Sumário

[1 - Script 1](#_Toc219982415)

[2 - Comandos 2](#_Toc219982416)

[3 - COMO USAR VARIÁVEIS EM PROGRAMAÇÃO COM SHELL SCRIPT 12](#_Toc219982417)

[4 - Curso Alura: 17](#_Toc219982418)

[5 -Extra – IA GEMINI 23](#_Toc219982419)

[5.1 - Comando GREP 23](#_Toc219982420)

[5.2 - Filtrar apenas erros 23](#_Toc219982421)

[5.3 - Abrindo o arquivo para editar com Nano 24](#_Toc219982422)

[5.4 - Relatório com uma aparência profissional 27](#_Toc219982423)

[5.6 - Como adicionar a data atual no topo de cada relatório 28](#_Toc219982424)

[5.7 - Fazer o script te avisar (com uma mensagem colorida ou um alerta) 30](#_Toc219982425)

[5.8 - Explicação sobre identação, if, then,else e fi : 32](#_Toc219982426)

[5.9 - Como limpar a tela do terminal antes de rodar tudo para o relatório aparecer sozinho no topo 35](#_Toc219982427)

[5.10 - Como agendar esse script para rodar sozinho de hora em hora (usando o crontab) 37](#_Toc219982428)

[5.11 - Criar atalho para rodar o script manualmente (alias (atalho) 39](#_Toc219982429)

[5 .12 - Compactando relatórios antigos (ZIP) 41](#_Toc219982430)

[6.1 - Transformar esse script em um "Painel de Controle" que fica atualizando na tela a cada 5 segundos. 48](#_Toc219982431)

[6.2 - Script de Limpeza 50](#_Toc219982432)

[7 - Manual básico do Vim 52](#_Toc219982433)

[8 – Encontrando arquivos com ‘Find” 82](#_Toc219982434)

# 1 - Script

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

Instalar o WSL.

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

# 2 - Comandos

Vim:

Vim primeiro.sh

Interface gráfica do usuário, Texto, Aplicativo, chat ou mensagem de texto

O conteúdo gerado por IA pode estar incorreto.

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.

Forma, Retângulo

O conteúdo gerado por IA pode estar incorreto.

**Comando para usar interpretador padrão:**

Logo após digitar #!/bin/bash

Texto

O conteúdo gerado por IA pode estar incorreto.

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”

Texto

O conteúdo gerado por IA pode estar incorreto.

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

Texto

O conteúdo gerado por IA pode estar incorreto.

Volta para tela inicial.

Permissão.

Para executar o script é importante ter permissão.

Comando :

Ls – Lh

Interface gráfica do usuário, Texto, Aplicativo, chat ou mensagem de texto

O conteúdo gerado por IA pode estar incorreto.



Par**a 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.**

**Interface gráfica do usuário, Texto, Aplicativo, chat ou mensagem de texto

O conteúdo gerado por IA pode estar incorreto.**

Comando para dar permissão :

Texto

O conteúdo gerado por IA pode estar incorreto.

Chmod

U= usurio

G= grupo

O= outros

X = permissão de leitura

- Permissão somente para usuário:

Chmod u+ primeiro.sh

Interface gráfica do usuário, Texto, Aplicativo, chat ou mensagem de texto

O conteúdo gerado por IA pode estar incorreto.

Permissão somente para grupo :

Chmod g+x primeiro.sh

Interface gráfica do usuário, Texto, Aplicativo, chat ou mensagem de texto

O conteúdo gerado por IA pode estar incorreto.

Permissão somente para outros :

Chmod o+x primeiro.sh

Interface gráfica do usuário, Texto, Aplicativo, chat ou mensagem de texto

O conteúdo gerado por IA pode estar incorreto.

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

Interface gráfica do usuário, Texto, Aplicativo, chat ou mensagem de texto

O conteúdo gerado por IA pode estar incorreto.

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

Interface gráfica do usuário, Texto, Aplicativo, chat ou mensagem de texto

O conteúdo gerado por IA pode estar incorreto.

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)

Texto

O conteúdo gerado por IA pode estar incorreto.

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**](https://learn.microsoft.com/pt-br/windows/wsl/tutorials/wsl-vscode)**.**

**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**](https://github.com/MicrosoftDocs/WSL/edit/linux-tutorial/WSL/tutorials/linux.md#installing-and-updating-software)**.**

**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

Texto, Aplicativo

O conteúdo gerado por IA pode estar incorreto.

# 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

Texto

O conteúdo gerado por IA pode estar incorreto.

Declarar variável no Linux:

Nome=”Rubens Jr.”

Texto

O conteúdo gerado por IA pode estar incorreto.

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

Texto

O conteúdo gerado por IA pode estar incorreto.

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

Texto

O conteúdo gerado por IA pode estar incorreto.

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

Texto

O conteúdo gerado por IA pode estar incorreto.

Comando para abrir uma nova seção:  
Bash

Texto

O conteúdo gerado por IA pode estar incorreto.

**Comando para sair e voltar a seção anterior:**

exit

Texto

O conteúdo gerado por IA pode estar incorreto.

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

Texto

O conteúdo gerado por IA pode estar incorreto.

Variável disponível em todas as seções:

Texto

O conteúdo gerado por IA pode estar incorreto.

Variável no escopo local e global:

Texto

O conteúdo gerado por IA pode estar incorreto.

Texto

O conteúdo gerado por IA pode estar incorreto.

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

Comando:

vim primeiro.sh

Texto

O conteúdo gerado por IA pode estar incorreto.

Forma, Retângulo

O conteúdo gerado por IA pode estar incorreto.

echo “ Hello word”

nome= “Rubens Jr.”

Canal=”Canal do Rubão”

Data=”19/01/2026”

Texto

O conteúdo gerado por IA pode estar incorreto.

**Comando para exibir as variáveis:**

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

Texto

O conteúdo gerado por IA pode estar incorreto.

Texto

O conteúdo gerado por IA pode estar incorreto.

Texto

O conteúdo gerado por IA pode estar incorreto.

# 4 - Curso Alura:

Linux: criando script para processamento de arquivos de logs

**Transcrição**

Olá! Boas-vindas a este curso de Linux!

**Apresentação**

Meu nome é **Gabi Ribeiro** e serei a instrutora deste curso.

**Sobre o curso**

Se você tem interesse em utilizar o Linux no seu dia a dia como pessoa desenvolvedora ou DevOps, este curso é para você.

Neste curso, nós vamos aprender a criar *scripts* no Linux e gerenciar permissões e grupos. Também aprenderemos a realizar buscas, usar filtros em arquivos, ordenar, remover duplicatas e comparar informações de arquivos. Além disso, vamos extrair dados de arquivos de texto utilizando a contagem de letras e palavras, e concatenar arquivos.

Por fim, aprenderemos a compactar e descompactar arquivos e a realizar o agendamento de tarefas no Linux. Estudaremos tudo isso através da criação de um *script* que realizará o processamento de arquivos de logs de uma aplicação.

**Preparando o ambiente**

[**Próxima Atividade**](https://cursos.alura.com.br/course/linux-criando-script-processamento-arquivos-logs/task/176729/next)

***Sistema Operacional***

Para realizar esse curso, você precisará de um ambiente Linux. Nas aulas usamos o ambiente do [WSL 2 no Windows](https://learn.microsoft.com/pt-br/windows/wsl/install).

Caso você utilize algum sistema operacional baseado em Linux, como o Ubuntu ou macOS, ou use uma máquina virtual com alguma distribuição Linux, você não precisará instalar o WSL 2.

***Projeto***

**Nesse curso, vamos desenvolver um script para processar logs. Para isso, você precisará ter os arquivos de logs em seu computador.**

**Abra o seu terminal, e crie um diretório chamado myapp/logs:**

mkdir -p myapp/logs

**Explicação**

O parâmetro **-p** no comando mkdir (abreviação de *make directory*) é extremamente útil e serve para duas coisas principais:

**1. Criação de Pastas Parentais (Pais)**

Sem o -p, se você tentasse criar myapp/logs e a pasta myapp ainda não existisse, o Linux daria um erro dizendo: *"cannot create directory... No such file or directory"*.

