Sumário

1 - Script

Um arquivo texto que contem uma sequência de comando de forma ordenada (sequencial).

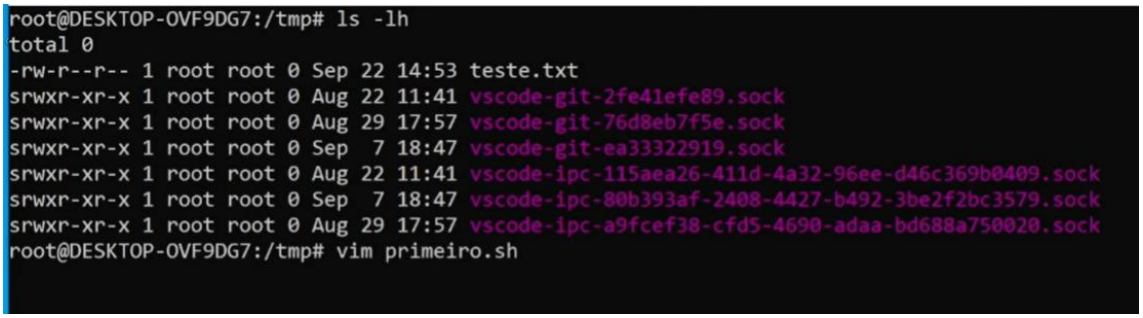
Instalar o WSL.

- Como boa pratica é importante dar uma extensão para o arquivo.

2 - Comandos

Vim:

Vim primeiro.sh



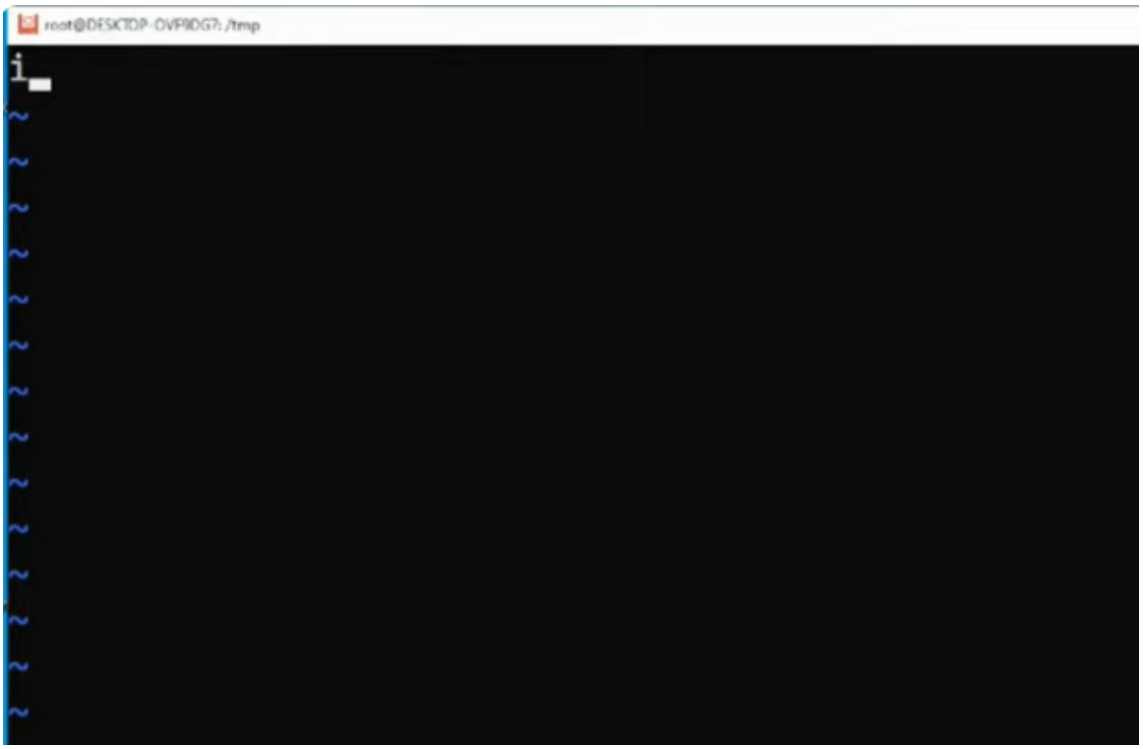
primeiro = arquivo

.sh = extensão do arquivo

Para criar um script tem que colocar uma primeira marcação.
Essa marcação se chama shibei.

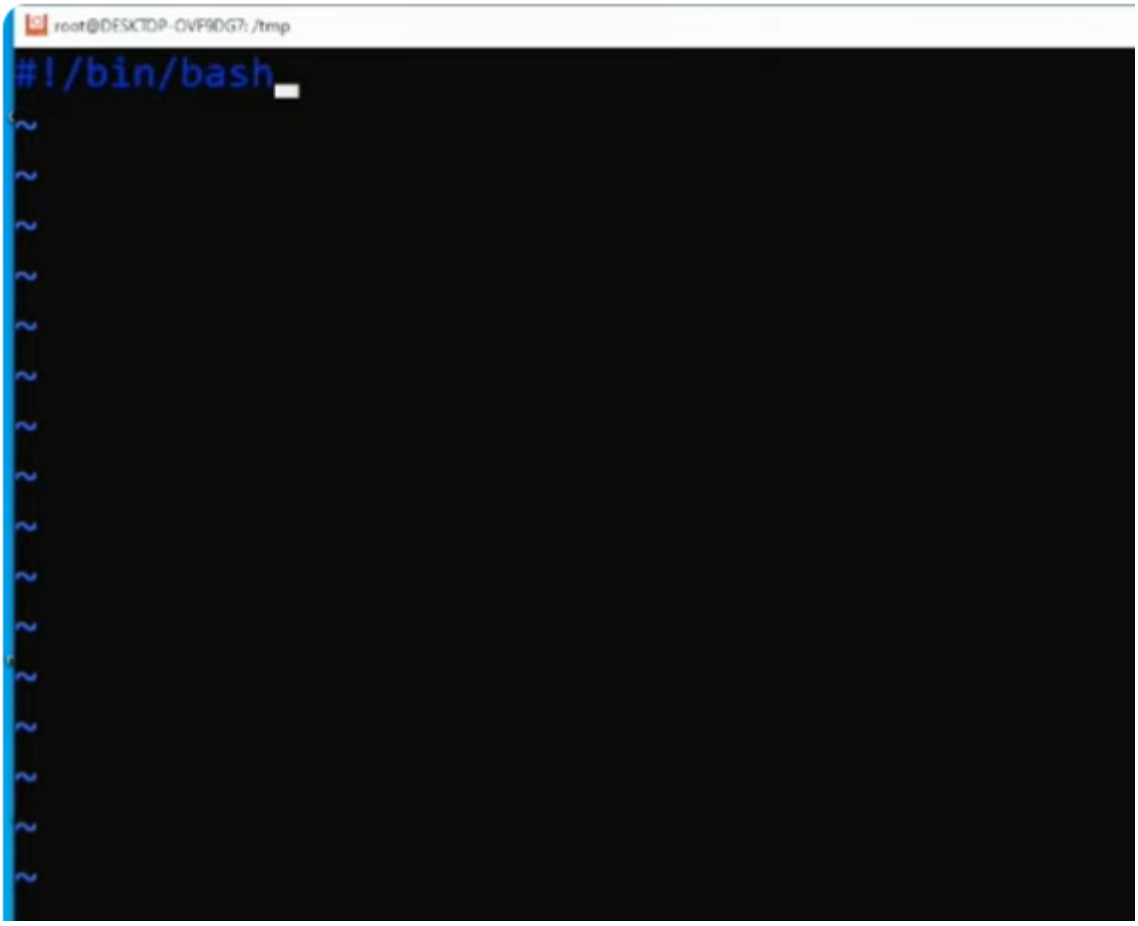
Comando para abrir modo de edição:

Digitar “i” para entrar no modo edição.



Comando para usar interpretador padrão:

Logo após digitar #!/bin/bash



Esse comando informa que, caso eu não especificar o shell interpretador use esse. (Caso eu não especifique = “(#!)”.

Primeira linha de código de programação:
“Hello Word”

Comando para exibir mensagem na tela: Echo “ Hello word”

Comando de saída do terminal = Esc, shift : wp, para sair e salvar

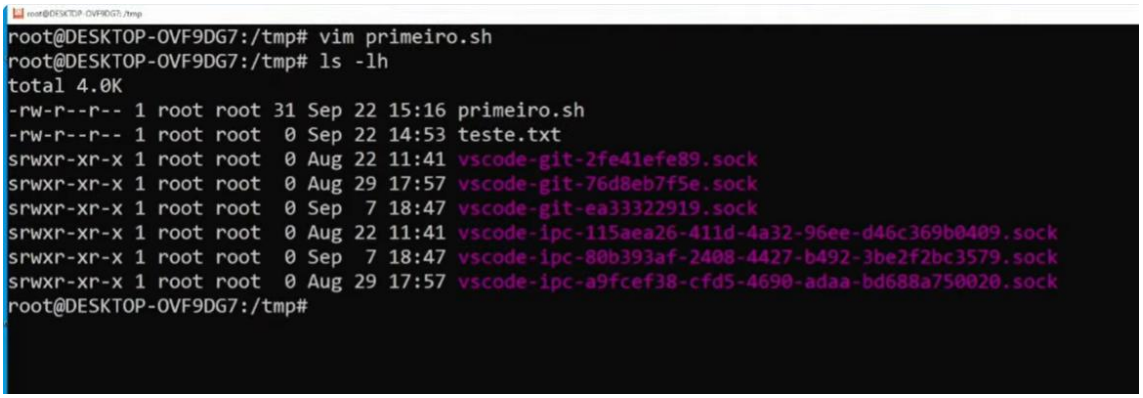
Volta para tela inicial.

Permissão.

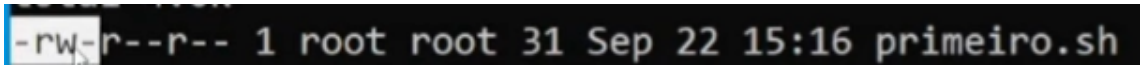
Para executar o script é importante ter permissão.

Comando :

Ls – Lh

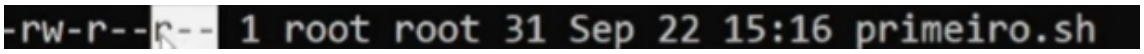


Para **o dono do arquivo**: Tem permissão de leitura e escrita (r e w), mas não de execução

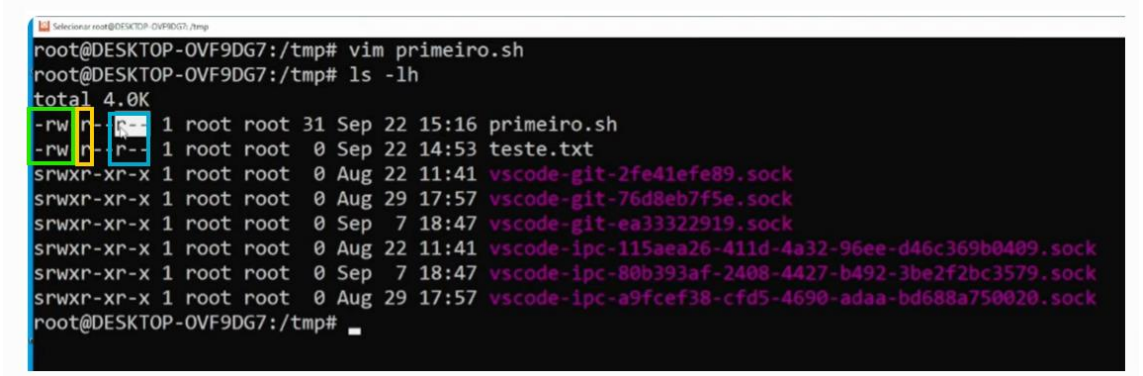


Para o grupo do dono do arquivo: tem permissão de leitura (r), mas não tem para escrita e execução.

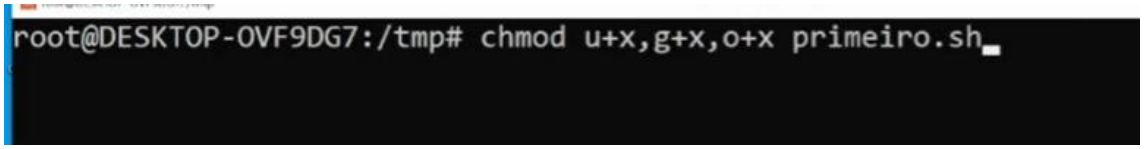
Para outros: tem permissão de leitura (r), mas não tem para escrita e execução.



O primeiro bloco informa as permissões do usuário, o segundo bloco do grupo do usuário e o terceiro bloco para outros.



