

DIGITAL INNOVATION ONE – GIT E GITHUB

1. INTRODUÇÃO AO GIT

O foco desse curso é ensinar as funcionalidades do Git e as funcionalidades do repositório GitHub. O Git é um sistema de repositório distribuído, criado por Linus Torvalds, que foi quem criou o SO Linux. Linus criou o Git porque ele sentiu a necessidade de um Software que monitorasse diferentes versões de um código e comportasse a colaboração de diversas pessoas. Os programas de versionamento de código da época não possuíam diversas funcionalidades que Linus achava importante.

O Git cuida dessa parte de versionamento de códigos e outros softwares, como o GitHub e o GitLab guardam esses códigos em repositórios de forma online. O GitHub é uma empresa da Microsoft e muito bem estabelecida no mercado e por isso ele aparece muitas vezes lado a lado do Git, porém são softwares diferentes. Ao trabalhar com Git e GitHub nós iremos aprender 5 tópicos importantes no desenvolvimento de software:

1. Controle de Versão
2. Armazenamento em Nuvem
3. Trabalho em Equipe
4. Melhorar o Código
5. Reconhecimento

Nesse curso iremos treinar na prática a utilizar o Git e o GitHub. Vamos desenvolver uma aplicação, que é a criação de um livro de receitas simples. Através da criação desse projeto simples iremos aprender conceitos mais abstratos do Git.

2. Navegação via Command Line Interface e Instalação

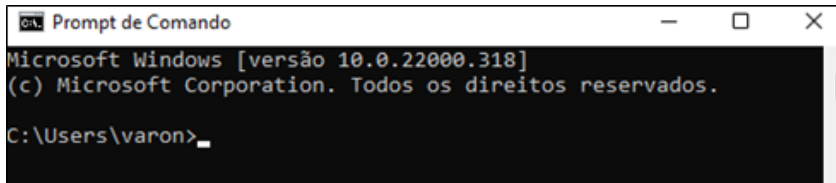
Navegação básica no terminal e instalação

A maioria dos programas operacionais possuem uma interface gráfica, sendo programas do tipo GUI (*graphic use interface*). O Git não possui uma interface gráfica, sendo um *software* do tipo CLI (*command line interface*). Existem programas que pegam o GIT e implementam uma interface gráfica, porém não iremos utilizar esse recurso no curso. Nós iremos interagir apenas com linhas de comando.

Nessa primeira parte aprenderemos a usar comandos comum do terminal, que permitem:

- Mudar de pastas;
- Listar as pastas;
- Criar pastas e arquivos;
- Deletar pastas e arquivos.

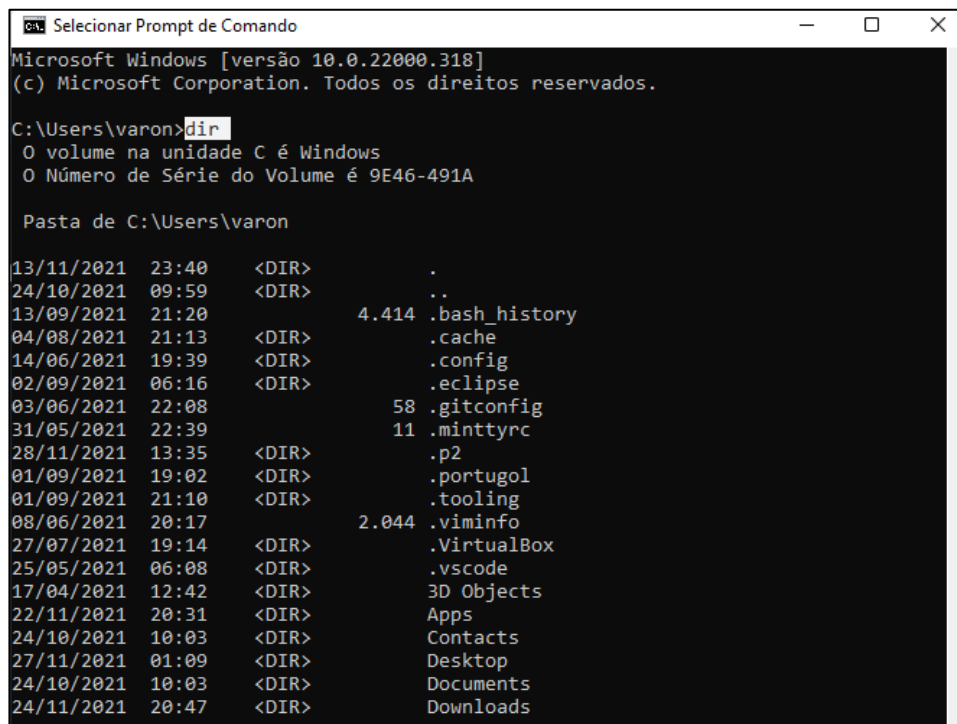
O terminal do Windows é chamado de **Prompt de comando**. Para abrir ele basta pesquisar no Windows por “cmd”.



```
Microsoft Windows [versão 10.0.22000.318]
(c) Microsoft Corporation. Todos os direitos reservados.

C:\Users\varon>
```

- O primeiro comando que iremos aprender é o comando de listar os arquivos e diretórios dentro da pasta que estamos executando o prompt. No cmd digite “dir” e tecele enter.



```
Microsoft Windows [versão 10.0.22000.318]
(c) Microsoft Corporation. Todos os direitos reservados.

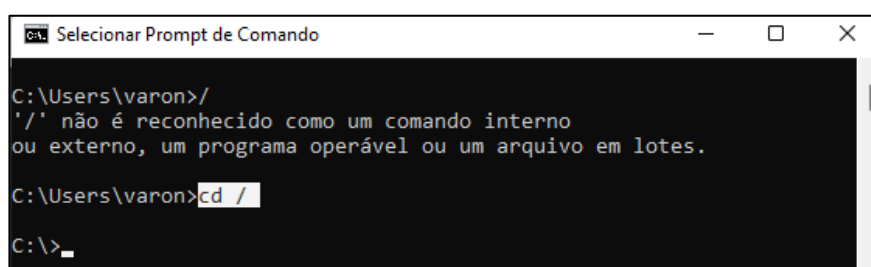
C:\Users\varon>dir
O volume na unidade C é Windows
O Número de Série do Volume é 9E46-491A

Pasta de C:\Users\varon

13/11/2021  23:40    <DIR>          .
24/10/2021  09:59    <DIR>          ..
13/09/2021  21:20             4.414 .bash_history
04/08/2021  21:13    <DIR>          .cache
14/06/2021  19:39    <DIR>          .config
02/09/2021  06:16    <DIR>          .eclipse
03/06/2021  22:08             58 .gitconfig
31/05/2021  22:39             11 .minttyrc
28/11/2021  13:35    <DIR>          .p2
01/09/2021  19:02    <DIR>          .portugol
01/09/2021  21:10    <DIR>          .tooling
08/06/2021  20:17             2.044 .viminfo
27/07/2021  19:14    <DIR>          .VirtualBox
25/05/2021  06:08    <DIR>          .vscode
17/04/2021  12:42    <DIR>          3D Objects
22/11/2021  20:31    <DIR>          Apps
24/10/2021  10:03    <DIR>          Contacts
27/11/2021  01:09    <DIR>          Desktop
24/10/2021  10:03    <DIR>          Documents
24/11/2021  20:47    <DIR>          Downloads
```

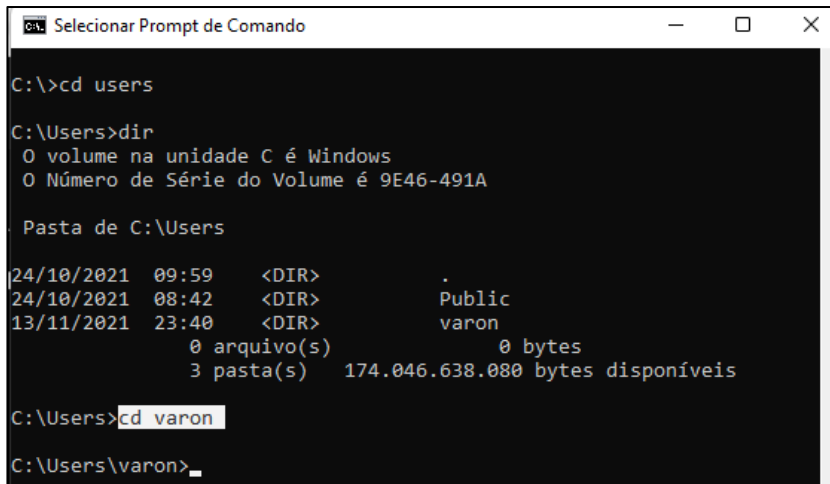
Todos os comandos que iremos aprender apresentam variâncias, ou *flags*, que são complementos que passamos para esses comandos e esse complementos acrescentam, modificam ou formatam a forma que esses comandos são devolvidos.

