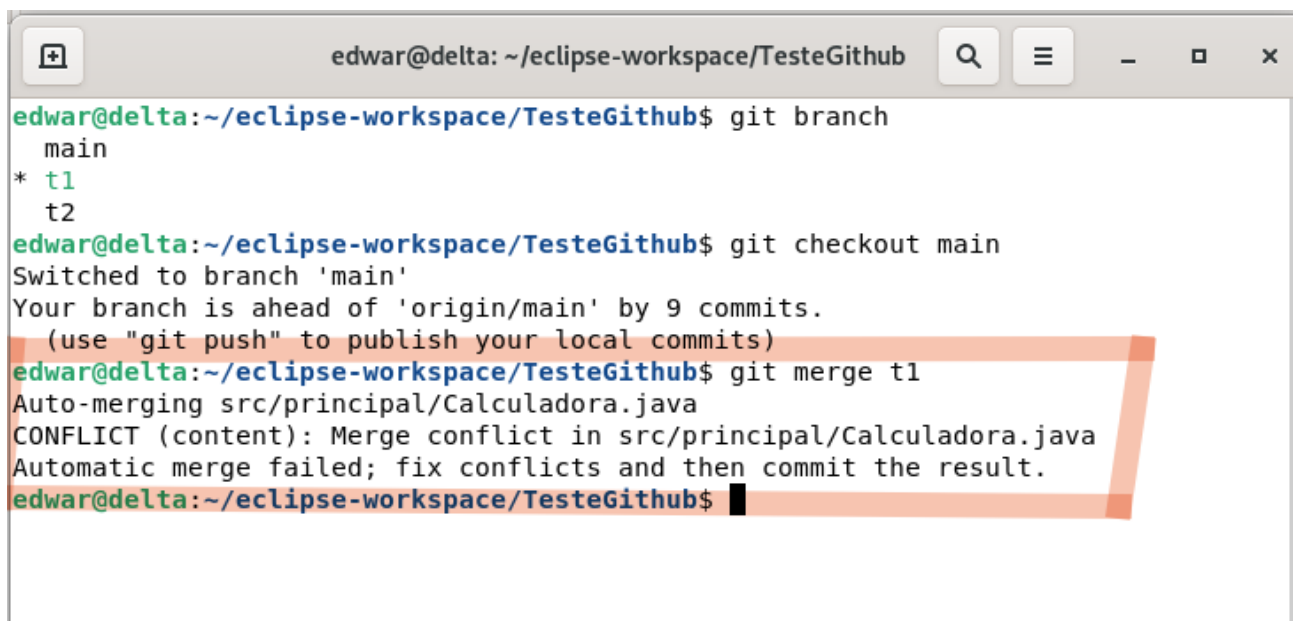
- para verificar as mudanças de um determinado *commit* será necessário ter o código do *commit*. Todo *commit* tem um código diferente, com este código “em mãos” basta usar o comando:
1. `git show codigoDoCommit`

Alguém pode ter feito modificações no *software*. Como faço para atualizar a versão do código-fonte que se encontra no meu computador?

- para sincronizar a versão de um *software* em um repositório local, com um determinado *branch* num repositório remoto, basta usar o comando:
1. `git pull origin nomeDoBranch`

Preciso fazer a fusão/mesclagem (*merge*) de um *branch* com o *branch* principal “main”

- para fazermos a fusão de dois *branches*, primeiramente deve-se estar no *branch* que vai receber a código do outro *branch*, no nosso caso, devemos estar no *branch main*. Estando no *branch main* basta executar o comando:
1. `git merge nomeDoBranch`
- se não houver conflito, então ótimo. Porém, se houver conflito então será mostrada uma mensagem semelhante à mostrada na Figura 10;