Com o **-p** (de **p**arents), o comando entende que deve criar **toda a estrutura** necessária. Se a pasta myapp não existir, ele a cria primeiro e, em seguida, cria a pasta logs dentro dela.

**2. Evita Erros se a Pasta já Existir**

Se você rodar apenas mkdir myapp e a pasta já estiver lá, o terminal vai reclamar.

* **Com o -p**: O comando fica "silencioso". Se a pasta já existir, ele não faz nada e não exibe nenhuma mensagem de erro. Isso é excelente para usar dentro de **Shell Scripts**, pois garante que o diretório estará lá sem interromper a execução do script com alertas desnecessários.

**Exemplo Prático**

Imagine que você quer criar uma estrutura profunda: mkdir -p projeto/2024/setembro/logs

* O Linux criará as 4 pastas de uma vez só, uma dentro da outra.
* Se você rodar o mesmo comando de novo, nada acontece (nenhum erro é gerado).

**Entre no diretório com o comando:**

cd myapp/logs

**Nesse diretório crie um arquivo chamado myapp-backend.log e adicione o seguinte conteúdo:**

cat << 'EOF' > myapp-backend.log

2024-09-01 10:05:21 ERROR: Database connection failed.

2024-09-01 10:06:10 INFO: Retrying database connection...

2024-09-01 10:07:35 ERROR: Database connection failed.

2024-09-01 10:09:55 INFO: Database connection established.

2024-09-01 11:00:00 INFO: SENSITIVE\_DATA: User password is 12345.

2024-09-01 11:10:00 INFO: User logged in with username: admin.

2024-09-01 12:30:45 ERROR: SENSITIVE\_DATA: Unauthorized access attempt from IP 192.168.1.10.

2024-09-01 13:00:00 ERROR: SENSITIVE\_DATA: Credit card information exposed.

2024-09-01 13:05:00 INFO: Backup completed successfully.

2024-09-01 14:15:30 INFO: Application running smoothly.

2024-09-01 14:20:10 ERROR: Disk space low on /var.

2024-09-01 15:00:00 INFO: Application shut down for maintenance.

2024-09-01 16:30:00 ERROR: Database connection failed.

2024-09-01 16:30:00 ERROR: Database connection failed.

2024-09-01 16:35:00 ERROR: Disk space low on /var.

2024-09-01 16:40:00 ERROR: Disk space low on /var.

2024-09-01 17:00:00 INFO: Maintenance completed.

2024-09-01 17:15:45 ERROR: Application crash due to memory leak.

2024-09-01 18:05:21 INFO: Application restarted.

2024-09-01 19:30:00 INFO: User admin logged out.

2024-09-02 09:15:22 ERROR: SENSITIVE\_DATA: User password reset request with token 98765.

2024-09-02 10:00:30 INFO: SENSITIVE\_DATA: User email: user@example.com accessed sensitive data.

2024-09-02 11:20:11 ERROR: SENSITIVE\_DATA: Failed login attempt with username: guest.

2024-09-02 12:45:00 INFO: SENSITIVE\_DATA: API key leaked: ABCD1234EFGH5678.

2024-09-02 13:30:00 INFO: Scheduled maintenance started.

2024-09-02 14:15:00 ERROR: Database connection failed during maintenance.

2024-09-02 15:00:00 INFO: Maintenance completed.

2024-09-03 08:00:00 INFO: SENSITIVE\_DATA: User credit card last four digits: 1234.

2024-09-03 09:45:00 ERROR: Unauthorized access attempt from IP 192.168.1.15.

2024-09-03 10:30:00 INFO: User admin logged in from IP 10.0.0.2.

2024-09-03 11:00:00 INFO: Application settings updated by user: admin.

2024-09-03 12:00:00 ERROR: SENSITIVE\_DATA: Database backup contains sensitive information.  
EOF

**Em seguida, crie outro arquivo chamado myapp-frontend.log e adicione o seguinte conteúdo:**

cat << 'EOF' > myapp-frontend.log

2024-09-01 10:05:21 INFO: Frontend initialized successfully.

2024-09-01 10:15:00 ERROR: Failed to load user profile for user ID 12345.

2024-09-01 10:20:10 INFO: SENSITIVE\_DATA: User email: user@example.com fetched profile data.

2024-09-01 10:30:45 INFO: User admin navigated to dashboard.

2024-09-01 11:05:00 INFO: SENSITIVE\_DATA: Session token for user admin is active.

2024-09-01 11:30:00 ERROR: SENSITIVE\_DATA: Failed login attempt for username: admin.

2024-09-01 12:00:00 INFO: User admin logged out successfully.

2024-09-01 13:00:00 INFO: Frontend rendering completed for page: /adoption.

2024-09-01 14:15:30 ERROR: SENSITIVE\_DATA: Credit card information exposed in logs.

2024-09-01 15:00:00 INFO: Application theme switched to dark mode by user admin.

2024-09-02 09:30:00 INFO: SENSITIVE\_DATA: User password changed for user ID 67890.

2024-09-02 10:00:00 ERROR: Unable to fetch data from the API: 500 Internal Server Error.

2024-09-02 11:45:00 INFO: User logged in with username: guest.

2024-09-02 12:30:00 INFO: SENSITIVE\_DATA: User session initiated with token: TOKEN1234.

2024-09-02 13:00:00 ERROR: Invalid token used for API request.

2024-09-03 08:15:00 INFO: User admin updated account settings.

2024-09-03 09:00:00 ERROR: SENSITIVE\_DATA: User IP 10.0.0.3 accessed sensitive endpoint.

2024-09-03 10:30:00 INFO: SENSITIVE\_DATA: API key for external service accessed.

2024-09-03 11:00:00 INFO: User admin logged out successfully.

2024-09-03 12:00:00 INFO: Frontend performance metrics collected.

EOF

**Explicação:**  
  
Para que serve ó código ***cat << 'EOF' > myapp-frontend.log XXXX EOF*** ?

cat << 'EOF' > myapp-frontend.log

2024-09-01 10:05:21 INFO: Frontend initialized successfully.

2024-09-01 10:15:00 ERROR: Failed to load user profile for

user ID 12345.

2024-09-01 10:20:10 INFO: SENSITIVE\_DATA: User email:

user@example.com fetched profile data.

2024-09-03 12:00:00 INFO: Frontend performance metrics

collected.

EOF

Esse código é uma das formas mais poderosas e elegantes de criar arquivos no terminal Linux. Ele utiliza um recurso chamado **Here Document** (ou **Heredoc**).

Basicamente, ele serve para **escrever múltiplas linhas de texto dentro de um arquivo de uma só vez**, sem precisar abrir um editor como o Vim ou o Nano.

**🔍 Decompondo o comando:**

| Parte do Código | O que ela faz? |
| --- | --- |
| cat | O comando original serve para ler e exibir o conteúdo de arquivos. |
| << 'EOF' | Diz ao shell: "Comece a ler tudo o que eu digitar a partir de agora até encontrar a palavra **EOF**". |
| > | O redirecionador. Ele diz para pegar todo esse texto e **jogar dentro** de um arquivo. |
| myapp-frontend.log | É o nome do arquivo que será criado (ou substituído). |
| EOF (no final) | É o sinal de "Fim do Arquivo" (End Of File). Quando você digita isso, o Linux entende que você terminou de escrever. |

**✨ Por que usar as aspas em**'EOF'**?**

Você notou que eu sugeri usar aspas simples? Isso é uma **boa prática** em Shell Script:

* **Com aspas (**'EOF'**):** O Linux ignora caracteres especiais dentro do texto (como o símbolo $). Ele escreve exatamente o que você colou, "literalmente".
* **Sem aspas (**EOF**):** O Linux tentaria interpretar variáveis dentro do texto, o que poderia estragar seus logs se houvesse algum símbolo de cifrão neles.