Depois de aprender o comando para listar os diretórios, vamos ver como subimos ou descemos de níveis nesses diretórios. O próximo comando que iremos aprender é o “cd”, que vai possibilitar que naveguemos entre as pastas. Para mudar de diretório basta digitar o nome do diretório junto do comando e para voltar um diretório, deve-se digitar “..”. Para ir direto na base do diretório. basta com o comando cd, digitar “/”.



```
C:\Users\varon>/
'/' não é reconhecido como um comando interno
ou externo, um programa operável ou um arquivo em lotes.

C:\Users\varon>cd /
C:\>
```



```
C:\>cd users

C:\Users>dir
O volume na unidade C é Windows
O Número de Série do Volume é 9E46-491A

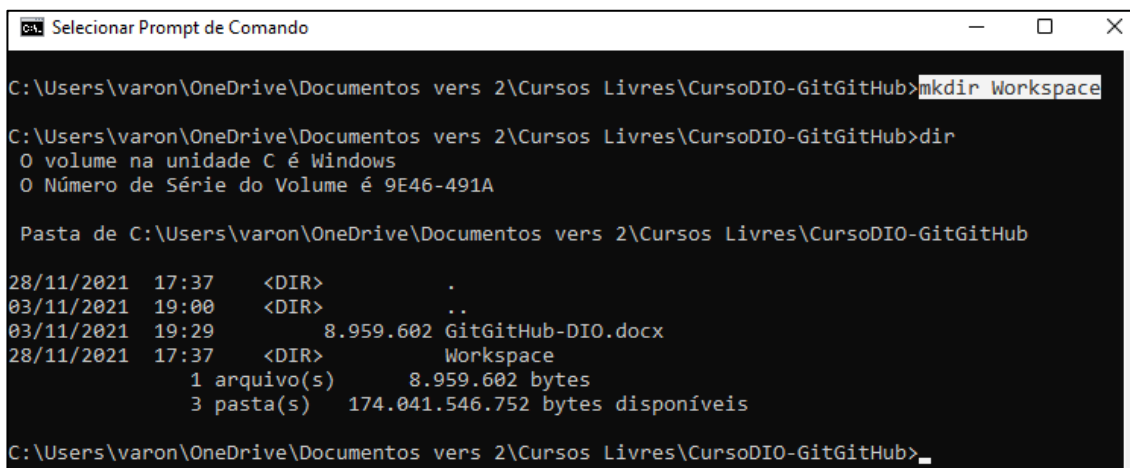
Pasta de C:\Users

24/10/2021  09:59    <DIR>        .
24/10/2021  08:42    <DIR>        Public
13/11/2021  23:40    <DIR>        varon
               0 arquivo(s)          0 bytes
               3 pasta(s) 174.046.638.080 bytes disponíveis

C:\Users>cd varon
C:\Users\varon>
```

Sempre que estamos interagindo com o terminal podemos optar em deixar a tela do terminal limpa. Para isso utilizamos o comando “cls” (*clear scream*). Outra função que facilita o trabalho no terminal é a função de autocompletar, que é executada apenas com a tecla Tab.

Vamos agora navegar entre as pastas no terminal e acessar a pasta do curso de Git e GitHub da DIO. Dentro dessa pasta vamos criar um diretório chamado “workspace”.



```
C:\Users\varon\OneDrive\Documentos vers 2\Cursos Livres\CursoDIO-GitGitHub>mkdir Workspace

C:\Users\varon\OneDrive\Documentos vers 2\Cursos Livres\CursoDIO-GitGitHub>dir
O volume na unidade C é Windows
O Número de Série do Volume é 9E46-491A

Pasta de C:\Users\varon\OneDrive\Documentos vers 2\Cursos Livres\CursoDIO-GitGitHub

28/11/2021  17:37    <DIR>        .
03/11/2021  19:00    <DIR>        ..
03/11/2021  19:29      8.959.602 GitGitHub-DIO.docx
28/11/2021  17:37    <DIR>        Workspace
               1 arquivo(s)      8.959.602 bytes
               3 pasta(s) 174.041.546.752 bytes disponíveis

C:\Users\varon\OneDrive\Documentos vers 2\Cursos Livres\CursoDIO-GitGitHub>
```

Agora vamos criar alguns arquivos dentro dessa pasta para aprender como deletar arquivos e repositórios. Podemos criar um arquivo txt dentro do terminal mesmo. Vamos criar uma arquivo txt chamado de hello e dentro dele vamos inserir a frase “Hello World”. Para isso, basta digitar o comando “echo Hello World > hello.txt”

```

C:\Users\varon\OneDrive\Documentos vers 2\Cursos Livres\CursoDIO-GitGit\Hub\Workspace>echo Hello World > hello.txt

C:\Users\varon\OneDrive\Documentos vers 2\Cursos Livres\CursoDIO-GitGit\Hub\Workspace>dir
O volume na unidade C é Windows
O Número de Série do Volume é 9E46-491A

Pasta de C:\Users\varon\OneDrive\Documentos vers 2\Cursos Livres\CursoDIO-GitGit\Hub\Workspace

28/11/2021  17:52    <DIR>          .
28/11/2021  17:37    <DIR>          ..
28/11/2021  17:53                14 hello.txt
               1 arquivo(s)                14 bytes
               2 pasta(s) 174.035.894.272 bytes disponíveis

C:\Users\varon\OneDrive\Documentos vers 2\Cursos Livres\CursoDIO-GitGit\Hub\Workspace>

```

Perceba na imagem acima que o arquivo hello.txt foi criado com sucesso. O termo *Silence on success* é outro conceito onde se acontecer tudo certo o programa fica em silêncio. Para verificar se o arquivo foi realmente criado, basta digitar o comando “dir” para listar os arquivos.

- Para apagar todos os arquivos que estão dentro de um diretório usamos o comando “del”. Porém esse comando não irá apagar o diretório.

```

C:\Users\varon\OneDrive\Documentos vers 2\Cursos Livres\CursoDIO-GitGit\Hub\workspace>del workspace
C:\Users\varon\OneDrive\Documentos vers 2\Cursos Livres\CursoDIO-GitGit\Hub\workspace\*, Tem certeza (S/N)? s

```

```

C:\Users\varon\OneDrive\Documentos vers 2\Cursos Livres\CursoDIO-GitGit\Hub\Workspace>dir
O volume na unidade C é Windows
O Número de Série do Volume é 9E46-491A

Pasta de C:\Users\varon\OneDrive\Documentos vers 2\Cursos Livres\CursoDIO-GitGit\Hub\Workspace

28/11/2021  23:06    <DIR>          .
28/11/2021  17:37    <DIR>          ..
               0 arquivo(s)                0 bytes
               2 pasta(s) 175.542.083.584 bytes disponíveis

C:\Users\varon\OneDrive\Documentos vers 2\Cursos Livres\CursoDIO-GitGit\Hub\Workspace>

```

- Para apagar diretamente uma pasta e todos os arquivos dentro dele usamos o comando “rmdir” e o nome da pasta a ser deletada.

```

C:\Users\varon\OneDrive\Documentos vers 2\Cursos Livres\CursoDIO-GitGitHub>rmdir workspace

C:\Users\varon\OneDrive\Documentos vers 2\Cursos Livres\CursoDIO-GitGitHub>dir
O volume na unidade C é Windows
O Número de Série do Volume é 9E46-491A

Pasta de C:\Users\varon\OneDrive\Documentos vers 2\Cursos Livres\CursoDIO-GitGitHub

28/11/2021  23:12    <DIR>          .
03/11/2021  19:00    <DIR>          ..
03/11/2021  19:29                8.959.602 GitGitHub-DIO.docx
               1 arquivo(s)          8.959.602 bytes
               2 pasta(s) 175.543.365.632 bytes disponíveis