Comando para dar permissão :



Chmod

U= usurio

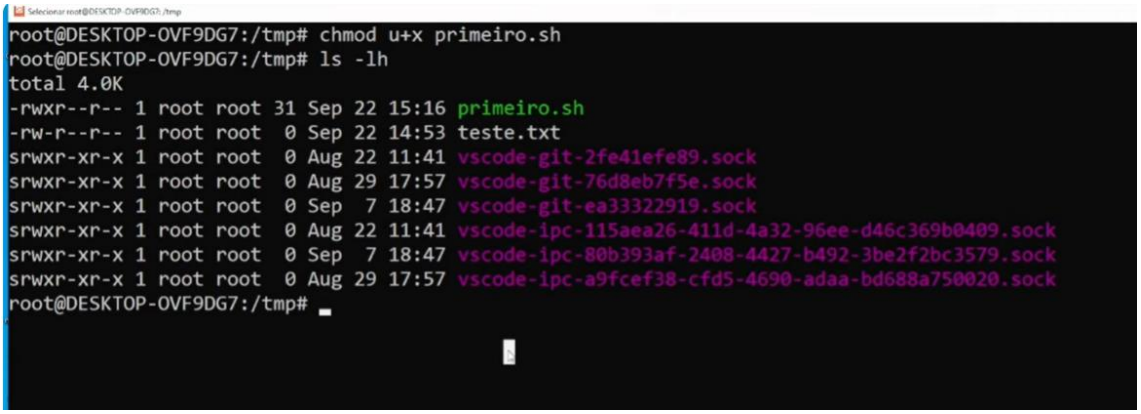
G= grupo

O= outros

X = permissão de leitura

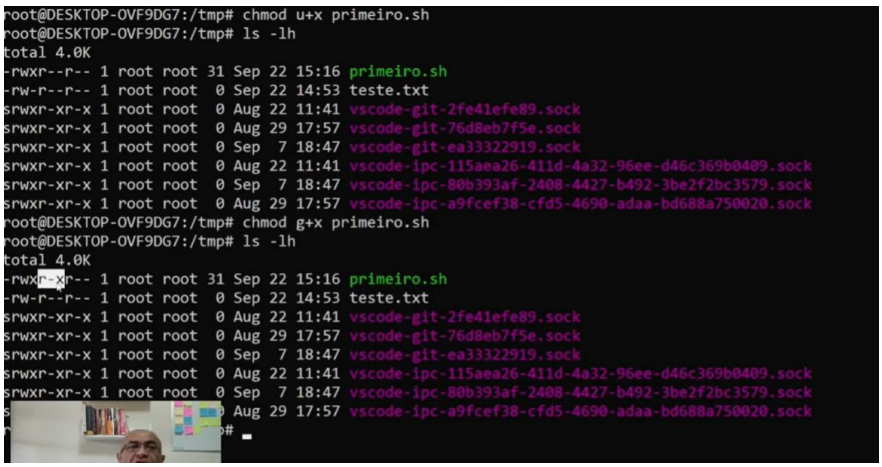
- Permissão somente para usuário:

Chmod u+ primeiro.sh



Permissão somente para grupo :

Chmod g+x primeiro.sh



Permissão somente para outros :

Chmod o+x primeiro.sh

```
root@DESKTOP-OVF9DG7:/tmp# chmod o+x primeiro.sh
root@DESKTOP-OVF9DG7:/tmp# ls -lh
total 4.0K
-rwxr-xr-x 1 root root 31 Sep 22 15:16 primeiro.sh
-rw-r--r-- 1 root root  0 Sep 22 14:53 teste.txt
srwxr-xr-x 1 root root  0 Aug 22 11:41 vscode-git-2fe41efe89.sock
srwxr-xr-x 1 root root  0 Aug 29 17:57 vscode-git-76d8eb7f5e.sock
srwxr-xr-x 1 root root  0 Sep  7 18:47 vscode-git-ea33322919.sock
srwxr-xr-x 1 root root  0 Aug 22 11:41 vscode-ipc-115aea26-411d-4a32-96ee-d46c369b0409.sock
srwxr-xr-x 1 root root  0 Sep  7 18:47 vscode-ipc-80b393af-2408-4427-b492-3be2f2bc3579.sock
srwxr-xr-x 1 root root  0 Aug 29 17:57 vscode-ipc-a9fcef38-cfd5-4690-adaa-bd688a750020.sock
root@DESKTOP-OVF9DG7:/tmp# ch
```

Comando direto para dar permissão todos no meu arquivo:
chmod +x primeiro.sh

```
root@DESKTOP-OVF9DG7:/tmp# chmod +x primeiro.sh
root@DESKTOP-OVF9DG7:/tmp# ls -lh
total 4.0K
-rwxr-xr-x 1 root root 31 Sep 22 15:16 primeiro.sh
-rw-r--r-- 1 root root  0 Sep 22 14:53 teste.txt
srwxr-xr-x 1 root root  0 Aug 22 11:41 vscode-git-2fe41efe89.sock
srwxr-xr-x 1 root root  0 Aug 29 17:57 vscode-git-76d8eb7f5e.sock
srwxr-xr-x 1 root root  0 Sep  7 18:47 vscode-git-ea33322919.sock
srwxr-xr-x 1 root root  0 Aug 22 11:41 vscode-ipc-115aea26-411d-4a32-96ee-d46c369b0409.sock
srwxr-xr-x 1 root root  0 Sep  7 18:47 vscode-ipc-80b393af-2408-4427-b492-3be2f2bc3579.sock
srwxr-xr-x 1 root root  0 Aug 29 17:57 vscode-ipc-a9fcef38-cfd5-4690-adaa-bd688a750020.sock
```

Comando para tirar permissões:
Chmod u-x,g-x,o-x primeiro.sh

```
root@DESKTOP-OVF9DG7:/tmp# chmod o-x,g-x,u-x primeiro.sh
root@DESKTOP-OVF9DG7:/tmp# ls -lh
total 4.0K
-rw-r--r-- 1 root root 31 Sep 22 15:16 primeiro.sh
-rw-r--r-- 1 root root  0 Sep 22 14:53 teste.txt
srwxr-xr-x 1 root root  0 Aug 22 11:41 vscode-git-2fe41efe89.sock
srwxr-xr-x 1 root root  0 Aug 29 17:57 vscode-git-76d8eb7f5e.sock
srwxr-xr-x 1 root root  0 Sep  7 18:47 vscode-git-ea33322919.sock
srwxr-xr-x 1 root root  0 Aug 22 11:41 vscode-ipc-115aea26-411d-4a32-96ee-d46c369b0409.sock
srwxr-xr-x 1 root root  0 Sep  7 18:47 vscode-ipc-80b393af-2408-4427-b492-3be2f2bc3579.sock
srwxr-xr-x 1 root root  0 Aug 29 17:57 vscode-ipc-a9fcef38-cfd5-4690-adaa-bd688a750020.sock
root@DESKTOP-OVF9DG7:/tmp#
```

Comandos para executar um script (Três tipos);

1ª “ ./primeito.sh ” (Usa-se o bin/bash)

2ª “ . primeito.sh ” (Usa-se o bin/bash)

3ª “ sh primeito.sh ” (Sh é o Shell interpretador, esse comando desconsidera a primeira linha #!/bin/bash)

Comando para verificar qual intrepretador estásendo usado:
cat primeiro.sh

Comandos de Saida:

No editor de texto **Vim** (ou **Vi**) e, acidentalmente, ativou a função de **gravação de macro** (aquele recording @a lá embaixo).

O Ctrl + r no Bash funciona de um jeito, mas dentro do Vim ele serve para "Refazer" (Redo).

Como sair dessa tela e voltar para o terminal:

Para sair do Vim e descartar qualquer alteração, digite exatamente esta sequência:

1. Aperte a tecla **Esc** (para garantir que você saiu do modo de gravação/edição).
2. Digite :w
3. Aperte **Enter**. Para salvar
4. Digite **:q!**
5. Aperte **Enter**. Para sair do editor vim.

Salvar e sair

- **Para salvar as alterações:** no modo normal, digite :w e pressione Enter.
- **Para sair do Vim:**
    - :q: sai se não houver alterações.
    - :q!: sai sem salvar as alterações.
    - :wq: salva e sai.

Por que os comandos anteriores não funcionaram?

No Linux, o comportamento das teclas muda dependendo do programa que está aberto:

- **No Terminal (Bash):** Ctrl + r pesquisa o histórico.
- **No Vim (Editor de texto):** O terminal "some" e você entra em um ambiente de edição. Para sair do Vim, os atalhos de terminal como Ctrl + c ou Ctrl + g geralmente não fecham o programa, apenas cancelam comandos internos.

Dica de Shell Script

Se você caiu nessa tela tentando editar um arquivo (ex: vim script.sh), lembre-se desses 3 comandos básicos do Vim:

- **i**: Entra no modo de **Inserção** (para você escrever seu código).
- **Esc**: Sai de qualquer modo e volta para o modo de comando.
- **:wq**: Salva e sai (Write and Quit).

Resumo de Comando:

Para exibir o caminho do diretório em que você está no momento, use o comando pwd:

Bash

Pwd

Para criar um novo diretório, use o mkdir comando seguido pelo nome do diretório que você deseja criar:

Bash

mkdir hello_world

Para alterar diretórios, use o cd comando seguido pelo nome do diretório para o qual você deseja navegar:

Bash

cd hello_world

Para ver o conteúdo no diretório em que você está atualmente, digite ls na linha de comando:

Bash

ls

Por padrão, o ls comando imprimirá o nome de todos os arquivos e diretórios somente. Para obter informações adicionais, como a última vez que um arquivo foi modificado ou permissões de arquivo, use o sinalizador -l:

Bash

ls -l

Você pode criar um novo arquivo por meio do touch comando seguido pelo nome do arquivo que deseja criar:

Bash

touch hello_world.txt

Você pode editar arquivos usando qualquer editor de texto gráfico baixado ou a extensão DO VS Code Remote – WSL. Você pode saber mais sobre como começar a usar o VS Code aqui.

Se você preferir editar um arquivo diretamente da linha de comando, precisará usar um editor de linha de comando, como vim, emacsou nano. Muitas distribuições vêm

com um ou mais desses programas instalados, mas você sempre pode instalar esses programas seguindo as instruções de instalação descritas no guia acima.

Para editar seu arquivo com seu método preferencial de edição, basta executar o nome do programa seguido pelo nome do arquivo que você deseja editar:

Bash

code hello_world.txt

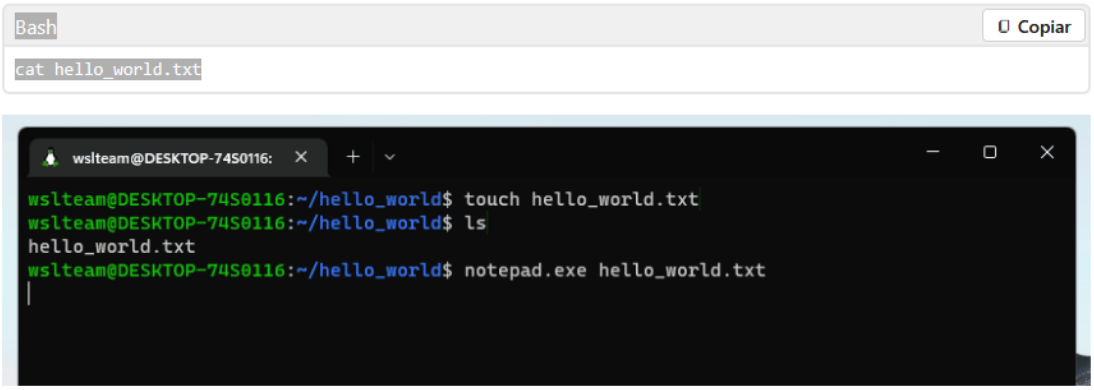
Bash

notepad.exe hello_world.txt

Para ver o conteúdo de um arquivo na linha de comando, use o cat comando seguido pelo arquivo que você deseja ler:

Bash

cat hello_world.txt



3 - COMO USAR VARIÁVEIS EM PROGRAMAÇÃO COM SHELL SCRIPT

O que é uma variável: Um espaço em memória alocado para armazenar alguma informação, na qual o sistema vai utilizar.

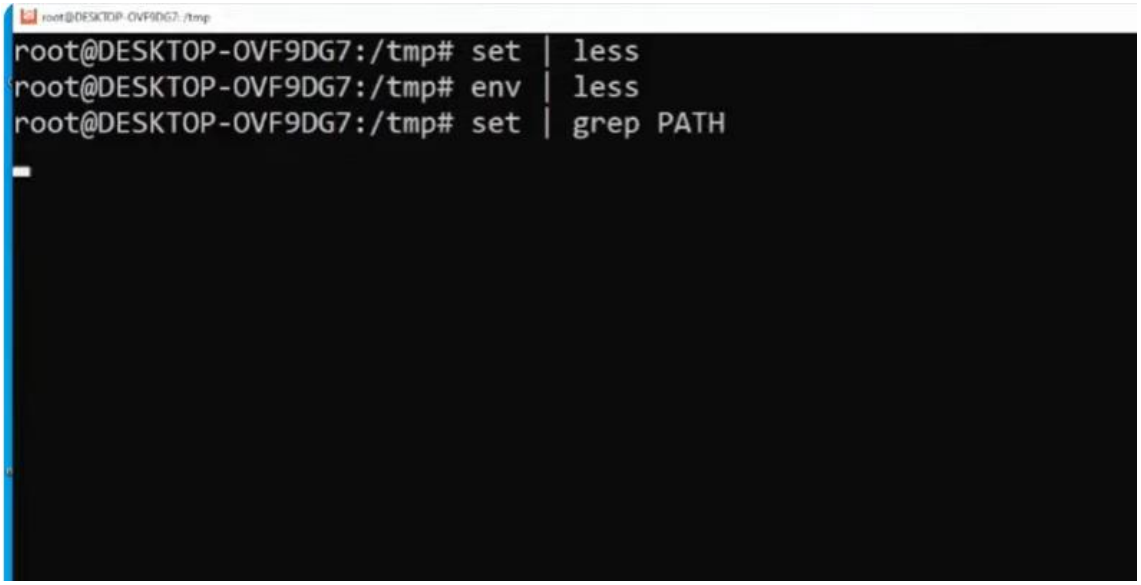
Variáveis de escopo local: de acordo com a seção que está.