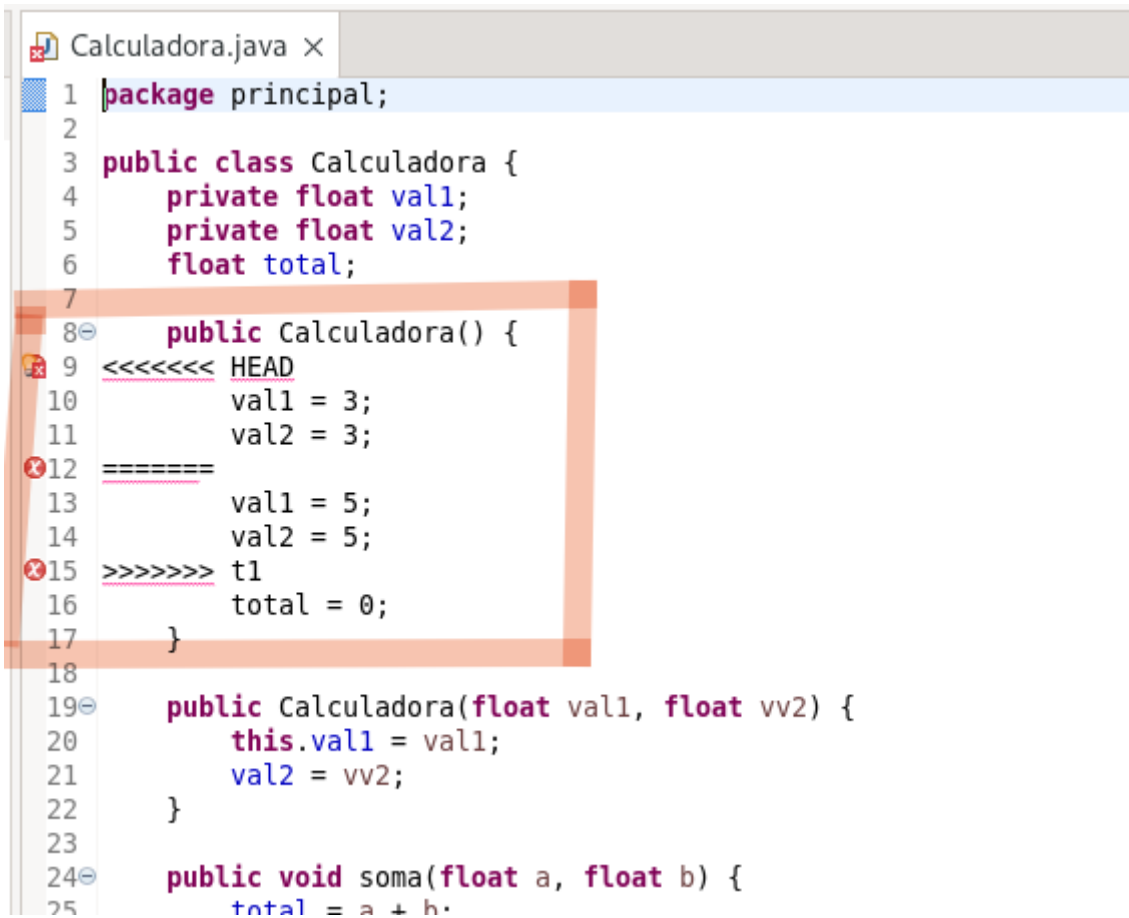


```
edwar@delta: ~/eclipse-workspace/TesteGithub
edwar@delta:~/eclipse-workspace/TesteGithub$ git branch
main
* t1
  t2
edwar@delta:~/eclipse-workspace/TesteGithub$ git checkout main
Switched to branch 'main'
Your branch is ahead of 'origin/main' by 9 commits.
(use "git push" to publish your local commits)
edwar@delta:~/eclipse-workspace/TesteGithub$ git merge t1
Auto-merging src/principal/Calculadora.java
CONFLICT (content): Merge conflict in src/principal/Calculadora.java
Automatic merge failed; fix conflicts and then commit the result.
edwar@delta:~/eclipse-workspace/TesteGithub$
```

Figura 10: Conflito ao tentar efetuar uma fusão (*merge*) de códigos.