```

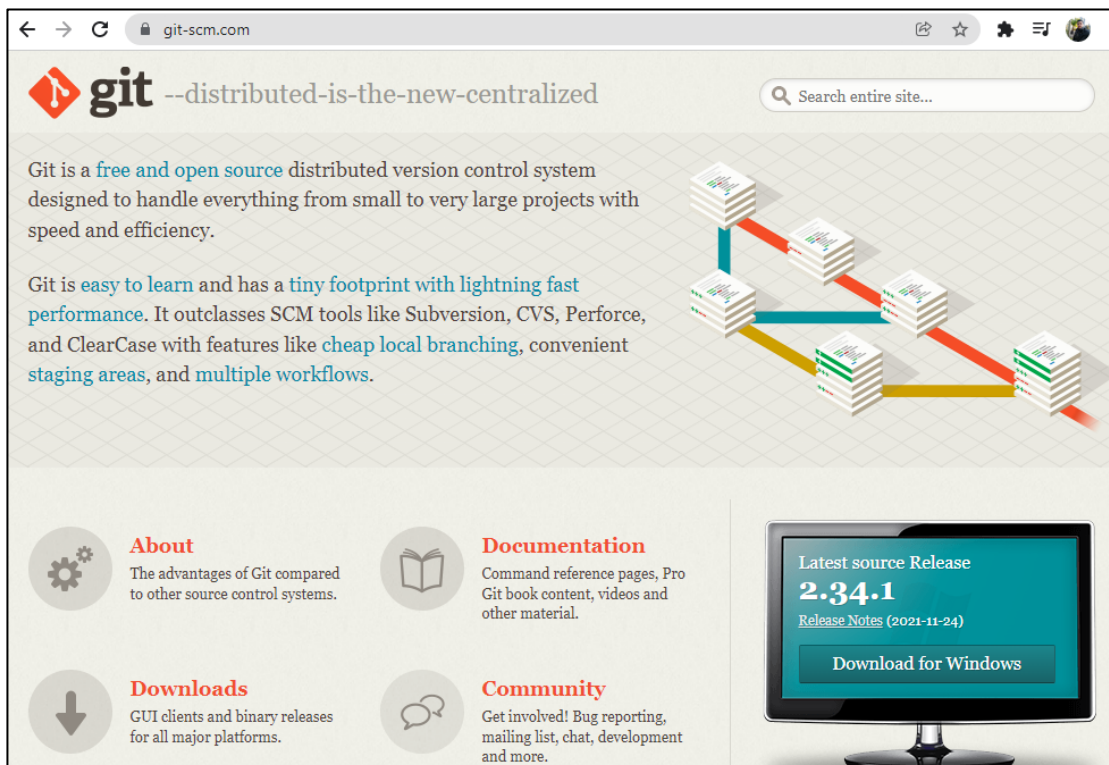
RESUMINDO

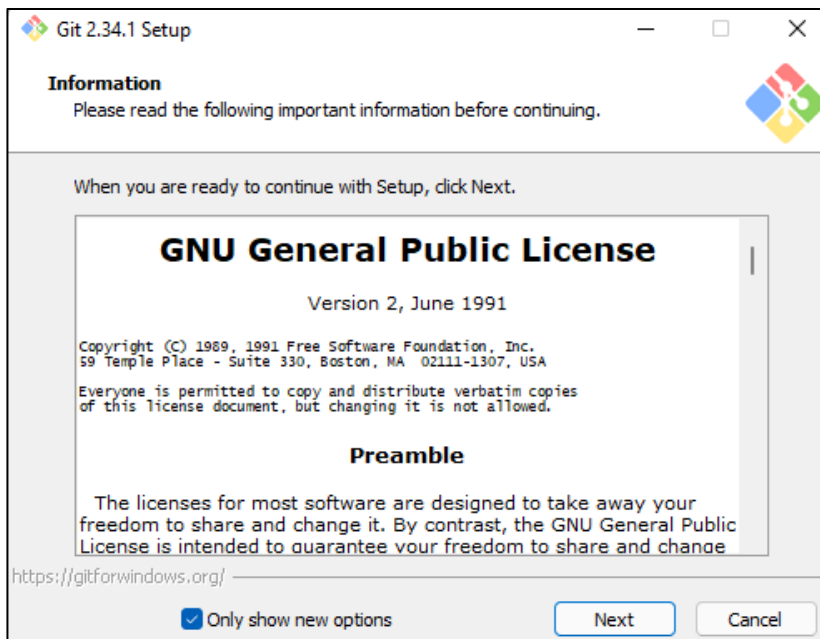
Comando básico para aprendermos a navegar pelo terminal:

- **cd**: muda de diretório para diretório.
- **dir**: lista os diretórios e arquivos contidos dentro da pasta.
- **mkdir**: comando que cria diretórios.
- **del / rmdir**: deleta permanentemente os arquivos e diretórios.

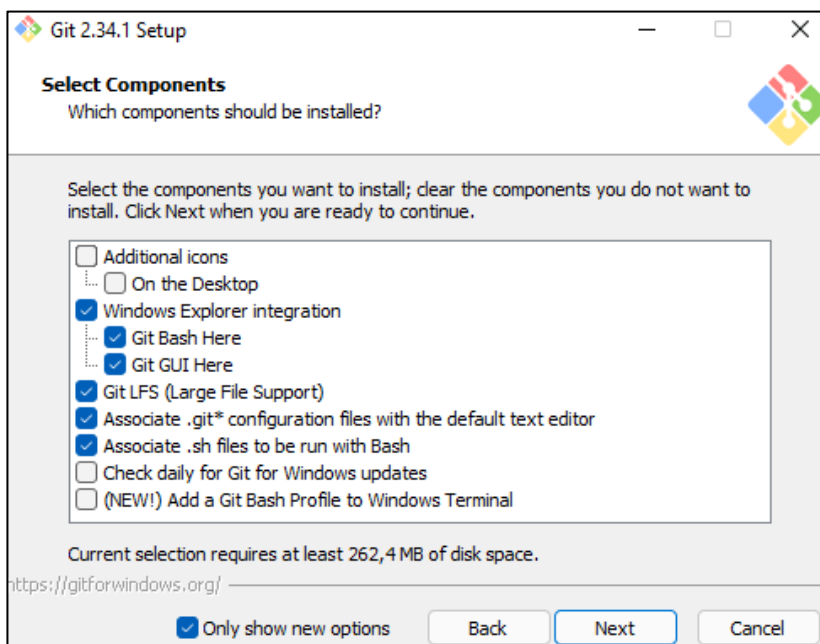
Realizando a Instalação do GIT

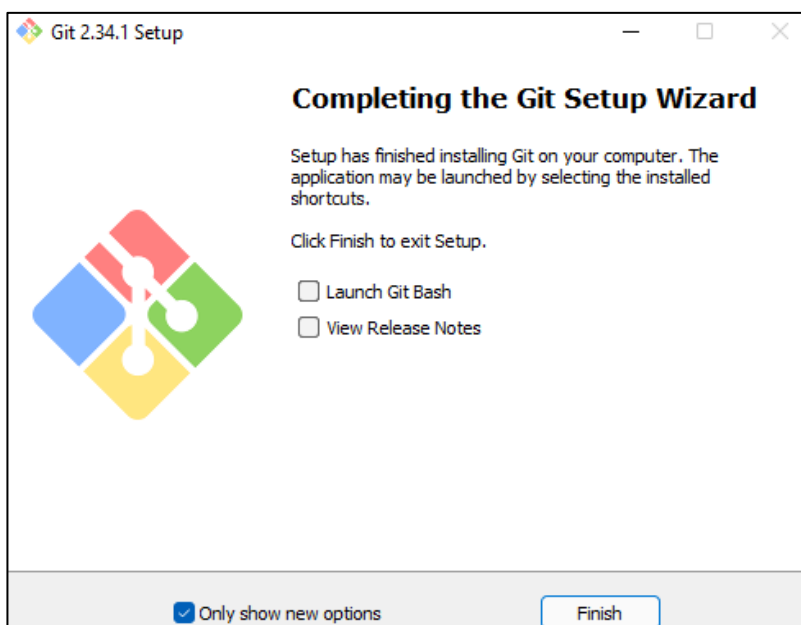
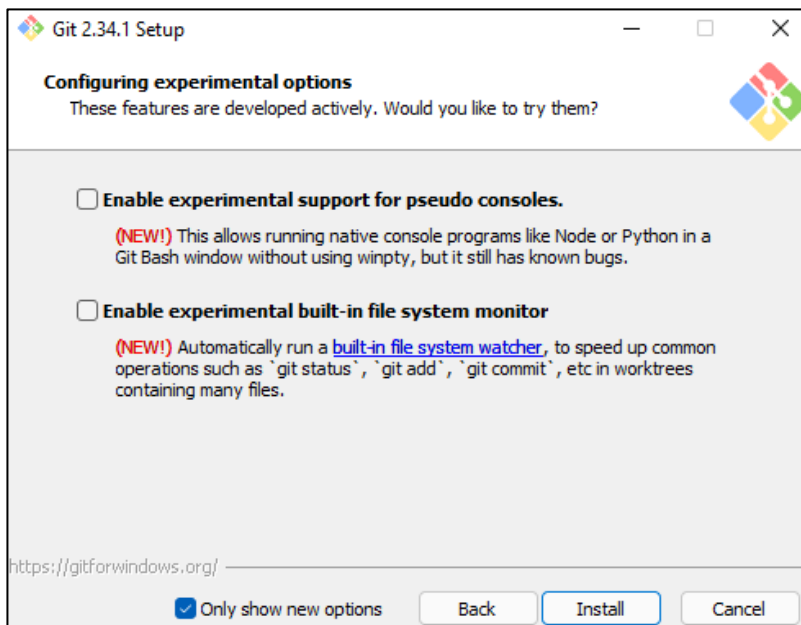
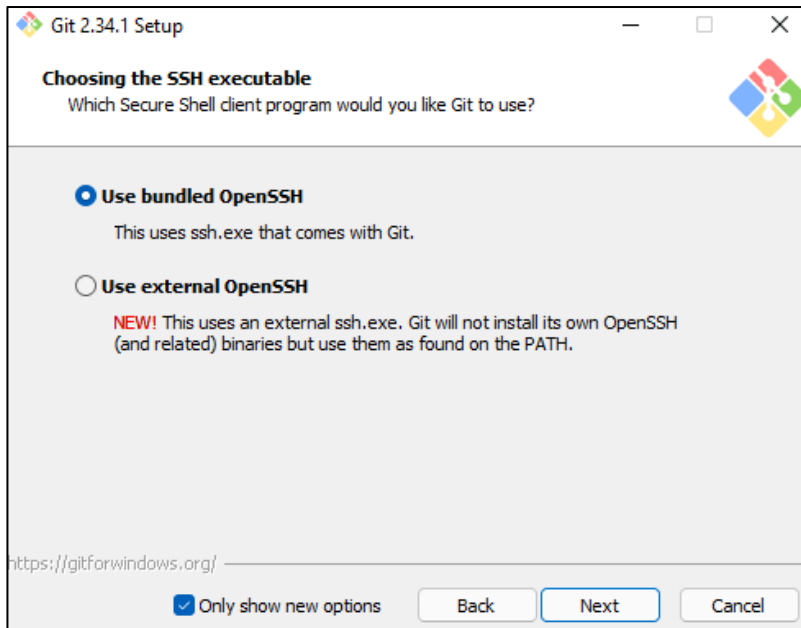
O link do site para baixarmos o Git é o <https://git-scm.com/>. Basta escolher a versão de acordo com o SO da máquina. Clique no botão “Download for Windows que irá abrir uma janela para escolher onde o instalador será baixado.