Variáveis de escopo global: todo sistema faz uso dessas variáveis e independente da seção que você está, terá visibilidade dessa variável.

Comandos:

set = Visibilidade de variáveis local

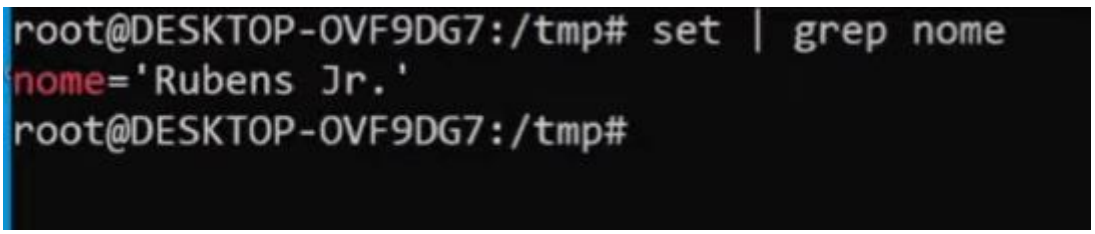
env – Visibilidade de variáveis global



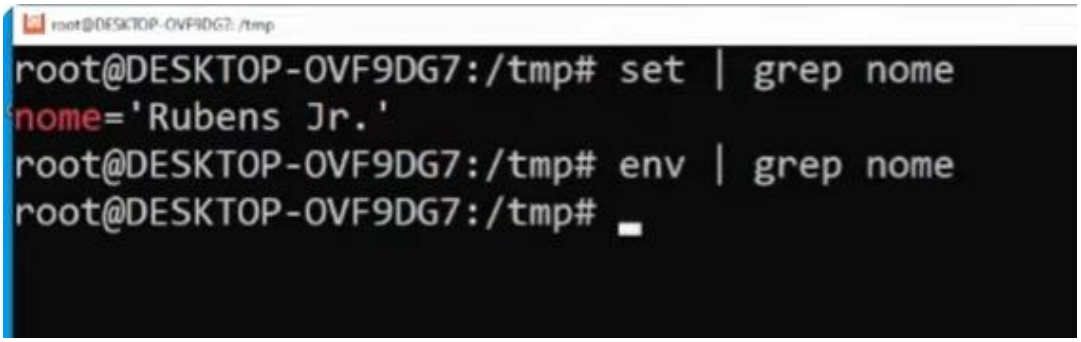
Declarar variável no Linux:

Nome=”Rubens Jr.”

Para verificar o conteúdo da variável local = set | grep nome

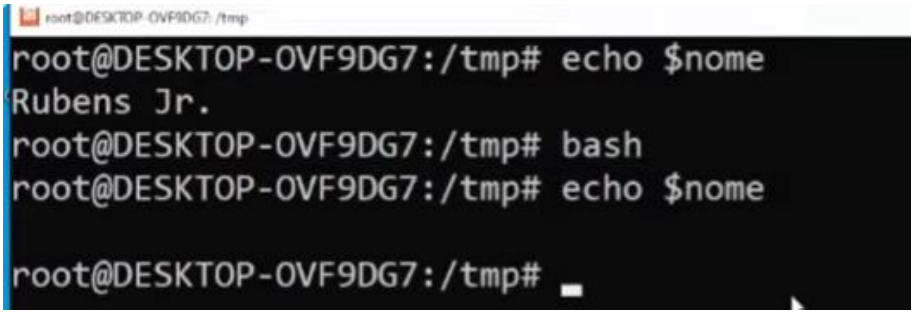


Para verificar o conteúdo da variável global = env | grep nome



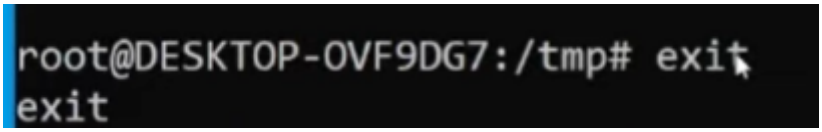
Para ler o conteúdo de uma variável usa-se o comando(Echo+ símbolo de dólar junto com o nome da variável):
echo $nome

Comando para abrir uma nova seção:
Bash



Comando para sair e voltar a seção anterior:

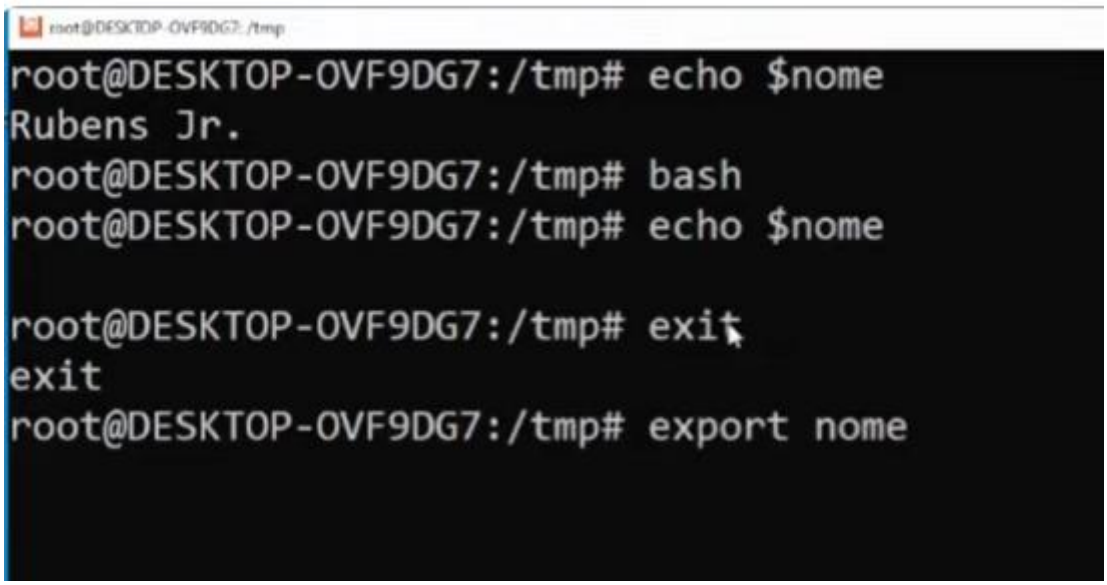
exit



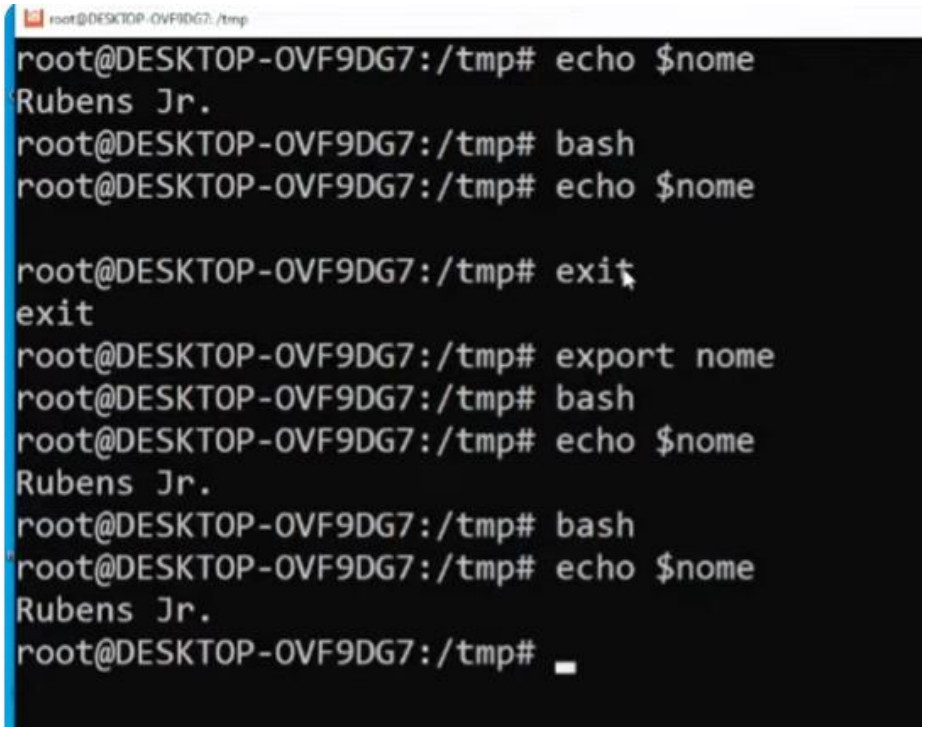
Para essa variável se tornar disponível para demais seções deve ser usado o comando exportar (fica disponível para local e global)

Comando para exportar variável:

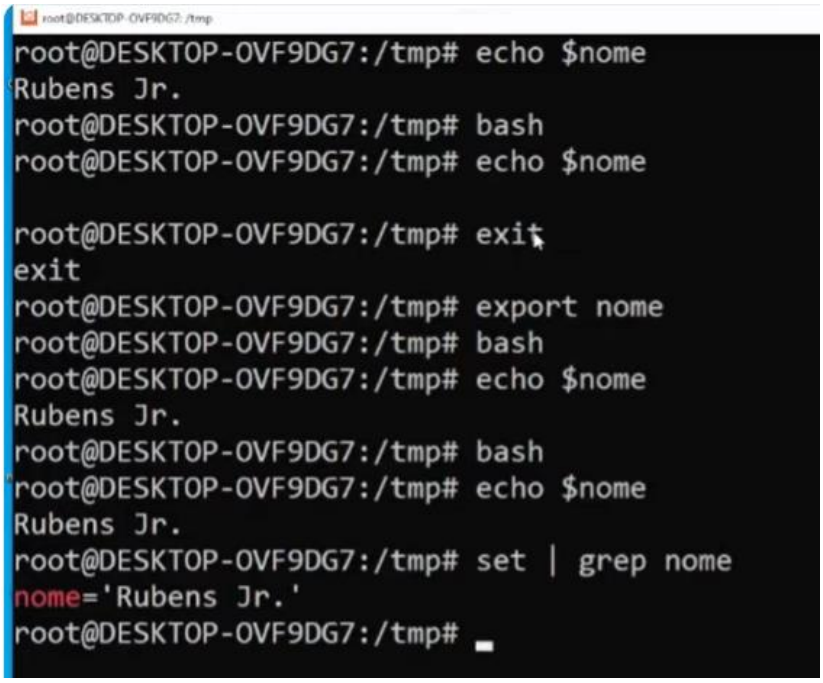
export nome

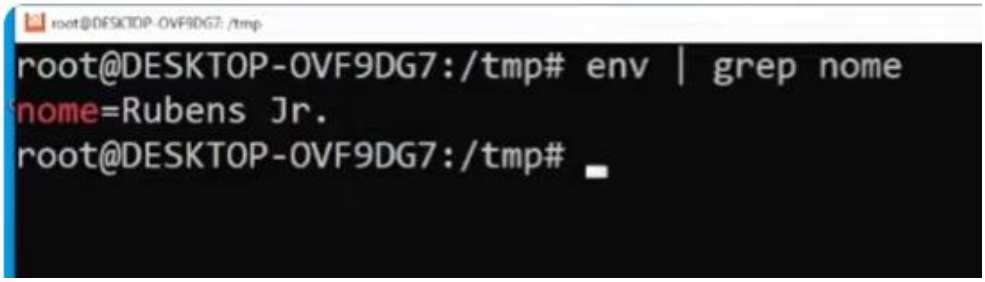


Variável disponível em todas as seções:



Variável no escopo local e global:

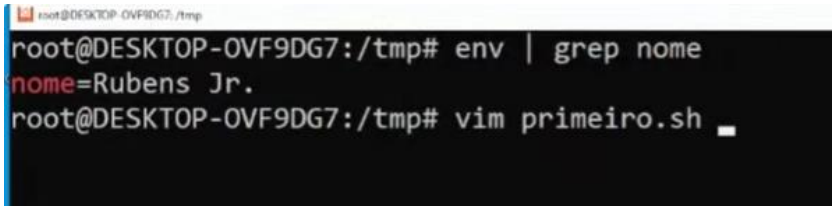




Abrir o script e colocar os comandos para que ele possa executar:

Comando:

vim primeiro.sh

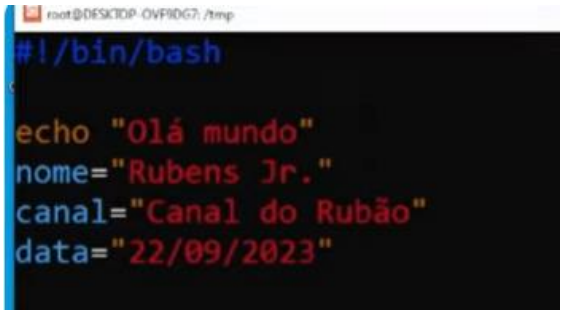


echo “ Hello word”

nome= “Rubens Jr.”