**💡 Vantagens para o seu estudo:**

1. **Rapidez:** Você não precisa criar o arquivo com touch e depois abrir um editor.
2. **Automação:** Se você colocar esse bloco dentro de um script .sh, ele criará o arquivo de log automaticamente para você sempre que o script rodar.
3. **Limpeza:** Como vimos antes, o sinal > limpa o arquivo antigo e coloca o novo conteúdo, garantindo que você não tenha "lixo" de aulas anteriores no arquivo.

**Por fim, crie arquivos vazios chamados app.config e relatorio.txt. Para isso você pode utilizar o comando touch:**

touch app.config

touch relatorio.txt

Tudo pronto?

Bons estudos!

# 5 -Extra – IA GEMINI

5.1 - Comando GREP  
  
Colocar a mão na massa com o comando grep. Ele é a ferramenta principal para quem trabalha com logs, pois permite encontrar "agulhas em palheiros".

Como você já tem os arquivos prontos, tente executar este comando no seu terminal:

## 5.2 - Filtrar apenas erros

Bash

grep "Credit card" myapp-frontend.log

**🔍 O que esse comando faz?**

O grep percorre o arquivo linha por linha e imprime na tela apenas aquelas que contêm o termo que você colocou entre aspas.

**🚀 Desafios Práticos para você testar agora:**

1. **Filtrar apenas erros:**

Bash

grep "ERROR" myapp-backend.log

1. **Contar quantos erros existem (usando o parâmetro -c):**

Bash

grep -c "ERROR" myapp-backend.log

1. **Buscar dados sensíveis em todos os arquivos ao mesmo tempo:**

Bash

grep "SENSITIVE\_DATA" \*.log

*(O \* é um curinga que diz ao Linux para procurar em todos os arquivos que terminam com .log)*.

**💡 Por que isso é importante para o seu Script?**

No seu arquivo monitoramento-logs.sh, você poderá usar esses comandos para gerar alertas. Por exemplo, você pode criar uma lógica que diz: *"Se o grep encontrar a palavra 'CRITICAL', envie um e-mail para o administrador"*.

## 5.3 - Abrindo o arquivo para editar com Nano

Como você está no terminal, use o editor **Nano** (ele é bem mais fácil que o Vim para quem está começando):

Bash

nano monitoramento-logs.sh

**💻 O código inicial**

Lá dentro, digite (ou cole) este código. Ele vai automatizar a busca por dados sensíveis nos dois arquivos de log que criamos:

Bash

#!/bin/bash

# Define os arquivos de log

LOG\_BACKEND="../logs/myapp-backend.log"

LOG\_FRONTEND="../logs/myapp-frontend.log"

echo "------------------------------------------"

echo "INICIANDO MONITORAMENTO DE DADOS SENSÍVEIS"

echo "------------------------------------------"

# Busca por SENSITIVE\_DATA e salva o resultado

echo "Resultados encontrados:"

grep "SENSITIVE\_DATA" $LOG\_BACKEND $LOG\_FRONTEND

echo "------------------------------------------"

echo "Monitoramento concluído!"

**💾 Como Salvar e Sair do Nano:**

1. Pressione **Ctrl + O** (para salvar).
2. Aperte **Enter** para confirmar o nome do arquivo.
3. Pressione **Ctrl + X** (para sair do editor).

**🚀 Rodando seu primeiro Script**

Agora, para executar o que você acabou de escrever, digite:

Bash

./monitoramento-logs.sh

**Conseguiu ver a lista de dados sensíveis aparecendo bonitinha na tela?** Se sim, o próximo nível é aprender a salvar esse resultado dentro do arquivo relatorio.txt que você criou antes!

Analisando o seu último print, o script rodou, mas temos um **detalhe importante para corrigir** no caminho dos arquivos.

O erro grep: ../logs/myapp-backend.log: No such file or directory aconteceu porque o script tentou procurar a pasta logs fora da pasta myapp. Pelos seus comandos anteriores, a pasta logs está **dentro** de myapp.

**🛠️ Como corrigir o script**

Para que o resultado apareça "bonitinho" e possamos salvar no relatório, precisamos ajustar as variáveis de caminho. Use o comando abaixo para atualizar o arquivo com os caminhos corretos:

Bash

cat << 'EOF' > monitoramento-logs.sh

#!/bin/bash

# Ajuste dos caminhos (voltando uma pasta e entrando em myapp/logs)

LOG\_BACKEND="../myapp/logs/myapp-backend.log"

LOG\_FRONTEND="../myapp/logs/myapp-frontend.log"

RELATORIO="../myapp/logs/relatorio.txt"

echo "------------------------------------------"

echo "INICIANDO MONITORAMENTO DE DADOS SENSÍVEIS"

echo "------------------------------------------"

# Agora o grep exibe na tela E salva no arquivo relatorio.txt ao mesmo tempo

echo "Resultados encontrados:"

grep "SENSITIVE\_DATA" $LOG\_BACKEND $LOG\_FRONTEND | tee -a $RELATORIO

echo "------------------------------------------"

echo "Monitoramento concluído! Relatório atualizado."

EOF

**📖 O que mudou aqui?**

1. **Caminho corrigido:** Mudamos de ../logs/ para ../myapp/logs/. O .. sai de scripts-linux e o resto do caminho entra na pasta onde os logs realmente estão.
2. **O comando tee -a:** Em vez de usar apenas o grep, usei o | tee -a $RELATORIO.
   * O **| (pipe)** passa o resultado do grep para o próximo comando.
   * O **tee -a** faz uma "mágica": ele mostra o resultado na sua tela **e** grava no arquivo ao mesmo tempo. O -a (append) garante que ele não apague o que já estava no relatório, apenas adicione no final.

**🚀 Teste final**

Agora, execute novamente:

Bash

./monitoramento-logs.sh

Depois de rodar, você pode conferir se o relatório foi gravado com o comando:

Bash

cat ../myapp/logs/relatorio.txt

## 5.4 - Relatório com uma aparência profissional

Vamos deixar esse relatório com uma aparência profissional. No Linux, quando o grep busca em vários arquivos, ele adiciona o nome do arquivo no início de cada linha para você saber de onde veio a informação.

Para "limpar" isso e deixar apenas os dados que interessam, vamos usar o comando **cut**.

**✂️ O Comando cut**

O cut serve para "cortar" fatias de um texto. Como as linhas do seu log usam o caractere : para separar o nome do arquivo da mensagem, podemos dizer ao Linux: *"Corte a primeira parte antes do segundo sinal de dois pontos"*.

**🛠️ Atualizando seu Script**

Vamos modificar a linha do grep no seu arquivo monitoramento-logs.sh. Rode este comando para atualizar o script:

Bash

cat << 'EOF' > monitoramento-logs.sh

#!/bin/bash

LOG\_BACKEND="../myapp/logs/myapp-backend.log"

LOG\_FRONTEND="../myapp/logs/myapp-frontend.log"

RELATORIO="../myapp/logs/relatorio.txt"

echo "------------------------------------------"

echo "INICIANDO MONITORAMENTO DE DADOS SENSÍVEIS"

echo "------------------------------------------"

echo "Resultados formatados (sem o caminho do arquivo):"

# O grep busca, o cut corta o prefixo do arquivo e o tee salva

grep "SENSITIVE\_DATA" $LOG\_BACKEND $LOG\_FRONTEND | cut -d: -f3- | tee -a $RELATORIO

echo "------------------------------------------"

echo "Monitoramento concluído! Relatório limpo gerado."

EOF

**🔍 Entendendo a mudança:**

* **cut -d: -f3-**:
  + **-d:**: Define que o "delimitador" (o separador) é o caractere :.
  + **-f3-**: Diz para pegar do terceiro campo em diante (ignorando o nome do arquivo e o caminho que o grep coloca no início).