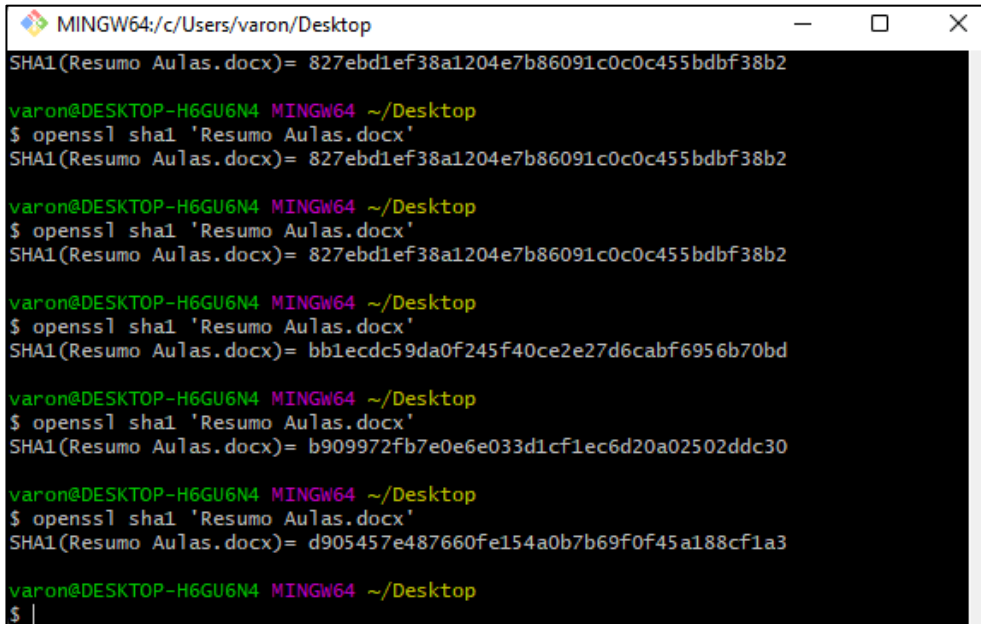
- Certifique-se que as opções “Git Bash Here” e “Git GUI Here” estão marcadas.





3.Entendendo como o Git Funciona por Baixo dos Panos

- **SHA1:** é a sigla para *Secure Hash Algorithm* (Algoritmo de Hash seguro). É um conjunto de funções hash criptográficas projetadas pela NSA (Agência de Segurança Nacional dos EUA). Essa encriptação gera um conjunto identificador de caracteres de 40 dígitos.



```

MINGW64:/c/Users/varon/Desktop
SHA1(Resumo Aulas.docx)= 827ebd1ef38a1204e7b86091c0c0c455bdbf38b2

varon@DESKTOP-H6GU6N4 MINGW64 ~/Desktop
$ openssl sha1 'Resumo Aulas.docx'
SHA1(Resumo Aulas.docx)= 827ebd1ef38a1204e7b86091c0c0c455bdbf38b2

varon@DESKTOP-H6GU6N4 MINGW64 ~/Desktop
$ openssl sha1 'Resumo Aulas.docx'
SHA1(Resumo Aulas.docx)= 827ebd1ef38a1204e7b86091c0c0c455bdbf38b2

varon@DESKTOP-H6GU6N4 MINGW64 ~/Desktop
$ openssl sha1 'Resumo Aulas.docx'
SHA1(Resumo Aulas.docx)= bb1ecd59da0f245f40ce2e27d6cabf6956b70bd

varon@DESKTOP-H6GU6N4 MINGW64 ~/Desktop
$ openssl sha1 'Resumo Aulas.docx'
SHA1(Resumo Aulas.docx)= b909972fb7e0e6e033d1cf1ec6d20a02502ddc30

varon@DESKTOP-H6GU6N4 MINGW64 ~/Desktop
$ openssl sha1 'Resumo Aulas.docx'
SHA1(Resumo Aulas.docx)= d905457e487660fe154a0b7b69f0f45a188cf1a3

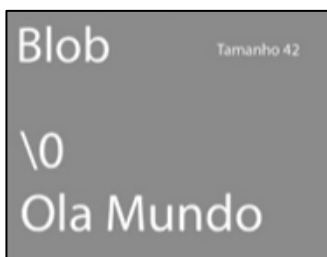
varon@DESKTOP-H6GU6N4 MINGW64 ~/Desktop
$
  
```

Objetos Fundamentais

Existem 3 tipos básicos de objetos do Git: BLOBS, TREES e COMMITS.

- **Blobs:** o jeito que o Git lida e manipula é através de objetos específicos. Os arquivos ficam guardados dentro de objetos chamados **blob**. Esse objeto possui metadados do Git nele.

O objeto “blob” vai ter o tipo do objeto, nesse caso blob, o tamanho dessa String ou desse arquivo, um “\0” e o conteúdo desse arquivo, que são os metadados do Git.



- **Trees:** as **trees** armazenam os blobs. Verificamos aqui uma crescente, onde o “blob” é o bloco básico de composição, e a “tree” é quem aponta e armazena os diferentes tipos de blobs.

As “trees” também contêm metadados, possuindo o “\0”. Elas apontam para o blob, que por sua vez tem o SHA1. Enquanto o blob só guarda o SHA1 do arquivo, a “tree” guarda o nome do arquivo e é a responsável por montar toda a estrutura de onde estão os arquivos.

Assim como no SO um diretório pode conter outro diretório, uma tree pode apontar para outras trees e elas também tem um SHA1 desse metadado.

Os objetos ficam armazenados entre si para sempre. Se mudarmos um caractere no arquivo, o SHA1 da blob irá mudar e SHA1 da tree também irá mudar.

Tree		<tamanho>
\0		
blob	sa4d8s	texto.txt

- **Commit:** esse é o objeto mais importante de todos pois é ele que vai juntar tudo o que vai dar sentido para a alteração que está sendo feita. O **commit** pode apontar para uma **tree** ou para um “parente”, que é o último commit realizado antes dele. O commit também aponta para um autor e para uma mensagem.

Commit		<tamanho>
tree	s4a5sq1	
parente	a98acq1	
autor	perkles	
mensagem	"inicia ..."	
timestamp		

A mensagem é passada no objeto commit e é ela que vai explicar e dar significado para as alterações feitas nesses arquivos e nessas pastas. O autor que esse objeto leva é o nome de quem fez a alteração. Os commits também possuem um *timestamp*, que é um carimbo do tempo (data e hora) de quando ele foi criado. Os commits também possuem a encriptação do SHA1 que é o identificador dos seus metadados.

Se alterarmos um dado dentro de uma blob de um arquivo, vai ser gerado um novo SHA1 dessa blob. Como tem uma tree apontando para essa blob, os metadados da tree também serão alterados. Consequentemente, como o commit aponta para a tree, se o metadado dessa tree é alterado, o metadado do commit também será alterado, gerando uma nova encriptação do SHA1. O commit é único para cada autor que está realizando a alteração.

O Git se torna um **sistema distribuído seguro** porque os **commits** são impossíveis de serem alterados. Em um projeto de software onde diversas pessoas contribuem no contigo, tanto na versão que está no servidor quanto na versão que está com qualquer uma das pessoas do grupo, também são versões confiáveis.

