MARÇO 31, 2018ABRIL 1, 2018 *by* SEMANICKZ

Linux Listas de: comando, atalhos, caracteres e vim

- ANOTAÇÕES LINUX, APRENDENDO A USAR O BASH, ARTIGOS, BASH, GNU/LINUX, LAZER, CULTURA E ENTRETENIMENTO, LINHA DE COMANDO, SHELL SCRIPT, TERMINAL LINUX BASH, UNIX
- ATALHOS, COMANDO, META-CARACTERES, VIM
- DEIXE UM COMENTÁRIO

```
comando, atalhos, meta-caracteres, vim

.

Terminal: Teclas de Atalho + Função
Ctrl + a (cursor p/ início da linha)
Ctrl + e (cursor p/ fim da linha)
Ctrl + l (Limpa tela)
Ctrl + u (Apaga do cursor p/ trás)
Ctrl + k (Apaga do cursor p/ frente)
Ctrl + w (Apaga antes do cursor)
Ctrl + Shift + _ (Desfaz mudanças)
!! (Exec último cmd no hist)
```

!abc (Exec último cmd no hist começando com abc)
!n (Exec cmd de núm n no histórico)

.

LISTA DE COMANDOS:
1) pwd (Informa o nome do diretório corrente)
2) cd (Navega entre diretórios)
3) cd (Volta para a home)
4) cd . (Diretório atual)
5) cd .. (Retrocede um diretório)
6) cd - (Avança para o último diretório em que esteve)
7) ls, ls1 (Lista arquivos e diretórios)
8) ls -a (Lista diretórios, arquivos e arquivos ocultos)
9) ls -t (Lista arquivos e diretórios por data de modificação)
10) cp (Copia arquivos e diretórios)
11) mv (Move ou renomeia arquivos e diretórios)
12) ln (Estabelece ligações entre arquivos)
13) ln -s (Estabelece ligações simbólicas entre arquivos)
14) mkdir (Cria um diretório)
15) mkdir -p pasta1/sub-pasta1 (Cria um diretório e um sub-diretório)
16) mkdir ../nome-pasta-a-ser-criada (Cria pasta abaixo da pasta onde está)
17) rmdir (Remove um diretório vazio)
18) rm -f (Apaga arquivos)
19) rm -r (Apaga pastas/diretórios)
20) rm -I (Pede confirmação antes de remover)
21) file (Indica tipo de arquivo)
22) grep (Pesquisa arquivos por conteúdo)
23) wc (Conta palavras, linhas e caracteres)
24) df -h (confere espaço em disco)
25) more (ler arquivos de texto)
26) clear (limpa a tela do terminal)
27) cat (exibe o que tem dentro de um arquivo)
28) find (procura por arquivos em pastas)
29) du -h, du *, du arq.txt (informa espaço utilizado)

.

LISTA DE COMANDOS VIM:
1) Inserção (de texto) -> Tecla i
2) Comandos (manipular texto) -> Tecla Esc
3) Linha de comando (manipular arquivo) -> Tecla :
4) Visual (seleção visual de texto) -> Tecla v
5) Busca (busca de padrões de texto) -> Tecla /
6) Reposição (insersão sobrescrevendo) -> Tecla R
7) Sair e salvar - aperto a tecla Esc e digito -> :x (salva e sai)
PARA ABRIR O VIM:
1) vim nome-do-arquivo.sh (se o arquivo não existe, ele cria)
2) vim nome-do-arquivo.txt (abre o arquivo com o cursor no fim do arquivo)
3) vim nome-do-arquivo.sh.txt +9 (abre o arquivo com o cursor na linha 9)
4) vim MeuArquivoVim.txt +/frase (abre o arquivo na primeira vez que tiver a palavra frase)

.

VIM:

.

COPIAR/COLAR
1) Aperto ESC
2) Posiciono o cursor no início do texto que quero copiar
3) Digito v minúsculo
Usando as teclas de direção, marco o texto a ser copiado
4) Digito y minúsculo
5) Posiciono o cursor no ponto onde desejo colar o texto
6) Digito p minúsculo

.

RECORTAR/COLAR
1) ESC
2) Posicione o cursor no início do texto que quer recortar
3) Digite v minúsculo
4) Usando as teclas de direção, marque o texto a ser recortado
5) Digite d minúsculo
6) Posicione o cursor no ponto onde deseja colar o texto
7) Digite p minúsculo

.

____

Linux comandos e shell script 2018

.

Para usar sistemas operacionais Linux, não é necessário saber nada do que está escrito aqui.

.

Vou aprender a usar o terminal do Linux e shell scripts. Este é o motivo de executar este artigo.

.

"Para algumas pessoas o Linux não tem como objetivo lucro. Para estes, o objetivo do GNU/Linux é garantir a nossa liberdade. Trabalho colaborativo, produz muitos softwares que funcionam bem e estão disponíveis gratuitamente a todo o mundo.

.

Comandos Linux e Shell Script

.

Este artigo é para iniciantes (EU). Para quem não sabe nada. Pretende oferecer o máximo de informação possível em pouco tempo, para que a pessoa ao terminar de EXECUTAR este artigo, entenda bastante coisa sobre Linux, use o terminal fluentemente e saiba escrever e executar Shell Scripts. Escrevi este artigo para meu uso pessoal (memória ruim). Tudo que tem neste artigo executei por minha conta e risco sem problemas. Tudo foi bem. Recomendo que faça o mesmo (por sua conta e risco).

.

VAMOS LÁ!

APRENDENDO COMANDOS LINUX E SHELL SCRIPT

.

Enquanto eu aguentar digitar comandos, digito. Quando cansar ai sim, copio e colo.

.

Programas CLI, são programas em modo texto. Por exemplo programas cli podem interagir com o terminal. Um programa cli bastante usado no Linux é o Dialog por exemplo.

.

FLUXOS DE ENTRADA/SAÍDA PADRÃO E OPERADORES DE FLUXO

.

Os fluxos padrão são canais de entrada/saída (E/S) entre um programa de computador e o ambiente (tipicamente um terminal de texto) que são pré conectados no início da execução.

.

A entrada padrão Stdin do Linux é o teclado. A gente se comunica com o sistema operacional e programas usando o teclado. Claro que podemos usar tela sensível ao toque, mouse, joystick, toutchpad, etc...

Stdin = Standart Input

É representada pelo número 0.

.

Abro meu terminal pelo menu do sistema. Digito/Copio e colo no meu terminal:

ls /dev/stdin ; echo Stdin = Standart Input - valor = 0

.

A saída padrão Stdout do Linux é o monitor. Quase sempre as informações

mais importantes são obtidas pelo terminal do sistema operacional. A gente recebe informação do sistema operacional e programas pelo monitor.

Stdin = Standart Output

É representada pelo número 1.

.

Copio e colo no meu terminal:

ls /dev/stdout

.

A saída de erro padrão Stderr do Linux é o monitor. Por ele são enviadas mensagens de erro geradas por aplicativos. A gente recebe informação de erro do sistema operacional e programas pelo monitor. Estas informações são muito úteis para podermos corrigir problemas. Quase sempre as informações de erro mais importantes são obtidas pelo terminal do sistema operacional.

Stderr = Standart Error

É representada pelo número 2.

.

Copio e colo no meu terminal:

ls /dev/stderr

.

Os 3 são usados por redirecionadores. Você redireciona a saída deles para um arquivo de texto por exemplo. Podemos redirecionar a saída de um comando para um arquivo em determinado diretório. Então lembre que:

1) Stdin=0

2) Stout=1

3) Stderr=2

.

Podemos manipular as saídas e entradas com 3 operadores.

O Pipe (|) Liga a saída de um programa a entrada de outro.

O Write (>) Redireciona para arquivo de texto por exemplo sobrescrevendo.

O Append (>>) Redireciona para arquivo de texto por exemplo, sem apagar o que já estiver escrito.

OS PROMPTS PADRÃO DO LINUX SÃO:

.

1) Para root:

- # -

2) Para os outros usuários:

- $ -

.

Exemplo:

~$ echo -e "\nBem Vindo(a) \n \na \n \nComandos Linux \n \ne \n \nShell Script \n"

.

SISTEMA DE PERMISSÃO DE ARQUIVOS

.

Todo arquivo possui uma dada permissão para cada usuário.

Leitura (r), escrita (w) e execução (x)

.

TUDO É ARQUIVO:

Pasta é arquivo.
Dispositivo é arquivo.
Usuário é arquivo.

.

Temos arquivos de texto, arquivos de áudio, arquivos de vídeo, arquivos de imagem, etc.

.

O SHELL SCRIPT É O MAIS ALTO NÍVEL QUE UM SISTEMA LINUX COMUM POSSUI.

.

O Shell padrão do Linux é o Bash. Quando o usuário digita comandos no terminal eles são interpretados pelo Bash.

.

Se o usuário escreve um arquivo de texto com comandos da maneira correta este arquivo de texto se torna um shell script.

.

O shell script é bastante usado pelas pessoas para a automação de tarefas. É muito perigoso executar Shell Scripts como root, isto pode danificar o sistema.

.

O MELHOR TERMINAL PARA USAR É O QUE JÁ VEM INSTALADO NO TEU SISTEMA. MAS EXISTEM OUTROS TERMINAIS QUE PODE USAR. TEM O TILIX, TEM O TERMINATOR POR EXEMPLO.

USAR O TERMINAL CHAMADO TERMINATOR

.

Tem um terminal legal para estudar comandos e shell script Linux chamado TERMINATOR.
Ele pode ser dividido horizontalmente (Ctrl+Shift+O) e verticalmente (Ctrl+Shift+E).
Pode redimensionar as divisões do terminator com (Ctrl+Shift+seta).
Para mover de uma janela de terminal para outra (Ctrl+Shift+N) (Ctrl+Tab).
Para fechar o terminal (Ctrl+Shift+W).
Procuro o TERMINATOR pelo software center da minha distro. Instalo. Abro ele pelo menu do sistema.
Com o TERMINATOR posso ler este artigo em uma subdivisão do terminal e usar outras para executar as instruções do artigo por exemplo.
Isto porque é só copiar este texto para um editor de texto salvar como estudo-linux.txt e abrir ele no terminal.

TERMINATOR EXEMPLO DE USO

.

APERTO AS TECLAS:

1) Ctrl+Shift+O
2) Ctrl+Shift+E
3) Ctrl+Shift+N
4) Ctrl+Shift+Seta para esquerda/ direita/ cima/ baixo
5) Ctrl+Shift+W

.

Para copiar no terminal: Shift Ctrl C

Para selecionar tudo: Shift Ctrl A (depois seta para cima)

.

Para pesquisar:

1) Shift Ctrl F (localizar)

2) Shift Ctrl G (anterior)
3) Shift Ctrl H (próximo)

.

Para ampliar: Ctrl +
Para reduzir: Ctrl -
Tamanho normal: Ctrl 0
Tela cheia: F11

.

Para colar no terminal: Shift Ctrl V

.

Após escrever comandos no meu terminal, eles são examinados pelo Shell, que no meu caso é o Bash. Após o exame do Shell, os comandos informados são passados para o Linux, que executa o comando ou exibe uma mensagem de erro que ajuda quando for executar o comando correto.

.

COMANDOS DO LINUX TEM QUASE SEMPRE A SEGUINTE FORMA:

.

comando [-opções] [argumentos]

Exemplo:

ls -t /Downloads/pasta_teste

.

comando [argumento]

Exemplo:

cat /Downloads/pasta_teste/arquivo1.txt

.

comando [- opções]

Exemplo:

ps -aux

.

O que é um argumento?

Um argumento é uma informação extra, como o nome ou tamanho de um arquivo a ser encontrado por um comando.

.

O que é parâmetro? É o mesmo que opção. Um comando quase sempre, tem como opção vários parâmetros. Por exemplo o parâmetro de ajuda do comando ls é --help. Fica assim:

ls --help (o ls tem várias opções)

.

Para me confundir menos, por enquanto, comandos Linux tem opções e argumentos.

.

Primeiro preciso de um local no sistema para estudar para não se misturar com meus outros arquivos. Abro o terminal pelo menu do sistema e colo:

cd ; cd Downloads/ ; mkdir EstudoEmDown ; cd EstudoEmDown/ ; clear

.

Agora estou no local (cd ; cd Downloads/EstudoEmDown/) que daqui para frente, será o lugar onde posso estudar comandos e shell script. Pelo menos por enquanto. Beleza.

.

Agora vou ler este artigo e executar comandos no terminal. Vamos?

.

1) Digito no terminal: whoami
2) Aperto a tecla Enter.
3) Digito no terminal: cd
4) Aperto a tecla Enter.
5) Digito no terminal: cd -
6) Digito no terminal: clear

.

Uso o exemplo abaixo para digitar no terminal:

ls -tli /home/seu_user_é_whoami/Downloads/

ls -tli ~/Downloads/

ls -t /home/seu_user_é_whoami/Downloads/

ls -t ~/Downloads/

.

No exemplo acima:

1) ls (É o comando.)
2) -tli (É a opção.)
3) /home/seu_user_é_whoami/Downloads/ (É o argumento)

.

comando + opção + argumento

.

O QUE SÃO STRINGS?

.

"Expressão contendo qualquer caracter alfanumérico, formando ou não palavras."

.

São cordas. Longas cadeias de comandos. Strings possuem comandos, opções, argumentos, metacaracteres, expressões regulares, pipes e palavras. Podem ser executadas no terminal, mas a casa delas é nos shell scripts.
Strings creio que são palavras dentro de um arquivo. Comandos, códigos, palavras executadas em um terminal.

Dentro de um arquivo existem carcteres, palavras, frases e parágrafos (strings).

Podemos digitar códigos/palavras/comandos (strings) no terminal também.

As palavras podem ser "palavras exatas" ou "expressões regulares".

As strings podem conter "comandos" "opções" "argumentos" "palavras exatas" ou "expressões regulares".

.

Executo no terminal o exemplo abaixo:

1)

STRING="Isso ===== é ===== uma STRING!"

2)

echo 'Quer saber o que é uma string?' ; echo $STRING

3)

unset STRING

4)

clear

5)

exit

.

O QUE É EXPRESSÃO REGULAR?

Expressões regulares acho que são maneiras de usar curingas e caracteres especiais importantes no Bash que é o interpretador de comandos padrão no terminal do Linux.

.

O QUE É SED?

Sed creio que é um editor de fluxo para os sistemas operacionais Unix e Linux. Ele pode ser utilizado para manipular o texto de várias maneiras, incluindo a procurar e substituindo cadeias encontradas em ficheiros de texto.

.

O formato básico para a substituição com sed é:

sed -i ‘s/FIND_STRING/REPLACE_STRING/g’

sed -i "s/encontrar_string/substituir_string/g" nome_do_arquivo

sed -i "s/find_string/replace_string/g" filename

.

Explicação:

O "-i" opção diz para editar a linha de texto.

Em outras palavras, o arquivo é editado sem a criação de um segundo arquivo.

A seção "s/find_string/replace_string/g", diz para substituir ("s"), o "replace_string" para o " find_string" globalmente ("g").

A última parte do comando ("filename") é o nome do arquivo a ser editado.

.

Se você quiser guardar uma cópia do arquivo original, pode omitir o "-i" opção e especificar um arquivo de saída como:

sed "s/find_string/replace_string/g" filename > saída

.

Se omitir o "g" do comando , o sed irá substituir apenas a primeira ocorrência do "find_string" e parar.

.

O QUE É PERL?

Perl penso ser uma linguagem de programação que pode ser usado a partir da linha de comando para manipular arquivos de texto.

linha de comando para manipular arquivos de texto.

Perl foi criado para manipular relatórios de texto ASCII de forma eficiente.

O método de Perl para procurar e substituir as cordas é semelhante ao comando sed. O formato é o seguinte:

Perl- pi -e 's/find_string/replace_string/g' nome-do-arquivo

.

EXECUTO OS EXEMPLOS ABAIXO PARA ENTENDER COMO OBTER AJUDA:

1) man --help

2) man grep (para sair aperto a tecla Q)

3) apropos mkdir

4) whatis bash

5) whereis sh

.

QUERO SABER A VERSÃO DA DISTRIBUIÇÃO DIGITO:

.

1) lsb_release -a
2) cat /etc/issue
3) reset

.

ME INFORMO SOBRE O KERNEL E ARQUITETURA DA MÁQUINA (faço sempre que posso pra eu não esquecer):

.

1) uname -a
2) uname -r
3) arch

.

EU QUERO APRENDER ATALHOS DE TERMINAL DO GNU/LINUX ENTÃO EU LEIO O TEXTO ABAIXO:

.

Teclas de Atalho + Função

Ctrl + a (Move o cursor para o início da linha)

Ctrl + e (Move o cursor para o final da linha)

Ctrl + l (Limpa a tela, semelhante ao comando clear)

Ctrl + u (Apaga do cursor para trás)

Ctrl + k (Apaga do cursor para frente)

Ctrl + w (Apaga uma palavra antes do cursor)

Ctrl + Shift + _ (Desfaz as últimas mudanças)

!! (Executa o último comando no histórico)

!abc (Executa último comando no histórico começando com abc)

!n (Executa comando de número n no histórico)

.

PARA APRENDER O QUE LI ACIMA, EXECUTO O ENSINAMENTO ABAIXO:

.

Abro o terminal pelo menu do sistema.

.

Colo o texto abaixo no terminal, mas poderia ser qualquer frase:

.

Frase: "Ctrl + Apaga do cursor ao início da linha."

.

Agora aperto duas teclas, as teclas:

Ctrl + A

.

Aperto a seta do teclado para a direita quatro vezes. Então aperto as teclas:

.

Ctrl + U

.

Aperto duas teclas, as teclas Ctrl + E

(Control e E)

.

Aperto a tecla seta para esquerda 7 vezes.

.

Aperto as teclas Ctrl + K

.

Aperto as teclas Ctrl + W

.

Aperto 3 teclas: Crtl + Shift + _

(Control, Shift e Underline.)

.

Aperto a tecla seta para cima algumas vezes.

.

Aperto a tecla seta para baixo algumas vezes.

.

Pronto. Já estou sabendo alguma coisa.

Com isto, já aprendi bastante sobre atalhos de teclado para o terminal Linux.

Aperto Ctrl+U e digito: history (aperto a tecla Enter).

Digito: !w (aperto a tecla Enter).

Digito: !whe (aperto a tecla Enter).

Digito: !who (aperto a tecla Enter).

Digito: !ar (aperto a tecla Enter).

Digito: !4 (aperto a tecla Enter).

Digito: !5 (aperto a tecla Enter).

.

Até Agora Tudo Beleza!!!

.

RECORDO QUE:

Terminal: Teclas de Atalho + Função
Ctrl + a (cursor p/ início da linha)
Ctrl + e (cursor p/ fim da linha)
Ctrl + l (Limpa tela)

Ctrl + u (Apaga do cursor p/ tras)
Ctrl + k (Apaga do cursor p/ frente)
Ctrl + w (Apaga antes do cursor)
Ctrl + Shift + _ (Desfaz mudanças)
!! (Exec último cmd no hist)
!abc (Exec último cmd no hist começando com abc)
!n (Exec cmd de núm n no histórico)

.

O PIPE "|"

.

O pipe "|" é um operador que liga um comando a outro comando. Liga a saída de um comando a entrada de outro comando.

.

Lembro que eu criei um local de estudo em Downloads com o comando abaixo:

```
cd ; cd Downloads/ ; mkdir EstudoEmDown
```

.

Para navegar até ela, uso o comando abaixo:

```
cd ; cd Downloads/EstudoEmDown
```

.

OBS:
Quando for estudar comandos e shell script abro o terminal na pasta EstudoEmDown.

.

LER UM ARQUIVO DE TEXTO PELO TERMINAL DO LINUX

.

Copio e colo o código abaixo no meu terminal Linux:

```
echo -e '\nQuando a luz dos olhos meus\nE a luz dos olhos teus\nResolvem se encontrar\nAi que bom que isso é meu Deus\nQue frio que me dá o encontro desse olhar\nMas se a luz dos olhos teus\nResiste aos olhos meus só para me provocar\nMeu amor, juro por Deus me sinto incendiar\nMeu amor, juro por Deus\nQue a luz dos olhos meus já não pode esperar\nQuero a luz dos olhos meus\nNa luz dos olhos teus sem mais lará-lará\nPela luz dos olhos teus\nEu acho meu amor que são se pode achar\nQue a luz dos olhos meus precisa se casar.\n' > arq-poema.txt
```

.

Para ler o arq-poema.txt todo pelo terminal copio e colo os comandos abaixo no meu terminal Linux um de cada vez:

abaixo no meu terminal Linux um de cada vez:

1) cat arq-poema.txt

2) cat -n arq-poema.txt

.

Para ler linhas do poema. Executo:

1) cat arq-poema.txt | head -n 3

2) cat arq-poema.txt | head -n 5

3) cat -n arq-poema.txt | head -n 3

4) cat -n arq-poema.txt | head -n 5

5) clear

.

Para ler as três ultimas linhas do poema:

1) cat arq-poema.txt | tail -n 3

2) cat -n arq-poema.txt | tail -n 3

.

Será que nas quatro primeiras linhas existe a palavra olhos? Vamos ver.
Copio e colo o comando abaixo no terminal:

1) cat arq-poema.txt | head -n 4 | grep olhos

2) cat -n arq-poema.txt | head -n 4 | grep olhos

.

Quais das 7 primeiras linhas que NÃO POSSUEM a palavra olhos?

1) cat arq-poema.txt | head -n 7 | grep -v olhos

2) cat -n arq-poema.txt | head -n 7 | grep -v olhos

.

Crio um arquivo comprimido. Um arquivo tar. com o comando:

tar cvf arq-poema.tar arq-poema.txt

ls -t

.

Extraio o arquivo txt do arquivo tar que criei:

tar xvf arq-poema.tar

.

Vejo o arquivo tar existente:

tar tvf arq-poema.tar

.

Para remover arq-poema.txt uso o comando abaixo no mesmo terminal em que está executando os comandos até o momento:

1) rm -f arq-poema.txt (não removo)

.

ACHOU INTERESSANTE ATÉ AGORA?

.

ENTÃO SIGAMOS EM FRENTE!

.

Copio e colo no terminal o texto abaixo. Vai criar um arquivo de texto onde vou repetir os comandos que já aprendi e mais adiante, alguns que irei aprender. Aperto Enter:

echo -e '\nEntão queres ser um escritor? \nSe não sair de ti explodindo apesar de tudo, não o faças. \nA menos que saia sem perguntar do teu coração e da tua cabeça e da tua boca e das tuas entranhas, não o faças. \nSe tens que sentar por horas olhando a tela do teu computador ou curvado sobre a tua máquina de escrever procurando palavras, não o faças. \nSe o fazes por dinheiro ou fama, não o faças. \nSe o fazes porque queres mulheres na tua cama, não o faças. \nSe tens que te sentar e reescrever uma e outra vez, não o faças. \nSe dá trabalho só pensar em fazê-lo, não o faças. \nSe tentas escrever como algum outro escreveu, não o faças. \nSe tens que esperar para que saia de ti a gritar, então espera pacientemente. \nSe nunca sair de ti a gritar, faz outra coisa. \nSe tens que o ler primeiro à tua mulher ou namorada ou namorado ou pais ou a quem quer que seja, não estás pronto. \nNão sejas como muitos escritores, não sejas como milhares de pessoas que se consideram escritores, não sejas estúpido nem enfadonho e pedante, não te consumas com auto-devoção. \nAs bibliotecas de todo o mundo têm bocejado até adormecer com os da tua espécie. \nNão sejas mais um. \nNão o faças. \nA menos que saia da tua alma como um míssil, a menos que o estar parado te leve à loucura ou ao suicídio ou homicídio, não o faças. \nA menos que o sol dentro de ti te esteja a queimar as tripas, não o faças. \nQuando chegar mesmo a altura, e se foste escolhido, vai acontecer por si só e continuará a acontecer até que tu morras ou morra em ti. \nNão há outra forma. \nE nunca houve. \n \n(Charles Bukowski)\n' > charles_texto-escritor.txt

.