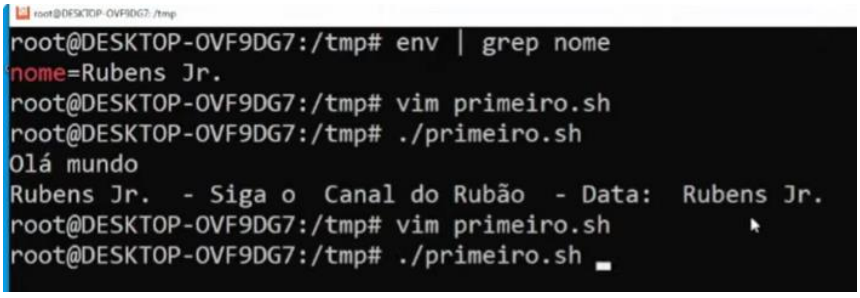
Canal=”Canal do Rubão”

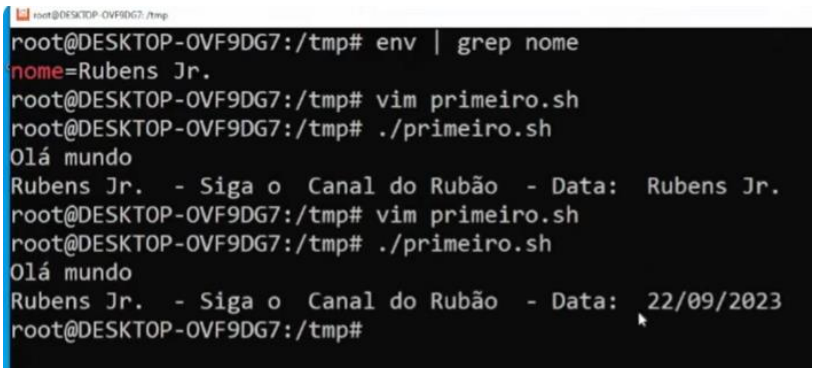
Data=”19/01/2026”



Comando para exibir as variáveis:

echo $nome “- siga o “ $canal “ – Data: “ $data “





Curso Alura:

Para saber mais: manual básico do Vim

Próxima Atividade

-
-

O Vim é um editor de texto poderoso e amplamente utilizado no ambiente Linux. Deixamos esse guia que cobre os comandos essenciais para começar a usar o Vim:

1. **Iniciando o Vim**

Para abrir um arquivo:

vim nome_do_arquivo

Copiar código

2. **Modos do Vim**

O Vim possui diferentes modos, sendo os principais:

- **Modo Normal:** Usado para navegação e comandos (o modo padrão ao abrir o Vim).
- **Modo de Inserção:** Usado para inserir texto.
- **Modo de Comando:** Usado para executar comandos do Vim.

3. **Mudando de modos**

- Para entrar no **modo de inserção**: pressione i (inserir antes do cursor) ou a (inserir após o cursor).
- Para voltar ao **modo normal**: pressione Esc.

4. **Navegação**

É possível usar as teclas de seta para navegar pelo arquivo no Vim, sendo:

- **Seta para cima:** move o cursor uma linha para cima.
- **Seta para baixo:** move o cursor uma linha para baixo.
- **Seta para esquerda:** move o cursor um caractere para a esquerda.
- **Seta para direita:** move o cursor um caractere para a direita.

Embora usar as teclas de seta seja conveniente, muitos usuários preferem os comandos de navegação do Vim, devido a eficiência que proporcionam, especialmente quando você está digitando e não quer tirar as mãos do teclado.

Assim, as seguintes teclas podem ser utilizadas para a navegação no Vim:

- **h:** move o cursor para a esquerda.
- **j:** move o cursor para baixo.
- **k:** move o cursor para cima.
- **l:** move o cursor para a direita.

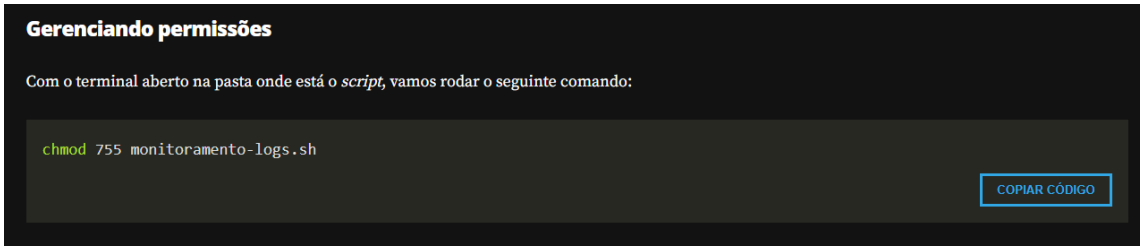
Outros comandos de navegação que podem ser úteis:

- **gg:** ir para o início do arquivo.
- **G:** ir para o final do arquivo.
- **0:** ir para o início da linha.
- **$:** ir para o final da linha.

5. **Salvar e sair**

- **Para salvar as alterações:** no modo normal, digite :w e pressione Enter.
- **Para sair do Vim:**
    - :q: sai se não houver alterações.
    - :q!: sai sem salvar as alterações.
    - :wq: salva e sai.

Tipos de Permissão;



Gerenciando permissões

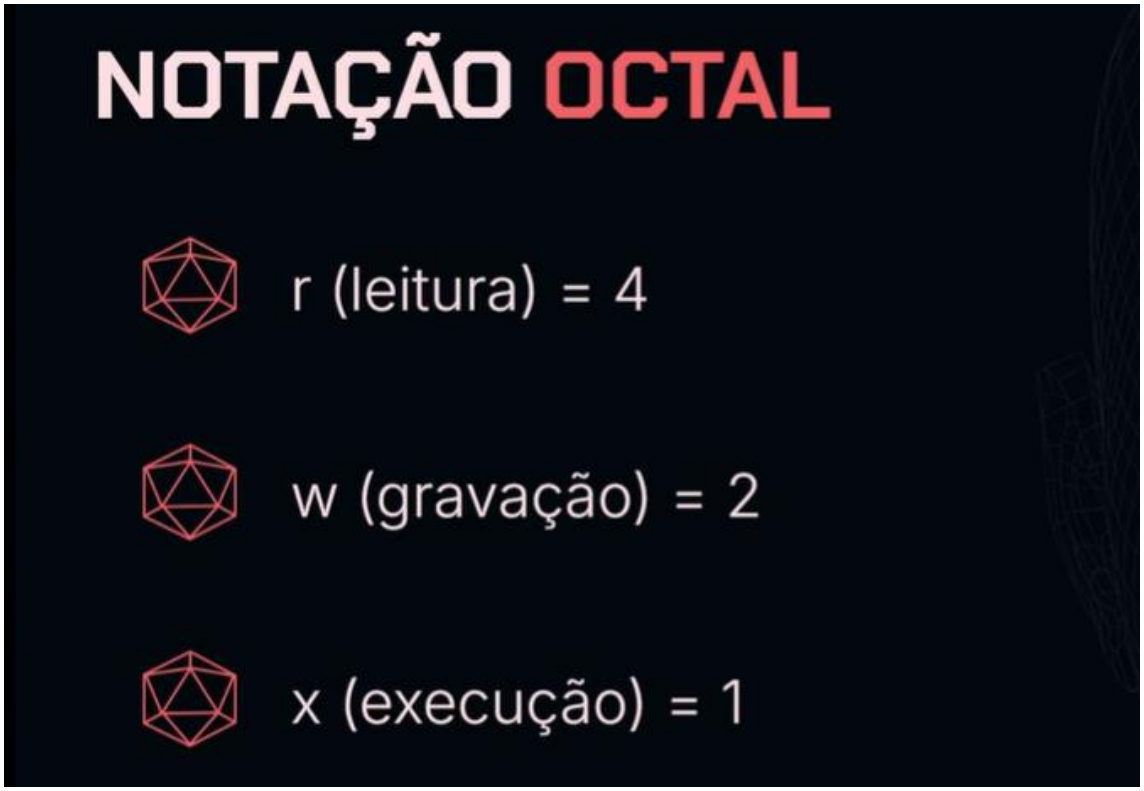
Com o terminal aberto na pasta onde está o *script*, vamos rodar o seguinte **comando:**

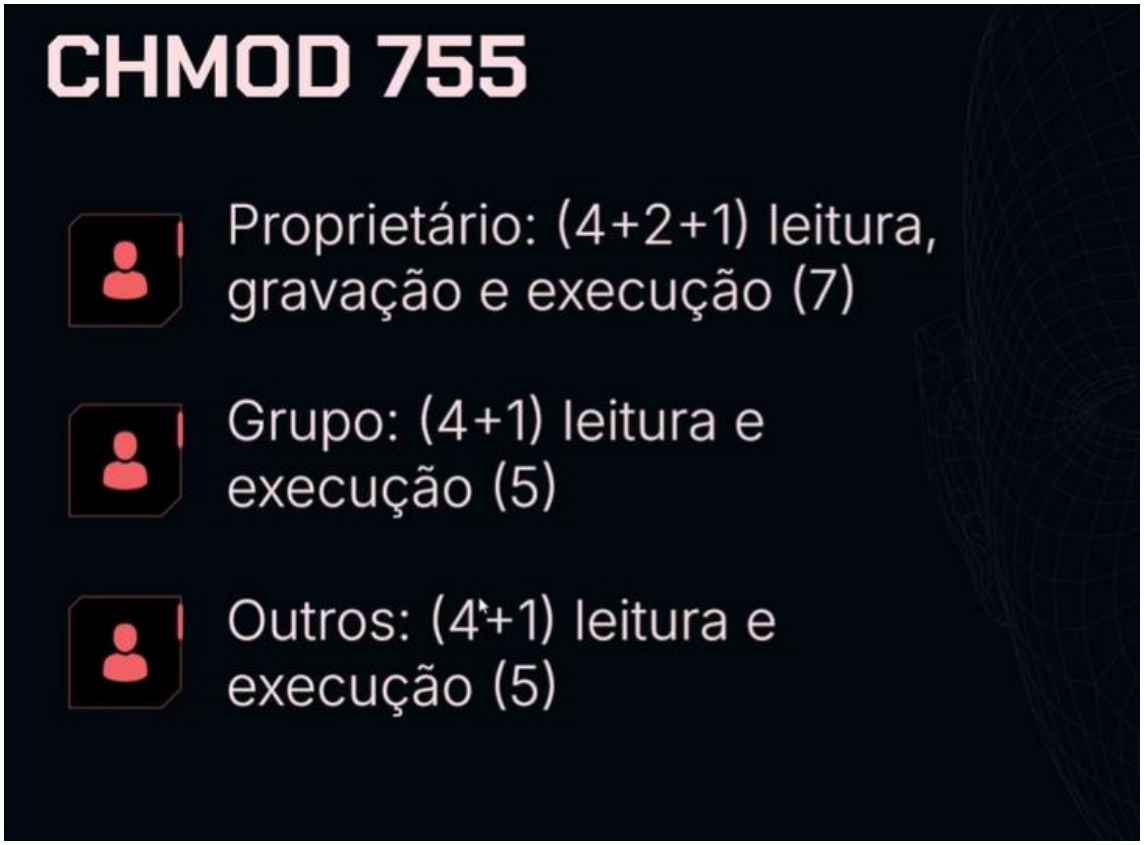
chmod 755 monitoramento-logs.sh

Para verificar as permissões de um arquivo, usamos o comando ls -l, que lista os arquivos e suas permissões: **ls – l (“l” de laranja)**

ls -l

Notação octal:





Cada número em octal (0-7) é calculado somando os valores das permissões:

Permissão	Símbolo	Valor Octal
Nenhuma	---	0
Executar	--x	1
Escrever	-w-	2
Escrever + Executar	-wx	3
Ler	r--	4
Ler + Executar	r-x	5
Ler + Escrever	rw-	6
Ler + Escrever + Executar	rwx	7

Notação Simbólica

CHMOD 755 no formato simbólico?

Executando o *script*

Agora que entendemos a gestão de permissões, podemos executar nosso *script*. Basta executar o seguinte comando:

./monitoramento-logs.sh

Transcrição

Já criamos nosso *script* com a estrutura inicial e agora queremos executá-lo para verificar se está tudo funcionando conforme o esperado. Porém, no Linux, sempre que desejamos executar um *script* Bash pela primeira vez, precisamos conceder **permissão** de execução para ele.

Gerenciando permissões

Com o terminal aberto na pasta onde está o *script*, vamos rodar o seguinte comando:

chmod 755 monitoramento-logs.sh

Copiar código

Após pressionar "Enter", já teremos aplicado a permissão de execução.

O comando chmod no Linux é utilizado para gerenciar permissões, adicionando ou removendo permissões em arquivos e diretórios. Mas o que significam os números que utilizamos? Como sabemos quais números usar para adicionar e remover essas

permissões? Vamos entender em detalhes como funciona essa gestão de permissões no Linux.

Tipos de permissões

É importante compreender esses detalhes de gerenciamento de permissões, pois são fundamentais para a segurança e administração do sistema. Existem diferentes tipos de permissões que podemos conceder ou retirar de arquivos e diretórios.

A primeira é a **permissão de leitura**, representada pela letra **r** (*read*, em inglês), que permite acessar o conteúdo de um arquivo ou listar arquivos de um diretório.

A segunda é a **permissão de escrita**, representada pela letra **w** (de *write*, em inglês), que permite modificar arquivos ou o conteúdo de um diretório.

Por fim, temos a **permissão de execução**, representada pela letra **x** (de *execute*, em inglês), que permite executar códigos, *scripts* e acessar diretórios.

Classes de usuário

Além das permissões, existem **classes de usuários** no Linux.

A primeira classe é o **proprietário**, representado pela letra **u**, que é o dono do arquivo. No nosso *script*, somos os proprietários.

Em seguida, temos o **grupo**, representado pela letra **g**, que é um grupo de usuários que não são os donos do arquivo, mas podem ter acesso a ele. Por exemplo, em uma instituição, é interessante que o time de desenvolvimento tenha acesso ao *script*. Então, podemos colocar essas pessoas em um grupo e conceder acesso ao grupo.