Chaves SSH e Tokens

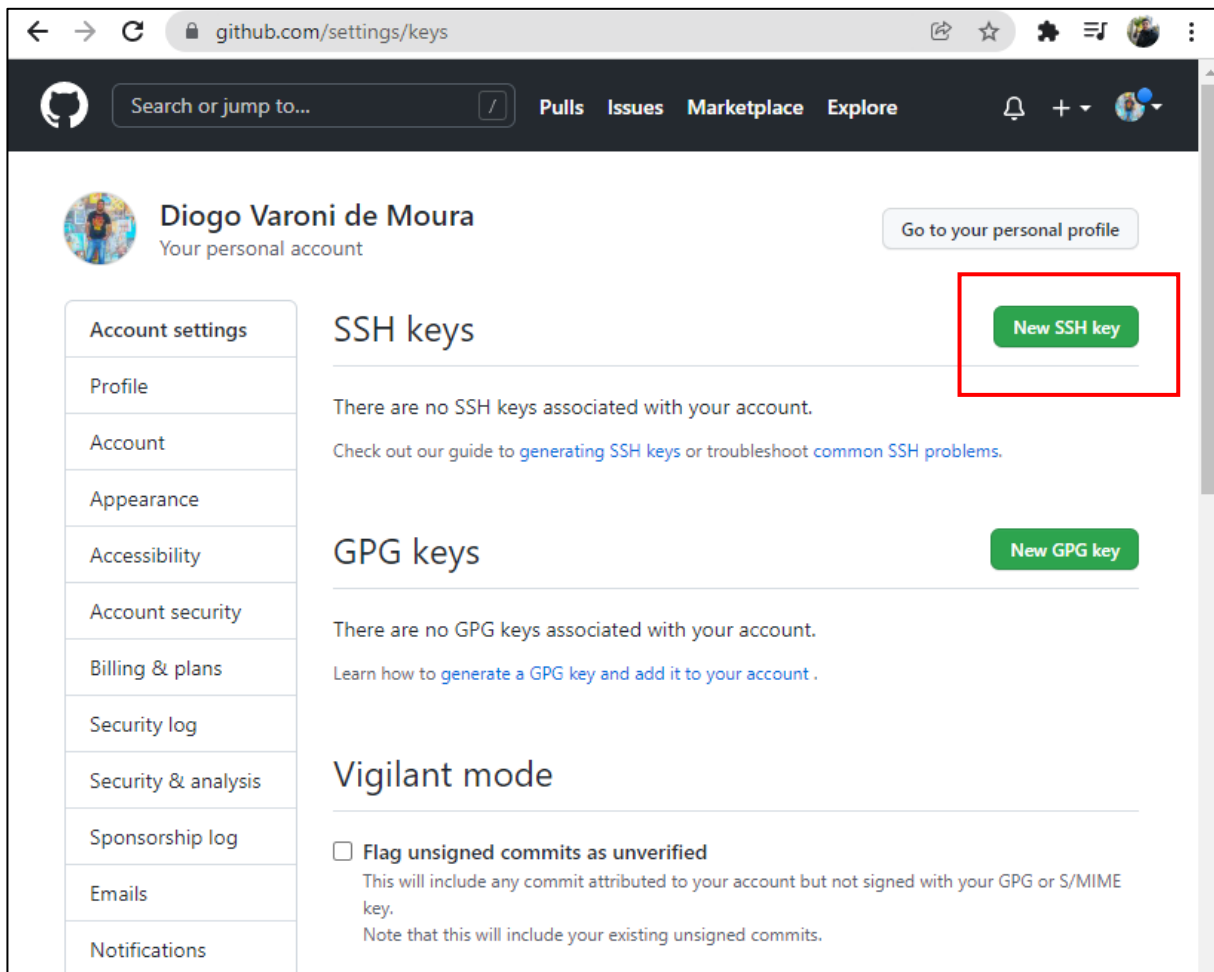
Esses processos de autenticação surgiram para fornecer mais segurança na hora de jogar o código para o GitHub. Antes esse processo era feito apenas com nome de usuário e senha mas o GitHub viu a necessidade de aumentar a segurança criando métodos de autenticação de seus usuários.

- **Chave SSH:** é uma forma de estabelecer uma conexão segura e encriptada de duas máquinas. Como iremos nos conectar com o servidor do GitHub, nós iremos configurar nossa máquina local como uma máquina confiável para o GitHub, estabelecendo essa conexão com duas chaves, sendo uma máquina pública e outra privada. Siga os passos a seguir para criar a chave SSH.

1º Passo: na plataforma do GitHub clique na imagem do perfil e em seguida no botão *Settings*.

The screenshot shows the GitHub profile page for Diogo Varoni de Moura. The profile picture is a circular image of a man standing in front of a wall covered in posters. The name 'Diogo Varoni de Moura' and the username 'diogovaroni' are displayed below the profile picture. A button labeled 'Edit profile' is visible. The page shows 'Popular repositories' and a message 'You don't have any public repositories yet.' Below this, it indicates '22 contributions in the last year' with a calendar grid showing contributions for June, July, August, and September. A dropdown menu is open on the right side of the page, showing the user is signed in as 'diogovaroni'. The menu includes options like 'Set status', 'Your profile', 'Your repositories', 'Your codespaces', 'Your projects', 'Your stars', 'Your gists', 'Upgrade', 'Feature preview', 'Help', 'Settings' (highlighted in blue), and 'Sign out'. At the bottom of the menu, there is a 'Less' button and a 'More' button.

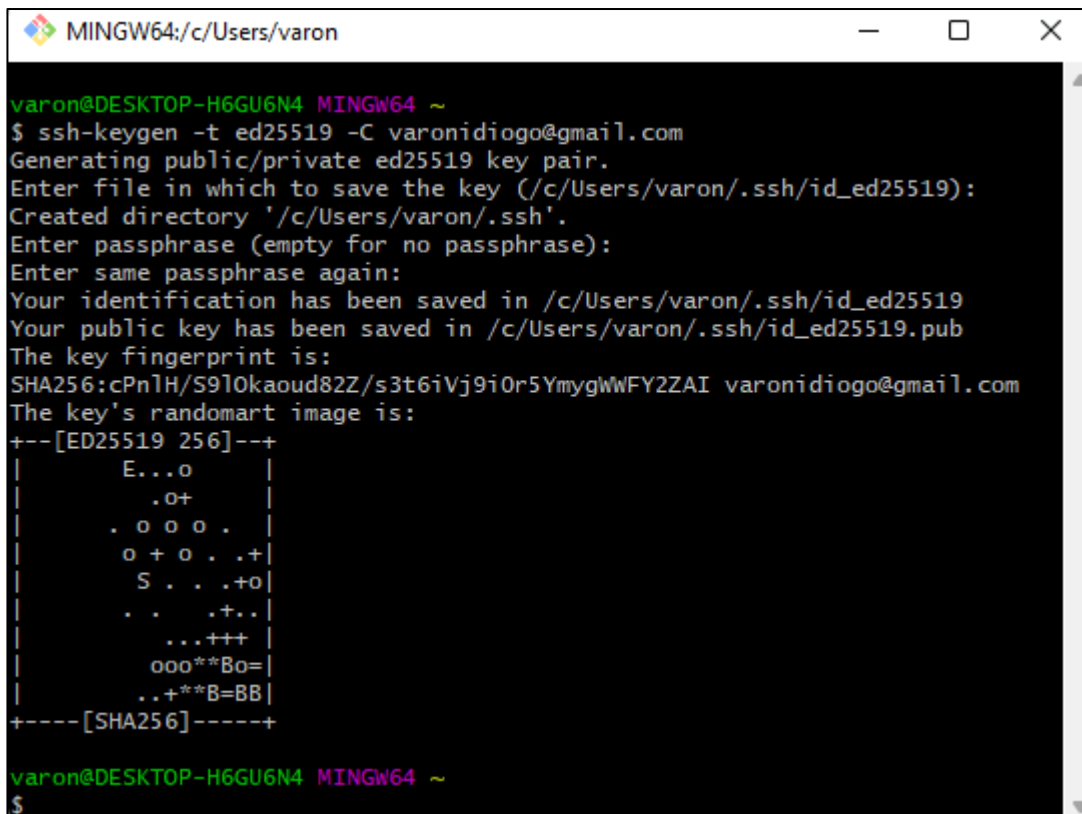
2º Passo: clique na opção “SSH and GPG Keys”. Em seguida clique no botão verde “New SSH key”.