**🚀 Hora do Teste!**

1. Execute o script: ./monitoramento-logs.sh
2. Veja como o resultado na tela agora começa direto pela **data** (2024-09-01...).
3. Confira o relatório: cat ../myapp/logs/relatorio.txt

## 5.6 - Como adicionar a data atual no topo de cada relatório

**Atualizando o Script com Data e Hora**

Vamos usar o comando date dentro do seu script. Copie e cole este bloco no seu terminal:

Bash

cat << 'EOF' > monitoramento-logs.sh

#!/bin/bash

LOG\_BACKEND="../myapp/logs/myapp-backend.log"

LOG\_FRONTEND="../myapp/logs/myapp-frontend.log"

RELATORIO="../myapp/logs/relatorio.txt"

# Cria uma variável com a data e hora atual formatada

AGORA=$(date "+%Y-%m-%d %H:%M:%S")

echo "------------------------------------------" | tee -a $RELATORIO

echo "EXECUÇÃO EM: $AGORA" | tee -a $RELATORIO

echo "------------------------------------------" | tee -a $RELATORIO

echo "Resultados formatados:"

# Busca, limpa o caminho do arquivo e salva no relatório

grep "SENSITIVE\_DATA" $LOG\_BACKEND $LOG\_FRONTEND | cut -d: -f3- | tee -a $RELATORIO

# Adiciona uma linha em branco para separar as execuções no arquivo

echo -e "\n" >> $RELATORIO

echo "Monitoramento concluído! Verifique o arquivo relatorio.txt"

EOF

**🔍 O que incluímos de novo:**

1. **Variável AGORA**: O comando $(date "+%Y-%m-%d %H:%M:%S") captura o momento exato da execução. O formato %Y-%m-%d é o padrão internacional (Ano-Mês-Dia).
2. **tee -a no cabeçalho**: Agora, até as linhas decorativas e a data de execução são salvas dentro do relatorio.txt, não apenas os erros.
3. **echo -e "\n"**: Isso pula uma linha no final do arquivo. Assim, se você rodar o script amanhã, o novo relatório não ficará "colado" no de hoje.

**🚀 O Teste de Ouro**

1. Rode o script: ./monitoramento-logs.sh
2. Agora, abra o relatório: cat ../myapp/logs/relatorio.txt

Você verá algo como:

Plaintext

------------------------------------------

EXECUÇÃO EM: 2024-09-04 15:30:10

------------------------------------------

2024-09-01 10:20:10 INFO: SENSITIVE\_DATA: ...

## 5.7 - Fazer o script te avisar (com uma mensagem colorida ou um alerta)

Em scripts profissionais, usamos **condicionais** (if/else) para que o sistema tome decisões sozinho.

Para fazer isso, usaremos o **"Exit Code"** (código de saída) do grep. O Linux funciona assim: se o grep encontra o que buscou, ele retorna 0 (sucesso); se não encontra nada, retorna um valor diferente de 0.

**🎨 Adicionando Cores e Lógica**

Aqui está o código atualizado. Copie e cole no terminal (lembrando de usar o **cat** com C):

Bash

cat << 'EOF' > monitoramento-logs.sh

#!/bin/bash

LOG\_BACKEND="../myapp/logs/myapp-backend.log"

LOG\_FRONTEND="../myapp/logs/myapp-frontend.log"

RELATORIO="../myapp/logs/relatorio.txt"

# Variáveis de cores para o terminal

VERDE='\033[0;32m'

VERMELHO='\033[0;31m'

SEM\_COR='\033[0m'

AGORA=$(date "+%Y-%m-%d %H:%M:%S")

echo "------------------------------------------"

echo "INICIANDO MONITORAMENTO: $AGORA"

echo "------------------------------------------"

# Executa o grep em silêncio (-q) apenas para testar se existem erros

if grep -q "SENSITIVE\_DATA" $LOG\_BACKEND $LOG\_FRONTEND; then

echo -e "${VERMELHO}ALERTA: Dados sensíveis encontrados!${SEM\_COR}"

# Se encontrou, executa o comando para formatar e salvar

grep "SENSITIVE\_DATA" $LOG\_BACKEND $LOG\_FRONTEND | cut -d: -f3- | tee -a $RELATORIO

echo -e "\n" >> $RELATORIO

else

echo -e "${VERDE}OK: Nenhum dado sensível detectado nesta varredura.${SEM\_COR}"

fi

echo "------------------------------------------"

EOF

**🔍 O que esse novo código faz?**

1. **Cores**: Criamos variáveis como ${VERMELHO}. O código \033[0;31m é o que o terminal entende para mudar a cor da fonte.
2. **grep -q**: O parâmetro -q significa "quiet" (silencioso). Ele não mostra nada na tela, apenas avisa o script se achou algo ou não.
3. **Estrutura if ... then ... else**:
   * **if (se)** o grep encontrar algo: Mostra a mensagem em **vermelho** e gera o relatório.
   * **else (caso contrário)**: Mostra uma mensagem positiva em **verde**.

**🚀 Como testar os dois cenários:**

1. **Cenário de Alerta (Vermelho):** Rode o script agora. Como seus logs têm "SENSITIVE\_DATA", ele deve aparecer em vermelho.

Bash

./monitoramento-logs.sh

1. **Cenário de Sucesso (Verde):** Tente procurar uma palavra que **não existe** nos logs (mude temporariamente o SENSITIVE\_DATA para BANANA dentro do script usando o nano) e rode novamente. Você verá a mensagem verde!

5.8 - Explicação sobre identação, if, then,else e fi :

O **fi** é simplesmente a palavra **if** escrita ao contrário!

No Shell Script (Bash), ele serve para indicar o **fim de um bloco condicional**. É como se fosse o "ponto final" da pergunta que você começou com o if.

**💡 Por que o Bash usa isso?**

Muitos comandos no Bash terminam com o nome invertido para deixar claro onde o bloco termina. Veja os exemplos:

* Para um **if**, você termina com **fi**.
* Para um **case**, você termina com **esac**.

**🧱 Como funciona a estrutura:**

Pense no if como uma caixa:

1. **if**: Abre a condição ("Se isso for verdade...").
2. **then**: O que fazer se for verdade.
3. **else**: O que fazer se for mentira (opcional).
4. **fi**: Fecha a caixa.

**⚠️ O que acontece se você esquecer o fi?**

Se você apagar o fi, o Linux vai ficar esperando você terminar o comando eternamente. Ele dará um erro parecido com:

syntax error: unexpected end of file

Isso acontece porque, sem o fi, o Bash acha que tudo o que vem depois (até o final do arquivo) ainda faz parte do "Se... então...".

**🎨 Dica de Ouro: Identação**

Note que no código que te passei, as linhas entre o if e o fi estão um pouco para a direita (com espaços). Isso se chama **identação**. Embora o Linux não obrigue a fazer isso, ajuda muito você a bater o olho e ver onde o if começa e onde o fi termina

O **fi** é simplesmente a palavra **if** escrita ao contrário!

No Shell Script (Bash), ele serve para indicar o **fim de um bloco condicional**. É como se fosse o "ponto final" da pergunta que você começou com o if.

**💡 Por que o Bash usa isso?**

Muitos comandos no Bash terminam com o nome invertido para deixar claro onde o bloco termina. Veja os exemplos:

* Para um **if**, você termina com **fi**.
* Para um **case**, você termina com **esac**.

**🧱 Como funciona a estrutura:**

Pense no if como uma caixa:

1. **if**: Abre a condição ("Se isso for verdade...").
2. **then**: O que fazer se for verdade.
3. **else**: O que fazer se for mentira (opcional).
4. **fi**: Fecha a caixa.