Executo o comando abaixo para ler o arquivo criado:

cat charles_texto-escritor.txt

.

Para criar uma pasta pelo terminal uso o comando mkdir. Executo:

mkdir pasta-teste

.

Para criar um arquivo uso o direcionador write (>). Executo:

Executo:

> arquivo-teste.txt

ls -t

.

Poderia criar um arquivo com o direcionador append (>>).

Executo:

>> arq-test5.txt

ls -t

.

Pego este texto:

.

Quando eu estava subindo a escada,
Encontrei um homem que não estava lá.
Ele não estava lá hoje também.
Ah, como eu queria que ele fosse embora!

--Willian Hughes Mearns.

.

Executo:

echo -e '\nQuando eu estava subindo a escada, \nEncontrei um homem que não estava lá. \nEle não estava lá hoje também. \nAh, como eu queria que ele fosse embora!\n \n --Willian Hughes Mearns\n' > arq_nao_estava.txt >> arquivo-teste.txt

ls -t

.

Leio o texto com o comando:

cat arq_nao_estava.txt

cat arquivo-teste.txt

.

Conto palavras:

wc -w arq_nao_estava.txt

wc -w arquivo-teste.txt

.

Para ver arquivos criados e me localizar digito:

ls -t

pwd

ls -l -i

ls -at ~/

pwd ~/

ls -t ~/Downloads/

clear

pwd ~/Downloads/

.

Para mover o arquivo-teste.txt para pasta-teste uso o comando mv:

mv arquivo-teste.txt pasta-teste

.

Para navegar para a pasta-teste uso o comando cd:

cd pasta-teste/

.

Para ver o que tem na pasta-teste uso o comando ls -t:

ls -t

.

Para voltar uma pasta atrás uso o comando cd ..:

cd ..

.

Confiro:

ls -t

.

Navego para a pasta anterior:

cd -

.

Para remover a pasta-teste uso o comando rm -r:

rm -r pasta-teste (mas não removo)

.

Se removi confiro se a pasta foi removida:

ls -t

.

EXECUTANDO MAIS COMANDOS ÚTEIS PELO TERMINAL (É comum repetir comandos já executados. Vou me acostumar.)

.

VERIFICO INFORMAÇÕES DA MINHA CPU:

cat /proc/cpuinfo

.

Me lembro que o TERMINAL do Linux tem SENSIBILIDADE a letras MAIÚSCULAS e MINÚSCULAS. Um arquivo chamado Texto.txt é diferente de texto.txt

.

VERIFICO INFORMAÇÕES SOBRE A MEMÓRIA:

cat /proc/meminfo

.

Quero saber que dia é hoje para o terminal, digito:

1) date

2) d

.

Quero saber quando cairá certo dia em outro ano anterior ou posterior usando pipe "|" e "less":

1) cal 2017 | less (para sair aperto a tecla Q)

2) cal 2018 | less (aperto: / digito: do aperto: Enter)

2) cal 2018 | less (aperto: 7, digito: do, aperto: Enter)

3) cal 2019 | less

.

Aperto a tecla Q para sair do calendário. O pipe "|" serve para unir um comando a outro. No GNU/Linux posso unir programas que trabalharão juntos.

.

Se quiser mandar o calendário para um arquivo de texto para depois imprimir este arquivo de texto em um folha de papel:

1) cal 2018 | less > calendario-de-2018.txt

2) ls -t

3) cat calendario-de-2018.txt | less

.

EM QUAL DIA DA SEMANA CAIRÁ O DIA DAS CRIANÇAS?

date --date='12 Oct' +%a

.

EM QUAL DIA DA SEMANA CAIRÁ O DIA DAS CRIANÇAS ANO QUE VEM?

date --date='12 Oct 1 year' +%a

.

QUE DIA FOI ONTEM?

date --date='1 day ago'

.

QUE DIA SERÁ AMANHÃ?

date --date='1 day'

.

A PARTIR DESTE DIA, DAQUI A UM ANO UM MÊS E UM DIA, QUE DIA SERÁ?

date --date='1 day 1 month 1 year'

.

QUE DIA FOI A UM ANO UM MÊS E UM DIA?

date --date='1 day 1 month 1 year ago'

.

PARA SABER INFORMAÇÃO SOBRE O SISTEMA E O HARDWARE:

PARA SABER INFORMAÇÃO SOBRE O SISTEMA E O HARDWARE:

(os comandos que começarem com sudo, pedirão senha então caso não queira, pule eles.)

.

Info CPU

cat /proc/cpuinfo

.

Info memória

cat /proc/meminfo

.

Detalhes da versão

cat /proc/version

.

Detalhes da partição

cat /proc/partitions

.

LER O .BASHRC E .BASH_HISTORY

1) cat ~/.bashrc

2) cat ~/.bash_history

.

Detalhes dispositivos SCSI/Sata

cat /proc/scsi/scsi

.

Info dispositivos SATA

hdparam /dev/sda1

.

Lista componentes do Hardware

sudo lshw

.

Imprime info do hardware

sudo hwinfo --short

.

Lista dispositivos scsi

sudo lsscsi

.

Lista todos os dispositivos PCI

lspci

.

Lista dispositivos USB

lsusb

.

Lista dispositivos de bloco

lsblk

.

Mostra informação sobre a arquitetura da CPU

lscpu

.

COMANDOS PARA MANIPULAÇÃO DE ARQUIVOS E DIRETÓRIOS:

.

LISTA DE COMANDOS:
1) pwd (Informa o nome do diretório corrente)
2) cd (Navega entre diretórios)
3) cd (Volta para a home)
4) cd . (Diretório atual)
5) cd .. (Retrocede um diretório)
6) cd - (Avança para o último diretório em que esteve)
7) ls, ls1 (Lista arquivos e diretórios)
8) ls -a (Lista diretórios, arquivos e arquivos ocultos)
9) ls -t (Lista arquivos e diretórios por data de modificação)
10) cp (Copia arquivos e diretórios)
11) mv (Move ou renomeia arquivos e diretórios)
12) ln (Estabelece ligações entre arquivos)
13) ln -s (Estabelece ligações simbólicas entre arquivos)
14) mkdir (Cria um diretório)
15) mkdir -p pasta1/sub-pasta1 (Cria um diretório e um sub-diretório)
16) mkdir ../nome-pasta-a-ser-criada (Cria pasta abaixo da pasta onde está)

17) rmdir (Remove um diretório vazio)
18) rm -f (Apaga arquivos)
19) rm -r (Apaga pastas/diretórios)
20) rm -I (Pede confirmação antes de remover)
21) file (Indica tipo de arquivo)
22) grep (Pesquisa arquivos por conteúdo)
23) wc (Conta palavras, linhas e caracteres)
24) df -h (confere espaço em disco)
25) more (ler arquivos de texto)
26) clear (limpa a tela do terminal)
27) cat (exibe o que tem dentro de um arquivo)
28) find (procura por arquivos em pastas)
29) du -h, du *, du arq.txt (informa espaço utilizado)

#

OBS:

Grep pesquisa nos arquivos de entrada (ou na entrada padrão caso nenhum arquivo seja informado ou o nome do arquivos seja igual a -), por linhas que contenham o padrão informado. Por padrão, grep lista as linhas coincidentes. O pacote "grep" instala: grep, egrep e fgrep.

.

(comandos para saber sobre o grep usando o terminal)

1) man grep
2) grep --help
3) info grep
4) man egrep
5) man fgrep
6) man pgrep

#

Exemplos:

1) grep palavra-que-procura nome-do-arquivo.txt (deve estar com o terminal aberto onde está o arquivo-de-texto.txt)

2) grep -i palavra-que-procura arquivo-de-texto.txt

3) grep -r palavra-que-procura ~/Downloads/

.

Executo:

1) grep não arq-poema.txt

2) grep -r teus ~/Downloads/

3) grep -n root /etc/passwd

4) grep -v bash /etc/passwd | grep -v nologin

4) grep -v bash /etc/passwd | grep -v nologin

5) grep -c false /etc/passwd

.

Exemplos:

grep -i arquivo-que-procura caminho/para/pasta

grep -i aluno arq-poema.txt /home/Downloads/EstudoEmDown

grep "frase que procuro em minusculas" -r /home/seu-usuario-whoami/nome-da-pasta-onde-esta-o-arquivo/ (Procurar uma frase em todos os arquivos de um diretório)

.

Executo:

grep -r teus arq-poema.txt

.

Exemplo:

grep -i "frase que procuro em minusculas" -r /home/seu-usuario-whoami/nome-da-pasta-onde-esta-o-arquivo/

.

Executo:

grep -i "Mas se a luz dos olhos teus" -r

.

Executo:

1) echo 'Linux Unix linux unix festa mais menos FESTA MAIS MENOS escritor bibliotecas resolvem teus faça não Aluno8 Carlos Charles Mariana nano gedit NaNo GediT universo festa mais teus menos sonda nave tempestades biBliOtecAs Sonda navE' >> arq-grep-palavras.txt

2) grep -lr "nano"

3) grep -Lr "universo"

4) grep -r --color=always "teus"

5) grep "tempestades" arq-grep-palavras.txt

6) grep -E "unix|nave" arq-grep-palavras.txt

7) grep -E --color=always "unix|nave" arq-grep-palavras.txt

#

OBS:
Para procurar em um arquivo as linhas que contenham uma palavra OU outra palavra deve estar com o terminal aberto onde está o arquivo que contém a palavra.

Exemplo:

egrep '(palavra_um|palavra2)' nome-do-arquivo

.

Executo:

egrep '(olhos|luz)' arq-poema.txt

egrep '(mais|nave)' arq-grep-palavras.txt

clear

.

COMANDO FGREP:

.

"O fgrep retorna as ocorrências de palavras simples na linha de comando."

.

Executo:

1) fgrep "universo" arq-grep-palavras.txt

2) fgrep -i "nano" arq-grep-palavras.txt

3) fgrep -c "nano" arq-grep-palavras.txt

4) fgrep -n "sonda" arq-grep-palavras.txt

5) clear

.

COMANDO EGREP:

"O egrep, por padrão, reconhece e utiliza expressões regulares simples e estendidas."

.

Executo:

1) egrep "mais" arq-grep-palavras.txt

2) egrep -i "nano" arq-grep-palavras.txt

3) egrep -c "menos" arq-grep-palavras.txt

4) egrep -n "universo" arq-grep-palavras.txt

5) clear

#

O COMANDO FIND:

"find (Localiza arquivo por suas características)"

.

Exemplos:

find ./ (exibe os arquivos existentes na pasta onde está)

find -name "palavra-termo-de-busca"

find -iname "palavra-termo-de-busca"

.

Exemplo:

find -name nome-do-arquivo -exec rm {} \; (Executa o comando cmd .

A finalidade do comando é considerada encerrada quando um ponto e vírgula (;) é encontrado. A cadeia {} é substituída pelo nome de cada arquivo que satisfaz ao critério de pesquisa e a linha assim formada é executada. Assim como foi dito para a opção - name, o ponto e vírgula (;)deve ter antes uma contrabarra, ou deve estar entre aspas ou apóstrofos)

.

Executo (faço o teste pelo terminal):

touch 01-arquivo.txt

ls -t

find -name 01-arquivo.txt -exec rm {} \;

ls -t

.

Executo:

> 01-arqvo.txt

echo 'Locate, Find, Estudo, Linux, pipe, casar, anos, Casar, Zuenir, Xerox, Caesar, caesar, Aluno5, um' > 01-arqvo.txt

cat 01-arqvo.txt

find 01-arqvo.txt

.

"O find é para realizar buscas por arquivos e diretórios, mas também, através da utilização de diversos parâmetros adicionais, serve para realizar outras operações, em conjunto."

.

Executo os comandos abaixo:

.

1)

```
cd ; mkdir ~/Downloads/EstudoEmDown/
```

2)

```
cd ; mkdir ~/Downloads/EstudoEmDown/pasta-find/ ; cd ~/Downloads/EstudoEmDown/pasta-find/ ; echo -ne 'amore \njuros \nencontrar \nbola \nbibliOteCAs \nalunos6 \nescritor \nAlunos6 \nfalantes \nmarina \nluz \nCharles \nalto \nfeira \nescritor \nMoreira \nZuenir \nfaça \nolhos \nescada \nluz \nhomem \nfalantes \nfaça \nesfinge \nporta \nEscada \nmartelo \nAluno6 \nescritor \ncasar \nvulcano \nalto \nporta \nMariana \nfeira \nmarina \nmarina \nAluNos6 \nalto \nLuz \nisto \nescritor \nfeira \nporta \nFeira \nEsfinge \nporta \nPorta \nluz \nmoreira \ncasar \nAna \nCharles \nana \ncharles \nbibliotecas \nMartelo \nAlto \nFeira \nhomem \nEsfinge \nOlhos \nMartelo \namor' > arq-find.txt
```

.

Executo:

1) `find . -name "arq-find.txt"`

2) `find ~/Downloads/EstudoEmDown/pasta-find/ "arq-find.txt"`

3) `find . -iname "aRq-fInD.txt"`

4) `find . -iname "aRq-fInD.txt" -exec ls -lt {} \;`

5) `find . -iname Arq-FiNd.txt -print0 | xargs -0 ls -til`

6) `find . -name "*.txt" -print0 | xargs -0 grep -Hin "amor" {} \;`

7) `find . -name "*.txt" -print0 | xargs -0 grep -Hin "Esfinge" {} \;`

8) `find . -name "*.txt" -mtime -2 -exec grep -Hin --color=always "ana" {} \;`

9) `find . -name "*.txt" -mtime -2 -exec grep -Hin --color=always "escada" {} \;`

10) `find . -name "*.txt" -mtime -2 -exec grep -Hin --color=always "arcano" {} \;`

11) `find . -name "*.txt" -mtime -2 -exec grep -Hin --color=always "luz" {}`

\;

12) find . -name "*.txt" -mtime -2 -exec grep -Hin --color=always "marciano" {} \;

13) find . -name "*.txt" -exec grep -l "faça" {} \;

14) ls -tli ~/Downloads/EstudoEmDown/pasta-find

15) find . -type f -print0 | xargs -0 ls -l

16) find . -maxdepth 1 -print0 | xargs -0 ls -l

17) find . -maxdepth 1 -type f -exec ls -l {} \; | less

.

OBS:
Aperto a tecla Q

.

18) find ~/Downloads/ mais

.

OBS:
Executar o comando 19 com cautela/cuidado.

.

19) find . -maxdepth 1 -type f -exec rm -f {} \;

20) find ~/Downloads -type f -atime -5 (Procura arquivos acessados há mais de 5 horas.

.

COMANDO BASENAME:

.

basename nome-do-arquivo (Devolve o nome de um arquivo recebendo o caminho completo)

Exemplo:

~ $basename Documentos/
Documentos

.

COMANDO DIRNAME:

dirname nome-do-diretório (Devolve o nome do diretório recebendo o caminho completo)

.

Executo:

> nome-do-arquivo.extensão (Cria arquivo vazio)

touch nome-do-arquivo.extensão1 nome-do-arquivo2.extensão (Cria um ou mais arquivos vazios)

cat nome-do-arquivo.extensão > nome-do-arquivo.extensão (Cria arquivo)

ls -alit

ls -l

cat nome-do-arquivo.extensão

cd ~

pwd

cd -

pwd

.

Executo:

echo -e 'Linux Olhar nada Faça Tudo Nada\n nada isso isto perfil luz \nUnix Festa Sonia amos Nada \ntudo Carlos martelo nada tordos \npoente perfil Aluno1 Maria nada \nluz Sonia Tudor nada Charles \nlinux UNIX unix LINUX nada \nhomem escada tambem lá estava nada' > arq-faz-de-conta.sh

cat arq-faz-de-conta.sh

.

REPETINDO:

1) cd - Navegando entre pastas/diretórios

2) cd . (pasta atual)

3) cd .. (pasta anterior)

4) cd ~ (pasta home do usuário)

5) cd - (última pasta)

.

COMANDO LS

ls - Lista arquivos e pastas

.

ls [opções] [arquivo/diretório/pasta]

.

Executo:

ls -l

.

ls -t

.

ls *.txt

(asterisco representa qualquer coisa ligada a .txt)

.

ls *.sh

(asterisco representa qualquer coisa ligada a .sh)

.

ls -lat

.

ls -lah

.

ls ?????????.txt (procura arquivos .txt com 9 caracteres)

(o ponto de interrogação substitui um e somente um caractere)

.

ls ????*.sh

.

ls [at]*.mp3

(listar todos os arquivos começados por a ou t, seguido por qualquer coisa * e terminados por .mp3).

.

Exemplo:

$ls [at]*.mp3
alcione_ne_me_quitte_pas.mp3 alex_cohen_quem_de_nos_dois.mp3
alex_cohen_hotel_california.mp3

.

.

Executo:

1) ls [ap]*.txt

2) ls [b-f]*.txt

3) ls [a-z]*.txt

.

CP - COPIA ARQUIVOS E DIRETÓRIOS

.

cp [opções]

.

1) Se usar:

cp -i

.

É o modo interativo. Talvez uso se não tenho certeza de que o arquivo foi copiado previamente, pois copiar novamente sobrescreve o arquivo copiado e posso perder alguma coisa...

.

2) Se usar:

cp -v

.

Mostra o que está copiando

.

Executo:

cp -v arq-faz-de-conta.sh arq-faz-de-conta-backup.sh

ls1

.

3) Se usar:

cp -r

.

Copia recursivamente arquivos pastas e subpastas

.

COMANDO: MV

mv - MOVER ARQUIVOS E PASTAS/DIRETÓRIOS

.

É usado para renomear arquivos, é quase o mesmo que copiar o arquivo origem para o arquivo destino e depois remover o arquivo origem. As opções do mv são parecidas com as do comando cp.

.

Se eu crio o arquivo-teste.txt pelo terminal com o comando touch e escrevo algo nele com o comando echo redirecionando a saída do echo para o arquivo-teste.txt, depois leio o que escrevi com o comando cat eu posso copia-lo para o 2arquivo-teste-novo.txt com o comando mv.

.

EXECUTO COPIANDO E COLANDO OS COMANDOS ABAIXO UM DE CADA VEZ:

.

```
echo -e '\nOlá!\nTudo\nFirme?' >> 05-arq.txt

echo -e '2-Olá!\n2-Tudo\n2-Firme?\n' >> 05-arq.txt

echo "Aprendendo Linux !" >> 05-arq.txt

echo -e '\nOs dedos dormentes !\n \nOs olhos vermelhos !\n' >> 05-arq.txt

echo -e '\nTodos esses que aí estão \nAtravandando meu caminho, \nEles passarão... \nEu passarinho! \n \n-- Mario Quintana \n' > parq-passarinho.txt

ls -tl1

cat 05-arq.txt

cat parq-passarinho.txt

cat -n 05-arq.txt

cat -n parq-passarinho.txt

touch arquivo-teste.txt

ls -li

ls -t1

find -iname 05-arq.txt

find ./* -type f -exec grep -l Tudo {} \; (Este busca textos ou strings dentro de arquivos)
```

```
find ./* -type f -exec grep -l passarinho {} \;

echo 'Aprendendo Linux !' >> arquivo-teste.txt

cat -n arquivo-teste.txt

echo 'Estou Aprendendo Linux ! Nós Estamos Aprendendo Linux ! Tudo Linux !
passaro Linux tudo' > arquivo-teste.txt

find ./* -type f -exec grep -l Linux {} \;

grep "Linux" -r ./

find /home -size +100k

grep luz arq-poema.txt

egrep "faça" -r ./

egrep "faça" -r ?*.txt

find ./* -size +100k

find ./* -type f -exec grep -l faça {} \;

grep "Tudo" -r ./

egrep "passarinho" -r ./

find ./* -size +20k

fgrep "Linux" -r ./

echo 'Aprender Linux é Divertido !' >> arquivo-teste.txt

clear

cat arquivo-teste.txt

find -name "arq*"

find -iname "arq*"

find *.txt

find *.txtt

find /home -size -1G

find ./* -size -1G

find ./* -size -190k

mv arquivo-teste.txt 2arquivo-teste-novo.txt

ls1

ls2
```

ls3

ls0

clear

ls -R

ls -li

ls -t

cat 2arquivo-teste-novo.txt

find -name 2arq*

find -iname "2arq*"

find ./* -type f -exec grep -l Divertido {} \;

find /home -size +2G (encontrar arquivos pelo tamanho)

find /home/* -mtime 30

whoami

find /home/user-whoami -mtime 15 (encontrar arquivo modificado nos últimos 15 dias)

cd ..

find . -mtime 2

find . -size +2048 -print

find . -size +2M -print (procura arquivos maiores que 2048 que é igual a 2 megabytes)

find . -atime +3 -print | head (procura no diretório corrente arquivos que não foram acessados a mais de 3 dias)

find . -iname "*.jpg" | wc -l

cd -

.

MAIS COMANDOS...

Por exemplo, o comando mv é útil quando eu tenho um arquivo que vou refazer mas quero guardar o arquivo antigo como base para o novo arquivo.

.

O comando mv pode mover o 2arquivo-teste-novo.txt para a pasta Downloads.

.

Executo os comandos abaixo:

cd

ls -li

ls -t

pwd

mv 2arquivo-teste-novo.txt Downloads/

cd ..

pwd

ls -li

ls -t

cat 2arquivo-teste-novo.txt

find -name "2arq*"

find *.txt

cd -

cd Downloads/EstudoEmDown/

ls1

.

OBS:
Agora renomeio o 2arquivo-teste-novo.txt para 3arquivo-teste-velho.txt:

.

mv 2arquivo-teste-novo.txt 3arquivo-teste-velho.txt

cat 3arquivo-teste-velho.txt

find -name "3ar*"

.

CONTINUO EXECUTANDO UM COMANDO DE CADA VEZ.

.

pwd

ls -t

mv -b 3arquivo-teste-velho.txt 3arquivo-teste-back.txt

```
ls -t

pwd

cat 3arquivo-teste-back.txt

mv -i 3arquivo-teste-back.txt 4arquivo-teste-velho.txt

ls -t

pwd

cat -n 4arquivo-teste-velho.txt

find -name "4arq*"

find *.txt

find -iname "4arq*" -exec rm {} \;

ls -t

find *.txt
```

.