3º Passo: agora vamos fazer o processo pelo terminal (CLI) do Git para gerar essa chave e deixar ela ligada na nossa máquina. Abra o Git Bash na área de trabalho e digite o comando **ssh-keygen -t ed25519 -c 'e-mail'**. Com esse comando serão criadas as chaves e irá mostrar o local onde essas chaves irão ‘morar’.

```
MINGW64:/c/Users/varon
varon@DESKTOP-H6GU6N4 MINGW64 ~
$ ssh-keygen -t ed25519 -C varonidiogo@gmail.com
Generating public/private ed25519 key pair.
Enter file in which to save the key (/c/Users/varon/.ssh/id_ed25519): |
```

Perceba que na última linha do CLI é mostrado onde a chave gerada será salva, sendo nesse caso salva na pasta “usuarios”, dentro do diretório “varon”. Esse é o melhor caminho para salvar a chave gerada. Ao teclar “Enter” o CLI irá solicitar a senha do usuário para gerar a chave SSH e caso não haja senha, basta teclar Enter mais uma vez.



```

MINGW64:/c/Users/varon
varon@DESKTOP-H6GU6N4 MINGW64 ~
$ ssh-keygen -t ed25519 -C varonidiogo@gmail.com
Generating public/private ed25519 key pair.
Enter file in which to save the key (/c/Users/varon/.ssh/id_ed25519):
Created directory '/c/Users/varon/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /c/Users/varon/.ssh/id_ed25519
Your public key has been saved in /c/Users/varon/.ssh/id_ed25519.pub
The key fingerprint is:
SHA256:cPn1H/S9l0kaoud82Z/s3t6iVj9i0r5YmyglWwFY2ZAI varonidiogo@gmail.com
The key's randomart image is:
+--[ED25519 256]--+
|      E...o      |
|      .o+       |
|    . o o o .   |
|    o + o . .+  |
|    S . . .+o   |
|    . . . .+..  |
|    ...+++      |
|    ooo**Bo=    |
|    ..+**B=BB   |
+-----[SHA256]-----+

varon@DESKTOP-H6GU6N4 MINGW64 ~
$

```

Finalizado esse processo de gerar a chave, vamos navegar entre os diretórios até onde a chave foi salva e visualizar essa chave. Para navegar direto para esse diretório, basta digitar o comando **“cd /c/Users/varon/.ssh/”**. Em seguida digite o comando **“ls”** para listar as chaves pública e privada.



```

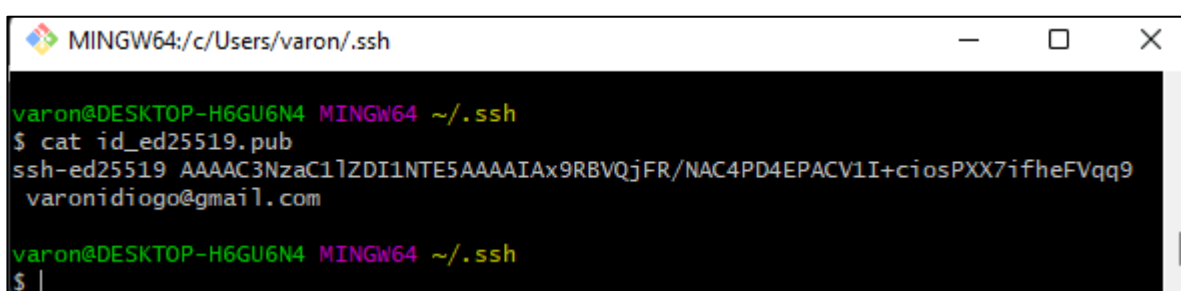
MINGW64:/c/Users/varon/.ssh
varon@DESKTOP-H6GU6N4 MINGW64 ~
$ cd .ssh/

varon@DESKTOP-H6GU6N4 MINGW64 ~/.ssh
$ ls
id_ed25519  id_ed25519.pub

varon@DESKTOP-H6GU6N4 MINGW64 ~/.ssh
$ |

```

Agora vamos usar um comando especial para visualizar o conteúdo dessas chaves. Esse é o conteúdo que iremos utilizar no Git Hub. Digite o comando **“cat id_ed25519.pub”**, lembrando que a chave que iremos utilizar e expor no Git Hub é a chave pública.



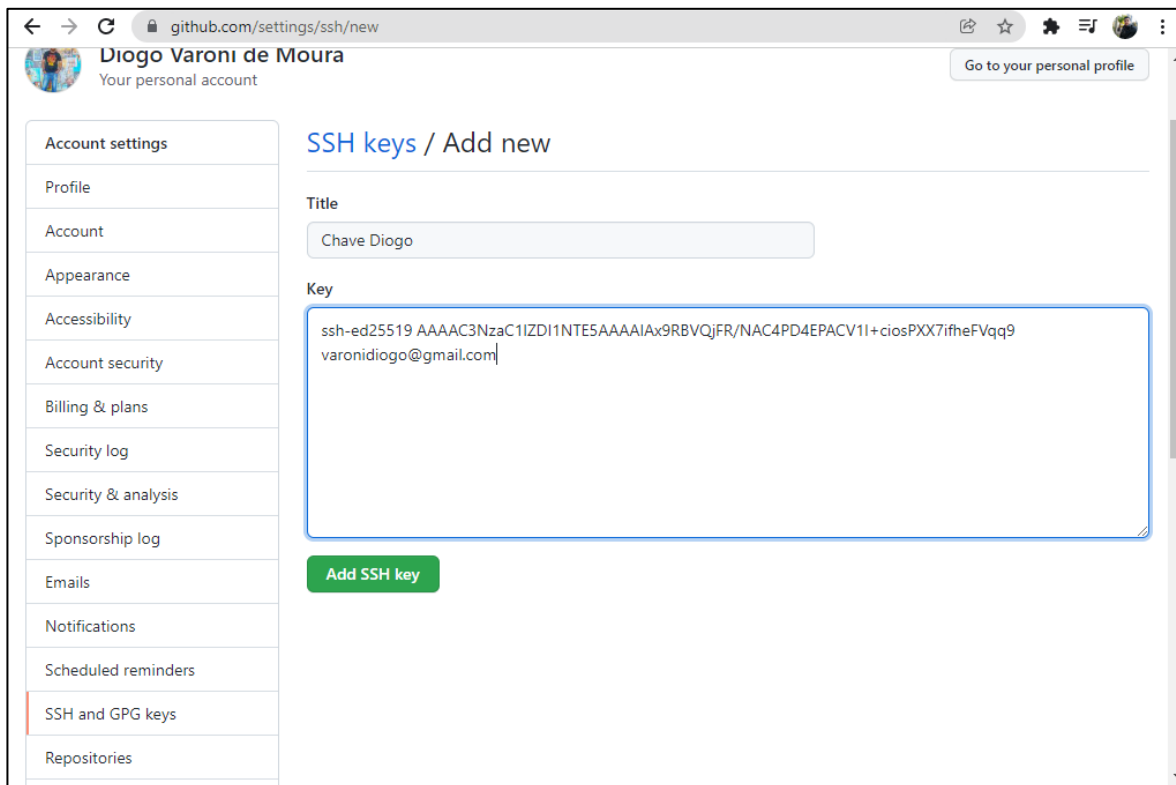
```

MINGW64:/c/Users/varon/.ssh
varon@DESKTOP-H6GU6N4 MINGW64 ~/.ssh
$ cat id_ed25519.pub
ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAIAx9RBVQjFR/NAC4PD4EPACV1I+ciosPXX7ifheFVqq9
varonidiogo@gmail.com

varon@DESKTOP-H6GU6N4 MINGW64 ~/.ssh
$ |

```

Vamos copiar esse conteúdo e ir lá no GitHub para colar ele. O GitHub irá solicitar que o usuário se autentique com senha.