**⚠️ O que acontece se você esquecer o fi?**

Se você apagar o fi, o Linux vai ficar esperando você terminar o comando eternamente. Ele dará um erro parecido com:

syntax error: unexpected end of file

Isso acontece porque, sem o fi, o Bash acha que tudo o que vem depois (até o final do arquivo) ainda faz parte do "Se... então...".

**🎨 Dica de Ouro: Identação**

Note que no código que te passei, as linhas entre o if e o fi estão um pouco para a direita (com espaços). Isso se chama **identação**. Embora o Linux não obrigue a fazer isso, ajuda muito você a bater o olho e ver onde o if começa e onde o fi termina

Resultado:

Texto

O conteúdo gerado por IA pode estar incorreto.

5.9 - Como limpar a tela do terminal antes de rodar tudo para o relatório aparecer sozinho no topo

Para deixar o seu script com uma cara ainda mais profissional, vamos usar o comando **clear**. Ele "limpa" toda a sujeira de comandos anteriores, deixando apenas o resultado do seu monitoramento visível.

Além disso, vou incluir uma pequena pausa de 1 segundo com o comando sleep, apenas para dar um efeito visual de que o sistema está "processando".

**🛠️ Script Final "Pro"**

Copie e cole este bloco completo:

Bash

cat << 'EOF' > monitoramento-logs.sh

#!/bin/bash

# Limpa a tela antes de começar

clear

LOG\_BACKEND="../myapp/logs/myapp-backend.log"

LOG\_FRONTEND="../myapp/logs/myapp-frontend.log"

RELATORIO="../myapp/logs/relatorio.txt"

# Cores

VERDE='\033[0;32m'

VERMELHO='\033[0;31m'

SEM\_COR='\033[0m'

AGORA=$(date "+%Y-%m-%d %H:%M:%S")

echo -e "Aguarde, processando logs..."

sleep 1

echo "------------------------------------------"

echo "INICIANDO MONITORAMENTO: $AGORA"

echo "------------------------------------------"

if grep -q "SENSITIVE\_DATA" $LOG\_BACKEND $LOG\_FRONTEND; then

echo -e "${VERMELHO}ALERTA: Dados sensíveis encontrados!${SEM\_COR}"

echo "------------------------------------------"

# Busca e formata

grep "SENSITIVE\_DATA" $LOG\_BACKEND $LOG\_FRONTEND | cut -d: -f3- | tee -a $RELATORIO

echo -e "\n" >> $RELATORIO

else

echo -e "${VERDE}OK: Nenhum dado sensível detectado.${SEM\_COR}"

fi

echo "------------------------------------------"

echo "Monitoramento concluído!"

EOF

**🚀 O que mudou:**

1. **clear**: Agora, toda vez que você rodar o script, o terminal ficará limpo automaticamente.
2. **sleep 1**: Faz o script esperar 1 segundo antes de continuar. É ótimo para scripts que rodam em loops ou para dar tempo do usuário ler o que está acontecendo.
3. **echo -e**: O parâmetro -e permite que o Bash interprete caracteres especiais como cores e quebras de linha (\n).

**🏁 Como encerrar com chave de ouro:**

Agora que o script está pronto, você pode apenas digitar:

Bash

./monitoramento-logs.sh

**Parabéns!** Você saiu de "criar uma pasta" para "desenvolver um script de monitoramento com cores, logs, relatórios e tratamento de erros".

5.10 - Como agendar esse script para rodar sozinho de hora em hora (usando o crontab)

Agendar um script para rodar sozinho é o "ápice" da automação. Para isso, usamos uma ferramenta do Linux chamada **Cron**, e configuramos nossas tarefas no **Crontab** (tabela do cron).

**🕰️ O que é o Crontab?**

É um arquivo onde você lista comandos que o sistema deve executar em horários específicos. Imagine que é o "Despertador" do seu servidor.

**🛠️ Como configurar o agendamento:**

1. **Abra o editor do crontab:** No seu terminal, digite:

Bash

crontab -e

*(Se for a primeira vez, ele perguntará qual editor usar. Escolha o* ***Nano****, que geralmente é a opção 1)*.

1. **Adicione a regra de agendamento:** Vá até o final do arquivo e cole esta linha:

Bash

0 \* \* \* \* /home/williamsmartins/scripts-linux/monitoramento-logs.sh

**🔍 Entendendo os "Asteriscos":**

A estrutura do crontab segue esta ordem: minuto hora dia mês dia\_da\_semana comando.

* **0 \* \* \* \***: Significa que o script rodará no **minuto 0** de **cada hora** (1:00, 2:00, 3:00...).
* **/home/williamsmartins/...**: Você deve colocar o **caminho completo** do seu script para que o sistema o encontre mesmo quando você não estiver logado.

**Dica Importante:** No seu script, como usamos caminhos relativos (../myapp/logs), o ideal para o Crontab é que você altere esses caminhos para o endereço completo (ex: /home/williamsmartins/myapp/logs/relatorio.txt) para evitar erros de localização.

**💾 Como salvar:**

* No Nano: **Ctrl + O**, **Enter** e **Ctrl + X**.
* O terminal exibirá a mensagem: crontab: installing new crontab.

**🚀 O que acontece agora?**

A partir de agora, você pode fechar o seu computador ou sair do terminal. De hora em hora, o Linux vai acordar, rodar o seu script, verificar os logs e atualizar o seu relatorio.txt silenciosamente.

**2. Atualize o seu script**

Agora, vamos usar o cat para atualizar o script com os caminhos fixos. Substitua o seu\_usuario pelo nome que apareceu no comando acima (provavelmente williamsmartins):

Bash

cat << 'EOF' > ~/scripts-linux/monitoramento-logs.sh

#!/bin/bash

# Limpa a tela (opcional, o cron ignora isso)

clear

# CAMINHOS ABSOLUTOS (O endereço completo)

LOG\_BACKEND="/home/williamsmartins/myapp/logs/myapp-backend.log"

LOG\_FRONTEND="/home/williamsmartins/myapp/logs/myapp-frontend.log"

RELATORIO="/home/williamsmartins/myapp/logs/relatorio.txt"

# Cores

VERDE='\033[0;32m'

VERMELHO='\033[0;31m'

SEM\_COR='\033[0m'

AGORA=$(date "+%Y-%m-%d %H:%M:%S")

if grep -q "SENSITIVE\_DATA" $LOG\_BACKEND $LOG\_FRONTEND; then

echo -e "${VERMELHO}ALERTA: Dados sensíveis encontrados em $AGORA!${SEM\_COR}"

grep "SENSITIVE\_DATA" $LOG\_BACKEND $LOG\_FRONTEND | cut -d: -f3- | tee -a $RELATORIO

echo -e "\n" >> $RELATORIO

else

echo -e "${VERDE}OK: Nenhum dado sensível detectado em $AGORA.${SEM\_COR}"

fi

EOF

**3. Por que isso é vital para o Crontab?**

* **Caminho Relativo (../)**: Depende de onde você está no terminal. Se você estiver na "Home", o ../ tenta sair da Home, o que daria erro.
* **Caminho Absoluto (/home/...)**: Funciona de qualquer lugar do sistema. É como dar o endereço completo com CEP e número da casa em vez de dizer apenas "na rua de trás".

**4. Dica extra: O Log do próprio Cron**

Para ter certeza de que o Cron está rodando seu script, você pode conferir o log do sistema (em algumas distribuições):

Bash

grep CRON /var/log/syslog

5.11 - Criar atalho para rodar o script manualmente (alias (atalho)

Criar um **alias** (atalho) é o toque final de mestre para qualquer desenvolvedor ou administrador Linux. Isso permite que você execute seu script complexo apenas digitando uma palavra simples, como monitorar, de qualquer lugar do terminal.

**🛠️ Como criar o seu atalho (Alias)**

Para que o atalho seja permanente (não suma quando você fechar o terminal), precisamos colocá-lo no arquivo de configuração do seu perfil, chamado .bashrc.