LN - ESTABELECE LIGAÇÕES ENTRE ARQUIVOS

.

```
ln [-s]
```

.

O comando ln cria ligações (links).

.

Existem dois tipos de ligações:

HARD LINK E LINK SIMBÓLICO.

.

Hard Link: neste caso, os dois compartilham os dados. Se apagar o arquivo original o hardlink permanece funcionando. Somente é possível fazer hardlink em arquivos que estejam em uma mesma partição de disco. Somente o usuário proprietário do sistema pode criar/desfazer hardlinks.

.

Executo:

```
> arq3-test.txt

ls -t

ln arq3-test.txt arq3-hard-link
```

ln arq3-test.txt arq3-hard-link

ls -lit (veja se possuem o mesmo número Inode e o mesmo device)

echo '1-Aprendendo sobre hard link.' >> arq3-test.txt

cat arq3-hard-link

echo '2-Mais uma linha escrita: Aprendendo sobre hard link.' >> arq3-hard-link

ls -lit

cat arq3-test.txt

rm -f arq3-test.txt

ls -lit

cat arq3-hard-link

.

Link Simbólico é somente o caminho do arquivo-origem. Diferente do hard link pode-se fazer links simbólicos em arquivos e pastas. Nenhum dos dois precisam estar na mesma partição de disco.

.

Uso mais Link Simbólico. Para entender vou criar links. Executo os comandos abaixo um de cada vez.

.

Ex:

touch arq4-test.txt

ls -t

ln -s arq4-test.txt arq4-link-simb

ls -tli

echo 'Aprendendo sobre link simbolico.' >> arq4-link-simb

cat arq4-test.txt

ls -t

rm -f arq4-test.txt (Este é o arquivo origem)

cat arq4-link-simb

ls -t

rm arq4-link-simb

.

Comandos que leem da entrada padrão (stdin/teclado) e escrevem na saída padrão (stdout/monitor de vídeo). O propósito destes filtros quase sempre é o de modificar a saída de outros, por isso, geralmente são utilizados em combinação com mais comandos. Exemplo:

1) cat
2) wc
3) sort
4) head
5) tail

.

1) cat (Exibe conteúdo de arquivos)

2) wc (Conta caracteres, palavras e/ou linhas de arquivos)

3) sort (Ordena o conteúdo de arquivos)

4) head (Exibe o início dos arquivos)

5) tail (Exibe o final dos arquivos)

.

Os comandos/filtros acima se não forem direcionados para um arquivo, irão aparecer na tela do terminal.

.

APRENDENDO SHELL SCRIPT - COMANDOS MAIS USADOS DO SHELLSCRIPT

.

SOBRE O COMANDOS MAIS USADOS DO SHELLSCRIPT:

É para iniciantes. Pretende oferecer o máximo de informação possível em pouco tempo, para que a pessoa ao terminar de EXECUTAR este artigo entenda bastante coisa sobre shell script e seja capaz de criar scripts úteis para si mesma. Desde o primeiro artigo eu editei ele umas oito ou nove vezes. Para mim este ficou mais interessante. Longe de ser perfeito, peço que faça vista grossa a erros de digitação ou definições ligeiramente incoerentes. O negócio é que este artigo é realmente útil para mim e espero que seja útil para você também. A maior parte deste artigo é pesquisa, mas alguns pedaços foram reescritos.

.

Não esqueça de dar intervalos em seus estudos para se esticar, hidratar, etc.

.

SABE POR QUE?

SABE POR QUE?

Porque depois de executar alguns trechos deste artigo a ficha cai, a pessoa fica animada e não quer parar (tipo eu).

.

VAMOS LÁ?

.

Qual é o seu usuário?
Se você esqueceu ou não tem certeza qual é o seu usuário, use o comando "whoami" sem aspas, para saber.
Abra o terminal pelo menu do sistema. Não mude de terminal, eu recomendo.

.

DIGito ESTES COMANDOS ABAIXO UM DE CADA VEZ:

.

```
apropos shell

whereis bash

whatis bash

echo $SHELL

echo $$

clear

echo

echo ''

echo ""

echo "Olá!"

printf 'Bem vindo ao bash!' (Aperte a tecla enter e digito: ls)

echo '#!/bin/bash'

echo "#!/bin/bash"

ls -t

ls -a

echo ; echo "Olá!" ; echo

echo -e 'Bom\nDia\nMundo!'

echo -e "Bom\nDia\nMundo!"
```

```
echo -e "\nBom\nDia\nMundo! \n"
echo "Hello world!"
echo "Hello "world"!"
echo "Hello \"world\"!"
echo "Alô \"Mundo\"!"
echo -e "1\t2\t3"
echo -e '1\t2\t3'
echo -e '1\t\v2\t\v3'
echo -e "Nós\tVós\tEles(as)"
echo -e 'Olá\t\vMundo\t\vLinux'
echo -e 'Olá\t\vMundo\t\vLinux\t\vpassarinho\t\volhos'
echo -e 'Olá\t\vMundo\t\vLinux\t\vpassarinho\t\volhos' >> test.txt
cat test.txt
echo -e 'Nós\tVós\tEles(as)'
printf "Hello world" (Aperto a tecla enter e digito: df -h)
pwd
clear
ls -tali
sleep 7
echo ; echo 'Olá!' ; sleep 3 ; ls -t
free -h -t
free -th
sleep 4
date
date -d "yesterday"
date -d "2 days ago"
date +%d
date +%m
date +%Y
date +"%d/%m/%Y"
```

```
date +"%d/%m/%Y"
cal
du -h
du -hc
clear
du -hcs
uptime
uname -a
free -tmlh
whatis df
df -a -h
df -k -l
df -T -m
whatis history
history
echo ; whoami ; echo
ls ; echo ; pwd ; echo
echo ; ls -at ; echo
echo "sou \n um \n texto"
echo -e "sou \n\v um \n\v texto"
echo -e "\nsou \n\t\v um \n\t\v texto\n"
whatis du
du *
ls -lah
du * | ls -lah
du -hcs
du -ach
clear
du * | ls -t | df -h
lsb_release
```

```
lsb_release

lsb_release -a

whatis cat

cat /etc/hostname

cat -n /etc/hostname

cd ..

pwd

cd -

ls -t ; w

VARIAVEL1 = Valor

echo $VARIAVEL1

VARIAVEL1=Valor

echo $VARIAVEL1

VARIAVEL2 = linux comandos shell script

echo $VARIAVEL2

VARIAVEL2= "linux comandos shell script"

echo $VARIAVEL2

VARIAVEL2 ="linux comandos shell script"

echo $VARIAVEL2

VARIAVEL2="linux comandos shell script"

echo $VARIAVEL2

echo $$ (mostra o pid da seção atual)

ps aux |grep bash

VARIAVEL="Meu diretório atual é o `pwd`"

echo $VARIAVEL

VARIAVEL="Este diretório contem: `ls -t`"

echo $VARIAVEL

VARIAVEL="Hoje é: `d`"

echo $VARIAVEL

unset VARIAVEL
```

```
unset VARIAVEL

echo $VARIAVEL

ls

cd -

pwd

ls

clear

echo ; echo 'df: Relata o espaço de disco usado pelo sistema (Usado e
Livre)' ; echo ; df -h ; echo

ls -hat

echo ; echo "du: Relata o espaço utilizado no disco de tal arquivo ou
diretório" ; echo ; du -hcs ; echo

echo ; echo 'env (Este comando oferece uma lista de variáveis. Entre elas
PWD, USER, SESSION_MANAGER e LANG)' ; sleep 4 ; echo ; env ; echo

clear

echo $PWD

echo ${PWD}

echo $USER

echo "Eu estou logado como usuário $USER"

echo $SESSION_MANAGER

echo ${SESSION_MANAGER}

echo $LANG

VALOR="Linux"

echo $VALOR

echo ${VALOR}

echo $VALOR $VALOR

echo ${VALOR} ${VALOR}

VALOR='ls -t'

$VALOR

${VALOR}

VALOR='history'
```

```
echo $VALOR

$VALOR

echo ${VALOR}

${VALOR}

unset VALOR

VALOR=$(cat /etc/hostname)

echo $VALOR

$VALOR

clear

VALOR='free -h -t'

$VALOR

${VALOR}

echo $VALOR

echo ${VALOR}

echo VALOR

echo -e '\nSaiba que ao usar o comando read VALOR\n \n(Aperte a tecla
enter, digito: "ls" sem aspas e aperte enter.)\n \nDepois execute $VALOR\n
\nVamos ver isto logo abaixo:\n'

read VALOR

$VALOR

${VALOR}

read VALOR

.

OBS: Aperto a tecla enter, digito: "uptime" sem aspas e aperte enter.

.

echo $VALOR

$VALOR

${VALOR}

echo "Entre com o valor para a variável: " ; read VARIAVEL

$VARIAVEL
```

```
unset VARIAVEL

VALOR='du -hcs'

echo ; $VALOR ; echo ; $VALOR ; echo

echo ; $VALOR ; sleep 4 ; echo ; $VALOR ; echo

echo ; ${VALOR} ; sleep 4 ; echo ; ${VALOR} ; echo

clear

unset VALOR

echo ; $VALOR

$VALOR

${VALOR}

echo ; ${VALOR}

VALOR='lsb_release -a'

$VALOR

VALOR=$(lsb_release -a)

echo $VALOR

echo -e '\nOlá!\nVamos \nSaber \nOs \nComandos \ndo \nShellscript
\nLinux!'

clear ; echo -e '\n \nOlá!\n \nVamos\n \nSaber\n \nOs\n \nComandos\n
\ndo\n \nShellscript Linux!\n'

HOJE=$(lsblk)

echo "Informação sobre dispositivos de bloco: $HOJE"

echo "Informação sobre dispositivos de bloco: ${HOJE}"

HOJE=$(cal)

echo "Informação sobre o calendário atual: $HOJE"

HOJE=$(uptime)

echo "Informação sobre tempo de funcionamento desta máquina: $HOJE"

clear

HOJE=$(lsblk)

echo 'Informação sobre dispositivos de bloco: $HOJE'

echo 'Informação sobre dispositivos de bloco: ${HOJE}'
```

```
unset HOJE
echo $HOJE
echo ${HOJE}
echo `expr 3 + 2`
echo $((3+2))
echo `expr 9 + 4`
echo $((9+4))
echo $((2*3))
echo 'dois vezes três é:' $((2*3))
echo $((2*4-2/2+3))
VALOR=44
echo $VALOR
echo $((VALOR*1))
echo $((VALOR*2))
echo $((VALOR*3))
echo $VALOR
VALOR=$((VALOR+1))
echo $VALOR
VALOR=$((VALOR+11))
echo $VALOR
VALOR=$((VALOR+1))
echo $VALOR
VALOR=$((VALOR+11))
echo $VALOR
unset VALOR
echo $VALOR
VALOR="echo -e \nBom\nDia\nMundo\nLinux\n"
$VALOR
echo $VALOR
```

```
echo ${VALOR}

VALOR=$(uname -a)

echo $VALOR

HOJE=$(arch)

echo $HOJE

clear

VALOR=$(uname -a) ; echo ; echo "Informação sobre o kernel: $VALOR" ;
sleep 4 ; echo ; VALOR=$(arch) ; echo "Informação sobre a arquitetura do
sistema: $USER" ; echo ; sleep 2

unset VALOR

unset HOJE

clear

printf "%-5s %-10s %-4s\n" No Nome Pontos

printf "%-5s %-10s %-4.2f\n" 1 Marta 8

printf "%-5s %-10s %-4.2f\n" 2 Joel 9

printf "%-5s %-10s %-4.2f\n" 3 Carlos 7

echo -e "\e[1;31m Este é o texto em vermelho \e[0m"

echo -e "\e[1;32m Este é o texto em verde \e[0m"

echo -e "\e[1;32m Este é o \e[1;34mtexto \e[1;31mmisturado \e[0m"

VALOR="echo -e \e[1;31m\nBom\nDia\nMundo\nLinux\n\e[0m"

$VALOR

echo $VALOR

echo ${VALOR}

unset VALOR

$VALOR

echo $VALOR

unset HOJE

echo $HOJE

STRING="Isso é uma string"

echo 'Quer saber o que é uma string?' ; echo $STRING
```

unset STRING

clear

exit

.

Muito bom que você digitou um comando de cada vez.

Os comandos mais longos pode copiar e colar no terminal se quiser, mas no início seria melhor digitar.

Saiba que a partir deste momento você já está entendendo coisas extremamente úteis para escrever shellscript usando o Bash!

Uma coisa interessante sobre a maior parte dos comandos acima, é que para estudar eles, você não precisa estar conectado a internet.

Os comandos acima oferecem informações interessantes e úteis. Podemos colocar todos eles em um só script e executar. Dependendo do caso, é mais fácil que digitar um a um de cada vez toda vez que precisar.

.

SE PUDER, ANTES DE CONTINUAR EXECUTE OS COMANDOS ACIMA QUE TE CHAMARAM MAIS A ATENÇÃO DE NOVO POIS, EXECUTAR ELES É MUITO DIDÁTICO. NADA MELHOR QUE APRENDER PRATICANDO.

.

ALGUMAS DICAS ÚTEIS QUE SERÃO REPETIDAS DURANTE O CAMINHO (Vou repetir muitas coisas pelo caminho):

.

1
Não execute shell script como root.

.

2
O shell script Linux usando o interpretador de comandos Bash começa com o shebang:

#!/bin/bash

.

3
Abro um editor de texto e colo nele o script salvando com um nome que termine com extensão .sh

.

4

Dar permissão para executar o script pode ser só para você ou para todos usarem:

chmod +x MeuScript.sh (só você)
chmod a+x MeuScript.sh (todos os usuários do PC)

.

OBS:
Para remover a permissão de um shell script:

chmod -x MeuScript.sh
chmod a-x MeuScript.sh

.

5
Um script pode ser executado de 3 modos:

bash MeuScript.sh
sh MeuScript.sh
./MeuScript.sh

.

Exemplo para começar bem. Copio e colo o código abaixo no terminal que você vai abrir pelo MENU DO SISTEMA:

.

echo -e '#!/bin/bash \nclear \necho \necho Olá usuário Linux! \nsleep 2 \necho \necho Olá Mundo! \nsleep 2 \necho \nVALOR=GNU/Linux \nVALOR2=whoami \necho Você quer aprender comandos $VALOR? \nsleep 2\necho \necho Então vamos aprender comandos $VALOR!\nsleep 2 \necho \necho Vamos aprender Shell Script!\nsleep 2\necho\necho 'Até Breve usuário Linux:' ; sleep 2 ; echo ; $VALOR2\necho\nsleep 2\nexit' > 03-olauser.sh ; chmod +x 03-olauser.sh ; bash 03-olauser.sh

.

Para remover o script criado pelo código acima, o arquivo 03-olauser.sh, use o mouse e o gerenciador de arquivos ou copio e colo o comando abaixo no mesmo terminal onde executou o código:

.

clear ; unset VALOR ; sleep 2 ; rm -f 03-olauser.sh ; sleep 2 ; ls -t ; sleep 2 ; exit

.

CRIAR, TRABALHAR E SALVAR ARQUIVO NO VIM (Atualizado)

.

Questões de diversos concursos estão ligadas ao Vim. Exames para obter

certificações Linux vão pedir conhecimento sobre ele então é bom procurar saber alguma coisa.

.

O Vim é um bicho brabo. Tudo é pelo teclado. O Vim usa todas as teclas do teclado para comandos e quando acabam-se as minúsculas ele usa comandos em letras maiúsculas. Para mim o Vim não é para usuários comuns (EU) e sim para programadores, porém, eu quero e vou saber alguma coisa deste editor de texto. Só usaria diariamente se fosse caso de vida ou morte.

.

Instalo o Vim pelo gerenciador de pacotes da minha Distro.

.

Digito: man vim

.

Por tudo que é mais sagrado! A man page do Vim é vastíssima e toda em inglês e para mim extremamente técnica. Dá para pescar uma coisa aqui e alí, mas quem me salva é o DuckDuckGo.

.

Como sempre eu caio no VOL, no E. Praciano, no TerminalRoot, no Aurélio, na Wikipedia. Pensamentos me passam pela cabeça meio de lado...

.

Longe, longe do aconchego do lar quando se trabalha profissionalmente com shell script e programas, dizem que quase sempre não tem ambiente gráfico. Então há de se acostumar a usar editores de texto que funcionam no terminal. Por exemplo o nano e o Vim. O mais usado pelos profissionais parece que é o Vim então mesmo sendo iniciante, alguns exercícios básicos de como usar o Vim são importantes para mim. Vou pesquisar e fazer isto. Vou tentar aprender a usar o editor de textos Vim.

.

No Vim as teclas mais apertadas por mim são a tecla "Esc" e a tecla "i". A tecla "Esc" faz o editor entrar em modo de comando para poder salvar e se movimentar pelo texto entre outras coisas. A tecla "i" é o modo de inserção onde posso digitar palavras entre outras coisas.

.

NO VIM PARECE QUE O ESQUEMA É MAIS OU MENOS ASSIM:

.

LISTA DE COMANDOS VIM:
1) Inserção (de texto) -> Tecla i
2) Comandos (manipular texto) -> Tecla Esc

3) Linha de comando (manipular arquivo) -> Tecla :
4) Visual (seleção visual de texto) -> Tecla v
5) Busca (busca de padrões de texto) -> Tecla /
6) Reposição (insersão sobrescrevendo) -> Tecla R
7) Sair e salvar - aperto a tecla Esc e digito -> :x (salva e sai)
PARA ABRIR O VIM:
1) vim nome-do-arquivo.sh (se o arquivo não existe, ele cria)
2) vim nome-do-arquivo.txt (abre o arquivo com o cursor no fim do arquivo)
3) vim nome-do-arquivo.sh.txt +9 (abre o arquivo com o cursor na linha 9)
4) vim MeuArquivoVim.txt +/frase (abre o arquivo na primeira vez que tiver a palavra frase)

.

NO TERMINAL DIGITO: vim UmArquivoVim.txt

.

ESCREVENDO NO VIM:

Com o vim aberto, aperto a tecla Esc. Aperto a tecla i. Digito algumas linhas. Exemplo:

.

" 2018 - Uma frase escrevo no editor de texto - Lista de palavras: lux olhos não maria Maria joão João isto Linux Unix Carlos Aluno4 " ! @ # $ %

.

SAIO E SALVO:

Aperto Esc, digito ZZ (maiúsculo)

.

Volto ao Vim com o comando: vim nome-do-arquivo.txt

.

COPIAR/COLAR
1) Aperto ESC
2) Posiciono o cursor no início do texto que quero copiar
3) Digito v minúsculo
Usando as teclas de direção, marco o texto a ser copiado
4) Digito y minúsculo
5) Posiciono o cursor no ponto onde desejo colar o texto
6) Digito p minúsculo

.

RECORTAR/COLAR
1) ESC
2) Posicione o cursor no início do texto que quer recortar
3) Digite v minúsculo
4) Usando as teclas de direção, marque o texto a ser recortado

5) Digite d minúsculo
6) Posicione o cursor no ponto onde deseja colar o texto
7) Digite p minúsculo

.

PARA ABRIR VÁRIOS ARQUIVOS (janelas horizontais):

Exemplo:

vim -o arqu1.txt Arquiv2.txt arq-teste3.txt texto-teste4.txt

.

PARA ABRIR VÁRIOS ARQUIVOS (janelas verticais):

Exemplo:

vim -O arqu1.txt Arquiv2.txt arq-teste3.txt texto-teste4.txt

.

PARA TROCAR DE JANELAS APERTO AS TECLAS:

1) Esc

2) Ctrl ww

.

No editor de texto Vim quase tudo é feito apenas usando o teclado. Porem pode por exemplo copiar um texto selecionado com o mouse e colar no Vim. Então devo aprender a apertar teclas para fazer as coisas no Vim.

.

APERTANDO TECLAS:

.

i (Modo de Inserção)

No modo de inserção (Esc a) começo a digitar a partidor do local onde o cursor está.

No modo de inserção (Esc o) começo a digitar numa linha abaixo do cursor.

No modo de inserção (Esc r) começo a digitar sobrescrevendo.

Esc (Modo de Comandos)

Esc h (Vai para esquerda)

Esc j (Desce)

Esc k (Sobe)

Esc l (Vai para a direita)

Esc l (vai para a direita)

Esc :wq (salva e sai)

Esc :x (salva e sai)

Esc ZZ (zalva e zai)

Esq :q! (sai mesmo que não tenha salvo)

Esc qa (fecha todos os arquivos que não foram alterados)

Esc :q (sai sem ligar se salvou, modificou ou não)

Esc :wqa (salva e sai de todos os arquivos que estiverem abertos)

Esc :W (salva o que foi escrito)

Esq :set autowrite (salva a cada operação feita)

Esc :set aw (salva a cada operação feita)

Esc :wa (salva todos os arquivos que estiverem abertos)

Esc :syntax on (Deixa o script colorido)

.

Executo este exercício no meu terminal abrindo ele pelo Menu do meu sistema:

.

Para entrar em MeuArquivoVim.txt em pasta-estudo-vim:

cd

cd Downloads/EstudoEmDown/

mkdir pasta-estudo-vim

cd pasta-estudo-vim/

touch MeuArquivoVim.txt

ls -t

vim MeuArquivoVim.txt

.

Para entrar em modo de inserção teclo:

i

.

Digito frases... Frases...

Mais frases....

La la la la la

Ba ba ba Ba Ba

Pi pi pi pi pi

Duma duma duma duma duma duma duma duma

Loba loba loba loba

Longe longe longe longe

frase Frase frase frase Frase frase Frase

Molinete molinete molinete molinete molinete

Otorrino otorrino otorrino otorrino

Rajada Rajada Rajada Rajada Rajada Rajada

(Aperto a tecla x, Aperto a tecla Delete)

Escrevo: Pedra pedra pedra pedra pedra pedra pedra

(Aperto Esc)

(Aperto as teclas h, j, k, l)

(Aperto as teclas :w)

(Aperto Enter)

(Aperto a tecla i)

Escrevo: Alguma palavra

(Aperto Esc)

(Aperto as teclas :syntax on)

Se este texto fosse um script, com :syntax on ele ficaria mais colorido.

(Aperto as teclas :set autowrite)

(Aperto Enter)

(Aperto a tecla i)

Escrevo: Mula mula Jaca janela

(Aperto Esc)

(Aperto as teclas :wq)

(Aperto a tecla i)

.

Volto ao modo de comandos teclando:

Esc

.

Salvar e sair:

:wq

.

Digito:

ls -t

du -h

.

Volto ao Vim:

vim MeuArquivoVim.txt

.

Aperto a tecla:

Esc

.

Saio do Vim:

ZZ (zalva e zai) ou:

:q!

.

Digito:

ls -t

pwd

.

Volto ao Vim:

vim MeuArquivoVim.txt

.

Aperto:

Esc

.

Saio do Vim:

:q!

.

Volto ao Vim:

vim MeuArquivoVim.txt

.

Para entrar em modo de inserção teclo:

i

.

Digito mais frases. Umas 6 linhas mais ou menos.

.

Volto ao modo de comandos teclando:

ESC

.