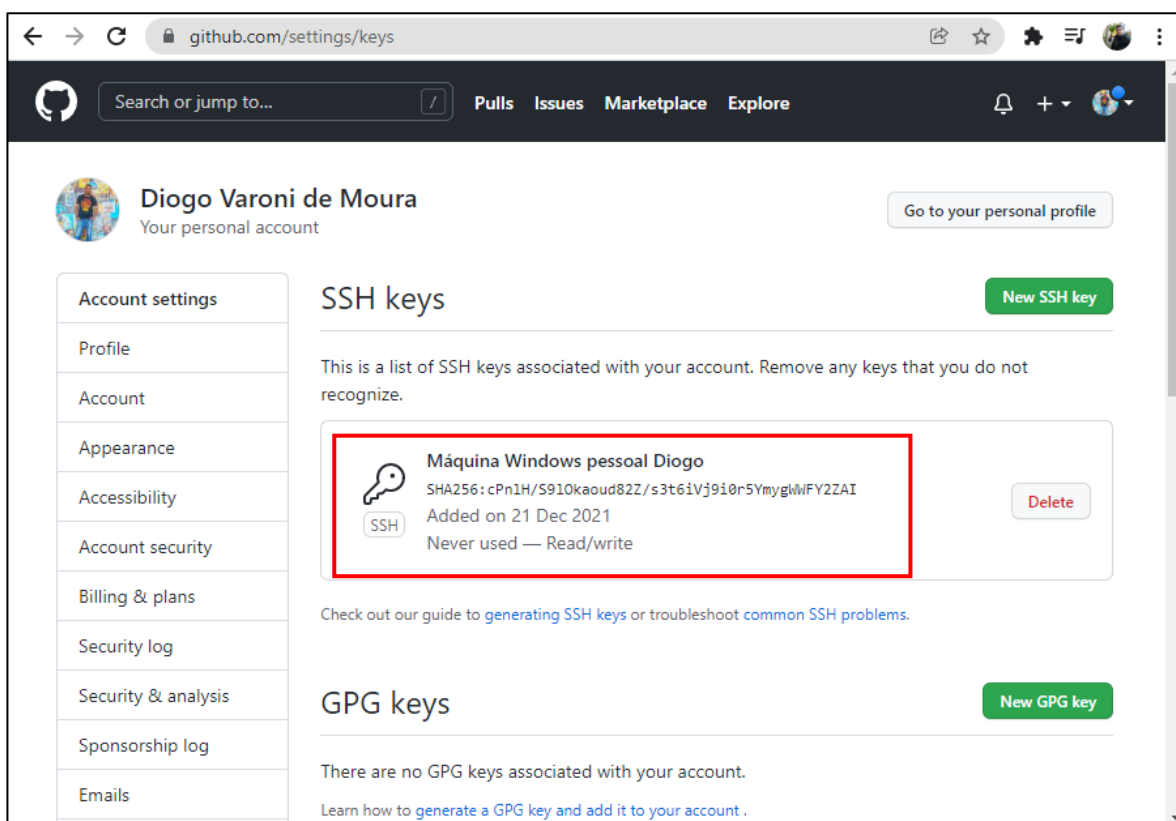
The screenshot shows the GitHub 'SSH keys / Add new' page for the user 'Diogo Varoni de Moura'. On the left is a sidebar with account settings. The main area has a form with a 'Title' field containing 'Chave Diogo' and a 'Key' text area containing the SSH key: `ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAIAx9RBVQJfR/NAC4PD4EPACV1I+ciosPXX7ifheFVqq9varonidiogo@gmail.com`. A green 'Add SSH key' button is at the bottom.

Account settings / SSH keys / Add new

Title: Chave Diogo

Key: `ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAIAx9RBVQJfR/NAC4PD4EPACV1I+ciosPXX7ifheFVqq9varonidiogo@gmail.com`


Add SSH key



The screenshot shows the GitHub 'SSH keys' page. It lists the SSH key added in the previous step, highlighted with a red box. The key is titled 'Máquina Windows pessoal Diogo' and was added on 21 Dec 2021. Below the list is a link to a guide on generating SSH keys. The 'GPG keys' section below is empty.

SSH keys

This is a list of SSH keys associated with your account. Remove any keys that you do not recognize.

SSH key icon	Key details	Action
	Máquina Windows pessoal Diogo SHA256:cPn1H/S910kaoud82Z/s3t6iVj9i0r5YmygwWfY2ZAI Added on 21 Dec 2021 Never used — Read/write	Delete

Check out our guide to [generating SSH keys](#) or troubleshoot [common SSH problems](#).

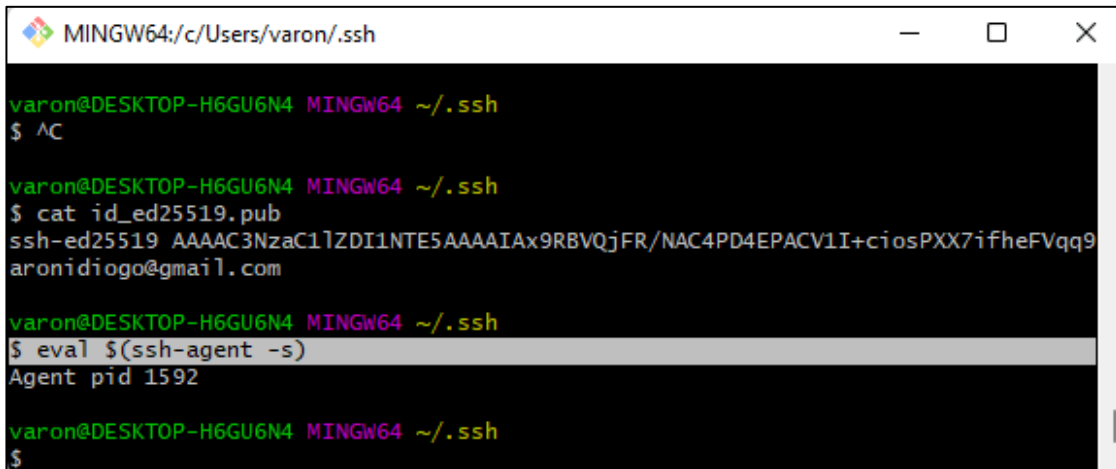
GPG keys

There are no GPG keys associated with your account.

Learn how to [generate a GPG key and add it to your account](#).

Pronto. A chave SSH foi gerada. Na página do GitHub é mostrado a lista de chaves criadas na conta, assim como a chave atual criada. Agora precisamos fazer mais um processo no **CLI** para que a chave funcione sem erros. Vamos inicializar o SSH Agent, que é uma

entidade que será encarregada de pegar as chaves e trabalhar com elas. Digite no CLI o comando “**eval \$(ssh-agent -s)**”.



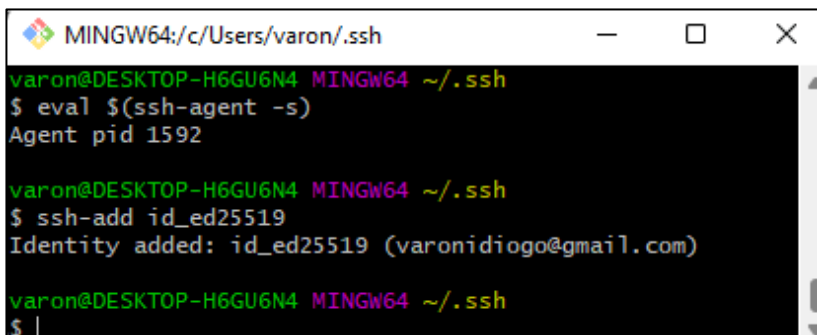
```
MINGW64:/c/Users/varon/.ssh
varon@DESKTOP-H6GU6N4 MINGW64 ~/.ssh
$ ^C

varon@DESKTOP-H6GU6N4 MINGW64 ~/.ssh
$ cat id_ed25519.pub
ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAIAx9RBVQjFR/NAC4PD4EPACV1I+ciosPXX7ifheFVqq9
aronidiogo@gmail.com

varon@DESKTOP-H6GU6N4 MINGW64 ~/.ssh
$ eval $(ssh-agent -s)
Agent pid 1592

varon@DESKTOP-H6GU6N4 MINGW64 ~/.ssh
$
```

Ao gerar o agente, vamos entregar a nossa chave **privada** para ele. Vamos digitar o comando “**ssh-add id_ed25519**”. Caso a chave não esteja no mesmo diretório em que o CLI está aberto, basta navegar até o diretório ou digitar o caminho direto no CLI. Toda vez que chegar a criptografia com essa chave, o agente irá descriptografar ela.



```
MINGW64:/c/Users/varon/.ssh
varon@DESKTOP-H6GU6N4 MINGW64 ~/.ssh
$ eval $(ssh-agent -s)
Agent pid 1592

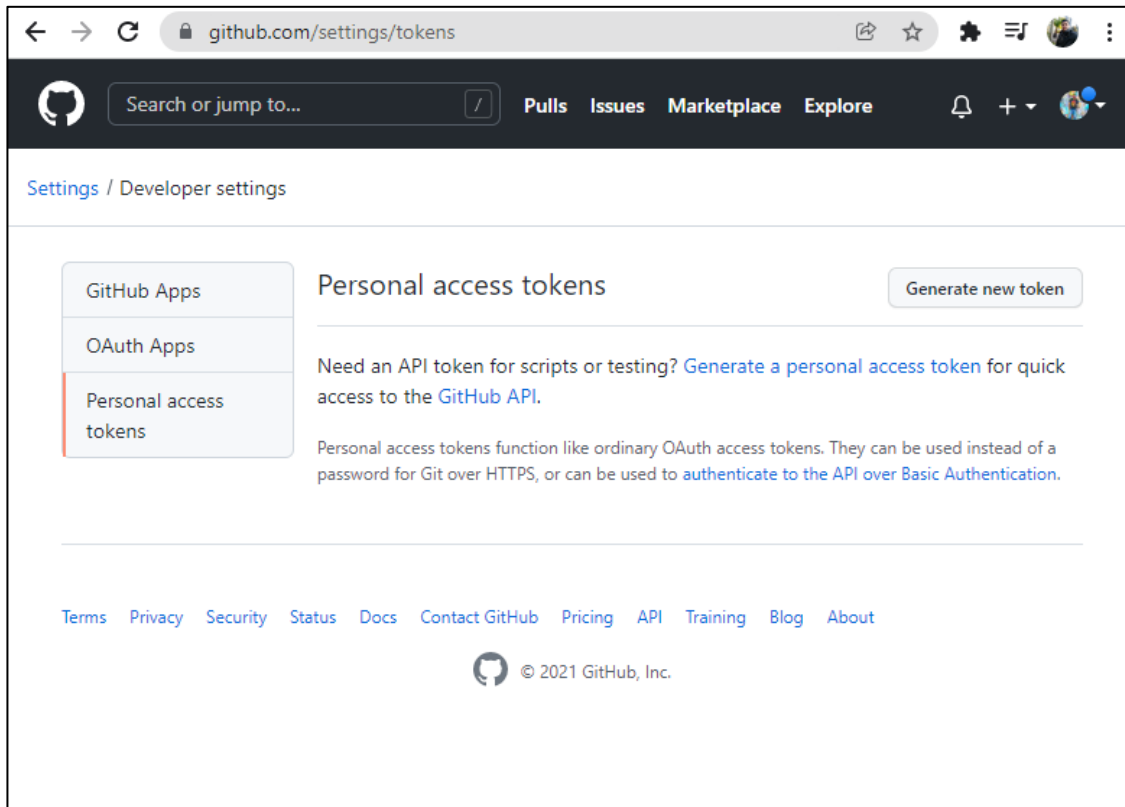
varon@DESKTOP-H6GU6N4 MINGW64 ~/.ssh
$ ssh-add id_ed25519
Identity added: id_ed25519 (varonidiogo@gmail.com)

varon@DESKTOP-H6GU6N4 MINGW64 ~/.ssh
$ |
```