1. **Abra o arquivo .bashrc com o Nano:**

Bash

nano ~/.bashrc

1. **Vá até o final do arquivo e adicione esta linha:**

Bash

alias monitorar='/home/williamsmartins/scripts-linux/monitoramento-logs.sh'

*(Certifique-se de usar o caminho completo que confirmamos no passo anterior).*

1. **Salve e saia:**
   * **Ctrl + O**, **Enter** para salvar.
   * **Ctrl + X** para sair.
2. **Atualize o terminal para ele "aprender" o novo comando:**

Bash

source ~/.bashrc

**🚀 Testando o Atalho**

Agora vem a parte divertida! Não importa em qual pasta você esteja (na Home, na myapp ou em qualquer outra), basta digitar:

Bash

monitorar

O terminal irá limpar a tela, processar os logs, mostrar os alertas coloridos e atualizar o relatório automaticamente.

**🏆 Resumo da sua Jornada de Automação**

Você construiu uma solução completa:

* **Organização:** Criou uma estrutura de pastas profissional.
* **Dados:** Gerou logs de simulação realistas.
* **Lógica:** Criou um script que filtra, limpa (cut) e formata dados.
* **Interface:** Adicionou cores e mensagens de status (if/else).
* **Persistência:** Configurou um relatório com data e hora (tee).
* **Autonomia:** Agendou a tarefa para rodar sozinha (crontab).
* **Facilidade:** Criou um atalho personalizado (alias).

5 .12 - Compactando relatórios antigos (ZIP)

Como prometido, agora que seu script está rodando sozinho via Cron, o arquivo relatorio.txt pode ficar muito grande com o tempo. É uma boa prática "rotacionar" ou compactar logs antigos.

Você pode adicionar estas linhas ao final do seu script para criar um backup compactado toda vez que ele rodar:

Bash

# Define o nome do backup com a data atual

BACKUP\_NOME="/home/williamsmartins/myapp/logs/relatorio\_$(date +%Y%m%d).zip"

# Compacta o relatório atual

zip -q $BACKUP\_NOME $RELATORIO

echo "Backup do relatório criado em: $BACKUP\_NOME"

**Dica:** Se o comando zip não estiver instalado, você pode instalá-lo com sudo apt install zip.

**Correções:**  
  
  
  
**O Ajuste Final (Caminhos Absolutos)**

Para que o seu monitoramento fique 100% limpo e sem erros de "No such file", precisamos garantir que as variáveis dentro do script apontem para o lugar exato. Com base no seu comando pwd da **image\_56f6a9.png**, o caminho correto é /home/williamsmartins/.

Vamos atualizar o script uma última vez com o comando cat:

Bash

cat << 'EOF' > ~/scripts-linux/monitoramento-logs.sh

#!/bin/bash

# Limpa a tela para o visual ficar profissional

clear

# CAMINHOS TOTAIS (Sem usar ../)

LOG\_BACKEND="/home/williamsmartins/myapp/logs/myapp-backend.log"

LOG\_FRONTEND="/home/williamsmartins/myapp/logs/myapp-frontend.log"

RELATORIO="/home/williamsmartins/myapp/logs/relatorio.txt"

# Cores

VERDE='\033[0;32m'

VERMELHO='\033[0;31m'

SEM\_COR='\033[0m'

AGORA=$(date "+%Y-%m-%d %H:%M:%S")

echo "------------------------------------------"

echo "INICIANDO MONITORAMENTO: $AGORA"

echo "------------------------------------------"

# Verifica se os arquivos existem antes de tentar ler

if [ ! -f "$LOG\_BACKEND" ] || [ ! -f "$LOG\_FRONTEND" ]; then

echo -e "${VERMELHO}ERRO: Arquivos de log não encontrados em /home/williamsmartins/myapp/logs/${SEM\_COR}"

exit 1

fi

if grep -q "SENSITIVE\_DATA" $LOG\_BACKEND $LOG\_FRONTEND; then

echo -e "${VERMELHO}ALERTA: Dados sensíveis encontrados!${SEM\_COR}"

grep "SENSITIVE\_DATA" $LOG\_BACKEND $LOG\_FRONTEND | cut -d: -f3- | tee -a $RELATORIO

echo -e "\n" >> $RELATORIO

else

echo -e "${VERDE}OK: Nenhum dado sensível detectado nesta varredura.${SEM\_COR}"

fi

echo "------------------------------------------"

EOF

**🧐 O que corrigimos agora?**

1. **Caminhos Estáticos:** Substituímos o .. pelo caminho completo. O .. significa "voltar uma pasta", mas se você rodar o script de um lugar inesperado, ele se perde. Com o caminho começando em /home/, ele funciona sempre.
2. **Verificação de Arquivo:** Adicionei um if [ ! -f ... ]. Isso faz o script checar se o arquivo de log realmente existe antes de tentar fazer a busca. Se não existir, ele te avisa com uma mensagem clara em vez do erro padrão do sistema.

**🚀 Teste o atalho**

Agora que o arquivo está salvo com o caminho fixo, tente usar o seu comando personalizado de qualquer pasta:

Bash

Monitorar

Pelas suas últimas imagens, finalmente conseguimos! A **image\_569fb4.png** mostra que o script rodou e validou que não havia dados sensíveis (mensagem em verde). No entanto, a **image\_569bd3.png** mostra um alerta em vermelho dizendo que os arquivos de log não foram encontrados no caminho específico /home/williamsmartins/myapp/logs/.

Isso aconteceu porque, no seu ambiente, a pasta myapp está diretamente na sua **Home**, mas o script está procurando dentro de uma subpasta que talvez não exista exatamente com esse nome.

**🛠️ O Ajuste Final do Caminho**

Vamos corrigir os caminhos uma última vez para que o grep encontre os arquivos. Execute este comando para atualizar o script:

Bash

cat << 'EOF' > ~/scripts-linux/monitoramento-logs.sh

#!/bin/bash

clear

# Ajustando para o caminho correto da sua Home

LOG\_BACKEND="$HOME/myapp/logs/myapp-backend.log"

LOG\_FRONTEND="$HOME/myapp/logs/myapp-frontend.log"

RELATORIO="$HOME/myapp/logs/relatorio.txt"

VERDE='\033[0;32m'

VERMELHO='\033[0;31m'

SEM\_COR='\033[0m'

AGORA=$(date "+%Y-%m-%d %H:%M:%S")

echo "------------------------------------------"

echo "INICIANDO MONITORAMENTO: $AGORA"

echo "------------------------------------------"

# Verifica se a pasta existe. Se não existir, avisa.

if [ ! -f "$LOG\_BACKEND" ]; then

echo -e "${VERMELHO}ERRO: Arquivo não encontrado em: $LOG\_BACKEND${SEM\_COR}"

exit 1

fi

if grep -q "SENSITIVE\_DATA" "$LOG\_BACKEND" "$LOG\_FRONTEND"; then

echo -e "${VERMELHO}ALERTA: Dados sensíveis encontrados!${SEM\_COR}"

grep "SENSITIVE\_DATA" "$LOG\_BACKEND" "$LOG\_FRONTEND" | cut -d: -f3- | tee -a "$RELATORIO"

echo -e "\n" >> "$RELATORIO"

else

echo -e "${VERDE}OK: Nenhum dado sensível detectado.${SEM\_COR}"

fi

echo "------------------------------------------"

EOF

**🔍 Por que usamos $HOME agora?**

* **Universalidade:** O $HOME é uma variável interna do Linux que aponta automaticamente para a sua pasta de usuário (ex: /home/williamsmartins), não importa o nome da máquina.
* **Fim dos Erros:** Isso evita que o grep diga "No such file or directory".

**🚀 Como testar se os arquivos estão no lugar certo?**

Antes de rodar o script, digite este comando no terminal para confirmar onde os logs estão:

Bash

ls -l ~/myapp/logs/

Se ele listar os arquivos myapp-backend.log e myapp-frontend.log, o script vai funcionar perfeitamente.