Por fim, temos **outros**, representados pela letra **o**, que são todos os usuários que não são nem o proprietário, nem parte de um grupo. Trata-se de pessoas de outros times que não estão trabalhando diretamente no projeto de processamento de *logs*.

Existe também a representação de **todos os usuários**, indicada pela letra **a**, que engloba proprietário, grupo e outros.

Notação Octal

Quando escrevemos o comando chmod 755, utilizamos uma **notação octal** para representar as permissões. Essa notação usa números para indicar o tipo de permissão que estamos aplicando ou removendo.

Cada permissão tem um valor diferente:

- leitura (r) tem o valor 4
- gravação (w) tem o valor 2
- execução (x) tem o valor 1

E o que significa o valor 755? Vamos entender por partes, a seguir.

Cada um desses três algarismos representa uma classe de usuários. O primeiro indica as permissões para a classe de usuário proprietário. O segundo para a classe grupo. O terceiro para outros.

Para chegar ao 7, somamos os valores das permissões. O proprietário tem permissão de leitura (4), gravação (2) e execução (1). Logo, 4 + 2 + 1 será 7.

Para o grupo e outros, que têm apenas permissão de leitura e execução, somamos apenas 4 + 1, resultando em 5. Assim, chegamos ao número 755, do chmod 755.

Notação simbólica

Além da notação octal, podemos usar a notação simbólica no chmod, que utiliza letras e símbolos para representar operações e usuários:

- o operador "+" adiciona permissões,
- o operador "-" remove permissões
- o operador "=" define permissões exatas

Por exemplo: chmod u+x arquivo.sh. A letra **u** indica o usuário proprietário. O símbolo de **+** adiciona uma permissão. A letra **x** representa a permissão de execução. Em outras palavras, estamos atribuindo a permissão de execução para o proprietário do arquivo arquivo.sh.

Outro exemplo: chmod g-w arquivo.sh. A letra **g** indica o grupo. O operador **-** remove uma permissão. O **w** representa a permissão de gravação. Ou seja, removemos a permissão de gravação do grupo.

Como seria o chmod 755 na notação simbólica? Poderíamos usar o chmod +x Como estamos atribuindo a permissão de execução para todas as classes, podemos emitir a primeira informação, apenas colocando o operador de **+** e o **x**, de execução.

Para escolher entre notação octal e simbólica, é interessante considerar alguns aspectos. Por exemplo, a facilidade de uso. A notação octal é mais compacta e simples, enquanto a simbólica é mais específica.

Outro aspecto a se considerar, é o escopo de modificação. Na notação octal, atribuímos permissões para todas as classes, enquanto a notação simbólica permite mais especificidade ao adicionar ou remover permissões.

Há uma atividade "Para saber mais" sobre as notações para você explorar mais o assunto.

Verificando permissões

Para limpar a tela do terminal, usa-se "Ctrl + L".

Para verificar as permissões de um arquivo, usamos o comando ls -l, que lista os arquivos e suas permissões:

ls -l

Copiar código

No início da linha, caracteres indicam o tipo de arquivo e suas permissões. No nosso caso, temos:

-rwxr-xr-x

O primeiro traço indica um arquivo regular. Caso fosse um diretório, seria a letra "d".

Os três primeiros dígitos representam as permissões do proprietário: **r** de leitura, **w** de escrita e **x** de execução.

Os três seguintes indicam permissões do grupo: **r** de leitura, um traço no lugar do **w** porque não há permissão de escrita, e o **x** de execução. Os últimos são as permissões de outros usuários, que são iguais ao anterior.

Em seguida, temos o nome do proprietário e o grupo atribuído ao arquivo.

Executando o *script*

Agora que entendemos a gestão de permissões, podemos executar nosso *script*. Basta executar o seguinte comando:

./monitoramento-logs.sh

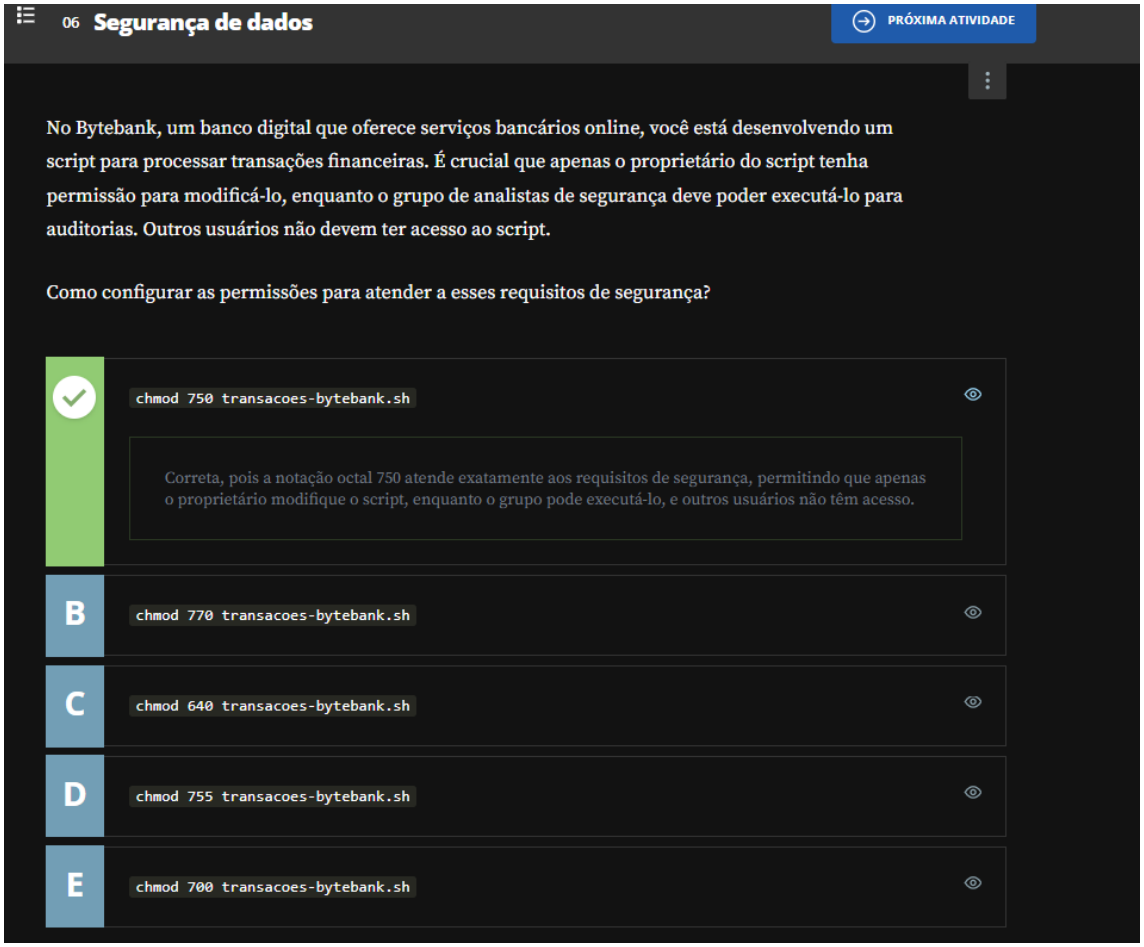
Copiar código

Dessa forma, veremos na tela a seguinte mensagem:

Verificando logs no diretorio ../myapps/logs

A saída na tela, gerada pelo comando echo, confirma que o *script* está verificando logs no diretório especificado.

Aprendemos a gerenciar permissões e executar scripts. Na sequência, vamos entender como gerenciar grupos.



Para saber mais: notação octal para permissões

Próxima Atividade

-
-

A notação octal utiliza três dígitos, cada um variando de 0 a 7, onde cada dígito representa um conjunto de permissões. A estrutura é a seguinte:

Permissões: [Usuário] [Grupo] [Outros]

Exemplo: **chmod** 755 **arquivo**.txt

Copiar código

Cada número em octal (0-7) é calculado somando os valores das permissões:

Permissão	Símbolo	Valor Octal
Nenhuma	---	0
Executar	--x	1

Permissão	Símbolo	Valor Octal
Escrever	-w-	2
Escrever + Executar	-wx	3
Ler	r--	4
Ler + Executar	r-x	5
Ler + Escrever	rw-	6
Ler + Escrever + Executar	rwx	7

Dessa forma, o número final representa as permissões para usuário, grupo e outros na ordem em que são escritos.

Exemplos de Uso

Se rodarmos o comando:

chmod 644 arquivo.txt

Copiar código

Teremos:

- 6 (Usuário): Leitura (4) + Escrita (2) = 6 (rw-).
- 4 (Grupo): Leitura = 4 (r--).
- 4 (Outros): Leitura = 4 (r--).

Neste caso, o proprietário pode ler e modificar o arquivo, enquanto o grupo e outros podem apenas lê-lo.

Se rodarmos o comando:

chmod 700 arquivo.txt

Copiar código

Teremos:

- 7 (Usuário): Leitura (4) + Escrita (2) + Execução (1) = 7 (rwx).
- 0 (Grupo): Sem permissão.
- 0 (Outros): Sem permissão.

Aqui, somente o proprietário tem todas as permissões, enquanto o grupo e outros não têm nenhuma.

Desse modo, a notação octal é mais compacta e rápida para definir um conjunto de permissões, alterando todas as permissões de uma vez.

Para saber mais: notação simbólica para permissões

Próxima Atividade

-
-

A notação simbólica usa letras e sinais (+, -, =) para definir permissões e grupos específicos. Nessa notação, você pode controlar quem tem permissões e quais permissões são aplicadas usando comandos como chmod.

Componentes da Notação Simbólica

1. **Categorias de Usuários**
    - u: refere-se ao usuário proprietário do arquivo ou diretório.
    - g: refere-se ao grupo ao qual o arquivo ou diretório pertence.
    - o: refere-se a outros, ou seja, todos os outros usuários do sistema.
    - a: refere-se a todos (equivalente a ugo), aplicando-se ao usuário, ao grupo e a outros.
2. **Tipos de Permissão**
    - r: Leitura – permite ver o conteúdo do arquivo ou listar os arquivos em um diretório.
    - w: Escrita – permite modificar o conteúdo de um arquivo ou alterar o conteúdo de um diretório (como criar, renomear ou remover arquivos).
    - x: Execução – permite executar um arquivo (caso seja um script ou executável) ou acessar um diretório.
3. **Operadores**
    - +: adiciona uma permissão.
    - -: remove uma permissão.
    - =: define a permissão exatamente como especificada, removendo qualquer permissão que não esteja incluída.

Exemplos de Uso

Vamos analisar alguns exemplos práticos para entender melhor a notação simbólica em comandos:

Se executarmos o comando:

chmod u+r arquivo.txt

Copiar código

Ele adiciona permissão de leitura apenas para o proprietário (u). Caso o usuário já tenha permissões adicionais, elas permanecem inalteradas.

Se executarmos o comando:

chmod g-w arquivo.txt

Copiar código

Ele remove a permissão de escrita do grupo (g). Isso é útil quando se deseja evitar que o grupo modifique o conteúdo de um arquivo.

Ao executar:

chmod o=x arquivo.txt

Copiar código

Definimos a permissão de execução apenas para outros (o) e removemos qualquer outra permissão que o proprietário e o grupo possam ter.

Já o comando:

chmod a+r arquivo.txt

Copiar código

Adiciona permissão de leitura para todos os tipos de usuários (a), ou seja, proprietário, grupo e outros. Isso é útil para tornar um arquivo acessível a todos, mantendo as permissões existentes.

Caso desejarmos, é possível combinar os tipos de usuários com diferentes permissões com essa notação, utilizando como separador a vírgula:

chmod u=rwx,g=rx,o=r arquivo.txt

Copiar código

Aqui, definimos permissões específicas para cada categoria:

- Usuário (u): tem leitura, escrita e execução (rwx).
- Grupo (g): tem leitura e execução (rx).
- Outros (o): têm apenas leitura (r).

Desse modo, a notação simbólica tem um formato mais explícito e pode ser mais intuitiva para alterações específicas, permitindo a mudança de permissões de forma mais granular.

Transcrição

Nós aprendemos sobre a gestão de permissões no Linux e como executar um *script*. Agora, vamos estudar sobre **gestão de grupos**.

Gerenciamento de grupos

Quando trabalhamos em um projeto de software, é comum haver todo um time envolvido. Nossa aplicação será normalmente executada em um servidor, em uma máquina onde todos terão acesso. Queremos que as pessoas possam acessar nosso *script* de processamento de *logs*.

No Linux, a questão das classes de usuários e dos grupos nos ajuda com isso, pois podemos criar um grupo e adicionar diversos usuários para facilitar a gestão de acesso. Assim, em vez de dar permissões individualmente para cada pessoa envolvida no time, podemos utilizar um grupo para facilitar esse processo e atribuir as permissões ao grupo.

Criando usuários

O meu usuário atual é "gabi", como aparece na tela do meu terminal. Ou seja, este é o nome do meu *host*. Podemos criar um novo usuário para o time usando o comando sudo adduser seguido do nome. Por exemplo, julia:

sudo adduser julia

Copiar código

Em seguida, precisamos informar a senha do usuário atual e responder a algumas perguntas para criar o novo usuário. A primeira delas é a senha do novo usuário. Após informá-la, é preciso repeti-la para confirmar.

Na sequência, há várias outras perguntas sobre nome e número da sala de trabalho, telefone e afins, mas podemos pressionar "Enter" para deixar essas informações em branco. No final, o sistema perguntará se as informações estão corretas. Vamos digitar Y e pressionar "Enter" para confirmar.