Salvar e sair (ESC + :wq) (ESC + ZZ):

:wq

ZZ (zalva e zai)

.

Listo arquivos e pastas:

ls -t

.

Volto ao arquivo usando o Vim:

vim MeuArquivoVim.txt

.

Volto ao modo de comandos teclando:

ESC

.

Vou para o modo de inserção teclando:

i

.

Aperto as teclas (estou no modo de inserção):

h, j, k, l

.

Aperto a tecla:

ESC

.

Aperto as teclas (estou no modo de comandos):

h, j, k, l

.

Vou para o início do arquivo teclando:

gg

.

Vou para o Final teclando:

G

.

Acesso a linha 8, 5, 1, 13, 15 e 11 digito:

1) 8G
2) 5G
3) 1G
4) 13G
5) 15G
6) 11G

.

Saio do Vim:

:q!

.

Volto ao Vim:

vim MeuArquivoVim.txt

.

Teclo:

ESC

.

Desço:

j

.

Subo:

k

.

Esquerda:

h

.

Direita:

l

.

Procuro pela palavra "frase" digito:

/frase

.

Aperto a tecla:

ESC

.

Aperto a tecla:

a (começa a digitar a direita do cursor)

.

Aperto a tecla:

O (modo de inserção uma linha antes do cursor)

.

Para copiar aperto a tecla:

Y

.

yy para copiar a linha atual

yw para copiar a próxima palavra.

.

Se quiser copiar as próximas 3 palavras, tento y3w.

y$ para copiar do ponto atual até o final da linha

Depois de copiado, basta mover o cursor para o local, no texto, em que desejo inserir o conteúdo do buffer e teclar p

.

Para colar aperto a tecla:

S

.

Copiar ou recortar palavras:

Esc

:

yw

dw

.

dd recorta a linha (atual) em que o cursor se encontra

.

dw recorta a palavra que se encontra à direita do cursor

.

db recorta a palavra à esquerda

.

3d recorta 3 linhas inteiras

.

O comando p é usado para inserir o conteúdo do buffer no texto.

.

Depois de recortar o que queria, basta posicionar o cursor no ponto em que deseja colar o texto “apagado” e pressionar ‘p’ - para reinserir o conteúdo do buffer.

conteúdo do buffer.

.

Se quiser, é possível multiplicar a quantidade de vezes em que é reinserido.

.

Tecle ‘5p’, por exemplo, para inserir o conteúdo do buffer 5 vezes.

.

Para selecionar um trecho:

V (para linhas)

v (para caracteres)

Uso: h j k l

Digito p para colar.

.

Para desfazer uma ação:

u

.

Para repetir um comando:

Ctrl+r

.

Buscar:

/palavra

.

Nova ocorrência:

N

.

:e - Abrir um arquivo.

.

Sintaxe

:e /root/teste.sh

.

:sp - Abrir em outra janela.

.

A Sintaxe pode ser :sp somente ou :sp /root/teste.sh

.

:enew - Novo documento.

A sintaxe

:enew

.

:sav - Salvar como.

A sintaxe

: sav nomedoarquivo.txt

.

u - Serve para Desfazer / Refazer.

.

A sintaxe " u " sem aspas, quando voce fizer uma coisa errada

.

. - Serve para repetir.

.

A Sintaxe " . " sem aspas.

.

[p - Serve para colar antes.

.

o comando eh simplesmente [p

.

]p - Server para colar depois.

.

o comando é simplesmente]p

.

ggVG - Seleciona tudo.

.

"+x - Serve para recortar.

.

A sintaxe é escrita com da forma a cima (Aspa dupla)+(Sinal de mais)+(x).

.

"+y - Serve para colocar .

.

A sintaxe é escrita com da forma a cima (Aspa dupla)

.

+(Sinal de mais)+(y).

.

:wqa - Serve para salvar e sair.

A sintaxe é

:wqa

.

:qa - Serve para sair somente.

.

A sintaxe eh :qa

.

x - Serve para deletar

.

O comando para deletar eh somente " x " sem as aspas.

.

:set hls! - Serve para Ativar / Desativar Realce de padroes

.

REVISANDO:

Para criar um arquivo de texto chamado MeuArquivoVim.txt, cujo conteúdo seria a frase "Teste tutorial criar e salvar arquivo no VIM" faria assim:

.

Executo o comando: vim MeuArquivoVim.txt

Executo o comando: vim MeuArquivoVim.txt

.

Agora estou no Vim.

.

Pressiono a tecla: i (para passar do MODO DE COMANDOS para o MODO DE INSERÇÃO)

.

Digito a frase: Teste tutorial criar e salvar arquivo no VIM

Digito a frase:

.

Volto ao modo de comandos apertando a tecla: ESC

.

Digito o símbolo e as 2 letras abaixo:

:wq

.

Pressiono Enter logo em seguida.

.

O comando (:wq) é utilizado para SALVAR (write) o arquivo e SAIR (quit) do Vim.

.

Repito esses passos em meu computador algumas vezes e, ao fim, tenho o arquivo MeuArquivoVim.txt em meu diretório de trabalho.

.

Para listar arquivos e verificar a presença do MeuArquivoVim.txt digito:

ls -tli

.

COMANDOS PRINCIPAIS DO VIM

Agora que aprendi como alternar entre os modos de comando e inserção e já executei o meu primeiro comando (:wq), é hora de aprender mais teclas de atalho e novos comandos. Mas, antes de prosseguir, vale a pena prestar a atenção em dois detalhes importantes.

.

Primeiro, lembro de retornar ao Modo de Comandos pressionando a tecla ESC, caso esteja digitando um texto no Vim.

.

Caso contrário, os comandos desta lista não funcionarão e meu arquivo de texto acabará com sequências do tipo :q! no meio de alguma palavra.

.

Em segundo lugar, noto que o Vim faz uso de quase todas as letras do alfabeto e, na falta de opções, acaba apelando para maiúsculas.

.

Isto quer dizer que os comandos o e O são diferentes.

.

MOVIMENTANDO-SE NO ARQUIVO

.

Com isso em mente, posso começar a movimentar o cursor do Vim entre as linhas e letras de um arquivo de texto.

.

Para isso, pressiono ESC (para sair do modo de inserção, caso esteja nele) e, em seguida, uso as teclas:

h, j, k, l

As teclas (h, j, k, l) movem o cursor para esquerda (h), para baixo (j), para cima (k) e para a direita (l).

.

As setas direcionais também podem ser usadas para esse fim, mas podem não funcionar em qualquer distro ou mapa de teclado.

.

Se quiser me mover para o início do arquivo, digito gg e, quando quiser ir ao final, pressiono G (maiúsculo).

.

Também é possível pular de palavra em palavra com teclas w e b, além de ir diretamente para uma linha em específico. Se quiser acessar a 27ª linha do texto, por exemplo, digito 27G.

.

ABRIR, SALVAR E OUTRAS OPERAÇÕES DE ARQUIVOS

.

Abrir um arquivo de textos com o Vim é simples.

.

Basta informar o caminho e nome do arquivo como parâmetro ao comando vim, quando for executá-lo.

.

Se quisesse abrir o arquivo /etc/passwd, por exemplo, executaria no terminal:

vim /etc/passwd

.

Para sair do editor de textos, digito :q.

.

Se quiser forçar a operação de sair, sem me preocupar com a possibilidade de salvar o arquivo que estava sendo editado, pressiono :q!.

.

Para salvar o arquivo no qual estou trabalhando, uso o comando :w, enquanto que, para salvar e sair, com já visto acima, uso :wq.

.

MUDAR DE MODOS

.

Então pressionar a tecla i faz com que o Vim passe para o Modo de Inserção.

.

Pressionar a tecla ESC retorna para o Modo de Comandos.

.

Porém, é possível fazer um uso mais preciso dessa operação. A tecla i, na verdade, inicia a inserção de texto à esquerda da localização atual do cursor.

.

Se quiser começar a digitar à direita do cursor, pressiono a.

.

Da mesma forma, caso deseje entrar no modo de inserção uma linha antes do cursor, pressiono O, e, se preferir, o para uma linha após o cursor.

.

PESQUISAS E COMANDOS AVANÇADOS

.

Apesar da simplicidade aparente do Vim, o editor permite realizar ações mais complexas.

.

Para começar, posso desfazer a última ação pressionando a tecla u.

.

Caso queira refazer ou repetir um comando, uso Ctrl+r.

.

Se quiser procurar por um termo ou frase no arquivo de texto, digito / seguido da expressão, sem espaço.

.

Para buscar a palavra MeuArquivo, por exemplo, uso /MeuArquivo.

.

Para procurar pela ocorrência seguinte da palavra, após ter realizado a pesquisa, basta pressionar n.

.

Se quiser voltar uma ocorrência, pressiono N.

.

ALGUNS DOS COMANDOS DO VIM NO MODO DE COMANDOS:

0 : mover o cursor para o início da linha em que o cursor está posicionado.

a : inserir texto após a posição atual do cursor.

A : inserir texto no final da linha atual.

dd : deletar linha atual.

[n]+dd : deletar n linhas a partir da linha atual.

G : ir para o fim do arquivo.

[n]+G : ir para a n-ésima linha do arquivo.

h : voltar um caractere.

H : ir para a primeira linha exibida na tela atual.

i : inserir texto a partir da posição atual do cursor.

I : inserir texto no início da linha atual.

j : descer uma linha.

J : juntar a linha atual com a linha seguinte.

[n]+J : juntar n linhas consecutivas a partir da linha atual.

k : subir uma linha.

l : avançar um caractere.

L : ir para a última linha exibida na tela atual.

n : procurar, a partir da posição atual do cursor, a próxima ocorrência do texto definido no último comando /.

N : procurar, a partir da posição atual do cursor e indo em direção ao início do arquivo, a próxima ocorrência do texto definido no último comando /.

o : inserir uma linha em branco após a linha atual.

O : inserir uma linha em branco acima da linha atual.

p : inserir linhas copiadas após a linha atual.

P : inserir linhas copiadas antes da linha atual.

r : substituir o caractere atual.

R : substituir um conjunto de caracteres.

s : deletar o caractere atual e inserir texto.

S : apagar linha e inserir novo texto na linha.

u : desfazer a última alteração feita no texto e ainda não desfeita.

U : desfazer a última alteração feita no texto.

x : apagar caractere onde o cursor está posicionado.

$: mover o cursor para o fim da linha em que o cursor está posicionado.

[n]+y : copiar n linhas a partir da linha atual.

yy : copiar a linha atual.

[n]+Y : copiar n linhas a partir da linha atual.

YY : copiar a linha atual.

CTRL+B : voltar uma página.

CTRL+F : avançar uma página.

F1 : exibir tela de ajuda.

[n]+ENTER : ir para n linhas abaixo da linha atual.

[n]+. : repetir o último comando que alterou o texto n vezes a partir da posição atual do cursor.

[n]+~+ENTER : inverter a caixa (case) dos n caracteres seguintes ao cursor.

/texto : procurar pela primeira ocorrência do texto especificado a partir da posição atual do cursor.

.

PROGRAMAÇÃO SHELL SCRIPT - REVISANDO COMANDOS BÁSICOS PARA SHELL SCRIPT

- Veja só! Execute este pequeno scriptzinho matemático no terminal do seu Linux (execução sequencial de comandos no shell):

```
cd ; cd Downloads/EstudoEmDown ; echo -e '#!/bin/bash\n x=8 # variável x valor 8\n y=4 # variável y valor 4\n# agora determinamos a soma de x e y para z:\n z=$(($x + $y))\necho\necho "A soma de $x + $y é $z"\necho\n# Fim do soma-scriptzinho.sh' > soma-scriptzinho.sh ; chmod +x soma-scriptzinho.sh ; sh soma-scriptzinho.sh
```

#

1) Copio e colo estes comandos no terminal do Linux. Será útil para aprender coisas interessantes.

#

Criando o local/pastas/arquivos de estudo em Downloads/:

```
cd ; cd Downloads/EstudoEmDown/ ; mkdir -p estudo-shell-script/arquivos ; cd estudo-shell-script/arquivos/ ; echo -e 'Ana Claudia Bodhi\nAna Claudia Vasconcelos\nAndre Gonçalves\nAntonio Silva\nBento Silva\nCarlos Augusto\nDaniel Sandra\nEliseu Padilha\nFernanda Padilha\nGustavo Rosa Bela\nHorácio Nunes\nIngrid Damacena\nMaria Antonieto Sousa\nOlimpio Silva\nPaulo Freitas\nRafaela dos Santos\nRoff Silvaciano\nSilvia Oliveira\nZuenir Mello\nXerxes Alvez' > alunos2.txt ; echo -e 'Aluno1\n Aluno2\n Aluno3\n Aluno4\n Aluno1\n Aluno4\n Aluno5\n Aluno6\n Aluno1\n Andre\n Paulo\n Junior \n Daiana\n Fernanda\n Fernanda\n Maria\n Daiana\n Maria\n Maria\n Nunes\n Gonçalo' > alunos.txt
```

#

Criando arquivos para estudo:

```
echo -e 'Ana Claudia\nAna Claudia Vasconcelos\nAndre Gonçalves\nAntonio Silva\nBento Silva\nCarlos Augusto\nCarlos Roberto \nDaniel Sandra\nEliseu Padilha\nFernanda Padilha \nLucas Carmo \nGustavo Rosa\nHorácio
```

```
Nunes\nIngrid Damacena\nMaria Antonieto Sousa\nOlimpio Silva\nPaulo Freitas\nRafaela dos Santos\nRoff Silvaciano\nSilvia Oliveira\nZuenir Mello\nXerxes Alvez' > alunos3.txt ; echo -e 'Aluno1\n Aluno2\n Aluno3\n Aluno4\n Aluno1\n Aluno4\n Aluno5\n Aluno6\n Aluno1\n Andre\n Paulo\n Pamela Castro\n Junior \n Daiana Neres\n Fernanda\n Fernanda\n Maria\n Daiana\n Maria\n Maria Fernanda\n Nunes\n Gonçalo' > alunos4.txt

#

2) Copio e colo no terminal (se já não fiz isto). Vai criar um arquivo de texto. Aperte Enter:

echo -e '\nEntão queres ser um escritor? \nSe não sair de ti explodindo apesar de tudo, não o faças. \nA menos que saia sem perguntar do teu coração e da tua cabeça e da tua boca e das tuas entranhas, não o faças. \nSe tens que sentar por horas olhando a tela do teu computador ou curvado sobre a tua máquina de escrever procurando palavras, não o faças. \nSe o fazes por dinheiro ou fama, não o faças. \nSe o fazes porque queres mulheres na tua cama, não o faças. \nSe tens que te sentar e reescrever uma e outra vez, não o faças. \nSe dá trabalho só pensar em fazê-lo, não o faças. \nSe tentas escrever como algum outro escreveu, não o faças. \nSe tens que esperar para que saia de ti a gritar, então espera pacientemente. \nSe nunca sair de ti a gritar, faz outra coisa. \nSe tens que o ler primeiro à tua mulher ou namorada ou namorado ou pais ou a quem quer que seja, não estás pronto. \nNão sejas como muitos escritores, não sejas como milhares de pessoas que se consideram escritores, não sejas estúpido nem enfadonho e pedante, não te consumas com auto-devoção. \nAs bibliotecas de todo o mundo têm bocejado até adormecer com os da tua espécie. \nNão sejas mais um. \nNão o faças. \nA menos que saia da tua alma como um míssil, a menos que o estar parado te leve à loucura ou ao suicídio ou homicídio, não o faças. \nA menos que o sol dentro de ti te esteja a queimar as tripas, não o faças. \nQuando chegar mesmo a altura, e se foste escolhido, vai acontecer por si só e continuará a acontecer até que tu morras ou morra em ti. \nNão há outra forma. \nE nunca houve. \n \n(Charles Bukowski)\n' > charles_texto-escritor.txt

#

3) Executando comandos de revisão:

echo $0

echo $SHELL

tail /etc/passwd

clear

#

4) Sem o software ls não tem o comando ls. Dizem que não existe terminal GNU/Linux sem o ls. Dizem também que sem ele (ls), todos os outros comandos não funcionam bem.

#
```

```
..

ls -ltr

ls -R

#

ls -1

ls -l /etc

ls -l /tmp

ls -la /var

ls -l /var/log/

ls -l /var/log/journal/

ls -l /var/log/samba/

ls -lh /var/lightdm

ls -lt /snap/

ls -lht /snap/bin/

ls -ltah /snap/bin/

ls -lit /root

clear

#

ps

#

ps axu

clear

ps -eF

ps -ejH

ps -eLf

ps axZ

clear

#

# ps axu - O que se usa mais no ps? Nome do usuario, pid do processo,
consumo de cpu, memoria, horario de execução, nome completo do processo.
```

```
#

(colo os comandos abaixo no terminal)

touch arquivo-teste

> arquivo-tester

ls -t

#

echo (Mostra na tela o parâmetro que vc deu)

echo linux shell script

echo O sol de manhã

echo "linux shell script"

clear

echo 'O echo pode ser acompanhado das opções -n (do not output the
trailing newline). E da opção -e (enable interpretation of backslash
escapes)'

echo -n "linux shell script" (Aperto enter)

# Não vai quebrar a linha por que pode precisar colocar um leitura de
variável, então é necessário não quebrar a linha.

echo -e

# Permite usar opções de tabulação.

echo -e "linux\nshell\nscript"

echo -ne "linux\nshell\nscript" (Aperto enter)

echo -e "curso shell\t script" # \t (Tabulação)

echo -ne "curso shell\t script" (Aperto enter)

echo -e "curso\nshell\tscript"

# \n e \t (Quebra de Linha e Tabulação)

echo -ne "curso\nshell\tscript"

echo -e "col1\tcol2\tcol3\t"

echo -ne "col1\tcol2\tcol3\t"

echo -e "\acol1\t\acol2\t\acol3\vcol5\vcol6\vcol7"

echo -ne "\acol1\t\acol2\t\acol3\vcol5\vcol6\vcol7"

echo -e "\acol1\t\acol2\t\acol3\vcol4
```

```
echo -e "\acol1\t\acol2\t\acol3\vcol4
vertical\vcol5vertical\vcol6vertical"

echo -ne "\acol1\t\acol2\t\acol3\vcol4
vertical\vcol5vertical\vcol6vertical"

#

mkdir pasta1

ls -tli

mkdir pasta1/sub-pasta1 # Porque a pasta1 já existe.

# Se uma pasta não existe para criar uma sub-pasta adicona a opção -p

mkdir -p pasta2/sub-pasta2

ls pasta2

rm -r pasta2

#

OBS:
rmdir (Só remove pasta vazia)

#

Sito PARA APRENDER/CONSULTAR COMANDOS:

http://explainshell.com/

#

(Digito/Copio e colo)

cat alunos.txt

cat alunos2.txt

cat -b alunos2.txt (numera e ordena de A a Z)

cat --number-nonblank alunos2.txt (numera e ordena de A a Z)

cat -n alunos2.txt

clear

cat --number alunos2.txt

cat -A alunos2.txt

cat --show-all alunos2.txt

cat -vET alunos2.txt

cat -vET charles_texto-escritor.txt
```

```
cat /snap/README

clear

#

man tac

tac alunos.txt

tac alunos2.txt

tac -b alunos.txt

tac -r alunos2.txt

#

echo -e 'Mais porem escrevendo todavia \nA forma controversa é literal
\nEscapa a fantasia pois \nO reduto da alegoria não \nSe refere a
longevida \nInerente a espaço de um objeto concreto \nPercebe-se a
listagem: \nLista1 \nLilista \nCiclista \nBravar \nLista2 \nAna Cara
\nZarabatana \nArcaido \nCaldo alto \nLuz Ribaltazar \nLista5 \nLista3
\nLista6 \nPossante \nLista7 \nLista8 \nLista9 \nLista10 \nLista11
\nEntretanto \nArquivo longo \nEstende' > arquivolongo.txt

tail arquivolongo.txt

tail -n5 arquivolongo.txt

tail -n7 charles_texto-escritor.txt

tail -4 arquivolongo.txt

tail -n5 alunos2.txt | wc -w

tail /etc/passwd | sort -k3

#

clear

head arquivolongo.txt

head -n5 arquivolongo.txt

head -n1 arquivolongo.txt

head -c10 arquivolongo.txt (aperte a tecla Enter)

#

man wc

whatis wc (cada letra é um byte)

whereis wc
```

.

O comando wc é utilizado para contar caracteres, palavras e/ou linhas dos dados da entrada padrão e apresenta o resultado na saída padrão.

.

Sintaxe:

Comando + parâmetros + arquivo

Parâmetros:

-l: conta as linhas;
-w: conta as palavras;
-c: conta os caracteres.

.

OBS:
Arquivo de entrada cujas palavras, linhas ou caracteres serão contados e exibidos na saída padrão. Se este parâmetro for omitido, o wc lê da entrada padrão.

.

Exemplo a ser executados:

```
echo -e '\nO Marco Marciano\n \nPelos alto-falantes do universo \nVou louvar-vos aqui na minha loa\nUm trabalho que fiz noutro planeta \nOnde nave flutua e disco voa \nFiz meu marco no solo marciano \nNum deserto vermelho sem garoa \n \nEste marco que eu fiz é fortaleza. \nElevando ao quadrado Gibraltar! \nTorreão, levadiço, raio-laser \nE um sistema internet de radar \nNão tem sonda nem nave tripulada \nQue consiga descer nem decolar. \n\nConstrui o meu marco na certeza \nQue ninguém, cibernético ou humano. \nPoderia romper as minhas guardas \nNem achar qualquer falha nos meus planos \nFicam todos em Fobos ou em Deimos \nContemplando o meu marco marciano \n \nO meu marco tem rosto de pessoa \nTem ruínas de ruas e cidades \nTem muralhas, pirâmides e restos \nDe culturas, demônios, divindades \nA história de Marte soterrada \nPelo efêmero pó das tempestades \n
\nConstrui o meu marco gigantesco \nNum planalto cercado por montanhas \nPrecipícios gelados e falésias \nProjetando no ar formas estranhas \nComo os muros Ciclópicos de Tebas \nE as fatais cordilheiras da Espanha \n \nBem na praça central. um monumento \nEmbeleza meu marco marciano \nUm granito em enigma recortado \nPelos rudes martelos de Vulcano \nUma esfinge em perfil contra o poente \nGuardiã imortal do meu arcano... \n \n -- Lenine \n' > arq_marciano.txt
```

.

wc -l arq_marciano.txt (conta linhas)

wc -w arq_marciano.txt (conta palavras)

wc -c arq_marciano.txt (conta caracteres)

wc -m arq_marciano.txt

wc arq_marciano.txt (não foi especificado nenhum parâmetro, o wc listou tudo caracteres, palavras e linhas).