**🏆 Missão Cumprida!**

Agora, ao digitar monitorar, você terá um sistema que limpa a tela, verifica os logs no lugar certo e te dá um relatório limpo.

6 - Script com Alerta de Incidente   
  
Fazer o script enviar um "e-mail fictício" ou criar um arquivo de alerta especial na sua área de trabalho se ele encontrar algo perigoso.

Vamos adicionar uma "Notificação de Incidente" ao seu script. Como estamos em um ambiente de aprendizado, em vez de enviar um e-mail real (que exigiria configurar um servidor de e-mail complexo), vamos criar um **Arquivo de Alerta Crítico** que aparecerá na sua pasta de usuário sempre que algo perigoso for detectado.

Isso simula o que ferramentas de monitoramento profissionais (como o PagerDuty ou Zabbix) fazem ao detectar uma falha.

**🛠️ Script com Alerta de Incidente**

Execute o comando abaixo para atualizar seu script com essa nova funcionalidade:

Bash

cat << 'EOF' > ~/scripts-linux/monitoramento-logs.sh

#!/bin/bash

clear

# Caminhos Absolutos

LOG\_BACKEND="$HOME/myapp/logs/myapp-backend.log"

LOG\_FRONTEND="$HOME/myapp/logs/myapp-frontend.log"

RELATORIO="$HOME/myapp/logs/relatorio.txt"

ARQUIVO\_ALERTA="$HOME/ALERTA\_CRITICO.txt"

# Cores

VERDE='\033[0;32m'

VERMELHO='\033[0;31m'

SEM\_COR='\033[0m'

AGORA=$(date "+%Y-%m-%d %H:%M:%S")

echo "------------------------------------------"

echo "INICIANDO MONITORAMENTO: $AGORA"

echo "------------------------------------------"

# Verifica se os logs existem

if [ ! -f "$LOG\_BACKEND" ]; then

echo -e "${VERMELHO}ERRO: Arquivo não encontrado!${SEM\_COR}"

exit 1

fi

# Lógica de Detecção

if grep -q "SENSITIVE\_DATA" "$LOG\_BACKEND" "$LOG\_FRONTEND"; then

echo -e "${VERMELHO}!!! ALERTA DE SEGURANÇA DETECTADO !!!${SEM\_COR}"

# Cria o arquivo de Alerta de Incidente

echo "INCIDENTE DETECTADO EM: $AGORA" > "$ARQUIVO\_ALERTA"

echo "Verifique o relatório em: $RELATORIO" >> "$ARQUIVO\_ALERTA"

# Salva no relatório principal

grep "SENSITIVE\_DATA" "$LOG\_BACKEND" "$LOG\_FRONTEND" | cut -d: -f3- | tee -a "$RELATORIO"

echo -e "\n" >> "$RELATORIO"

echo -e "Um arquivo de alerta foi gerado em: $ARQUIVO\_ALERTA"

else

echo -e "${VERDE}SISTEMA SEGURO: Nenhum dado sensível detectado.${SEM\_COR}"

# Remove o alerta antigo se o sistema estiver limpo agora

rm -f "$ARQUIVO\_ALERTA"

fi

echo "------------------------------------------"

EOF

**🔍 O que esse novo código faz?**

1. **Criação de Arquivo de Alerta**: Se o grep encontrar um erro, o script cria o arquivo ALERTA\_CRITICO.txt na sua Home. Isso serve como um sinal visual forte.
2. **Auto-Limpeza**: Se você rodar o script e os logs estiverem limpos (supondo que você corrigiu o problema), o comando rm -f "$ARQUIVO\_ALERTA" apaga o alerta automaticamente.
3. **Variável $HOME**: Continuamos usando $HOME para garantir que o script funcione independente de onde o Cron ou o Alias o chamarem.

**🚀 Teste o cenário de "Ataque" e "Defesa"**

1. **Teste o Alerta**: Digite monitorar. Como seus logs atuais têm erros, ele vai criar o arquivo de alerta.
   * Confira com: ls ~ | grep ALERTA
2. **Simule a Correção**:
   * Abra o log com o nano: nano ~/myapp/logs/myapp-backend.log
   * Apague as linhas que dizem SENSITIVE\_DATA.
   * Salve e saia.
3. **Teste a Limpeza**: Digite monitorar novamente. Agora ele deve mostrar a mensagem **verde** e o arquivo ALERTA\_CRITICO.txt terá sumido!

6 - Transformar esse script em um "Painel de Controle" que fica atualizando na tela a cada 5 segundos.  
  
Para criar esse efeito de "Painel de Controle" em tempo real, vamos usar um comando fantástico do Linux chamado **watch**. Ele serve para repetir qualquer comando ou script em um intervalo de tempo determinado, limpando a tela automaticamente a cada ciclo.

**🛠️ Como transformar seu script em um Monitor em Tempo Real**

Você não precisa mudar nada dentro do seu script monitoramento-logs.sh. Vamos apenas usar o comando watch para executá-lo continuamente.

No seu terminal, digite:

Bash

watch -n 5 -c monitorar

**🔍 O que esse comando faz?**

* **watch**: É o comando que "assiste" e repete a tarefa.
* **-n 5**: Define o intervalo de **5 segundos**. Você pode mudar para 2 ou 10, se preferir.
* **-c**: (Color) Instruiu o watch a interpretar as cores (Verde e Vermelho) que configuramos no script.
* **monitorar**: É o seu **alias** (atalho) que criamos anteriormente.

**🚀 O "Show" do Monitoramento**

Agora, para ver a mágica acontecer, o ideal é você ter **duas abas** do terminal abertas (ou dois terminais lado a lado):

1. **Na aba 1**: Deixe o comando watch -n 5 -c monitorar rodando. Ele estará verde dizendo "SISTEMA SEGURO".
2. **Na aba 2**: Vamos simular um incidente inserindo um dado sensível manualmente:

Bash

echo "2026-01-22 14:00:00 ERROR: SENSITIVE\_DATA: Vazamento de teste!" >> ~/myapp/logs/myapp-backend.log

1. **Resultado**: Em no máximo 5 segundos, a **aba 1** mudará automaticamente para **VERMELHO**, exibirá o alerta e gerará o arquivo de incidente.

**🛑 Como sair do Painel?**

Para parar o monitoramento em tempo real e voltar ao terminal normal, basta apertar: **Ctrl + C**

**🏆 Você acaba de criar um SOC (Security Operations Center)**

Você tem um script que:

1. **Filtra logs** com grep.
2. **Limpa dados** com cut.
3. **Decide ações** com if/else.
4. **Notifica visualmente** com cores.
5. **Registra histórico** com tee e data.
6. **Gera alertas de arquivo** para incidentes.
7. **Roda sozinho** via cron.
8. **Atualiza em tempo real** com watch.

6.1 - Script de Limpeza

Apaga os logs de teste e os relatórios antigos para começar do zero sempre que quiser.

Manter o ambiente limpo é fundamental para que seus testes não fiquem confusos com dados antigos. Vamos criar um script chamado reset-lab.sh que apaga os logs de teste, limpa o relatório e remove qualquer alerta crítico, deixando tudo como se fosse o "primeiro dia".