Agora, a nova usuária chamada julia foi criada. Para verificar se a criação foi bem-sucedida, podemos rodar o seguinte comando para listar os usuários:

cat /etc/passwd

Copiar código

Ao final da lista, temos a nova usuária julia, com seu diretório home.

Criando um grupo

Queremos que a Julia trabalhe conosco e ajude no *script* de processamento de *logs*, sendo capaz de executá-lo. Portanto, criaremos um grupo para adicionar a Julia e outras futuras pessoas colaboradoras do projeto.

Para criar um grupo, rodamos o comando sudo groupadd seguido do nome. Nosso grupo terá pessoas que desenvolverão e executarão o *script*, então o chamaremos de devs:

sudo addgroup devs

Copiar código

Para verificamos se o grupo foi criado com sucesso, rodamos o seguinte comando:

getent group devs

Copiar código

O sistema retornará que o grupo existe. Agora, adicionaremos a Julia ao grupo com o seguinte comando:

sudo usermod -aG devs julia

Copiar código

Para checar se ela foi adicionada com sucesso, basta executar novamente o comando getent group devs, que mostrará a usuária julia como parte do grupo.

Gerenciando acessos

O próximo passo é conceder acesso para que as pessoas do grupo, para que possam executar o *script*. Vamos entrar na pasta onde o *script* foi criado com cd scripts-linux e rodar o comando ls -ld para conferir as permissões:

drwxr-xr-x gabi gabi

Notamos que o grupo tem permissão de leitura e execução, mas não de escrita. Quando atribuímos essas permissões, é necessário que todos os diretórios-pais também tenham as permissões necessárias.

No momento, o grupo ainda não é devs, mas gabi. Se formos para o diretório-pai com cd .. e rodarmos ls -ld, temos as seguintes permissões:

drwxr-x--- gabi gabi

O grupo também tem permissão de leitura e execução, mas não de escrita. Além disso, o grupo é gabi. O grupo do diretório-pai deve ser o mesmo do diretório-filho que queremos acessar.

Se adicionarmos um grupo apenas na pasta "scripts-linux" e não alterarmos o grupo do diretório-raiz, não conseguiremos acessar a pasta. Portanto, adicionamos o grupo no diretório-pai para evitar problemas de permissão.

Para conferir o caminho da pasta atual, usa-se o comando pwd.

Para alterar o grupo na pasta atual ("/home/gabi"), utilizamos o seguinte comando:

sudo chown -R :devs /home/gabi

Copiar código

Essa opção -R aplica a modificação recursivamente a todos os subdiretórios. Ou seja, todas as pastas dentro do "/home/gabi" terão seu grupo alterado. Esse processo pode demorar um pouco, dependendo da quantidade de diretórios no seu computador.

Após a execução, vamos rodar o comando ls -ld e verificar que o grupo foi alterado para devs:

drwxr-x--- gabi devs

Na sequência, verificaremos se a usuária julia consegue executar o *script*. Vamos alterar o usuário que estamos usando com o seguinte comando:

su - julia

Copiar código

Após informar a senha, notaremos que o nome do *host* mudou. Vamos acessar a pasta dos *scripts*, executando o seguinte comando:

cd /home/gabi/scripts-linux

Copiar código

Lembre-se de remover "gabi" e alterar o comando para condizer com o seu nome de usuário.

Conseguimos acessar a pasta. Agora, vamos rodamos o *script* com o seguinte comando:

./monitoramento_logs.sh

Copiar código

Confirmamos que a usuária julia consegue executar o *script*!

Ao gerenciar uma máquina ou servidor, é importante verificar se queremos que todos os usuários do grupo tenham acesso a todas as pastas dentro do diretório. É possível conceder acesso a pastas específicas, sem a opção -R.

Para voltar ao usuário gabi, usamos su - gabi e informamos a senha novamente. Assim, retornamos ao usuário original.

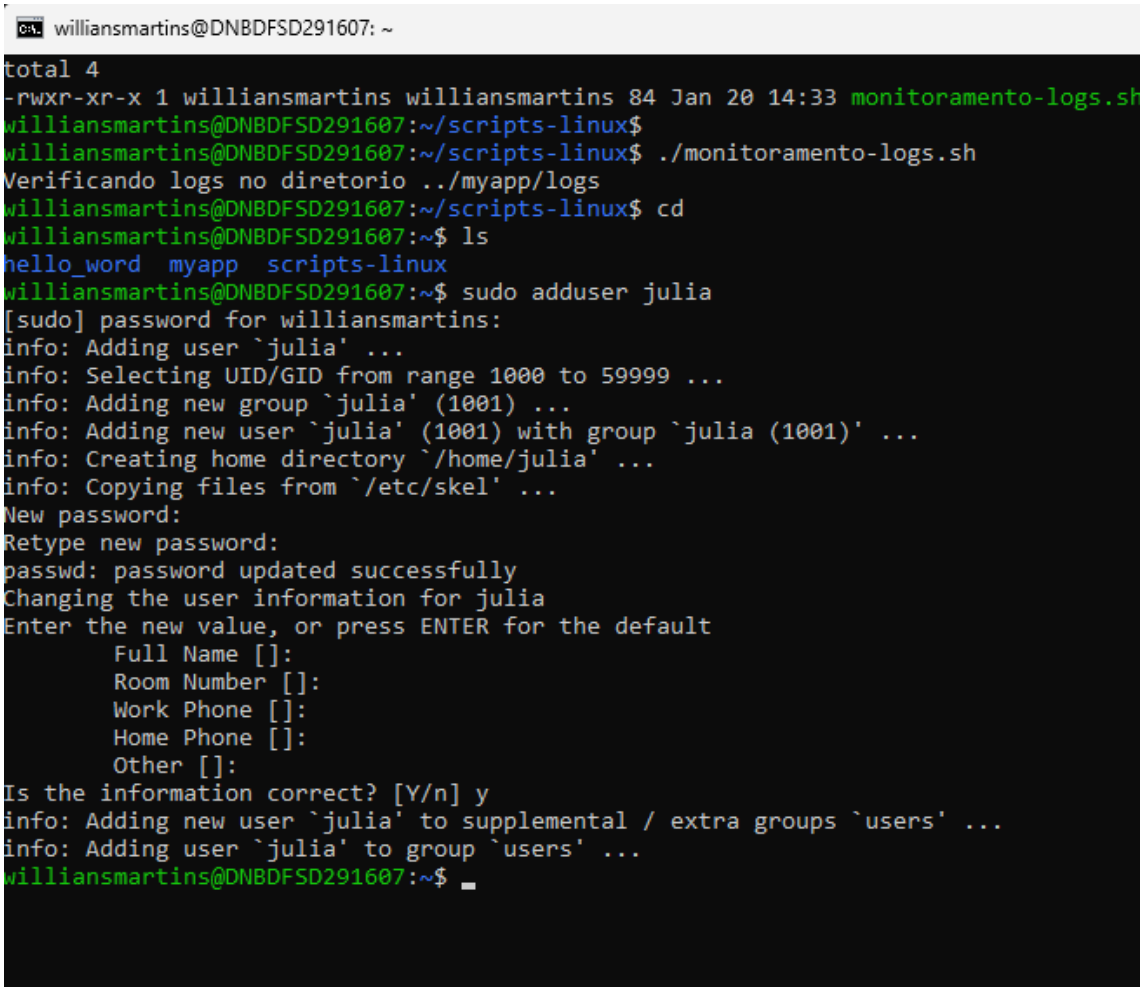
Descobrimos que o comando chown é útil na gestão de grupos e usuários no sistema. Aprendemos a gerenciar permissões, grupos e a criar um *script*. Na sequência, vamos incrementar nosso *script* de processamento de logs.

Criando usuários

O meu usuário atual é "gabi", como aparece na tela do meu terminal. Ou seja, este é o nome do meu *host*. Podemos criar um novo usuário para o time usando o comando sudo adduser seguido do nome. Por exemplo, julia:

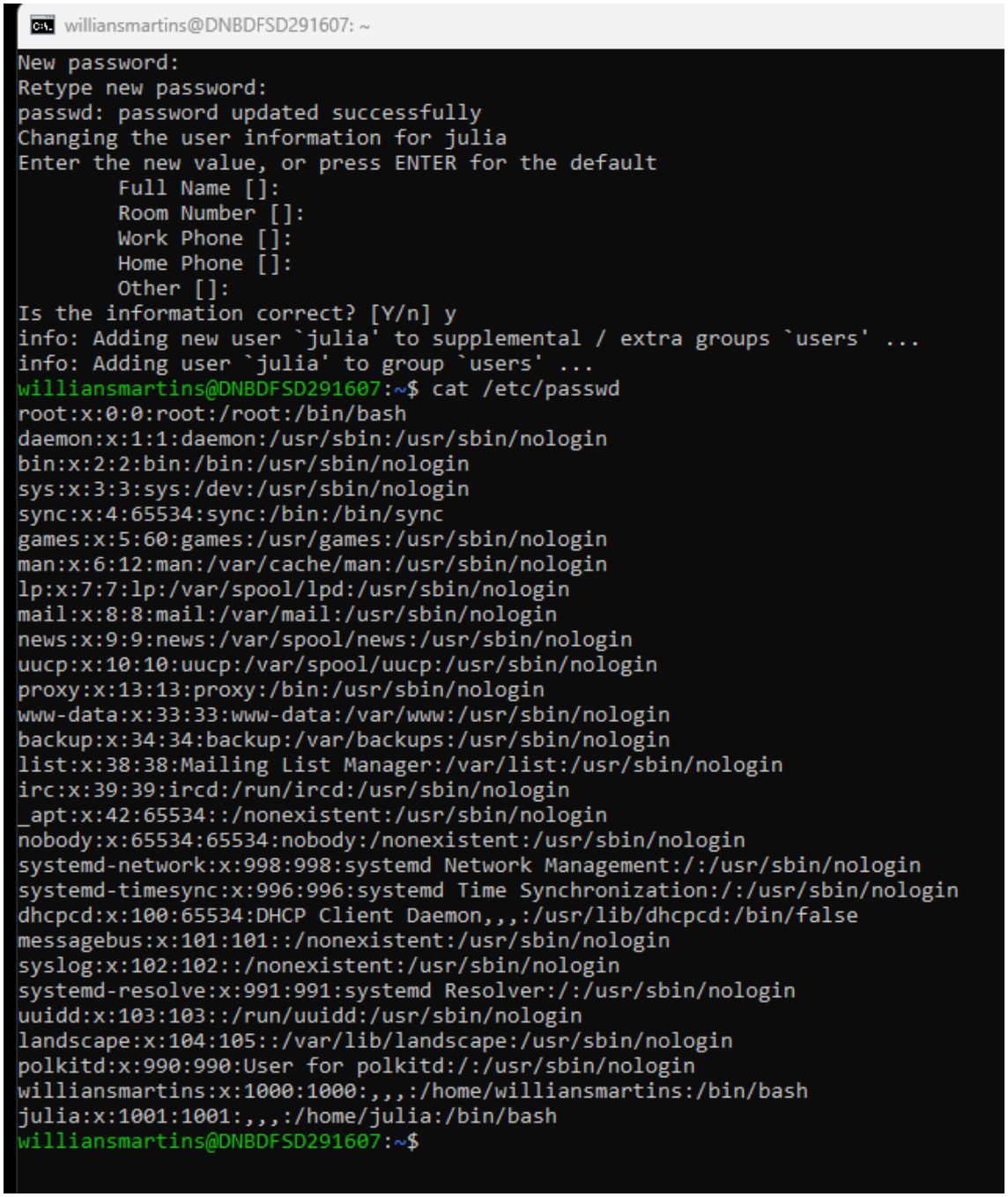
sudo adduser julia

Criação do novo usuário Julia com o password 1234



Agora, a nova usuária chamada julia foi criada. Para verificar se a criação foi bem-sucedida, podemos rodar o seguinte comando para listar os usuários:

cat /etc/passwd



Criando um grupo

Queremos que a Julia trabalhe conosco e ajude no *script* de processamento de *logs*, sendo capaz de executá-lo. Portanto, criaremos um grupo para adicionar a Julia e outras futuras pessoas colaboradoras do projeto.

Para criar um grupo, rodamos o comando sudo groupadd seguido do nome. Nosso grupo terá pessoas que desenvolverão e executarão o *script*, então o chamaremos de devs:

sudo addgroup devs

Copiar código

Para verificamos se o grupo foi criado com sucesso, rodamos o seguinte comando:

getent group devs

Copiar código

O sistema retornará que o grupo existe. Agora, adicionaremos a Julia ao grupo com o seguinte comando:

sudo usermod -aG devs julia

Copiar código

Para checar se ela foi adicionada com sucesso, basta executar novamente o comando getent group devs, que mostrará a usuária julia como parte do grupo.