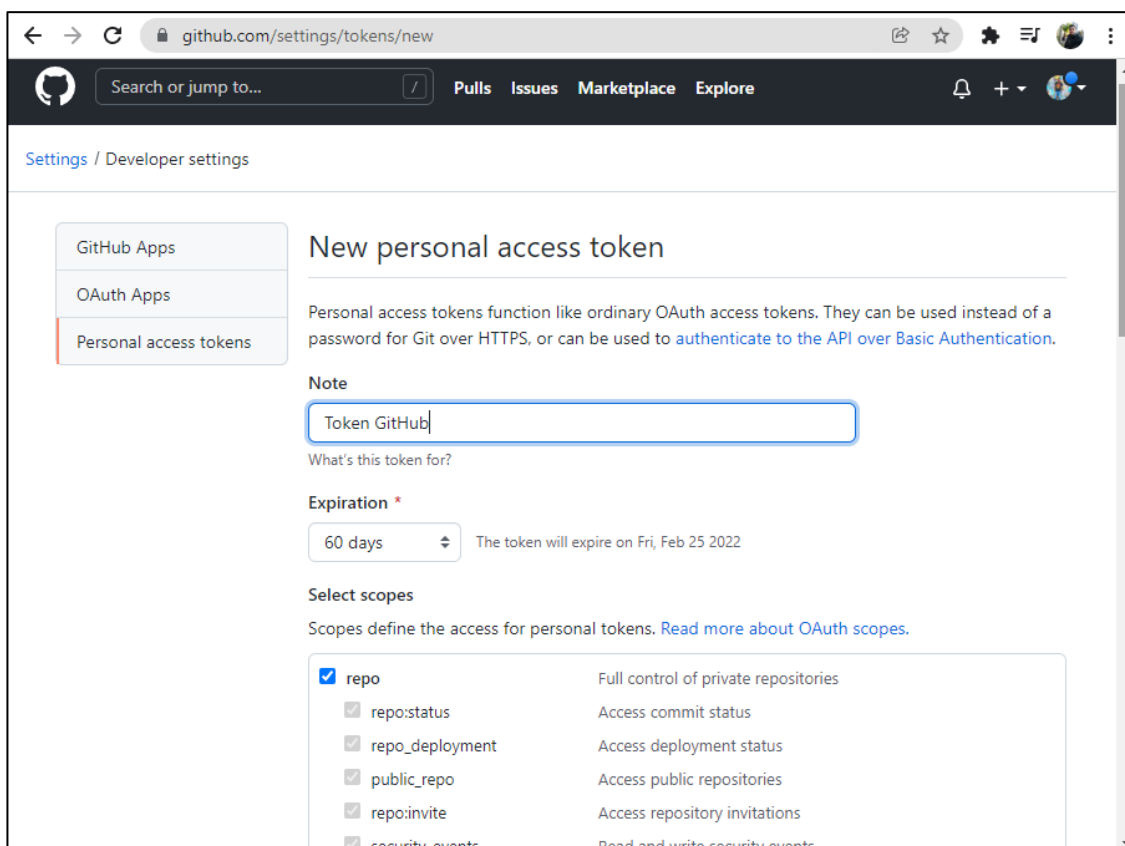
Pronto! O processo foi finalizado. Perceba que o CLI retornou que a identidade foi adicionada e mostrou o e-mail da conta no GitHub, que neste caso é varonidiogo@gmail.com.

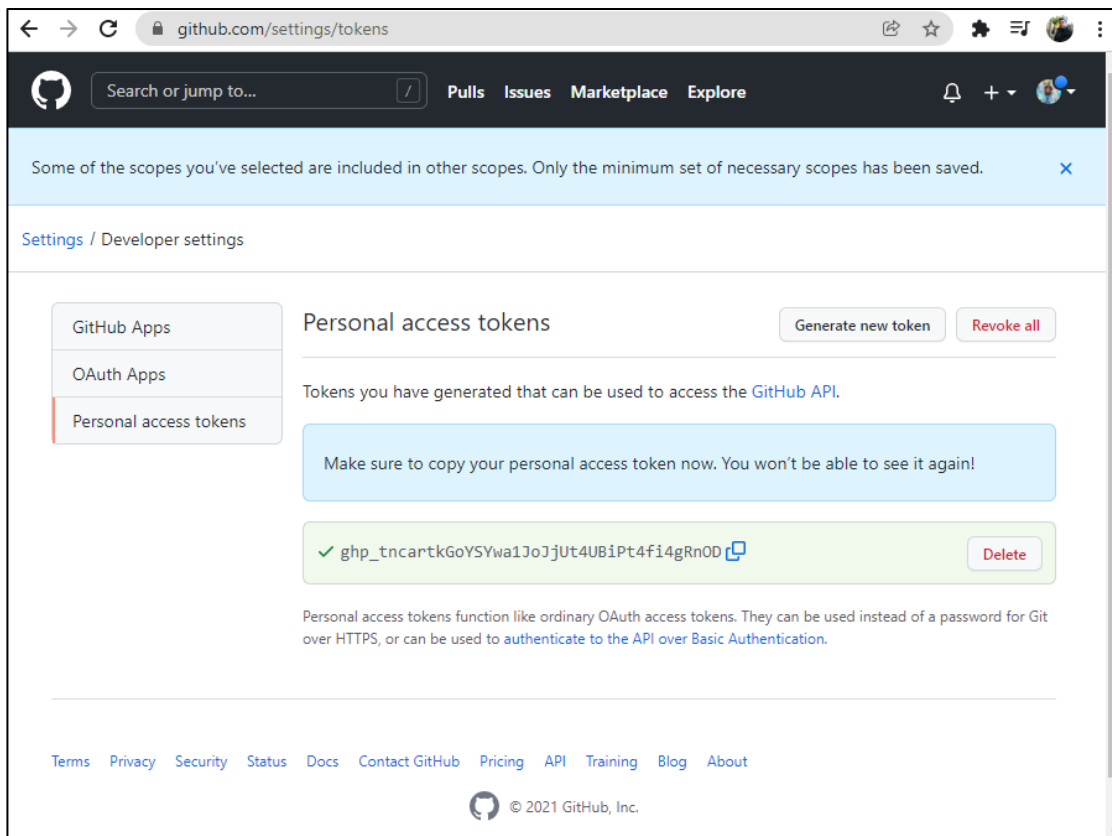
- **Token:** o token de acesso pessoal é a segunda forma de identificação segura que o GitHub oferece para os usuários. É um processo que se assemelha mais ao procedimento usando antes, que era digitar o nome de usuário e a senha. Iremos gerar o *Token* no GitHub, guardar ele na máquina e sempre que fizermos um commit, o Git irá solicitar o nome de usuário e a senha e em seguida o *Token* de acesso.

1º Passo: no GitHub selecione a opção *Settings* e em seguida a opção *Developer settings*. Agora selecione a opção **Personal access tokens** e clique no botão **Generate new token**.



2º Passo: preencha o campo **Note** e selecione a data que o token criado irá expirar. Em seguida marque a opção **Repo**. Agora clique no botão **Generate token**.





3º Passo: após gerar o Token copie e salve em algum arquivo seguro pois ele não fica salvo no GitHub. Se sairmos da página sem salvar o Token, teremos que gerar novamente. A seguir é dado o token criado: **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**.

Para clonar o repositório privado no GitHub usando o Token de acesso, devemos usar o protocolo HTTPS.