.

wc alunos.txt

clear

wc alunos2.txt

wc -l alunos2.txt

wc -l alunos2*

wc -l alunos*

wc -n alunos2.txt

clear

wc -w alunos2.txt

wc -c alunos2.txt

wc -m alunos2.txt

#

(Sort e Uniq, o sort vem antes do uniq)

#

sort alunos2.txt

sort -r alunos2.txt

sort -k2 alunos2.txt # Ordena pelo segundo campo

#

SITE PARA APRENDER COMANDOS:

http://explainshell.com/

#

sort alunos.txt # Primeiro sort para organizar

#

uniq alunos.txt # Depois pode usar o unic se não, o unic repete palavras iguais que não estiverem na sequência

```
#
sort alunos.txt |uniq -c
sort alunos.txt |uniq -d
sort alunos.txt |uniq -u
clear
#
(As principais opções do uniq são)
-u (o que apareceu uma só vez)
-d (duplicadas)
-c (conta repetições)
#
sort alunos.txt -u
sort alunos.txt -d
clear
sort alunos.txt -c
sort alunos.txt |uniq -d # Linhas duplicadas
sort alunos.txt |uniq -c |sort
clear
sort alunos.txt |uniq -c |sort -r
uniq -c alunos.txt
sort alunos.txt |uniq -c |sort -r|head -n1
#
(O que é tr?)
#
Ex:
$whatis tr
tr (1) - translate or delete characters
tr (1p) - translate characters
#
(whatis tr - traduz ou deleta caracteres - dentro de uma string)
```

#

(Trocar toda letra a pela letra e)

cat alunos.txt |tr a e

cat alunos.txt |tr a e > alunos-troca-letra-a-por-e.txt

cat alunos-troca-letra-a-por-e.txt

#

(O tr serve para substituir/trocar vários tipos de coisa)

#

cat alunos.txt | tr a-z A-Z (Vai trocar toda vez que encontrar caractere minúsculo)

cat alunos.txt | tr aei AEI

(Posso trocar por símbolos, espaços, tabulação, quebra de linha, etc...)

#

cat alunos2.txt |tr ' ' '\t'

#

Ex:

$cat alunos2 |tr ' ' '\t'

cat: alunos2: Arquivo ou diretório não encontrado

#

cat alunos2.txt |tr ' ' '\t'

#

cat alunos2.txt |tr ' ' '\t' | cat -A

#

Ex:

$cat alunos2.txt |tr ' ' '\t' | cat -A

Ana^IClaudia$
Ana^IClaudia^IVasconcelos$
Andre^IGonM-CM-'alves$
Antonio^ISilva$
Bento^ISilva$
Carlos^IAugusto$
Daniel^ISandro$
Eliseu^IPadilha$

Elised IPadilha$
Fernanda^IPadilha$
Gustavo^IRosa$
HorM-CM-!cio^INunes$
Ingrid^IDamacena$
Maria^IAntonieto^ISousa$
Olimpio^ISilva$
Paulo^IFreitas$
Rafaela^Idos^ISantos$
Roff^ISilvaciano$
Silvia^IOliveira$
Zuenir^IMello$
Xerxes^IAlvez$

#

cat alunos2.txt |tr -d aei (deletar as letras aei na saida do comando, o arquivo em si continua sem alteração)

#

$cat alunos2.txt |tr -d aei
An Clud
An Clud Vsconclos
Andr Gonçlvs
Antono Slv
Bnto Slv
Crlos Augusto
Dnl Sndro
Elsu Pdlh
Frnnd Pdlh
Gustvo Ros
Horáco Nuns
Ingrd Dmcn
Mr Antonto Sous
Olmpo Slv
Pulo Frts
Rfl dos Sntos
Rcrd Prodncno
Slv Olvr
Zunr Mllo
Xrxs Alvz

#

echo "Curso Shell Script" | tr l L

echo "Curso Shell Script" | tr S s

#

(comprimir toda vez que encontrar letras repetidas)

echo "Curso Shell Script" | tr -s 'l'

```
echo "Cuurso Sheeeelll Scriiiippttt" | tr -s 'u e l i p t'

#

echo Curso Shell Script | tr [:lower:] [:upper:]

#

(cut recorta pedaços de uma palavras escritas no terminal ou em shell
script, as chamadas strings eu acho. Pode cortar por caracteres e por
campos)

#

cut -d: -f 1 /etc/passwd

cut -d: -f 1,3 /etc/passwd

cut -c 1-10 /etc/passwd

date | cut -d: -f1

cat alunos2.txt | cut -c1-5

cat alunos2.txt | cut -c1,2,6

cat alunos2.txt | cut -c1,6,2

cat alunos2.txt | cut -c1,2,3

cat alunos2.txt | cut -c3,2,1

cat alunos2.txt | cut -c1,2,2

cat alunos2.txt | cut -c1,2

free | tr -s ' ' | sed '/^Mem/!d' | cut -d" " -f2

clear

#

(Cada comando GNU/Linux me parece que faz um e apenas um serviço bem
feito)

#

(Do 5 para frente)

cat alunos2.txt | cut -c5-

#

(Até o 5)

cat alunos2.txt | cut -c-5

#
```

(Até o 10)

cat alunos2.txt | cut -c-10

#

(1, 2 e do 10 para frente)

cat alunos2.txt | cut -c1,2,10-

#

(cada palavra é um campo. pode usar o cut com campos)

cat alunos2.txt | cut -f1 (falta alguma coisa)

clear

cat alunos2.txt | cut -d" " -f1

cat alunos2.txt | cut -d" " -f1,3 (1 e 3)

cat alunos2.txt | cut -d" " -f1-3 (1 ao 3)

cat alunos2.txt | cut -d" " -f2-

cat alunos2.txt | cut -d" " -f-2

clear

#

tail /etc/passwd

tail /etc/passwd | cut -d":" -f1,5

#

(comando diff compara 2 arquivos)

cat alunos.txt

cat alunos3.txt

diff alunos.txt alunos3.txt

(resultado)

$diff alunos.txt alunos3.txt

bash: diff: comando não encontrado
arquivos $diff alunos.txt alunos3.txt
1,9d0
< Aluno1

| STDER (2)
|---------------------->

```
|----------------------<

#

(direcionar a saida de erro)

ls -l alunos.txtt 2> log.out

cat log.out

ls -l alunos.txtx 2>> log.out

ls -l alunos.tdt > log.out 2> log-erro.out

#

(a saida de erro sera a mesma que a saida padrao)

ls -l alunos.txt3 > log.out 2>&1

ls -l alunos.txt3 >> log.out 2>&1

clear

(jogar a saida de erro para lugar nenhum)

ls -l alunos.txtxt 2> /dev/null

#

(redirecionamento da entrada usando o sinal de menor, o conteúdo do
arquivo vira a entrada do comando)

(util para redirecionamento de e-mail)

tr 'a' 'Z'

.

Exemplos:

.

mkdir ~/Downloads/EstudoEmDown/

touch ~/Downloads/EstudoEmDown/E_S_arq.txt

echo -e 'Bom\nDia\nMundo\nLinux\nLista:\nAzul, Beleza, Simone, Aluno3,
luz, olhos, marciano, Carlos, marciano, Mariana, Maria, Carlos, Mariana' >
~/Downloads/EstudoEmDown/E_S_arq.txt

ls ~/Downloads/EstudoEmDown/

cat ~/Downloads/EstudoEmDown/E_S_arq.txt

ls -lR ~/Downloads/EstudoEmDown/E_S_arq.txt

find ~/ -iname "*.txt" > ~/Downloads/EstudoEmDown/E_S_arq.txt 2>
```

~/Downloads/EstudoEmDown/ErroE_S_arq.txt

.

Acima o redirecionamento é destrutivo, isto é, ele apaga o conteúdo anterior do arquivo para onde será redirecionada a saída.

.

É possível redirecionamento NÃO-DESTRUTIVO. Usa-se a notação >> no lugar de > e implica em ACRESCENTAR a saída ao FINAL do arquivo para onde está sendo feito o redirecionamento.

.

Exemplos:

touch ~/Downloads/EstudoEmDown/Arquivos.txt

ls -lR ~/ >> ~/Downloads/EstudoEmDown/Arquivos.txt

find ~/ -iname "*.txt" >> ~/Downloads/EstudoEmDown/E_S_arq.txt 2> ~/Downloads/EstudoEmDown/ErroE_S_arq.txt

.

Quando o usuário deseja que a saída padrão ou a saída de erros NÃO sejam exibidas na tela do terminal e tampouco sejam redirecionadas para um arquivo adicional, usa-se o redirecionamento para um arquivo especial: /dev/null. No exemplo acima, as mensagens de erro do comando find não serão exibidas, pois foram redirecionadas para /dev/null.

.

2) Pipeline

.

Suponha o seguinte exemplo:

sort ~/Arquivos.txt > ~/ordenados.txt

less ~/ordenados.txt

.

PIPE É UMA FORMA DE REDIRECIONAMENTO QUE CONECTA A SAÍDA PADRÃO DE UM PROGRAMA À ENTRADA PADRÃO DE OUTRO PROGRAMA.

.

Com pipes (símbolo |) o exemplo acima se transforma em:

.

sort ~/Arquivos.txt | less

.

PIPELINES PODEM SER USADOS EM QUALQUER QUANTIDADE EM UMA LINHA DE COMANDO:

ls -lR ~ | sort -r | cut -c1-10 | less

.

PIPELINES E REDIRECIONAMENTOS PODEM SER USADOS EM UMA MESMA LINHA DE COMANDO:

find /var -print 2>/dev/null | sort > ~/LOG

.

No exemplo acima, a saída de erros (STDERR) do find é redirecionada para /dev/null (isto é, não aparecerá na tela), mas a saída padrão (STDOUT) será redirecionada para o pipe. O comando sort ordena o resultado do find e redireciona o resultado final para o arquivo ~/LOG.
Expressões Regulares Básicas: grep

.

Uma EXPRESSÃO REGULAR é uma forma compacta de especificar um padrão genérico de caracteres. Existem muitos filtros em UNIX, tais como grep, sed e awk que usam tanto padrões exatos quanto expressões regulares.

.

Executo:

awk '$2 ~ "luz" {print $1, "\t", $4}' arq-poema.txt

(imprime os campos 1 e 4 de cada linha de arq-poema.txt cujo segundo campo contenha "luz".)

.

Por exemplo, pode-se usar grep para encontrar em um arquivo por todas as linhas que tenham a letra ``H'', seguida de um número qualquer de letras minúsculas, seguida da letra ``m''.

.

EXPRESSÕES REGULARES são parte integrante de sistemas Linux e é EXTREMAMENTE IMPORTANTE aprender como usá-las.

.

Dentro de uma expressão regular, certos símbolos tem um significado especial, conforme mostrado na tabela 1. Mais símbolos com siginificado especial podem ser encontrado na seção REGULAR EXPRESSIONS no manual on-line de grep.

. (Ponto)

Significa: qualquer caracter simples, exceto quebra de linha (newline)

---

* (Asterisco)

Significa: zero ou mais ocorrências do caracter precedente

---

^

Significa: início de uma linha

$

Significa: final de uma linha

\

Significa: final de uma palavra

[ ]

Significa: um, e apenas um dos caracteres indicados pelos colchetes

---

[^]

Significa: quaisquer caracteres que não estejam entre os indicados pelos colchetes

---

\

Significa: toma o caracter seguinte literalmente

[[:alnum:]]

Significa: [0-9A-Za-z]

[[:alpha:]]

Significa: [A-Za-z]

[[:digit:]]

Significa: [$0-9]

[a-d]

Significa: [abcd]

[a-dP-T]

Significa: [abcdPQRST]

[]a^[-]

Significa: um dos caracteres: a, ^,], [ou -.

O símbolo ^ perde seu significado especial se não está no início da expressão entre colchetes.

O símbolo - perde seu significado especial se é o último caracter da expressão entre colchetes.

O símbolo] perde seu significado se é o primeiro caracter da expressão entre colchetes.

FIM DA: REVISÃO DE COMANDOS BÁSICOS PARA SHELL SCRIPT

.

CONTINUANDO...

.

(O nome deste símbolo é tralha)

.

! (O nome deste símbolo é exclamação)

./ (Ponto e barra significa onde você está)

#! (Tralha e exclamação significa Shebang)

.

DIRECIONADORES

.

(Write)

> (Este direcionador sobrescreve o que estiver escrito)

1) uptime > arq-uptime1.txt ; sleep 3

2) uptime > arq-uptime1.txt (leia o arquivo)

3) cat arq-uptime1.txt

.

(Append)

>> (Este direcionador não sobrescreve o que estiver escrito)

Exemplo execute os comandos abaixo um de cada vez:

1) uptime >> arq-uptime2.txt ; sleep 3

2) uptime >> arq-uptime2.txt (leia o arquivo)

3) cat arq-uptime2.txt

.

A Shebang para o Bash é a primeira coisa que se escreve em um shell scripr no Linux:

#!/bin/bash

.

Para ver qual é o shell padrão:

printenv SHELL

.

Para mudar o Shell padrão (não faça isto!) Ex:

chsh -s /bin/ash

.

Para sair do shell:

exit

.

Geralmente o primeiro script para quem nunca criou um shell script é criar uma pasta chamada pasta-estudos na pasta home.

cd ; mkdir pasta-estudos

.

Abrir a pasta-estudos e nela criar um arquivo de texto chamado de 01-script.sh.

cd pasta-estudos/

> 01-script.sh

.

Dá poder de execução ao arquivo 01-script.sh.

chmod a+x 01-script.sh

.

Abre o 01-script.sh com um editor de texto. Escreve com um editor de texto no arquivo 01-script.sh o seguinte:

#!/bin/bash

Um comentário

echo "Olá, mundo!"

Fim do Shell Script

.

Comentários também podem ser escritos assim:

#!/bin/bash # Um comentário

echo "Olá, mundo!" # Fim do Shell Script

.

SALVE O QUE FOI ESCRITO. FECHE O EDITOR DE TEXTO.

.

EXECUTA O SCRIPT USANDO UM COMANDO DE CADA VEZ.

comando 1:

sh 01-script.sh

comando 2:

./01-script.sh

comando 3:

bash 01-script.sh

.

TUDO ISTO ACIMA PODE SER FEITO DE UMA SÓ VEZ COM O CÓDIGO ABAIXO:

```
echo ; printenv SHELL ; echo ; cd ; mkdir pasta-estudos ; cd pasta-estudos/ ; > 01-script.sh ; chmod a+x 01-script.sh ; echo -e '#!/bin/bash\n# Um comentário\necho "Olá, mundo!"\n# Fim do Shell Script' > 01-script.sh ; sh 01-script.sh ; echo ; ./01-script.sh ; echo ; bash 01-script.sh ; ./01-script.sh ; sh 01-script.sh
```

.

Bloco de comentários tudo que estiver entre : bin-script.sh ; chmod a+x bin-script.sh ; cd ; cd Downloads/ ; bin-script.sh ; sleep 3 ; echo ; echo 'Estamos em:' ; echo ; sleep 2 ; echo ; pwd ; echo ; sleep 3 ; echo ; echo 'Funcionou porque o script foi executado na pasta Downloads' ; echo

.

VOCÊ NOTOU SE JÁ É CAPAZ DE LER ESTE CÓDIGO ACIMA E ENTENDER O QUE ELE FAZ?

.

ACRESCENTANDO O NOVO VALOR À VARIÁVEL $PATH:

```
cd

whoami

PATH=$PATH:/home/seu-usuario-whoami/bin/
```

.

PARA TORNAR O VALOR, QUE VOCÊ QUISER, PERMANENTE, TEM GENTE QUE FAZ O AJUSTE DA VARIÁVEL DENTRO DO ARQUIVO ".profile" OU ".bash_profile"

```
export PATH="$HOME/bin:$HOME/.local/bin:$PATH"
```

.

Exemplo:

```
cd

pwd

ls -a

echo 'PATH=$PATH:/home/seu-usuario-whoami/bin/' >> .profile

ls -a -t
```

.

PARA FUNCIONAR, TENTE EXECUTAR:

```
cd ; source .profile ; source .bashrc
```

.

PODE TAMBÉM ATUALIZAR E REINICIAR O SISTEMA. COMANDO PARA REBOOT:

```
sudo shutdown -r now
```

.

VERIFICANDO A NOVA VARIÁVEL:

```
echo $PATH
```

.

LINUX SHELL SCRIPT BRINCANDO COM 5 VARIÁVEIS:

```
#!/bin/bash

clear

# Este script testa o uso de variáveis

# Definindo 5 variáveis

echo ; echo 'Definindo 5 variáveis em 10 segundos.' ; sleep 3 ; echo

VALOR1='ls -t' ; sleep 2 ; VALOR2='pwd' ; sleep 2 ; VALOR3='cal' ; sleep 2
; VALOR4='uptime' ; sleep 2 ; VALOR5='whoami' ; sleep 2

# Executando as 5 variáveis

echo 'Executando as 5 variáveis com explicações.' ; sleep 4 ; echo

echo 'Vamos listar o conteúdo desta pasta:' ; echo ; $VALOR1 ; sleep 4 ;
echo ; echo 'Vamos saber onde estamos localizados no sistema:' ; sleep 4 ;
echo ; $VALOR2 ; sleep 4 ; echo ; echo 'Vamos ver o calendário atual' ;
sleep 4 ; echo ; $VALOR3 ; echo ; echo 'Vamos saber o tempo de
funcionamento da máquina:' ; sleep 4 ; echo ; $VALOR4 ; sleep 4 ; echo ;
echo 'Vamos saber qual é o usuário logado' ; sleep 4 ; echo ; $VALOR5 ;
echo

sleep 5

echo 'Executando apenas as variáveis:' ; echo ; $VALOR1 ; sleep 2 ;
$VALOR2 ; sleep 2 ; $VALOR3 ; sleep 2 ; $VALOR4 ; sleep 2 ; $VALOR5

# Removendo as 5 variáveis pois este script é apenas um teste

echo ; echo 'Removendo as 5 variáveis em 10 segundos, pois este script é
apenas um teste' ; sleep 4 ; echo

unset VALOR1 ; sleep 2 ; unset VALOR2 ; sleep 2 ; unset VALOR3 ; sleep 2 ;
unset VALOR4 ; sleep 2 ; unset VALOR5 ; sleep 2

echo ; echo '10 seg para testar as variáveis que não devem ecoar valores'
; sleep 4 ; echo
```

```
echo 'Testando as variáveis:' ; echo ; $VALOR1 ; sleep 2 ; $VALOR2 ; sleep 2 ; $VALOR3 ; sleep 2 ; $VALOR4 ; sleep 2 ; $VALOR5

exit

# Fim do script
```

.

LINUX SHELL SCRIPT - OlaUsuario.sh

.

```
#!/bin/bash

# Titulo: 01-olauser.sh

clear

USERL='Usuário-Linux'

echo

echo "Olá $USERL"

sleep 2

echo

echo "Tchau $USERL"

sleep 2

clear

exit

# Fim do script
```

.

ABRA O TERMINAL PELO MENU DO SISTEMA. ABRA O GERENCIADOR DE ARQUIVOS. COLOQUE OS DOIS LADO A LADO NA TELA DO COMPUTADOR. EXECUTE O SCRIPT ACIMA COPIANDO E COLANDO NO TERMINAL O CÓDIGO ABAIXO QUE VAI EXECUTAR O SCRIPT E DEPOIS VAI APAGAR O SCRIPT:

.

```
cd ; mkdir praticando-shellscript ; cd praticando-shellscript/ ; echo -e '#!/bin/bash\n \n# Titulo: 01-olauser.sh\n \nclear\n \nUSERL=Usuário-Linux\necho\necho Olá $USERL\n \nsleep 4\necho\necho Tchau $USERL\n \nsleep 4\necho\n \nsleep 2\nclear\n \nexit\n \n# Fim do script' > 01-olauser.sh ; chmod a+x 01-olauser.sh ; sh 01-olauser.sh ; sleep 2 ; cd .. ; unset USERL ; rm -rf praticando-shellscript/ ; echo ; pwd ; sleep 2 ; echo ; ls -t ; sleep 2 ; $USERL
```

.

LINUX SHELL SCRIPT IMPRIMIR SAÍDA COLORIDA:

.

Um script pode usar sequências de escape para produzir textos coloridos no terminal. As cores são representadas por códigos, temos 9 códigos:

reset = 0

black = 30

red = 31

green = 32

yellow = 33

blue = 34

magenta = 35

cyan = 36

white = 37

.

O caractere de escape para vermelho é: "\e[1;31m" após o texto em vermelho, usa "\e[0m" para resetar a cor voltando ao padrão. Substitua o código 31 por outra cor que desejar. Temos 9 códigos: 0, 30, 31, 32, 33, 34, 35, 36, 37.

.

Exemplo:

```
echo -e "\e[1;36m texto que vai ficar colorido \e[0m"
```

.

PARA IMPRIMIR UM TEXTO COLORIDO USE ESTES EXEMPLOS ABAIXO:

.

```
echo -e "\e[1;34m Este é o texto em azul! \e[0m"

echo -e "\e[1;30m Este é o texto em preto! \e[0m"

echo -e "\e[1;32m Este é o texto em verde! \e[0m"

echo -e "\e[1;33m Este é o texto em amarelo! \e[0m"

echo -e "\e[1;35m Este é o texto em magenta! \e[0m"

echo -e "\e[1;36m Este é o texto em cyan! \e[0m"

echo -e "\e[1;37m Este é o texto em branco! \e[0m"
```

echo -e "\e[1;37m Este é o texto em branco! \e[0m"

.

PODEMOS IMPRIMIR TODOS ESTE TEXTOS AO MESMO TEMPO PARA TESTAR.

.

Copio o código que eu escrevi abaixo e colo no meu terminal:

.

```
echo -e "\e[1;34m Este é o texto em azul! \e[0m" ; sleep 3 ; echo -e "\e[1;30m Este é o texto em preto! \e[0m" ; sleep 3 ; echo -e "\e[1;32m Este é o texto em verde! \e[0m" ; sleep 3 ; echo -e "\e[1;33m Este é o texto em amarelo! \e[0m" ; sleep 3 ; echo -e "\e[1;35m Este é o texto em magenta! \e[0m" ; sleep 3 ; echo -e "\e[1;36m Este é o texto em cyan! \e[0m" ; sleep 3 ; echo -e "\e[1;37m Este é o texto em branco! \e[0m"
```

.

DÁ PARA CRIAR UM SHELL SCRIPT COM ISTO VEJA SÓ:

.

```
#!/bin/bash

clear

echo

echo -e "\e[1;34mVamos criar uma \e[1;31mpasta. \e[0m"

echo

sleep 4

mkdir pasta-teste

echo

echo -e "\e[1;30mVamos ver se a \e[1;31mpasta \e[1;30mfoi criada. \e[0m"

echo

sleep 4

ls -t

echo

sleep 3

echo -e "\e[1;32mVamos criar um \e[1;31marquivo de texto \e[1;32mvazio \e[0m"

echo
```

```
sleep 4

> texto-teste.txt ; echo ; ls -t ; echo ; sleep 5 ; echo

echo -e "\n\e[1;33mVamos \e[1;31mescrever \e[1;33me \e[1;35mmover
\e[1;33mo \e[1;32mtexto-teste.txt \e[1;33mpara:\n \n\e[1;34mpasta-teste\n
\e[0m"

echo

sleep 4

echo -e "Esta frase\nserá escrita\nem\ntexto-teste.txt" > texto-teste.txt
; sleep 3 ; mv texto-teste.txt pasta-teste

echo

echo -e "\n\e[1;35mEntrar em \e[1;32mpasta-teste \e[1;35me conferir o
conteúdo dela \e[0m\n"

echo

cd pasta-teste/ ; echo ; ls -t ; sleep 4 ; echo ; pwd ; echo ; sleep 4

echo

echo -e "\e[1;36mCopiando \e[1;37mtexto-teste.txt \e[1;36mpara
\e[1;34mtexto-teste2.txt \e[0m"

echo

sleep 4

cp texto-teste.txt texto-teste2.txt ; echo ; ls -t ; echo ; sleep 4 ; echo
; pwd ; echo ; sleep 4

echo -e "\e[1;37mFim do script. \e[1;31mPode apagar tudo \e[1;32musando o
\e[1;37mmouse. \e[0m"

echo

sleep 4

echo -e "\e[1;37mOu pode executar o comando: \e[1;31mrm -rf pasta-
teste\e[1;32m mas antes confere os arquivos de texto.\e[0m"

echo

sleep 4

exit

# Fim do script

.

COMANDOS MAIS USADOS EM SHELL SCRIPT NO LINUX
```

.