Comando:

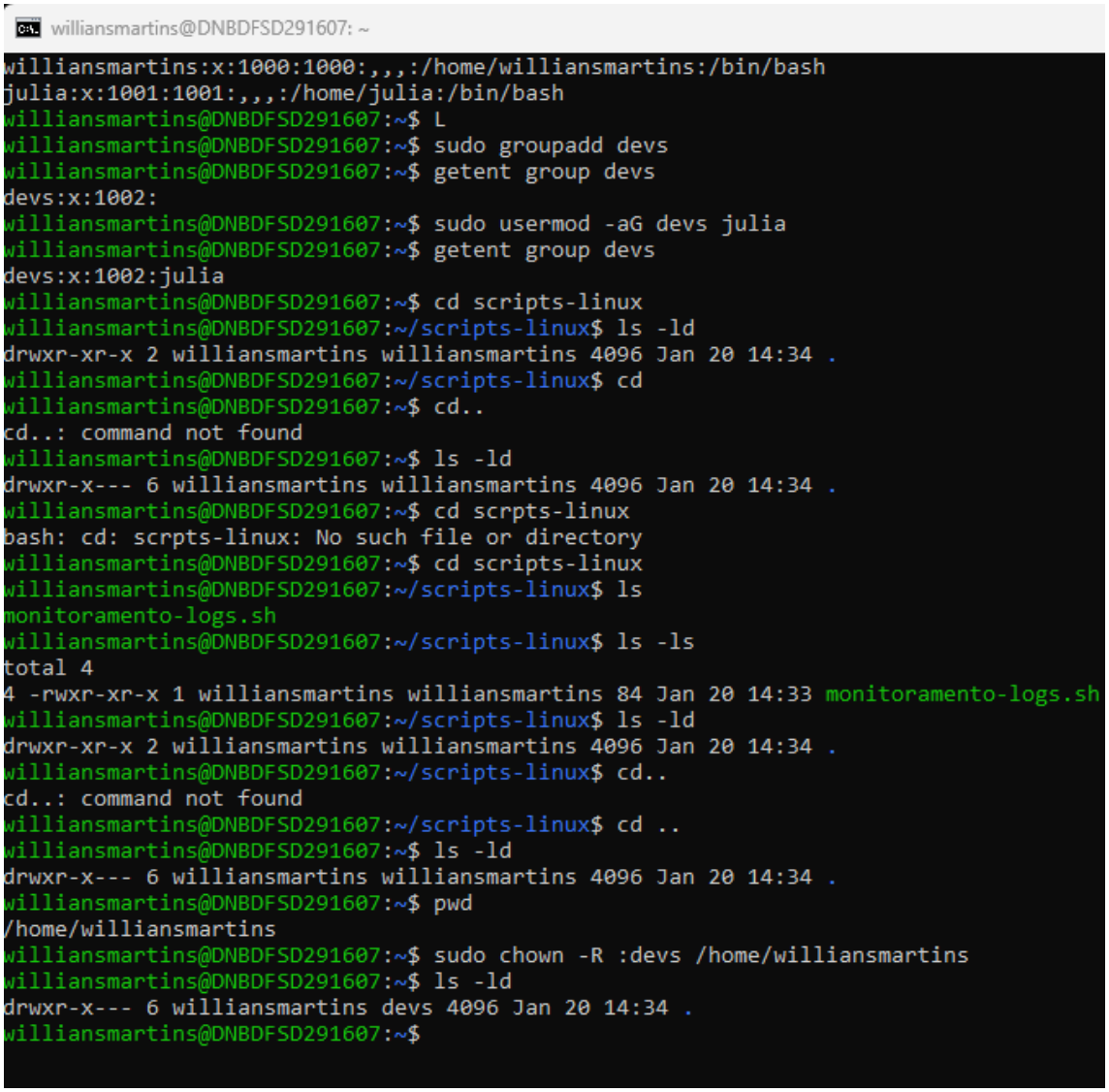
Cd .. para ir para diretório pai.

Para alterar o grupo dentro da pasta :

usar o comando:

Sudo chown -R /home/williansmartins

Usar comando ls -ld para ver permissões

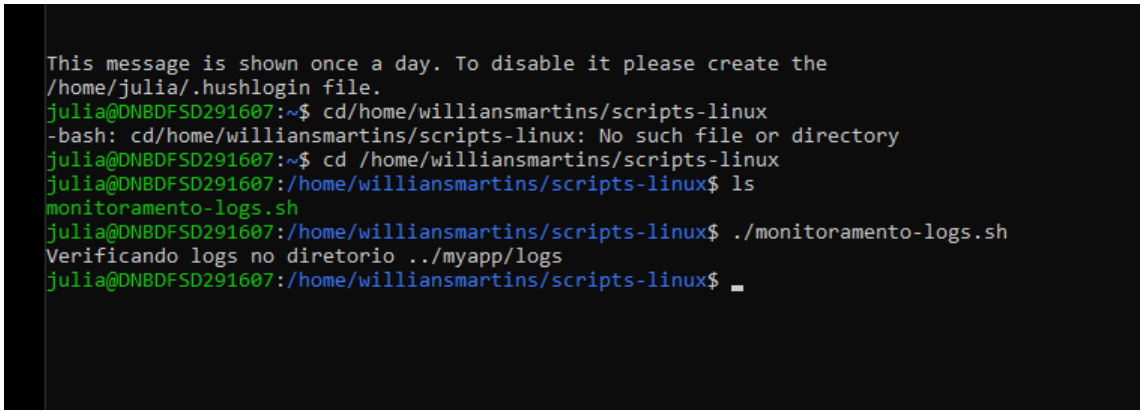


Nota-se que o grupo foi alterado

Comando para alterar o usuário utilizado;

Su – julia

Testando acesso no novo usuario



Para voltar para o usuário basta usar o comando:

su -Williansmartins (usuário

Transcrição

Nós aprendemos sobre a gestão de permissões no Linux e como executar um *script*. Agora, vamos estudar sobre **gestão de grupos**.

Gerenciamento de grupos

Quando trabalhamos em um projeto de software, é comum haver todo um time envolvido. Nossa aplicação será normalmente executada em um servidor, em uma máquina onde todos terão acesso. Queremos que as pessoas possam acessar nosso *script* de processamento de *logs*.

No Linux, a questão das classes de usuários e dos grupos nos ajuda com isso, pois podemos criar um grupo e adicionar diversos usuários para facilitar a gestão de acesso. Assim, em vez de dar permissões individualmente para cada pessoa envolvida no time, podemos utilizar um grupo para facilitar esse processo e atribuir as permissões ao grupo.

Criando usuários

O meu usuário atual é "gabi", como aparece na tela do meu terminal. Ou seja, este é o nome do meu *host*. Podemos criar um novo usuário para o time usando o comando sudo adduser seguido do nome. Por exemplo, julia:

sudo adduser julia

Copiar código

Em seguida, precisamos informar a senha do usuário atual e responder a algumas perguntas para criar o novo usuário. A primeira delas é a senha do novo usuário. Após informá-la, é preciso repeti-la para confirmar.

Na sequência, há várias outras perguntas sobre nome e número da sala de trabalho, telefone e afins, mas podemos pressionar "Enter" para deixar essas informações em branco. No final, o sistema perguntará se as informações estão corretas. Vamos digitar Y e pressionar "Enter" para confirmar.

Agora, a nova usuária chamada julia foi criada. Para verificar se a criação foi bem-sucedida, podemos rodar o seguinte comando para listar os usuários:

cat /etc/passwd

Copiar código

Ao final da lista, temos a nova usuária julia, com seu diretório home.

Criando um grupo

Queremos que a Julia trabalhe conosco e ajude no *script* de processamento de *logs*, sendo capaz de executá-lo. Portanto, criaremos um grupo para adicionar a Julia e outras futuras pessoas colaboradoras do projeto.

Para criar um grupo, rodamos o comando sudo groupadd seguido do nome. Nosso grupo terá pessoas que desenvolverão e executarão o *script*, então o chamaremos de devs:

sudo addgroup devs

Copiar código

Para verificamos se o grupo foi criado com sucesso, rodamos o seguinte comando:

getent group devs

Copiar código

O sistema retornará que o grupo existe. Agora, adicionaremos a Julia ao grupo com o seguinte comando:

sudo usermod -aG devs julia

Copiar código

Para checar se ela foi adicionada com sucesso, basta executar novamente o comando getent group devs, que mostrará a usuária julia como parte do grupo.

Gerenciando acessos

O próximo passo é conceder acesso para que as pessoas do grupo, para que possam executar o *script*. Vamos entrar na pasta onde o *script* foi criado com cd scripts-linux e rodar o comando ls -ld para conferir as permissões:

drwxr-xr-x gabi gabi

Notamos que o grupo tem permissão de leitura e execução, mas não de escrita. Quando atribuímos essas permissões, é necessário que todos os diretórios-pais também tenham as permissões necessárias.

No momento, o grupo ainda não é devs, mas gabi. Se formos para o diretório-pai com cd .. e rodarmos ls -ld, temos as seguintes permissões:

drwxr-x--- gabi gabi

O grupo também tem permissão de leitura e execução, mas não de escrita. Além disso, o grupo é gabi. O grupo do diretório-pai deve ser o mesmo do diretório-filho que queremos acessar.

Se adicionarmos um grupo apenas na pasta "scripts-linux" e não alterarmos o grupo do diretório-raiz, não conseguiremos acessar a pasta. Portanto, adicionamos o grupo no diretório-pai para evitar problemas de permissão.

Para conferir o caminho da pasta atual, usa-se o comando pwd.

Para alterar o grupo na pasta atual ("/home/gabi"), utilizamos o seguinte comando:

sudo chown -R :devs /home/gabi

Copiar código

Essa opção -R aplica a modificação recursivamente a todos os subdiretórios. Ou seja, todas as pastas dentro do "/home/gabi" terão seu grupo alterado. Esse processo pode demorar um pouco, dependendo da quantidade de diretórios no seu computador.

Após a execução, vamos rodar o comando ls -ld e verificar que o grupo foi alterado para devs:

drwxr-x--- gabi devs

Na sequência, verificaremos se a usuária julia consegue executar o *script*. Vamos alterar o usuário que estamos usando com o seguinte comando:

su - julia

Copiar código

Após informar a senha, notaremos que o nome do *host* mudou. Vamos acessar a pasta dos *scripts*, executando o seguinte comando:

cd /home/gabi/scripts-linux

Copiar código

Lembre-se de remover "gabi" e alterar o comando para condizer com o seu nome de usuário.

Conseguimos acessar a pasta. Agora, vamos rodamos o *script* com o seguinte comando:

./monitoramento_logs.sh

Copiar código

Confirmamos que a usuária julia consegue executar o *script*!

Ao gerenciar uma máquina ou servidor, é importante verificar se queremos que todos os usuários do grupo tenham acesso a todas as pastas dentro do diretório. É possível conceder acesso a pastas específicas, sem a opção -R.

Para voltar ao usuário gabi, usamos su - gabi e informamos a senha novamente. Assim, retornamos ao usuário original.

Descobrimos que o comando chown é útil na gestão de grupos e usuários no sistema. Aprendemos a gerenciar permissões, grupos e a criar um *script*. Na sequência, vamos incrementar nosso *script* de processamento de logs.

Para saber mais: mergulhando no comando chown

Próxima Atividade

-
-

O comando chown no Linux é usado para controlar quais usuários e grupos podem acessar e modificar arquivos específicos. Ele é fundamental em sistemas multiusuário, onde a segurança e a organização de permissões são importantes.

Sua sintaxe básica é:

chown [opções] novo_dono:novo_grupo arquivo

Copiar código

Modificando proprietário e grupo

Com esse comando, você pode **alterar o proprietário e grupo** de um arquivo ou diretório.

Quando você especifica ambos, proprietário e grupo, chown altera os dois de uma só vez. Como é o caso do exemplo a seguir:

chown usuario:grupo arquivo.txt

Copiar código

Se apenas o usuário for especificado o grupo permanece o mesmo. A sintaxe dessa alteração é mostrada a seguir:

chown usuario arquivo.txt

Copiar código

Da mesma forma, o comando abaixo altera apenas o grupo:

chown :grupo arquivo.txt

Copiar código

Recursividade

A opção -R permite que você aplique a alteração de propriedade a todos os arquivos e subdiretórios dentro de uma pasta:

sudo chown -R usuario:grupo /caminho/para/diretorio

Copiar código

Isso é útil ao modificar permissões em diretórios grandes, garantindo que todos os arquivos dentro tenham o mesmo proprietário e grupo.

Opção de referência

Com --reference, você define o proprietário e o grupo de um arquivo com base em outro:

sudo chown --reference=arquivo1 arquivo2

Copiar código

Dessa forma, arquivo2 assume o mesmo proprietário e grupo de arquivo1, o que ajuda a manter permissões consistentes entre arquivos semelhantes.

Opções de segurança e verificação

O chown também possui a opção -c (ou --changes), que exibe uma lista apenas das alterações feitas, o que pode ser útil para verificar que os comandos estão sendo aplicados corretamente.

Privilégios

Apenas o proprietário atual do arquivo ou um usuário com privilégios de root (ou com uso de sudo) pode mudar a propriedade. Isso ajuda a garantir a segurança dos arquivos e evitar acesso indesejado.