**🧹 Criando o Script de Limpeza (Reset)**

Execute este comando no seu terminal:

Bash

cat << 'EOF' > ~/scripts-linux/reset-lab.sh

#!/bin/bash

# Caminhos

LOG\_DIR="$HOME/myapp/logs"

ALERTA="$HOME/ALERTA\_CRITICO.txt"

echo "------------------------------------------"

echo "LIMPANDO AMBIENTE DE MONITORAMENTO..."

echo "------------------------------------------"

# 1. Limpa o conteúdo dos arquivos de log (sem apagá-los)

# O comando > limpa o arquivo mas mantém ele existindo

> "$LOG\_DIR/myapp-backend.log"

> "$LOG\_DIR/myapp-frontend.log"

> "$LOG\_DIR/relatorio.txt"

# 2. Remove o arquivo de alerta se ele existir

rm -f "$ALERTA"

echo -e "\033[0;32mSUCESSO: Logs e relatórios zerados!\033[0m"

echo "------------------------------------------"

EOF

# Dá permissão de execução

chmod +x ~/scripts-linux/reset-lab.sh

**🛠️ Criando um Atalho para a Limpeza**

Para facilitar ainda mais, vamos criar um alias para esse comando também:

1. Abra o seu perfil: nano ~/.bashrc
2. Adicione esta linha ao final: alias limpar='~/scripts-linux/reset-lab.sh'
3. Salve (**Ctrl+O**, **Enter**) e saia (**Ctrl+X**).
4. Atualize o terminal: source ~/.bashrc

**🔄 O Fluxo de Trabalho Completo**

Agora você tem o controle total do laboratório:

* **monitorar**: Executa a verificação uma única vez.
* **watch -n 5 -c monitorar**: Abre o painel de controle em tempo real.
* **limpar**: Reseta todo o ambiente para novos testes.

**Dica Pro:** Se você rodar o limpar enquanto o watch estiver aberto em outra aba, você verá o painel mudar instantaneamente de **VERMELHO** para **VERDE**, pois os logs de erro foram apagados!

# 7 - Manual básico do Vim

[**Próxima Atividade**](https://cursos.alura.com.br/course/linux-criando-script-processamento-arquivos-logs/task/176730/next)

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;

Texto

O conteúdo gerado por IA pode estar incorreto.

Interface gráfica do usuário, Texto

O conteúdo gerado por IA pode estar incorreto.

**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:  
Texto

O conteúdo gerado por IA pode estar incorreto.

Texto

O conteúdo gerado por IA pode estar incorreto.

Interface gráfica do usuário, Aplicativo

O conteúdo gerado por IA pode estar incorreto.

Notação Simbólica

Interface gráfica do usuário

O conteúdo gerado por IA pode estar incorreto.

Interface gráfica do usuário, Aplicativo

O conteúdo gerado por IA pode estar incorreto.

CHMOD 755 no formato simbólico?  
  
Texto

O conteúdo gerado por IA pode estar incorreto.

Interface gráfica do usuário, Aplicativo

O conteúdo gerado por IA pode estar incorreto.

**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.

Tela de celular com texto preto sobre fundo branco

O conteúdo gerado por IA pode estar incorreto.

**Para saber mais: notação octal para permissões**

[**Próxima Atividade**](https://cursos.alura.com.br/course/linux-criando-script-processamento-arquivos-logs/task/176736/next)

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:

**Permiss**õ**es**: [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 |
| 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**](https://cursos.alura.com.br/course/linux-criando-script-processamento-arquivos-logs/task/176737/next)

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

Texto

O conteúdo gerado por IA pode estar incorreto.

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

Texto

O conteúdo gerado por IA pode estar incorreto.

**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

Texto

O conteúdo gerado por IA pode estar incorreto.

Nota-se que o grupo foi alterado

Comando para alterar o usuário utilizado;

Su – julia

**Testando acesso no novo usuário**

Texto

O conteúdo gerado por IA pode estar incorreto.

**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**](https://cursos.alura.com.br/course/linux-criando-script-processamento-arquivos-logs/task/176738/next)

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**

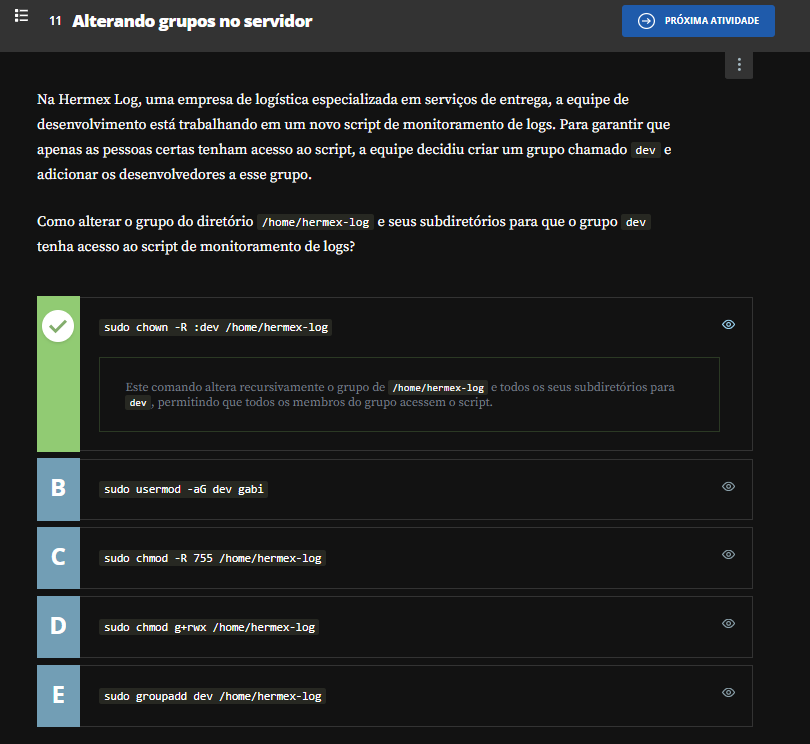
Imagine um servidor onde diferentes usuários precisam acessar uma pasta compartilhada. Ao configurar:

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**](https://cursos.alura.com.br/course/linux-criando-script-processamento-arquivos-logs/task/176743/next)

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**](https://cursos.alura.com.br/course/linux-criando-script-processamento-arquivos-logs/task/172093/next)

**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.

# 8 – Encontrando arquivos com ‘Find”

**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.

**Comando Para copiar dentro do terminal** , selecionar texto e usar as teclas control, shfth+c.

Interface gráfica do usuário, Texto

O conteúdo gerado por IA pode estar incorreto.

**Para saber mais: entendendo o laço de repetição while**

[**Próxima Atividade**](https://cursos.alura.com.br/course/linux-criando-script-processamento-arquivos-logs/task/176748/next)

O laço de repetição while em Bash é usado para executar um bloco de comandos repetidamente enquanto uma condição for verdadeira.

Ele verifica a condição antes de cada iteração e, se for verdadeira, executa o bloco de comandos dentro do loop. Quando a condição se torna falsa, o loop termina.

Sua estrutura básica é:

**while** [ condição ]; **do**

# comandos a serem executados enquanto a condição for verdadeira

**done**

Copiar código

Vamos ver um exemplo em que usamos o while para contar de 1 a 5:

#!/bin/bash

contador=1

**while** [ $contador -le 5 ]; **do**

echo "Contador: $contador"

((contador++)) # incrementa o valor de contador em 1

**done**

Copiar código

Nesse script:

* Inicializamos a variável contador com o valor 1;
* O loop while verifica se o valor de contador é menor ou igual a 5, através do operador de comparação -le. Esse operador verifica se um número é menor ou igual a outro;
* Enquanto a condição for verdadeira, o loop executa o comando echo para mostrar o valor atual de contador;
* O trecho ((contador++)) incrementa o valor de contador em 1 a cada iteração;
* O loop termina quando contador chega a 6, pois a condição se torna falsa.

A saída desse script ao executá-lo será:

**Contador: 1**

**Contador: 2**

**Contador: 3**

**Contador: 4**

**Contador: 5**

Copiar código

Você pode salvar esse script em seu computador e verificar o comportamento do laço de repetição while, caso deseje.

O while é especialmente útil quando não se sabe de antemão o número exato de repetições e quer-se continuar o loop enquanto uma condição específica é atendida.

Esse foi o caso do script de monitoramento de logs que criamos nesse curso. Não sabemos de antemão a quantidade de arquivos de logs para processar, pois podemos ter inúmeros arquivos no diretório. Assim, o uso do while foi crucial.