- um conflito é uma situação onde uma ou mais linhas de código de um determinado arquivo foram alteradas em paralelo. Ou seja, estas mesmas linhas de código-fonte foram alteradas num arquivo XYZ de um “*branch A*” e também no arquivo XYZ de um “*branch B*”. E desta forma o Git não saberá como resolver esta questão: qual das alterações ele deve considerar ao tentar fazer o “*merge*” (a fusão)?
- em um caso como este o Git gerará um arquivo com as duas versões do código-fonte que foi alterado, como mostrado na Figura 11. Observe que

o conflito aparece entre três *tag*'s: <<<<<< HEAD, ===== e >>>>>> t1 (t1 é o nome do *branch* que não é o *main*). O código-fonte que aparece depois da palavra *HEAD* e antes da linha duplamente tracejada ("=====") é o que se encontra no *branch main*. Já o código-fonte que se encontra abaixo da linha duplamente tracejada ("====="), é o que se encontra no outro *branch*. Como houve alteração nestas mesmas linhas em ambos os arquivos, tanto o que está no *branch main* quanto o que está no outro *branch*, então o Git não sabe resolver o conflito. E neste caso você, deverá fazer escolher qual das alterações apresentadas será postada no *branch main*. Você poderá optar por uma das duas, mesclar as duas ou ainda, caso você não queira nenhuma das duas, poderá excluí-las e colocar outro código-fonte totalmente diferente no lugar delas.



```
Calculadora.java x
1 package principal;
2
3 public class Calculadora {
4     private float val1;
5     private float val2;
6     float total;
7
8     public Calculadora() {
9 <<<<<< HEAD
10         val1 = 3;
11         val2 = 3;
12 =====
13         val1 = 5;
14         val2 = 5;
15 >>>>>> t1
16         total = 0;
17     }
18
19     public Calculadora(float val1, float vv2) {
20         this.val1 = val1;
21         val2 = vv2;
22     }
23
24     public void soma(float a, float b) {
25         total = a + b;

```

Figura 11: Exemplo de conflito identificado pelo Git.

- após o acerto, deve-se executar os seguintes comandos para efetivação das mudanças:
 1. git add .
 2. git commit -m "Um texto referente as alterações aplicadas."
 3. git push -u origin main

- e caso o *branch*, que não é o principal (*main*), não seja mais necessário, então deve-se apagá-lo:

1. `git branch -d nomeDoBranch`

Bibliografia

ALVES, Camila Fernanda. **Nova exigência do Git de autenticação por token, o que é e o que devo fazer?**. Disponível em: <<https://www.alura.com.br/artigos/nova-exigencia-do-git-de-autenticacao-por-token-o-que-e-o-que-devo-fazer>>. Acesso em: 10 abr. 2022.

GIT. In: WIKIPÉDIA: a enciclopédia livre. Wikimedia, 2021. Disponível em: <<https://pt.wikipedia.org/wiki/Git>>. Acesso em: 29 mar. 2021.

GITHUB. In: WIKIPÉDIA: a enciclopédia livre. Wikimedia, 2021. Disponível em: <<https://pt.wikipedia.org/wiki/GitHub>>. Acesso em: 02 abr. 2021.

GITHUB Training. **Folheto de Ajuda para Git do Github**. Disponível em: <https://training.github.com/downloads/pt_PT/github-git-cheat-sheet.pdf>. Acesso em 03 abr. 2021.

MOTA, Fernando Jorge. **.gitignore - Saiba como ignorar arquivos no Git facilmente**. Disponível em: <<https://fjorgemota.com/gitignore-ou-como-ignorar-arquivos-no-git/>>. Acesso em: 02 abr. 2021.

CAMPOS, Manoel. **Esqueça senhas: Autenticação baseada em chave no GitHub a partir do Linux**. Disponível em: <<https://www.youtube.com/watch?v=j7IX9ff9rC8>>. Acesso em: 15 Ago. 2022.

NÚCLEO (sistema operacional). In: WIKIPÉDIA: a enciclopédia livre. Wikimedia, 2021. Disponível em: <[https://pt.wikipedia.org/wiki/N%C3%BAcleo_\(sistema_operacional\)](https://pt.wikipedia.org/wiki/N%C3%BAcleo_(sistema_operacional))>. Acesso em: 31 mar. 2021.