Exemplo de caso prático

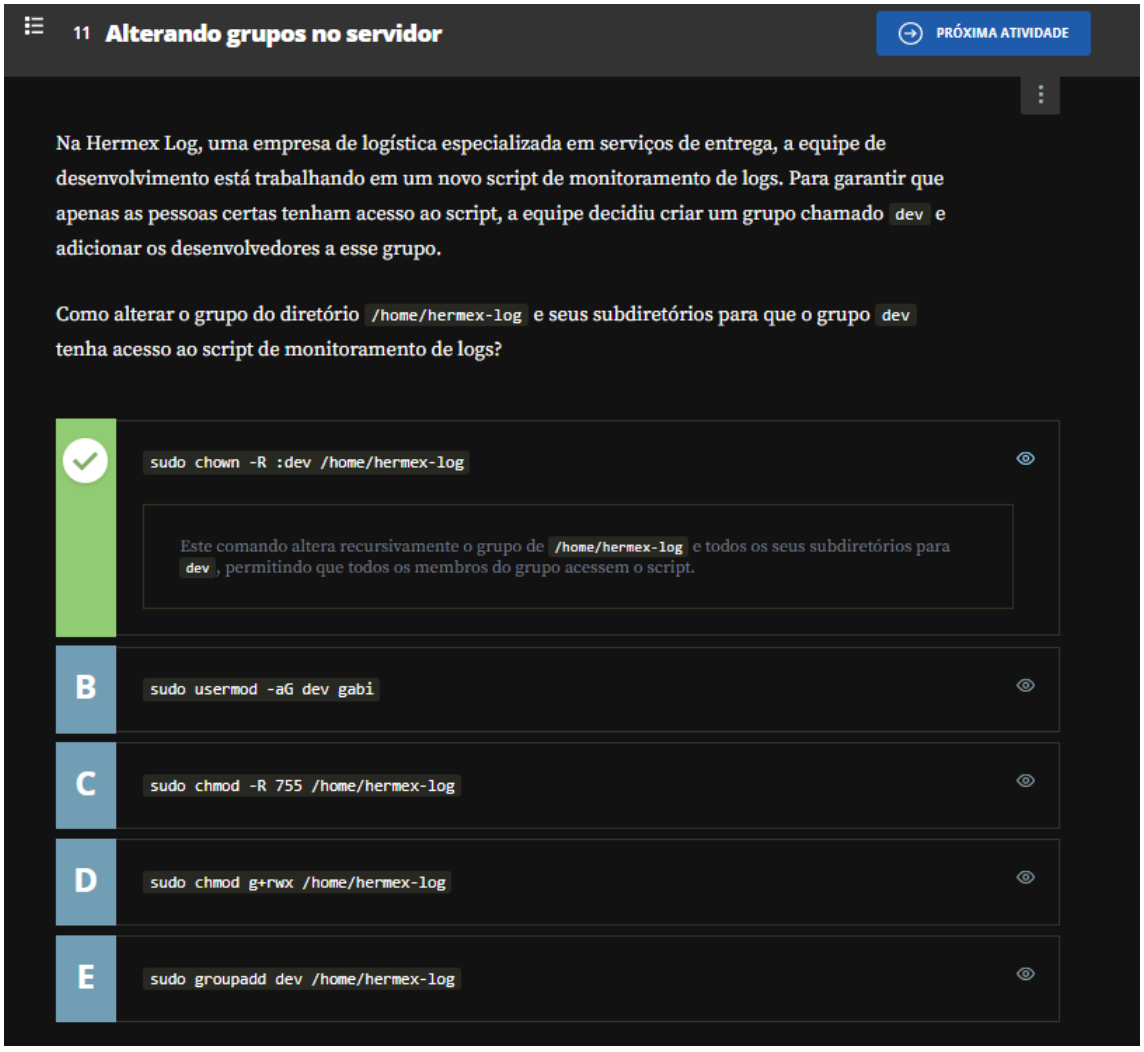
Imagine um servidor onde diferentes usuários precisam acessar uma pasta compartilhada. Ao configurar:

chown -R usuario:grupo /diretorio_compartilhado

Copiar código

Você garante que os arquivos nessa pasta sejam de responsabilidade de um grupo específico e estejam acessíveis apenas aos usuários que pertencem a esse grupo.

Em suma, o chown é essencial para a administração e organização de arquivos, especialmente em ambientes onde o compartilhamento de recursos deve ser gerenciado de forma segura.



Faça como eu fiz: criando um script de monitoramento de logs e gerenciando permissões e grupos do script

Próxima Atividade

-
-

Nesta aula criamos um script no Linux, que salva um diretório em uma variável e depois exibe uma mensagem na tela do terminal, referenciando a variável.

Também vimos como dar permissão de execução para o script e como executá-lo. Por fim, vimos como gerenciar o grupo de arquivos e diretórios.

Agora é com você!

Crie um novo script Linux para monitorar logs, habilite a permissão de execução para ele e execute-o. Para isso, siga os passos:

- Crie uma pasta chamada scripts-linux para salvar o script. Nesta pasta crie um script bash chamado monitoramento-logs.sh com o editor de texto vim;
- No script, crie uma variável que salve o caminho para a pasta onde os logs estão salvos;
- Faça com que o script exiba uma mensagem na tela do terminal com o comando echo, referenciando a variável criada;
- Dê permissão de execução para o script e execute-o;
- Crie um novo usuário chamado julia e um novo grupo chamado devs em seu Linux. Adicione esse usuário ao grupo criado;
- Altere o grupo da sua pasta home para o grupo criado de forma recursiva, aplicando assim a alteração aos subdiretórios;
- Acesse o usuário julia, abra a pasta que contém o script e execute-o com esse usuário.

Para saber mais detalhes sobre como realizar esses passos, acesse a opinião da pessoa instrutora.

Boa prática!

Ver opinião do instrutor

Opinião do instrutor - Resumo da criação do script

Crie uma pasta chamada scripts-linux com o comando:

mkdir scripts-linux

Entre na pasta com o comando:

cd scripts-linux

Utilize o editor de texto vim para criar um script bash chamado monitoramento-logs.sh:

vim monitoramento-logs.sh

Habilite o modo de inserção no vim, clicando na tecla i.

No script, crie uma variável que armazena o caminho para o diretório em que os arquivos de logs estão armazenados. Em seguida faça o script exibir uma mensagem na tela, conforme mostrado a seguir:

#!/bin/bash

LOG_DIR="../myapp/logs"

echo "Verificando os logs no diretorio $LOG_DIR"

Saia do modo de inserção e vá para o modo de comando do vim, clicando na tecla Esc.

Salve o script digitando o comando a seguir:

:w

Saia do editor de texto com o comando:

:**q**

Dê a permissão de execução para o script, rodando o comando:

chmod 755 monitoramento-logs.sh

Em seguida, execute o script com o comando:

./monitoramento-logs.sh

No terminal, execute o comando a seguir para criar um novo usuário chamado julia:

sudo adduser julia

Serão pedidas algumas informações. Informe uma senha para esse usuário. As demais informações são opcionais.

Verifique se o usuário foi criado com sucesso, listando os usuários do sistema, com o comando:

cat /etc/passwd

Isso mostrará o novo usuário na lista.

Crie um novo grupo com o comando:

sudo groupadd devs

Verifique se o novo grupo foi criado com sucesso com o comando:

getent **group** devs

Adicione o novo usuário julia ao grupo devs usando o comando:

sudo usermod -aG devs julia

Verifique se o usuário foi adicionado com sucesso, rodando o comando:

getent **group** devs

Vá para o diretório home do seu usuário e altere o grupo para devs:

sudo chown -R :devs /home/SEU-USUARIO

Lembre-se de substituir SEU-USUARIO, pelo nome do seu usuário no seu sistema Linux.

Para verificar se a alteração de grupo deu certo, rode o comando:

ls -ld

Altere para o usuário julia, usando o comando:

su - julia

Informe a senha que você utilizou para criar esse usuário.

Entre na pasta do script com o comando:

cd /home/SEU-USUARIO/scripts-linux

Lembre-se de substituir SEU-USUARIO, pelo nome do seu usuário no seu sistema Linux.

Execute o script com o comando:

./processamento-logs.sh

Para voltar para o seu usuário basta rodar o comando:

su - SEU-USUARIO

Substitua SEU-USUARIO pelo nome do seu usuário no seu sistema Linux.

O que aprendemos?

Próxima Atividade

-

-

Nessa aula, você aprendeu como:

- Criar um script bash no Linux, para processar logs de uma aplicação, utilizando o editor de texto vim;
- Navegar entre os modos de inserção e comando no editor de texto vim, clicando na tecla i para o modo inserção e na tecla ESC para o modo de comando;
- Salvar arquivos no vim com o comando :w e sair do vim com o comando :q;
- Utilizar o comando chmod para adicionar e remover permissões de arquivos e diretórios;
- Usar as notações octal e a simbólica para gerenciar permissões no Linux;
- Gerenciar proprietários e grupos de arquivos e diretórios, através do comando chown.

4 – Encontrando arquivos

Transcrição

Estamos trabalhando no nosso *script* de processamento de *logs* e temos uma variável que armazena o diretório onde estão os nossos *logs*.

Pode ser que dentro desse diretório existam outros arquivos que não queremos utilizar nesse processamento, pois não são arquivos de *log*. Então, vamos verificar o que temos dentro da pasta.

Encontrando e filtrando arquivos

No terminal, vamos acessar a pasta "/myapp/logs" e executar o comando ls para listar os itens da pasta. Temos alguns arquivos com a extensão .log, mas também há outros com extensões diferentes que não queremos processar.

É interessante **criar um filtro** para processar somente os arquivos myapp-backend.log e myapp-frontend.log, que contêm as informações de *log* da nossa aplicação.

Para fazer esse filtro e obter somente os nomes dos arquivos que nos interessam, podemos usar o comando find no Linux. Vamos digitar:

find . -name "*.log"

Copiar código

O retorno será:

./myapp-frontend.log

./myapp/backend.log

Esse comando retorna somente os nomes dos arquivos que terminam com .log. A seguir, vamos entender mais a fundo o que cada parte do comando faz.

O find realiza uma busca. O ponto especifica que a busca deve ser feita a partir do diretório atual. Caso queiramos buscar em outro diretório, podemos especificar o caminho no lugar do ponto.

A opção -name permite especificar o padrão de busca, no caso, o nome. Entre as aspas duplas, utilizamos o caractere especial de asterisco, que representa qualquer cadeia de caracteres em uma *string*. Assim, qualquer texto que termine com ".log" será retornado.

Portanto, esse comando filtra apenas os arquivo de *log* e podemos utilizá-lo no nosso *script* de monitoramento. Vamos copiar o comando com "Ctrl + Shift + C", sair da pasta com cd e entrar na pasta do *script* com cd scripts-linux. Para abrir o *script*, digitamos vim monitoramento-logs.sh.

Incrementando o *script* de monitoramento

No *script*, vamos habilitar a edição pressionando a tecla "I". Após o echo, pularemos uma linha e colaremos o comando copiado com "Ctrl + Shift + V":

#!/bin/bash

LOG_DIR="../myapp/logs"

echo "Verificando logs no diretorio $LOG_DIR"

find . -name "*.log"

Copiar código

Faremos algumas alterações no comando. Em vez de usar o ponto, chamaremos a variável LOG_DIR para utilizar o caminho do diretório onde estão armazenados os *logs*:

#!/bin/bash

LOG_DIR="../myapp/logs"

echo "Verificando logs no diretorio $LOG_DIR"

find $LOG_DIR -name "*.log"

Copiar código

Agora, podemos usar os nomes dos arquivos filtrados para percorrer cada um deles e fazer o devido processamento dos *logs*.

Laços de repetição

Para percorrer o conteúdo, utilizamos **laços de repetição**. Basta especificar uma condição e, enquanto ela for verdadeira, executamos uma ação dentro do laço.

Para redirecionar a saída do comando find para um laço de repetição, usamos o operador *pipe*, que é uma barra vertical. O laço de repetição que utilizaremos é o while. Antes de especificar as condições e as ações do laço, vamos entender sua estrutura.

Começamos escrevendo while, que significa "enquanto" em inglês. Em seguida, definimos a condição e um ponto e vírgula. Depois, escrevemos do, que significa "faça" em inglês. O próximo passo é inserir as ações. Por fim, usamos a palavra done, que significa "feito" em inglês. Ou seja, todas as ações já foram feitas:

find $LOG_DIR -name "*.log" | **while** [condição]; **do**

  [ações]

done

Copiar código

Os trechos entre colchetes serão substituídos a seguir.

Na sequência, vamos especificamos a condição e as ações a serem realizadas. Para a condição, usaremos o IFS= (*Internal Field Separator*) definido como vazio, para evitar que nomes de arquivos com espaços ou caracteres especiais sejam quebrados:

find $LOG_DIR -name "*.log" | **while** IFS= ; **do**

  ações

done

Copiar código

Em seguida, usamos read para ler os arquivos passados pelo find, com algumas opções extras. A opção -r impede a interpretação caracteres especiais e -d '' indica que delimitador é o caractere nulo:

find $LOG_DIR -name "*.log" | **while** IFS= read -r -d ''; **do**

  ações

done

Copiar código

Por padrão, o find não utiliza o delimitador nulo. Para alterar essa configuração, vamos inserir -print0 antes do *pipe*, garantindo que a saída do find utilize o delimitador nulo, que é esperado pelo read:

find $LOG_DIR -name "*.log" -print0 | **while** IFS= read -r -d ''; **do**

  ações

done

Copiar código

Por fim, especificaremos uma variável arquivo para armazenar o nome dos arquivos de *log*. Assim, conseguiremos trabalhar com cada um deles individualmente:

find $LOG_DIR -name "*.log" -print0 | **while** IFS= read -r -d '' arquivo; **do**

  ações

done

Copiar código

Para verificar se o laço está funcionando, vamos incluir uma ação nele.
Chamaremos echo "Arquivo encontrado $arquivo":

find $LOG_DIR -name "*.log" -print0 | **while** IFS= read -r -d '' arquivo; **do**

  echo "Arquivo encontrado $arquivo"

done

Copiar código

Vamos sair do modo de inserção com "Esc". Para salvar e sair do Vim, em vez de usar os comandos :w e :q individualmente, podemos concatená-los assim: :wq.

Para executar o *script*, rodamos:

./monitoramento-logs.sh

Copiar código

No terminal, temos a seguinte a saída:

Verificando logs no diretorio ../myapp/logs

Arquivo encontrado: ../myapp/logs/myapp-frontend.log

Arquivo encontrado: ../myapp/logs/myapp-backend.log

O laço de repetição e o comando find estão encontrando os arquivos, o que nos permitirá realizar o processamento na aula a seguir.