ASPAS SIMPLES ' E ASPAS DUPLAS ":

.

Aspas duplas permitem interpretar caracteres especiais.

Aspas simples desabilitam esta interpretação.

Ambas são úteis.

.

CARACTERES DE ESCAPE:

```
echo "Hello \"world\"!"
```

.

TODO SCRIPT ESCRITO PARA RODAR NO BASH COMEÇA COM:

```
#!/bin/bash
```

Após "#!/bin/bash" de um espaço entre linhas e então pode começar a digitar comandos.

.

Exemplo para executar:

```
#!/bin/bash

clear

echo ; date ; echo ; sleep 4

echo ; cal ; echo ; sleep 4

echo ; uptime ; echo ; sleep 4

echo ; df -h ; echo ; sleep 4

echo ; free -html ; echo ; sleep 4

echo ; whoami ; echo ; sleep 4

echo ; pwd ; echo ; sleep 4

echo ; ls -at ; echo ; sleep 4

echo ; whereis bash ; echo ; sleep 4

echo ; echo 'Este é o fim do script 01-script.sh' ; echo ; sleep 4

exit

# Fim do script
```

Fim do script

.

ESTE SCRIPT ÚTIL E INOFENSIVO ACIMA SERÁ SALVO NA PASTA HOME, A PASTA DA CASINHA, USANDO UM EDITOR DE TEXTO E TERÁ O NOME DE:

01-script.sh

.

Posso melhorar/tornar mais amigável este script acima explicando sobre cada comando:

.

```
#!/bin/bash

clear

echo ; echo 'Hoje é data:' ; echo ; sleep 2

echo ; date ; echo ; sleep 4

echo ; echo 'Hoje pelo calendário é:' ; echo ; sleep 2

echo ; cal ; echo ; sleep 4

echo ; echo 'Esta máquina está funcionando a:' ; echo ; sleep 2

echo ; uptime ; echo ; sleep 4

echo ; echo 'Sobre o tamanho desta pasta:' ; echo ; sleep 2

echo ; df -h ; echo ; sleep 6

echo ; echo 'Sobre a memória RAM:' ; echo ; sleep 2

echo ; free -html ; echo ; sleep 6

echo ; echo 'Você está logado como:' ; echo ; sleep 2

echo ; whoami ; echo ; sleep 4

echo ; echo 'Você está em:' ; echo ; sleep 2

echo ; pwd ; echo ; sleep 4

echo ; echo 'Neste diretório/pasta tem:' ; echo ; sleep 2

echo ; ls -at ; echo ; sleep 6

echo ; echo 'O Bash está em:' ; echo ; sleep 2

echo ; whereis bash ; echo ; sleep 4

echo ; echo 'Este é o fim do script 01-script.sh' ; echo ; sleep 4

exit
```

```
exit

# Fim do script
```

.

No Linux o script deve ter permissão de execução, isto pode ser feito abrindo o terminal pelo menu do sistema e executando o comando:

```
chmod +x 01-script.sh
```

.

Depois de salvo você tem que executar o arquivo, dessa forma:

```
./01-script.sh
```

.

Viu alguma utilidade neste pequeno script?

Então siga adiante.

.

IMPORTANTE:

Para estudar shell script tem que ser como usuário normal. Se você está acessando o sistema como usuário administrador (root), saia e entre como um usuário normal. É muito perigoso estudar shell usando o superusuário, você pode danificar o sistema com um comando errado.

Ok, continuemos.

.

Para exibir um manual do bash ou mesmo do comando 'chmod', digito na linha de comando:

```
man bash

man chmod
```

.

É possível executar o arquivo mesmo sem modificar a permissão de execução, por exemplo, se for um arquivo escrito para ser executado pelo bash, usar:

```
sh ./"Nome do arquivo, sem aspas"
```

.

SHELL

É importante saber o que é um Shell.

Na linha de comandos de um shell, podemos utilizar diversos comandos um após o outro, ou mesmo combiná-los numa mesma linha.

Se colocarmos diversas linhas de comandos em um arquivo texto simples, teremos em mãos um Shell Script, ou um script em shell, já que Script é uma descrição geral de qualquer programa escrito em linguagem interpretada, ou seja, não compilada.

Outros exemplos de linguagens para scripts são o PHP, Perl, Python, JavaScript e muitos outros. Podemos então ter um script em php, um script perl e assim em diante.

Uma vez criado, um ShellScript pode ser reutilizado quantas vezes for necessário.

Seu uso, portanto, é indicado na automação de tarefas que serão realizadas mais de uma vez.

Todo sistema Unix e similares são repletos de scripts em shell para a realização das mais diversas atividades administrativas e de manutenção do sistema.

Os arquivos de lote (batch - arquivos *.bat) do Windows são também exemplos de ShellScripts, já que são escritos em linguagem interpretada e executados por um Shell do Windows, em geral o command.com ou hoje em dia o cmd.exe.

Os Shells do Unix, porém, são inumeras vezes mais poderosos que o interpretador de comandos do Windows, podendo executar tarefas muito mais complexas e elaboradas.

.

OS SCRIPTS SHELL PODEM SER AGENDADOS PARA EXECUÇÃO ATRAVÉS DA TABELA CRONTAB, ENTRE OUTRAS COISAS.

.

O shell é uma ferramenta indispensável aos administradores de sistemas Unix.

O Shell mais comum e provavelmente o que possui mais scripts escritos para ele é também um dos mais antigos e simples, o sh.

Este shell está presente em todo o sistema tipo Unix, incluído o Linux, FreeBSD, AIX, HP-UX, OpenBSD, Solaris, NetBSD, Irix, etc. Por ser o shell nativo mais comum é natural que se prefira escrever scripts para ele, tornando o script mais facilmente portável para outro sistema.

Os Shells não estão diretamente associados a um ou outro tipo de Unix, embora várias empresas comerciais tenham suas próprias versões de Shell. No software livre o Shell utilizado em um sistema em geral é exatamente o mesmo utilizado em outro. Por exemplo, o bash encontrado no Linux é o mesmo shell bash encontrado no FreeBSD e pode também facilmente ser instalado no Solaris, Windows através do Cygwin [1] ou outros sistemas Unix comerciais para passar a ser utilizado como interface direta de comandos ou como interpretador de scripts. O mesmo acontece com o tcsh e

vários outros shells desenvolvidos no modelo de software livre.

.

INTERAGIR COM O USUÁRIO

.

Para o script ficar mais completo, vamos colocar uma interação mínima com o usuário, pedindo uma confirmação antes de executar os comandos.

.

```
#!/bin/bash

clear

echo "Vou buscar os dados do sistema. Posso continuar? [S/n] "

read RESPOSTA

test "$RESPOSTA" = "n" && exit

echo ; echo "Data e Horário:" ; echo

date

echo

echo "Uso do disco:" ; echo

df -ht

echo

echo "Usuários conectados:" ; echo

w

echo ; echo "Seu nome de login é:"

whoami

echo

exit

# Fim do script
```

.

O comando "read" leu o que o usuário digitou e guardou na variável RESPOSTA. Logo em seguida, o comando "test" verificou se o conteúdo dessa variável era "n". Se afirmativo, o comando "exit" foi chamado e o script foi finalizado. Nessa linha há vários detalhes importantes:

O conteúdo da variável é acessado colocando-se um cifrão "$" na frente

O comando test é útil para fazer vários tipos de verificações em textos e arquivos

O operador lógico "&&", só executa o segundo comando caso o primeiro tenha sido OK. O operador inverso é o "||"

.

MELHORAR O CÓDIGO DO SCRIPT

Com o tempo, o script vai crescer, mais comandos vão ser adicionados e quanto maior, mais difícil encontrar o ponto certo onde fazer a alteração ou corrigir algum erro. Para poupar horas de estresse, e facilitar as manutenções futuras, é preciso deixar o código visualmente mais agradável e espaçado, e colocar comentários esclarecedores.

.

```
#!/bin/bash

# nome-do-script - script que mostra informações sobre o sistema

# Autor: Fulano da Silva

# Pede uma confirmação do usuário antes de executar

clear

echo "Vou buscar os dados do sistema. Posso continuar? [S/n]"

read RESPOSTA

# Se ele digitou 'n', vamos interromper o script

test "$RESPOSTA" = "n" && exit

# O date mostra a data e a hora correntes

sleep 3 ; echo "Data e Horário:" ; echo

date

sleep 3

echo

# O df mostra as partições e quanto cada uma ocupa no disco

echo "Uso do disco:"

sleep 3

echo

df

echo
```

```
sleep 6

# O w mostra os usuários que estão conectados nesta máquina

echo "Usuários conectados:"

sleep 3

echo

w

sleep 3

echo

# Fim do script
```

.

Basta iniciar a linha com um "#" e escrever o texto do comentário em seguida. Estas linhas são ignoradas pelo shell durante a execução. O cabeçalho com informações sobre o script e seu autor também é importante para ter-se uma visão geral do que o script faz, sem precisar decifrar seu código. Também é possível colocar comentários no meio da linha # como este

.

FERRAMENTAS PARA FUNÇÕES

.

O COMANDO RETURN INFORMA O FIM DA FUNÇÃO. EXEMPLO:

```
MinhaFuncao(){
echo "Isto será exibido"
return
echo "Isso não será exibido, pois está depois de return"
}

MinhaFuncao
```

.

Se criar uma variável dentro de uma função, mesmo chamando ela fora da função, o valor da variável será exibido, pois ela será tratada como variável GLOBAL, caso tente imprimi-la, para que o valor de uma variável seja exibido somente dentro da função, precisamos usar o comando local antes da variável para que não seja exibido. Para entender isto só mesmo com exemplo na prática. Vamos a ele:

.

EU ABRO O EDITOR DE TEXTO E COLO O TEXTO ABAIXO NELE:

```
#!/bin/bash
```

```
# Meu comentário

VARIAVEL="Olá Mundo do Shell Script!"

echo $VARIAVEL

sleep 2

echo

echo "Vou buscar os dados do sistema. Posso continuar? [S/n] "

read RESPOSTA

test "$RESPOSTA" = "n" && exit

echo ; echo "Data e Horário:" ; echo

date

echo

echo "Uso do disco:" ; echo

df -hi

echo

echo "Usuários conectados:" ; echo

w

echo ; echo "Seu nome de login é:"

whoami

echo

ARRAY5=("Shell" "Madrugada" "Script" "Amanhece" "Linux")

echo "O ARRAY5 possui ${#ARRAY5[@]} elemento(s)"

MinhaFuncao(){
echo "Eu estou passando $# parâmetros"
#return
echo "Muitas coisas"
}

MinhaFuncao $@

echo 'Se o resultado é zero está tudo certo!'

sleep 2

echo

echo $?
```

```
echo $?

exit

# Fim do script: 09-ferramentas-funcoes.sh

.

SALVO COMO: 09-ferramentas-funcoes.sh

.

DOU PODER DE EXECUÇÃO:

chmod a+x 09-ferramentas-funcoes.sh

.

EXECUTO:

sh 09-ferramentas-funcoes.sh

sh 09-ferramentas-funcoes.sh Linux Brasil GNU

.

AGORA ALTERO O SCRIPT 09-ferramentas-funcoes.sh (REMOVO A TRALHA # DE
RETURN) E EXECUTO DE NOVO:

.

#!/bin/bash

# Meu comentário

VARIAVEL="Olá Mundo do Shell Script!"

echo $VARIAVEL

sleep 2

echo

echo "Vou buscar os dados do sistema. Posso continuar? [S/n] "

read RESPOSTA

test "$RESPOSTA" = "n" && exit

echo ; echo "Data e Horário:" ; echo

date

echo

echo "Uso do disco:" ; echo

df -hi
```

```
echo

echo "Usuários conectados:" ; echo

w

echo ; echo "Seu nome de login é:"

whoami

echo

ARRAY5=("Shell" "Madrugada" "Script" "Amanhece" "Linux")

echo "O ARRAY5 possui ${#ARRAY5[@]} elemento(s)"

MinhaFuncao(){
echo "Eu estou passando $# parâmetros"
return
echo $VARIAVEL
echo "Muitas coisas"
}

MinhaFuncao $@

echo 'Se o resultado é zero está tudo certo!'

sleep 2

echo

echo $?

exit

# Fim do script: 09-ferramentas-funcoes.sh
```

.

COPIO E COLO O COMANDO ABAIXO NO TERMINAL:

```
sh 09-ferramentas-funcoes.sh Amanhecer ANOITECER Estudar DORMIR
```

.

CONSTANTES NÃO MUDAM. EXEMPLO:

.

```
#!/bin/bash

MinhaFuncao(){
local OLA="Olá, mundo!"
echo "Eu estou passando $# parâmetro(s)"
return
echo $OLA
echo "Muita coisa"
```

```
echo "Nada coisa
}

MinhaFuncao

declare -r MinhaConstante='Estamos constantes'

echo $MinhaConstante

# Fim do 3d-script-constante.sh
```

.

PODER DE EXECUÇÃO:

```
chmod +x 3d-script-constante.sh
```

.

EXECUTAR O SCRIPT:

```
sh 3d-script-constante.sh

sh 3d-script-constante.sh LINUX BASH
```

.

PARA APAGAR CONSTANTES E FUNÇÕES É O MESMO PROCEDIMENTO DE APAGAR VARIÁVEIS (unset).

.

CONDIÇÕES EM SHELL SCRIPT

O Shell tem estruturas para se fazer condições. Por exemplo o comando TEST. O comando TEST é útil para fazer vários tipos de verificações em textos e arquivos, testa o conteúdo de uma string, poder ser um arquivo, uma variável, compara valores numéricos ou não. O test avalia uma determinada condição dada e se ela for verdadeira a variável $? é retornada com o valor zero (0) e se falsa o valor é 1.

.

COM O COMANDO TEST PODEMOS VER SE É UM DISPOSITOVO DE:

-b (Bloco), -c (Caractere), -d (Diretório), -e (Se existe), -f (Se é um arquivo normal), -G (O grupo do arquivo é o do usuário atual), -L (O arquivo é um link simbólico), -r (O arquivo tem permissão de leitura), -s (O tamanho é maior que zero), -nt (O arquivo é o mais recente), -ot (O arquivo é mais antigo), -w (Se o arquivo tem permissão de escrita), -x Se o arquivo tem permissão de execução), -ef (Se o arquivo é o mesmo), e por aí vai... etc;

.

Podemos usar a COMPARAÇÃO NUMÉRICA no comando test. Veja estas opções abaixo por exemplo:

.

-lt (É menor que)

-gt (É maior que)

-le (É menor igual)

-ge (É maior igual)

-eq (É igual)

-ne (É diferente)

.

COMPARAÇÃO DE STRINGS:

.

= (É igual)

!= (É diferente)

-n (É não nula)

-z (É nula)

.

OPERADORES LÓGICOS:

! (NÃO lógico)

-a (E lógico) (AND)

-o (OU lógico)

.

VAMOS EXECUTAR OS EXEMPLOS ABAIXO NO TERMINAL:

test 1 = 1 ; echo $? (É igual a...)

test 1 = 2 ; echo $? (É igual a...)

test 1 != 2 ; echo $? (É diferente de...)

test 1 != 1 ; echo $? (É diferente de...)

.

- PRATICANDO O COMANDO TEST -

.

Vamos testar se o arquivo: 08-test-script.sh existe? Se é uma pasta? Se é

um arquivo normal? Execute os comandos abaixo:

touch 08-test-script.sh

ls -t

test -e 08-test-script.sh ; echo $? (Verifica se existe)

test -d 08-test-script.sh ; echo $? (Verifica se é uma pasta)

test -f 08-test-script.sh ; echo $? (Verifica se é um arquivo comum)

test -s 08-test-script.sh (Verifica se o tamanho é maior que zero)

rm -fi 08-test-script.sh

.

Exemplo:

```
~ $rm -fi 08-test-script.sh
rm: remover arquivo comum vazio '08-test-script.sh'? S
```

.

Execute os comandos abaixo:

ls -t

test -e 08-test-script.sh ; echo $?

test -d 08-test-script.sh ; echo $?

test -s 08-test-script.sh

.

SE CORRESPONDE A OPÇÃO ESCOLHIDA A RESPOSTA É ZERO. SE NÃO A RESPOSTA É 1.

.

OS SCRIPTS SHELL PODEM CONTER ESTRUTURAS DE PROGRAMAÇÃO/condição TAIS COMO:

if, them, else, elif, fi

.

ESTRUTURAS DE DECISÃO (if)

If testa um comando e não uma condição. O comando que testa condições é o test. Usamos o test junto com o if. O if é um recurso utilizado para dar sequencia em fluxos de execução baseado em decisões. Cuja sintaxe é:

- Condição Verificada é o teste que definirá se controle deve ser passado para dentro do bloco then.

- Ação são comandos a serem executados em caso verdadeiro da condição

verificada.

.

OPERADORES PARA NÚMEROS

.

-eq Verifica se é igual,

-ne Verifica se é diferente,

-lt Verifica se é menor,

-gt Verifica se é maior,

-le Verifica se é menor ou igual,

-ge Verifica se é maior ou igual.

.

OPERADORES PARA TEXTO

.

!= Verifica se é diferente,

= Verifica se é igual.

.

OPERADORES LÓGICOS

.

! Lógica NOT,

-o Lógica OU, (OR) ou ||,

-a Lógica E, (AND) ou &&.

.

OPERADOR PARA arquivos/

.

-d Verifica se é diretório,

-f Verifica se é arquivo,

-e Verifica se existe.

.

EXEMPLOS A SEREM EXECUTADOS:

.

```
#!/bin/bash

# Este script é para saber se uma variável é maior
# ou menor # do que 10 e mostrar uma mensagem na
# tela informando.
# No fim do script fechamos a condição com fi.
# Não esqueça de fechar a condição com fi, se não dá
# erro.

VARIAVEL=9;
if test "$VARIAVEL" -gt 10
then
echo "é maior que 10"
else
echo "é menor que 10"
fi

# Fim do script: 07w-script-if.sh
```

.

COPIO E COLO O CÓDIGO ABAIXO NO TERMINAL E VEJA O RESULTADO:

.

```
echo -e '#!/bin/bash\n \nVARIAVEL=9;\nif test "$VARIAVEL" -gt 10\n then\n
echo "é maior que 10"\nelse\n echo "é menor que 10"\nfi\n \n# Fim do
script: 07w-script-if.sh' > 07w-script-if.sh ; chmod a+x 07w-script-if.sh
; sh 07w-script-if.sh
```

.

RESULTADO:

```
é menor que 10
```

.

MAIS UM EXEMPLO IF, THEN, ELSE, ELIF, FI:

.

No exemplo abaixo a variável é igual a 10.

Neste caso tem que usar o "elif" se não vai dar erro.

Sem o "elif" o script vai dizer que é MENOR que 10.

É mais ou menos o seguinte, se (if) não é (then) maior, ou (elif) é (then) igual ou então (else) é menor. Fim (fi).

.

EXECUTE NO TERMINAL O SHELL SCRIPT EXEMPLO ABAIXO:

.

```
#!/bin/bash

VARIAVEL=10;
if test "$VARIAVEL" -gt 10
then
echo "é maior que 10"

elif test "$VARIAVEL" -eq 10
then

echo "é igual a 10"
else
echo "é menor que 10"

fi

# Fim do script:
# 08kw-script-if-then-else-elif-fi.sh
```

.

RESULTADO:

```
~ $sh 08kw-script-if-then-else-elif-fi.sh
é igual a 10
~ $
```

.

SE A VARIÁVEL FOSSE 11 EXEMPLO:

.

```
#!/bin/bash

VARIAVEL=11;
if test "$VARIAVEL" -gt 10
then
echo "é maior que 10"

elif test "$VARIAVEL" -eq 10
then

echo "é igual a 10"
else
echo "é menor que 10"

fi

# Fim do script:

# 03kw-script-if-then-else-elif-fi.sh
```

.

O RESULTADO É:

```
~ $sh 03kw-script-if-then-else-elif-fi.sh
é maior que 10
```

.

COPIO E COLO O CÓDIGO ABAIXO NO TERMINAL E VEJA O RESULTADO:

```
echo -e '#!/bin/bash\nVARIAVEL=11;\nif test "$VARIAVEL" -gt 10\nthen\necho "é maior que 10"\nelif test "$VARIAVEL" -eq 10\nthen\necho "é igual a 10"\nelse\necho "é menor que 10"\nfi\n# Fim do script:\n# 03kw-script-if-then-else-elif-fi.sh' > 03kw-script-if-then-else-elif-fi.sh ; chmod a+x 03kw-script-if-then-else-elif-fi.sh ; sh 03kw-script-if-then-else-elif-fi.sh
```

.

Consegue entender (ler) o código acima e sabe o que ele faz?

.

IF VEM ACOMPANHADO DO THEN
ELIF VEM ACOMPANHADO DO THEN
ELSE E O SCRIPT TERMINA COM FI

.

EXECUTE ESTE EXEMPLO NO TERMINAL:

.

```
#!/bin/bash

# Uso de Estrutura de Decisão

clear

echo 'opções'

echo '======'

echo ' -> Data do Sistema'

echo ' -> Uso do Sistema'

read opcao

if [ "$opcao" -eq 1 ]

then

echo 'Data do sistema: ' && date

elif [ "$opcao" -eq 2 ]
```

..

```
then

echo 'Uso do disco: ' && df -Th

fi

# Fim do script: 08cw-script-if.sh
```

.

PODEMOS CRIAR UMA FUNÇÃO

.

```
#!/bin/bash

MinhaFuncao(){
VARIAVEL=$1;
if test "$VARIAVEL" -gt 10
then
echo "é maior que 10"

elif test "$VARIAVEL" -eq 10
then

echo "é igual a 10"
else
echo "é menor que 10"

fi
}

MinhaFuncao $1

# Fim do script:

# 09ks-funcao-if-then-else-elif-fi.sh
```

.

JÁ SABE OS PROCEDIMENTOS. EXECUTE O SCRIPT ACIMA ASSIM:

```
sh 09ks-funcao-if-then-else-elif-fi.sh 8

bash 09ks-funcao-if-then-else-elif-fi.sh 19

./09ks-funcao-if-then-else-elif-fi.sh 868

sh 09ks-funcao-if-then-else-elif-fi.sh 2
```

.

ESTRUTURAS DE REPETIÇÃO (for) (while)

.

ESTRUTURA DE REPETIÇÃO for:

Permite que ações de iteração sejam executadas sobre determinados comandos ou variáveis até que a condição seja satisfeita.

.

```
# !/bin/bash

clear

echo "DIAS DA SEMANA"

for dia in seg ter qua qui sex sab dom

do

echo "$dia"

done

# Fim do script 7s-etrut-repet-for-while.sh
```

.

RESULTADO:

.

```
DIAS DA SEMANA
seg
ter
qua
qui
sex
sab
dom
~ $
```

.

ESTRUTURA DE REPETIÇÃO while:

Em situações onde sabemos até onde o loop irá realizar uma contagem o ideal é usar o for entretanto em cenarios onde a iteração deve cessar somente após se satisfazer uma condição o uso do laço while é mais indicado. Ex:

.

```
# /bin/bash

clear

var=1

while [ $var -le 7 ]

do
```

do

```
echo "Valor de var: $var"

var=$((var+1))

done

# Fim do script 7g-estr-rep-while.sh
```

.

Resultado:

```
Valor de var: 1
Valor de var: 2
Valor de var: 3
Valor de var: 4
Valor de var: 5
Valor de var: 6
Valor de var: 7
~ $
```

.

UM USO INTERESSANTE DO WHILE PARA VER O LOOP FUNCIONANDO:

.

Por vezes queremos acompanhar a cópia de um arquivo na console do Linux e o caminho mais normal é abrir um outro terminal e ficar repetitivamente executando o comando ls, ou algum outro comando, haja dedo para apertar a seta pra cima e enter, seta pra cima e enter, seta pra cima e enter. Podemos resolver isto usando o comando while de forma bem simples, por exemplo se quisermos executar um ls por várias vezes, podemos fazer assim:

.

OBS: Execute um de cada vez, parar o comando com Ctrl+C.

.

1

```
while true; do ls; done;
```

2

```
while true; do ls; echo; sleep 5; done;
```

3

```
while true; do ls; echo; sleep 5; clear; done;
```

.

ISTO VAI EXECUTAR O COMANDO LS ATÉ PRESSIONARMOS CTRL + C PARA QUEBRÁ-LO.

.

FUNÇÕES E ARGUMENTOS

.

Ex:

.

```
# !/bin/bash
# REALIZAR BACKUP DO DIR
echo -e " \033[1;33m Digito o caminho de origem.: \033[0m "
read DIR_ORIGEM
clear
echo -e " \033[1;34m Digito o caminho de destino.: \033[0m "
read DIR_DESTINO
clear
verifica_argumentos(){
if [ $# -lt 1 ];
then
echo "Faltou informar um dos argumentos (parametros) necessarios!"
exit 1
fi
}
copia_arquivos(){
verifica_argumentos
clear
echo "Realizando backup..."
#Verificando se o dir de destino existe
if ! [ -d $DIR_DESTINO ]
then
mkdir $DIR_DESTINO
echo "Diretorio de Destino Criado"
```

```
fi

#COPIANDO ARQUIVOS

for arq in `ls $DIR_ORIGEM`

do

cp /$DIR_ORIGEM/$arq $DIR_DESTINO/$arq.bak

done

}

copia_arquivos

# Fim do script 08b-funcoes-e-arg.sh
```

.

DEFINIÇÕES DE VARIÁVEIS E ESCOPO DESTAS

.

Variáveis são definidas pela nomenclatura NOME_VARIAVEL="Valor da Variável". O valor pode ser tanto numérico quanto texto.

.

```
Nome="Joel"
```

.

Se quisermos acessá-la, basta fazer referência a ela com o caractere $ (cifrão) antes do nome: o comando echo $Nome, por exemplo, retornará a palavra "Joel".

.

Se quiser sabe informações sobre os sistemas de arquivo nos quais cada ARQUIVO reside ou, por padrão, sobre todos os sistemas de arquivos posso abrir um terminal e digitar:

```
VarInfo="df -h"
```

.

Depois digito no terminal "$VarInfo" sem aspas.

.

VARIÁVEIS DE AMBIENTE

.

As variáveis de ambiente independem da definição do usuario. Elas são criadas automaticamente, no momento em que se faz o login no sistema.

.

Ex:

PATH: define diretórios de procura por programas executados no shell;

USER: informa o nome do usuário do shell;

HOME: informa o caminho do diretório home do usuário;

PWD: diretório atual;

.

As variáveis são a base de qualquer script. É dentro delas que os dados obtidos durante a execução do script serão armazenados. Para definir uma variável, basta usar o sinal de igual "=" e para ver seu valor, usa-se o "echo":

.

Executo estes comandos abaixo no terminal:

.

```
VARIAVEL="um dois tres"

echo $VARIAVEL

echo $VARIAVEL $VARIAVEL
```

.

Para remover a variável acima:

```
unset VARIAVEL
```

.

Teste:

```
echo $VARIAVEL
```

.

É possível armazenar a saída de um comando dentro de uma variável. Ao invés de aspas, o comando deve ser colocado entre "$(...)", execute no terminal os comandos abaixo:

```
HOJE=$(date)

echo "Hoje é: $HOJE"

sleep 2

unset HOJE
```

```
echo $HOJE

HOJE=$(ls)

echo "O conteúdo desta pasta tem: $HOJE"

sleep 2

unset HOJE

echo $HOJE

HOJE=$(free -hmt)

echo "Informando sobre a memória desta máquina: $HOJE"

sleep 2

unset HOJE

echo $HOJE
```

.

EXEMPLOS DE USO DO SHELL SCRIPT:

.

Apagar arquivos velhos - Apagar periodicamente arquivos mais velhos que 30 dias do diretório /tmp:

.

```
#!/bin/bash

cd /tmp

find . -type f -mtime +30 -delete

# Fim do script
```

.

Este seria o conteúdo de um shell script que sempre que fosse executado apagaria arquivos com data de modificação maior que 30 dias a partir do diretório /tmp do sistema de arquivos.

.

Notem que ele é nada mais do que uma associação de 2 comandos (cd e find) em um arquivo para facilitar a repetição da tarefa. Este poderia ser, por exemplo, o conteúdo do arquivo /bin/limpatmp.sh e poderíamos chamar este script pela linha de comandos sempre que desejássemos repetir esta ação:

.

```
$ limpatmp.sh
```

.

Onde o símbolo "$" representa o prompt de comandos. Do ponto de vista do usuário este seria mais um comando disponível para uso.

.

Os scripts em shell são também muito empregados junto à inicialização do sistema (para auto iniciar tarefas) ou como mini aplicativos, que facilitam tarefas dos usuários, tais como montagem de dispositivos, menus de ajuda, etc.

.

Sua primeira linha obrigatoriamente começa com um "#!" (que não se deve confundir com um comentário qualquer, pois realmente é uma exceção; este par se chama, em inglês, de shebang), informando diretamente ao núcleo (kernel) qual interpretador ele deverá usar, juntamente com seu caminho, de acordo com a necessidade de cada caso. Exemplo:

```
#!/bin/bash
```

.

Em seguida, são adicionados os comandos desejados, um por linha, ou separados por ponto e vírgula. Exemplo:

.

```
mount -t reiserfs /dev/hda1 /mnt/hda1

ls /mnt/hda1

cp -r /mnt/hda1/* /home/user/backup

umount /dev/hda1
```

.

Por fim, dá-se a permissão de execução a este arquivo de texto simples ("chmod +x arquivo").

.

DATA ANTERIOR

.

```
#!/bin/bash

# Função em Bash para retornar a data anterior, levando em conta o mês e ano.

fn_data_anterior()

{
```

```
DIA=$D

MES=$M

ANO=$A

# Dado DIA, MES e ANO numéricos, obtém a data do dia anterior

DIA=`expr $DIA - 1`

if [ $DIA -eq 0 ]; then

MES=`expr $MES - 1`

if [ $MES -eq 0 ]; then

MES=12

ANO=`expr $ANO - 1`

fi

DIA=`cal $MES $ANO`

DIA=`echo $DIA | awk '{ print $NF }'`

fi

}

ano=`date +%Y`;

mes=`date +%m`;

let dia=10\#`date +%d`;

if (( $dia". Para guardar a saída do comando anterior no arquivo "saida", basta fazer:

cat /etc/passwd | grep root | cut -c1-10 > saida

cat saida

.

REPETINDO: O COMANDO TEST

.

O canivete suíço dos comandos do shell é o "test", que consegue fazer vários tipos de testes em números, textos e arquivos. Ele possui várias opções para indicar que tipo de teste será feito, algumas delas:

1) -lt Núm. é menor que (LessThan)

2) -d É um diretório
```

3) -gt Núm. é maior que (GreaterThan)

4) -f É um arquivo normal

5) -le Núm. é menor igual (LessEqual)

6) -r O arquivo tem permissão de leitura

7) -ge Núm. é maior igual (GreaterEqual)

8) -s O tamanho do arquivo é maior que zero

9) -eq Núm. é igual (EQual)

10) -w O arquivo tem permissão de escrita

11) -ne Núm. é diferente (NotEqual)

12) -nt O arquivo é mais recente (NewerThan)

13) = String é igual

14) -ot O arquivo é mais antigo (OlderThan)

15) != String é diferente

16) -ef O arquivo é o mesmo (EqualFile)

17) -n String é não nula

18) -a E lógico (AND)

19) -z String é nula

20) -o OU lógico (OR)

.

SCRIPT QUE TESTA ARQUIVOS

Tente fazer um script "testa-arquivos", que pede ao usuário para digitar um arquivo e testa se este arquivo existe. Se sim, diz se é um arquivo ou um diretório.

.

CONCEITOS MAIS AVANÇADOS:

.

CASE

.

O CASE É PARA CONTROLE DE FLUXO, TAL COMO É O IF. Mas enquanto o if testa expressões não exatas, o case vai agir de acordo com os resultados exatos. Abre com case e fecha com esac. Se nao fizer assim dá erro. Vejamos uns exemplos:

exemplos:

.

```
case $VARIAVEL in
10) echo "é 10" ;;
9) echo "é 9" ;;
7|8) echo "é 7 ou 8" ;;
*) echo "é menor que 6 ou maior que 10" ;;
esac
```

.

```
case $1 in
parametro1) comando1 ; comando2 ;;
parametro2) comando3 ; comando4 ;;
*) echo "Você tem de entrar com um parâmetro válido" ;;
esac
```

.

AQUI ACIMA ACONTECEU O SEGUINTE: o case leu a variável $1 (que é o primeiro parâmetro passado para o programa), e comparou com valores exatos. Se a variável $1 for igual à “parametro1″, então o programa executará o comando1 e o comando2; se for igual à “parametro2″, executará o comando3 e o comando4, e assim em diante. A última opção (*), é uma opção padrão do case, ou seja, se o parâmetro passado não for igual a nenhuma das outras opções anteriores, esse comando será executado automaticamente. Com o case fica muito mais fácil criar uma espécie de “menu” para o shell script do que com o if.

.

EXEMPLO DE SCRIPT PARA EXECUTAR NO TERMINAL:

```
#!/bin/bash

# Aprendendo shell script

MinhaFuncao (){

case $1 in
10) echo "é 10" ;;
9) echo "é 9" ;;
7|8) echo "é 7 ou 8" ;;
*) echo "é menor que 6 ou maior que 10" ;;
esac

}

MinhaFuncao $1

# Fim script 07hr-funcao-case-esac.sh
```

.

RESULTADOS:

```
~ $sh 07hr-funcao-case-esac.sh 8
é 7 ou 8

~ $sh 07hr-funcao-case-esac.sh 15
é menor que 6 ou maior que 10

~ $sh 07hr-funcao-case-esac.sh 9
é 9
```

.

REPETINDO: If, for e while

.

Assim como qualquer outra linguagem de programação, o shell também tem estruturas para se fazer condicionais e loop. As mais usadas são if, for e while.

.

ATÉ ESTE PONTO EM QUE ESTAMOS, JÁ SABEMOS O BÁSICO, O NECESSÁRIO PARA SE FAZER UM SCRIPT DE FUNCIONALIDADE MÍNIMA. E ESTE MÍNIMO PODE FAZER COISAS INCRÍVEIS. AVANÇAR:

case, if, for e while.

.

Comando if:

.

Comando if - else:

```
if ( condição ) {
comandos a serem executados se a condição for verdadeira;
}
else {
comandos a serem executados se a condição for falsa;
}
```

.

COMANDOS DE CONTROLE DE FLUXO

.

Controle de fluxo são comandos que vão testando algumas alternativas, e de acordo com essas alternativas, vão executando comandos. Um dos comandos de controle de fluxo mais usados é certamente o if, que é baseado na lógica “se acontecer isso, irei fazer isso, se não, irei fazer aquilo”.

.

EXEMPLO DE UM PEDAÇO DE CÓDIGO:

.

```
if [ -e $linux ]
then
echo 'A variável $linux existe.'
else
echo 'A variável $linux não existe.'
fi
```

.

O que este pedaço de código faz? O if testa a seguinte expressão: Se a variável $linux existir, então (then) ele diz que que existe com o echo, se não (else), ele diz que não existe. O operador "-e" é pré-definido, e você pode encontrar a listagem dos operadores na tabela:

.

-eq Igual
-ne Diferente
-gt Maior
-lt Menor
-o Ou
-d Se for um diretório
-e Se existir
-z Se estiver vazio
-f Se conter texto
-o Se o usuário for o dono
-r Se o arquivo pode ser lido
-w Se o arquivo pode ser alterado
-x Se o arquivo pode ser executado

.

Exemplo de uso do if:

.

```
if [COMANDOS]

then

comandos

else

comandos

fi

# Fim do script
```

.

Ex:

```
for VAR in LISTA
do
comandos
done
# Fim do script
```

.

Ex:

```
while COMANDO
do
comandos
done
# Fim do script
```

.

REPETINDO: Diferente de outras linguagens, o if testa um comando e não uma condição. Porém como já conhecemos qual o comando do shell que testa condições, é só usá-lo em conjunto com o if. Por exemplo, para saber se uma variável é maior ou menor do que 10 e mostrar uma mensagem na tela informando:

.

Ex:

```
if test "$VARIAVEL" -gt 10
then
echo "é maior que 10"
else
echo "é menor que 10"
fi
# Fim do script
```

.

Há um ATALHO PARA O TEST , que é o comando [. Ambos são exatamente o mesmo comando, porém usar o [deixa o if mais PARECIDO COM O FORMATO TRADICIONAL de outras linguagens:

.

O test pode ser escrito assim:

```
test 1 = 1 ; echo $?

test 1 != 1 ; echo $?
```

.

Ou assim:

```
[ 1 = 1 ] ; echo $?

[ 1 != 1 ] ; echo $?
```

.

Executo os comandos acima no terminal.

.

O Resultado é:

```
~ $test 1 = 1 ; echo $?
0
~ $test 1 != 1 ; echo $?
1
~ $[ 1 = 1 ] ; echo $?
0
~ $[ 1 != 1 ] ; echo $?
1
~ $
```

.

EXEMPLO A SER EXECUTADO:

.

```
#!/bin/bash

VARIAVEL=17;
if [ "$VARIAVEL" -gt 10 ]

then
echo "é maior que 10"

elif [ "$VARIAVEL" -eq 10 ]

then
echo "é igual a 10"

else
echo "é menor que 10"
fi
```

```
# Fim do script 06r-atalho-test.sh
```

.

Resultado:

```
~ $sh 06r-atalho-test.sh
é maior que 10
```

.

OUTRO EXEMPLO A SER EXECUTADO:

.

```
#!/bin/bash

VARIAVEL=4;
if [[ "$VARIAVEL" -gt 10 ]];

then
echo "é maior que 10"

elif [[ "$VARIAVEL" -eq 10 ]];

then
echo "é igual a 10"

else
echo "é menor que 10"
fi

# Fim do script 07r-atalho-test.sh
```

.

Resultado:

```
~ $sh 07r-atalho-test.sh
é menor que 10
```

.

Se usar o [, também é preciso fechá-lo com o], e sempre devem ter espaços ao redor. É recomendado evitar esta sintaxe para diminuir suas chances de erro.

.

Percebeu que neste ponto a gente está sabendo usar de verdade TEST IF THEN ELIF ELSE FI?

.

COMANDO WHILE

.

O while é um laço que é executado enquanto um comando retorna OK. Novamente o test é bom de ser usado. Por exemplo, para segurar o processamento do script enquanto um arquivo de lock não é removido:

.

Ex:

```
while test -f /tmp/lock

do

echo "Script travado..."

sleep 1

done

# Fim do script
```

.

O COMANDO FOR

.

E por fim, o "for" percorre uma lista de palavras, pegando uma por vez:

.

Ex:

```
for numero in um dois três quatro cinco

do

echo "Contando: $numero"

done

# Fim do script
```

.

Uma ferramenta muito útil para usar com o "for" é o "seq", que gera uma seqüência numérica.

.

Para fazer o loop andar 10 passos, pode-se fazer:

```
for passo in $(seq 10)
```

.

O mesmo pode ser feito com o while, usando um contador:

o mesmo pode ser feito com o while, usando um contador:

.

```
i=0

while test $i -le 10

do

i=$((i+1))

echo "Contando: $i"

done

# Fim do script
```

.

LOOPS EM SHELL QUANDO USAR?

.

Precisamos digitar um código talvez com ligeira mudança. Para não ter que repetir a tarefa digitando o código 20 vezes ou mais, usamos loops. Podemos usar vários loops. Tem pelo menos tres tipos de loops para o comando FOR por exemplo.

.

Exemplos abaixo:

.

```
for ((i=0;i 01-while1.sh ; chmod a+x 01-while1.sh ; ./01-while1.sh
```

.

Exemplo:

.

```
~ $cd ; mkdir pasta-while-loop ; cd pasta-while-loop/ ; echo -e '#!/bin/bash\n \nn=1\n \nwhile [ $n -le 5 ]\ndo\necho "Este é o loop $n"\n(( n++ ))\ndone' > 01-while1.sh ; chmod a+x 01-while1.sh ; ./01-while1.sh
Este é o loop 1
Este é o loop 2
Este é o loop 3
Este é o loop 4
Este é o loop 5
pasta-while-loop $
```

.

EXECUTO O SCRIPT ABAIXO:

.

```
#!/bin/bash

# Aprendendo loops Shell Script

# Nome: 06s-loop-while.sh

_INPUT_STRING="Olá"
while [[ "$_INPUT_STRING" != "tchau" ]]
do
echo "Você deseja ficar aqui ?"
read _INPUT_STRING

if [[ $_INPUT_STRING = 'tchau' ]]; then
echo "Você disse Tchau"
else
echo "Você ainda deseja ficar aqui"
fi
done

# Fim do 06s-loop-while.sh
```

.

Resultado:

```
~ $sh 06s-loop-while.sh
Você deseja ficar aqui ?
Sim
Você ainda deseja ficar aqui
Você deseja ficar aqui ?
Claro
Você ainda deseja ficar aqui
Você deseja ficar aqui ?
Tchau
Você ainda deseja ficar aqui
Você deseja ficar aqui ?
tchau
Você disse Tchau
~ $
```

.

O loop serve para scripts avançados, redes de computadores, e outros.

.

CRIE SEU PRÓPRIO EPUB USANDO O PROGRAMA SIGIL

.

LINUX COMANDOS DE MANIPULAÇÃO DE ARQUIVOS PARA USAR EM SHELL SCRIPT

.

Crio uma pasta, abro o terminal nesta pasta criada e executo:

```
.
ls -t -a -h -l
pwd
cd ..
ls -a
pwd
cd -
ls -t
pwd
touch nome-do-arquivo-a-ser-criado1.txt nome2.txt
ls -t
cat nome2.txt
echo 'Olá Mundo!' > nome2.txt
cat nome2.txt
ls ; echo ; echo "E ..." ; sleep 4 ; echo ; ls -t
clear
mkdir pasta-teste
ls -a
ls -t ; echo
rmdir pasta-teste
ls -t ; pwd
mkdir -p pasta-mama/pasta-filha
ls -t ; echo
> nome-do-arquivo-a-ser-criado3.txt ; > nome4.txt
ls -at ; echo
pwd
cp nome2.txt nome3.txt
ls -t ; echo
```

```
mv nome2.txt nome1.txt
ls -t
find -name nome2.txt
find -name nome3.txt
find -name nome1.txt
mv nome1.txt pasta-mama/
find -name nome1.txt > nome1.txt
ls
cat nome1.txt
find ./ -name nome3.txt
find ./ -name calendario-de-2018.txt
rm nome3.txt
find -name nome3.txt
ls -t
pwd
find nome-do-arquivo-a-ser-criado3.txt
rm nome-do-arquivo-a-ser-criado3.txt
ls -t
rm nome4.txt nome-do-arquivo-a-ser-criado1.txt
ls -t ; echo
clear
ls
cd pasta-mama/
cd ..
pwd
ls
cd -
ls ; echo
echo -e 'Este texto\n \né do arquivo\n \nnome1.txt\n' >> nome1.txt
cat nome1.txt
```

```
cat nome1.txt

ls

mv nome1.txt pasta-filha/

ls ; echo

mkdir pasta-bro

ls

cd pasta-filha/

> texto-filha.txt

ls -t

echo -e "\n \nEste texto\n \nEstá escrito em\n \ntexto-filha.txt!" >
texto-filha.txt

cat texto-filha.txt

echo -e "\n \nEste texto\n \nSobreescreve\n \ntexto-filha.txt!" > texto-
filha.txt

cat texto-filha.txt

echo -e "\n \nEste texto\n \nSerá adicionado a\n \ntexto-filha.txt\n" >>
texto-filha.txt

cat texto-filha.txt

ls -t

cat nome1.txt

cp nome1.txt nome8.txt

clear

ls -t

cat nome8.txt

cd ..

ls

rm -rf pasta-filha/

ls

rmdir pasta-bro/

ls

cd ..
```

```
pwd

ls

rmdir pasta-mama/

ls

clear

pwd

echo -e 'L1ee\n L2nn\nL3cc\nL4yy\nL5rr\nL6ii\nL7hh\n L8jj\nL9ss\n L10mm\n
L11ww\nL12oo\n L13ff' > 09-texto_teste.txt

mkdir 09-texto_teste ; mv 09-texto_teste.txt 09-texto_teste/ ; cd 09-
texto_teste/ ; ls -c

cat 09-texto_teste.txt

head -5 09-texto_teste.txt

head -2 09-texto_teste.txt

tail -6 09-texto_teste.txt

tail -3 09-texto_teste.txt

head -5 09-texto_teste.txt | tail

tail -6 09-texto_teste.txt | head

wc -m 09-texto_teste.txt

wc -w 09-texto_teste.txt

wc 09-texto_teste.txt

more 09-texto_teste.txt

cd ..

rm -r 09-texto_teste/

ls -t

clear

exit

.

CRIAÇÃO/BACKUP DE/EM LINKS COM LN:

ln -s

ln -b
```

ln -i

.

COMANDOS DE COMPACTAÇÃO / DESCOMPACTAÇÃO

.

TAR

Armazena ou extrai arquivos e diretórios dentro de um único arquivo ou dispositivo.

.

Sintaxe: tar [opções] arquivos_ou_diretórios

Opções:

-c :: cria um novo arquivo .tar e adiciona a ele os arquivos especificados

-x :: retira os arquivos agrupados no arquivo .tar

-f :: indica que o destino é um arquivo em disco e não uma fita magnética

-v :: exibe o nome de cada arquivo processado

-Z :: compacta ou descompacta arquivos utilizando o comando compress

-z :: compacta ou descompacta arquivos utilizando o comando gzip

-j :: compacta ou descompacta arquivos utilizando o comando bzip2

-M :: múltiplos volumes

-b n :: define o tamanho do bloco de dados utilizado pelo tar (n*512 bytes)

.

Exemplos:

1. Gera um arquivo de backup do diretório "documentos1":

tar -cvf documentos.tar documentos1

2. Exibe o conteúdo do arquivo "documentos.tar":

tar -tvf documentos.tar

3. Extrai o conteúdo do arquivo "documentos.tar":

tar -xvf documentos.tar

5. Gera um arquivo de backup compactado com bzip2 do diretório "documentos1":

tar -cvjf memorandos.tar.bz2 documentos1

6. Divide em vários disquetes o arquivo "documentos.tar.bz2":

tar -cvMf /dev/fd0 /operftp/documentos.tar.bz2

7. Extrai o arquivo de backup armazenado no disquete:

tar -xvMf /dev/fd0

.

CPIO

Executa funções de arquivamento de dados.

.

Sintaxe: cpio [opções]

Opções:

-o :: lê nomes de arquivos da entrada padrão e os copia para a saída padrão com a informação necessária para a sua recuperação posterior com o comando: cpio -i

-i :: lê da entrada padrão um arquivo criado pelo comando cpio -o e extrai os arquivos armazenados

-v :: exibe o nome de cada arquivo processado

.

Exemplos:

1. Copia todos os arquivos mencionados em "lista.txt" para o arquivo "backup.cpio":

cpio -o /operftp/lista.txt > /operftp/backup.cpio

2. Extrai todos os arquivos armazenados em "backup.cpio":

cpio -i procedimento.zip

.

COMPRESS

Compacta um ou mais arquivos utilizando a compactação Lempel-Ziv.

Sintaxe: compress [opções] arquivos

Opções:

-c :: grava o arquivo compactado na saída padrão e retém o arquivo original

-d :: descompacta o arquivo

-d :: descompacta o arquivo

-r :: compacta recursivamente arquivos em todos os subdiretórios

Exemplos:

compress documentos.tar
$ compress -d documentos.tar.Z

.

UNCOMPRESS

Descompacta um ou mais arquivos que tenham sido compactados com o comando compress.

Sintaxe: uncompress [opções] arquivos

Opções:

-c :: grava o resultado na saída padrão e retém o original

-r :: descompacta recursivamente arquivos em todos os subdiretórios

Exemplo:

uncompress documentos.tar.Z

.

GZIP

Compacta um ou mais arquivos.

Sintaxe: gzip [opções] arquivos

Opções:

-c :: grava o arquivo compactado na saída padrão e retém o arquivo original

-d :: descompacta arquivo. O mesmo que gunzip

-f :: sobrescreve arquivos já existentes

-h :: mensagem de ajuda

-l :: lista o conteúdo de um arquivo compactado

-t :: testa a integridade do arquivo compactado

-n :: não salva o nome original

-r :: compacta recursivamente arquivos em todos os subdiretórios

-L :: exibe a licença do comando

Exemplos:

gzip documentos.tar

$ gzip -d documentos.tar.gz

.

GUNZIP

Descompacta arquivos compactados pelos comandos gzip e compress. Utiliza as mesmas opções de gzip.

Sintaxe: gunzip [opções] arquivos

Exemplo:

gunzip documentos.tar.gz

.

BZIP2

Compacta um ou mais arquivos.

Sintaxe: bzip2 [opções] arquivos

Opções:

-z :: força a compressão

-c :: grava na saída padrão

-t :: testa a integridade do arquivo compactado

-f :: sobrescreve arquivos já existentes

-d :: descompacta arquivos. O mesmo que bunzip2

-k :: não apaga os arquivos de entrada

-L :: licença do comando

Exemplos:

bzip2 documentos.tar

$ bzip2 -d documentos.tar.bz2

.

BUNZIP2

Descompacta arquivos compactados pelos comandos gzip ou compress. Utiliza as mesmas opções de bzip2.

Sintaxe: bunzip2 [opções] arquivos

Exemplo:

```
bunzip2 documentos.tar.bz2
```

.

ZIP

Compacta um ou mais arquivos.

Sintaxe: zip [opções] arquivo-destino arquivo-origem

Opções:

-e :: permito encriptar o conteúdo de um arquivo ZIP através de senha. A senha será pedida no momento da compactação

-m :: apaga os arquivos originais após a compactação

-r :: compacta recursivamente arquivos em todos os subdiretórios

Exemplos:

```
zip documentos.zip *.txt
```

```
$ zip -r documentos.zip /usr/*.txt
```

.

UNZIP

Descompacta arquivos compactados pelo comando zip.

Sintaxe: unzip [opções] arquivo.zip arquivos-origem [diretório]

Opções:

-l :: exibe os arquivos existentes dentro do arquivo ZIP

-d :: diretório onde os arquivos serão descompactados

-o :: substitui arquivos existentes sem perguntar

Exemplos:

```
unzip documentos.zip
$ unzip documentos.zip -d /operftp
```

.

ZCAT, ZMORE, ZLESS, BZ2CAT

Visualiza o conteúdo de um arquivo texto compactado, sem precisar descompactar o arquivo.

Os comandos zcat, zless e zmore funcionam da mesma forma que cat, less e more. A única diferença, é que esses comandos podem ler diretamente arquivos compactados com gzip ou compress sem precisar descompactar os arquivos.

.

EXEMPLOS:

.

zcat nome_arquivo

zless nome_arquivo

zmore nome_arquivo

bz2cat nome_arquivo

.

LINUX PERMISSOES DE PASTAS E ARQUIVOS

.

As permissões são usadas para definir quem pode acessar determinados arquivos ou diretórios, assim mantendo segurança e organização em sistemas e redes. Cada arquivo ou pasta tem 3 permissões:

(Usuário Dono) (Grupo Dono) (outros)

.

Usuário dono: é o proprietário do arquivo. Grupo Dono: é um grupo, que pode conter vários usuários. Outros: se encaixam os outros usuários em geral. O comando "ls -lh" faz uma listagem longa e detalhada no diretório onde eu estiver com o terminal aberto.

.

PARA ENTENDER AS PERMISSÕES DE PASTAS E ARQUIVOS, VOU FAZER O SEGUINTE:

.

Abro o terminal pelo menu do sistema. Não sou root, sou um usuário comum com alguns poderes administrativos, mas o prompt do terminal não está em root (#) está ($).

.

CRIO UMA PASTA COM O COMANDO:

mkdir pasta-estudo-permissoes

.

ENTRO NA PASTA COM O COMANDO:

cd pasta-estudo-permissoes/

.

CRIO UM ARQUIVO DE TEXTO CHAMADO estudo-permissoes.txt COM O COMANDO:

> estudo-permissoes.txt

.

CRIO OUTRO ARQUIVO COM O COMANDO:

touch permissoes-estudos.txt

.

DIGITO:

ls -lh

.

O RESULTADO É:

```
pasta-estudo-permissoes $ls -lh
total 0
-rw-rw-r-- 1 user user 0 mar 8 05:15 estudo-permissoes.txt
-rw-rw-r-- 1 user user 0 mar 8 05:16 permissoes-estudos.txt
```

.

AS PERMISSÕES DE ARQUIVOS SÃO INFORMADAS A MIM POR 5 CARACTERES. SÃO ELES:

r (significa leitura), w (significa escrita), x (significa execução), - (hífen), d (indica que é uma pasta/diretório)

.

A PERMISSÃO DE ARQUIVOS E PASTAS ESTÁ DIVIDIDA EM QUATRO PARTES POR EXEMPLO:

Parte 1: -, Parte 2: rw-, Parte 3: r--, Parte 4: r--

.

NO TERMINAL ESTAS QUATRO PARTES SERÃO VISTAS ASSIM:

-rw-r--r--

.

EXEMPLO:

-rw-rw-r-- 1 user user 0 mar 8 05:15 estudo-permissoes.txt

.

Parte 1 (-)

hífen: o caractere "-" no início da sequência indica que aquele é um arquivo comum. Caso contrário, indica que a permissão de escrita, leitura

ou execução está negada.

.

Parte 2 (rw-)

r: leitura permitida do arquivo ou diretório

.

w: permite editar um arquivo ou modificar o conteúdo de um diretório

.

hífen: o caractere "-" no início da sequência indica que aquele é um arquivo comum. Caso contrário, indica que a permissão de escrita, leitura ou execução está negada.

.

Parte 3 (r --)

r: leitura permitida do arquivo ou diretório

.

hífen: o caractere "-" no início da sequência indica que aquele é um arquivo comum. Caso contrário, indica que a permissão de escrita, leitura ou execução está negada.

.

hífen: o caractere "-" no início da sequência indica que aquele é um arquivo comum. Caso contrário, indica que a permissão de escrita, leitura ou execução está negada.

.

Parte 4 (r--)

r: leitura permitida do arquivo ou diretório

.

hífen: o caractere "-" no início da sequência indica que aquele é um arquivo comum. Caso contrário, indica que a permissão de escrita, leitura ou execução está negada.

.

hífen: o caractere "-" no início da sequência indica que aquele é um arquivo comum. Caso contrário, indica que a permissão de escrita, leitura ou execução está negada.

.

EU POSSO ALTERAR AS PESMISSÕES DESTES ARQUIVOS USANDO CHMOD.

.

O CHMOD É UM COMANDO QUE ALTERA AS PERMISSÕES DE ARQUIVOS E DIRETÓRIOS.

.

APRENDER COMO INTERPRETAR AS PERMISSÕES AJUDA A USAR O COMANDO CHMOD, POIS QUE ESSE COMANDO TRABALHA COM UMA SINTAXE SEMELHANTE.

.

A "FÓRMULA" DE USO MAIS SIMPLES DO CHMOD É:

chmod ugoa+-=rwx arquivo/diretório

.

SENDO QUE:

1) u: define que as regras serão aplicadas ao usuário

2) g: define que as regras serão aplicadas ao grupo

3) o: define que as regras serão aplicadas aos outros usuários do sistema

4) a: define que as regras serão aplicadas a todos

5) +: adiciona permissão

6) -: remove permissão

7) =: informa que a permissão aplicada deve ser exatamente igual a que será indicada a seguir

8) r: atribui a permissão de leitura

9) w: atribui a permissão de escrita

10) x: atribui a permissão de execução

.

COM ESTE PEQUENO/GRANDE CONHECIMENTO ADQUIRIDO, POSSO EXECUTAR UM COMANDO ALTERANDO AS PERMISSÕES DE UM ARQUIVO QUE EU CRIEI. O ARQUIVO "permissoes-estudos.txt".

.

EXECUTO NO TERMINAL:

sudo chmod g+w permissoes-estudos.txt

.

O chmod precisa ser usado com o comando sudo, pois apenas o usuário root tem a permissão de executá-lo. Depois de informar a senha e de ter

concluído a execução, basta listar o arquivo com o ls -l para conferir a nova permissão. Percebo que onde antes era r--, agora se tornou rw-

.

CASO QUEIRA REMOVER A PERMISSÃO DE LEITURA, ESCRITA E EXECUÇÃO DE PESSOAS QUE NÃO PERTENCEM AO GRUPO, USO:

sudo chmod o-rwx permissoes-estudos.txt

.

SE QUISER DAR A PERMISSÃO DE ESCRITA E LEITURA PARA TODOS OS USUÁRIOS DO SISTEMA, BASTA EXECUTAR:

sudo chmod a+rw permissoes-estudos.txt

.

O PARÂMETRO a+rw VAI APENAS ADICIONAR A PERMISSÃO DE LEITURA E ESCRITA, ISTO É, CASO O DONO DO ARQUIVO, GRUPO E OUTROS USUÁRIOS JÁ POSSUAM A PERMISSÃO DE EXECUÇÃO, ELA NÃO SERÁ REVOGADA. SE EU QUISER IMPOR A REGRA DE APENAS LEITURA E ESCRITA, REVOGANDO UMA PERMISSÃO DE EXECUÇÃO PRÉ-EXISTENTE, USO O SINAL DE IGUAL:

sudo chmod a=rw permissoes-estudos.txt

.

OUTRA FORMA DE USAR O CHMOD

.

Existe mais de uma forma de usar o chmod e muitos preferem esta segunda, que ensinaremos a seguir. Em vez de digitar letras e operadores matemáticos, como o caso de a+rw, por exemplo, muitos administradores preferem estipular as permissões com códigos numéricos.

.

PARA ISSO, VOCÊ PRECISA PENSAR NA REGRA DE PERMISSÃO COMO SE FOSSE UMA SEQUÊNCIA DE BITS. A PERMISSÃO rwx, POR EXEMPLO, EQUIVALERIA A 111, ENQUANTO QUE RW- SE TRANSFORMARIA EM 110. RESUMINDO: 1 PARA LETRA, 0 PARA HÍFEN. CONVERTER ESSES NÚMEROS DE BASE BINÁRIA PARA DECIMAL, 111 VIRARIA 7, ENQUANTO QUE 110 SE TORNARIA 6.

.

VEJA O EXEMPLO ABAIXO BEM FÁCIL:

.

rwx
111
7

rw-

...
110
6

.

SEGUINDO ESSA LÓGICA, SE VOCÊ QUISESSE FORNECER AS PERMISSÕES DE LEITURA, ESCRITA E EXECUÇÃO PARA TODOS OS USUÁRIOS DO SISTEMA, PODERIA DIGITAR A SEGUINTE LINHA:

sudo chmod 777 nome-do-arquivo

.

SE PREFERIR MANTER A OPÇÃO DE EXECUTAR O ARQUIVO APENAS PARA O DONO E GRUPO DELE, ENTÃO A LINHA MUDA PARA:

sudo chmod 776 nome-do-arquivo

.

PARA FACILITAR E NÃO TER QUE CONVERTER DE CABEÇA TODOS ESSES NÚMEROS, EXISTE UMA PEQUENA TABELA DE APENAS OITO (8) PERMISSÕES, QUE PODE SER CONSULTADA SEMPRE QUE NECESSÁRIO:

.

Permissão1: 0 ; Permissão2: 1 ; Permissão3: 2 ; Permissão4: 3 ; Permissão5: 4 ; Permissão6: 5 ; Permissão7: 6 ; Permissão8: 7. Onde a:

.

A permissão 1:

Convertida para binário:

000

Tem o decimal:

0

A permissão 2:

--x

Convertida para binário:

001

Tem o decimal:

1

A permissão 3:

-W-

Convertida para binário:

010

Tem o decimal:

2

A permissão 4:

-WX

Convertida para binário:

011

Tem o decimal:

3

A permissão 5:

r--

Convertida para binário:

100

Tem o decimal:

4

A permissão 6:

r-x

Convertida para binário:

101

Tem o decimal:

5

A permissão 7:

rw-

Convertida para binário:

110

Tem o decimal:

6

A permissão 8:

rwx

Convertida para binário:

111

Tem o decimal:

7

.

CASO NÃO HAJA TODAS AS PERMISSÕES, PODERÁ APARECER INCOMPLETO:

rwxrw_ _ _x, ou seja, neste exemplo:

Dono do arquivo tem permissão de Ler, Escrever e executar (rwx).

Grupo tem permissão de Ler e Escrever (rw_).

Outros tem permissão apenas de executar. (_ _ x).

.

CONCLUSÃO:

.

EXISTEM DOIS MODOS DE DEFINIR UMA PERMISSÃO, ATRAVÉS DO MODO OCTAL E MODO TEXTUAL.

.

TEXTUAL - Usamos letras antes das permissões (chmod é o comando para modificar permissões de arquivos):

$ chmod u+rw, g+w, o-rwx teste.txt

Onde:

U - representa usuário;
G - representa grupo;
O - representa outros.

ugo

.

MODO OCTAL

.

O modo Octal tem a mesma função de definir permissões, só que em números. Exemplo:

$ chmod 620 permissoes-estudos.txt

$ chmod 620 permissoes_estudos.txt

(comando) (permissão) (arquivo)

.

Tipo de permissão Octal:

4 - Indica permissão de leitura;
2 - Permissão de escrita;
1 - Indica permissão de execução;
0 - Indica sem permissões.

.

Agora é simples, é só somar e ditar as permissões, exemplo:

4 + 2 + 1 = 7 (permissão de rwx)
4 + 2 = 6 (permissão rw)
4 = (permissão r)

.

Exemplo: A permissão 610 indica que o arquivo tem permissão:

6 para dono do arquivo
1 para grupo e
0 para outros ou seja

dono= (rw_) Grupo=(_ _ x) outros=(_ _ _)

.

Percebo que quando é feita a listagem longa "ls -l", o primeiro caractere não é uma permissão.

.

O primeiro caracter "d" indica o tipo do arquivo, neste caso "d" indica que é um diretório.

.

Existem também outras permissões especiais mas não quero saber no momento.

.

Com um pouco de prática talvez um dia eu esteja alterando permissões de arquivos e diretórios de maneira automática. Mas eu garanto que você e eu, nunca mais vamos esquecer do que faz um chmod 777.

.

Posso ler este texto em formato .txt no celular usando o FBreader ou algum outro programa do Android.

.

CREIO QUE ISTO É UM BOM COMEÇO. A GENTE NUNCA PARA DE APRENDER. QUER CONTINUAR?

.

CURIOSIDADE:

.

Quem desenvolve software com Linux para usar no Linux, tem que disponibilizar o código fonte.

Muitos desenvolvedores disponibilizam um código fonte sem muita informação, sem instrução detalhada de como compilar. Algo tão complexo e trabalhoso que, para dele se obter o software e usa-lo só pagando a um outro profissional especializado.

A complexidade do código fonte é uma barreira criada com propósito.

Um deles é este.

Nega o software a grupos, empresas e o usuário comuns por causa da sua extrema complexidade de compilação.

Pode ser que isto já seja a regra.

Tente instalar alguns programas usando o código fonte e tire sua própria conclusão.

.

Ganhar dinheiro vendendo serviços e criando programas com software Linux é bom.

.

Criar barreiras para dificultar o uso do código fonte não é bom.

.

Existem pessoas que creditam que melhores softwares podem resultar de um modelo
aberto de desenvolvimento do que de modelos proprietários.

Teoricamente, qualquer empresa criando um software para uso próprio pode economizar dinheiro adicionando suas contribuições as dos outros a fim de obter um produto final muito melhor para ela mesma.

Quem quer ganhar dinheiro com a venda de software precisa ser extremanente criativo atualmente.

A pessoa ou empresa pode vender o software que cria usando Linux, mas tem que incluir um software GPL, o código-
fonte dele que deve ser passado para frente.

Outros podem recompilar esse produto, e assim usar o software e pode até

revender sem custos.

.

ENTÃO PARA GANHAR DINHEIRO GRUPOS, PESSOAS E EMPRESAS PODEM:

Vender seus produtos com base em uma assinatura. Por uma determinada quantia de dinheiro por ano, você obtém o código binário para rodar o Linux (assim você não tem que compilar por conta própria), suporte garantido, ferramentas para monitoramento de hardware e software no seu computador, acesso à base de conhecimento da empresa e outros recursos.

.

A maioria dos sistemas operacionais oferecidos gratuitamente, inclui grande parte do mesmo software que outros oferecem com base em uma assinatura. Também estão disponíveis em forma binária, mas não há garantias associadas com o software ou futuras atualizações dele.

.

Dizem que pequeno escritório ou um usuário pessoal pode arriscar usar sistemas operacionais Linux excelentes e gratúitos, mas uma grande empresa que está executando aplicações de missão crítica provavelmente acabará investindo algum dinheiro em produtos com base em uma assinatura.

.

Para lucrar, gerar capital, ganhar dinheiro empresas que trabalham com Linux oferecem:

.

1
Treinamento e certificação

2
Recompensas de software

uma maneira fascinante de as empresas de software de código-fonte aberto fazerem dinheiro. Digamos que uma empresa/grupo/usuário está usando o pacote de software ABZ e precisa de um novo recurso imediatamente. Ao pagar uma recompensa de software para o projeto em si, ou para outros desenvolvedores, a empresa/grupo/usuário pode ter as melhorias que precisa deslocadas para o início da fila.

O software que empresa/grupo/usuário paga permanecerá coberto pela sua licença de código-fonte aberto, mas a empresa/grupo/usuário terá os recursos de que precisa provavelmente mais barato que o custo da construção do projeto a partir zero.

3
Doações

Muitos projetos de código-fonte aberto aceitam doações de pessoas físicas ou empresas de desenvolvimento de código fonte aberto que usam código a

ou empresas de desenvolvimento de código-fonte aberto que usam código a partir de seus projetos. Surpreendentemente, muitos projetos de código-fonte aberto suportam um ou dois desenvolvedores e funcionam exclusivamente com base em doações. Estojos, canecas e camisetas - Muitos projetos de código-fonte aberto têm lojas online onde podemos comprar CDs e uma variedade de canecas, camisetas, mouse pads e outros souvenires. Se tu amas um projeto de verdade, compre uma camiseta e uma caneca. Formas mais criativas estão a ser inventadas diariamente para apoiar quem esta a produzir software de código-fonte aberto.

4
Retorno obtido

Pessoas/Grupos se tornam colaboradoras e mantenedoras de software de código-fonte aberto porque precisavam ou queriam o software. As contribuições que elas fazem gratuitamente valem a pena pelo retorno que elas obtêm de outras pessoas que fazem o mesmo.

.

Não esqueça que para algumas pessoas, o Linux não tem como objetivo lucro. O objetivo do Linux é garantir a tua liberdade. O trabalho colaborativo produz ótimos softwares disponíveis a todos em toda parte.

.

APRENDENDO SHELL SCRIPT - CONTINUE, BREAK E EXIT

.

A instrução continue é usada para retomar a iteração seguinte do loop FOR, WHILE ou UNTIL.

Use a instrução break para sair de dentro de um loop FOR, WHILE ou UNTIL, isto é, pare a execução do loop.

O exit é usado para definir o status da execução do programa, se o valor de $? for 0 então tudo ocorreu naturalmente, se não houve erro.

Você pode por o exit 0 no final do seu script para sair sem problemas, e um echo $? para saber qual foi o código de retorno na execução do programa.

No exemplo abaixo só irá imprimir 8 e 9, perceba o uso continue , break e exit.

.

```
#!/bin/bash

for i in $(seq 1 10);
do
if [[ "$i" "9" ]]; then
break;
fi
```

```
echo $i;

done

exit 0;

# Fim do arquivo: i8-continue-break-exit.sh
```

.

EXECUTO O EXEMPLO ABAIXO:

.

```
#!/bin/bash

_INPUT_STRING="Olá"
while :
do
echo "Você deseja ficar aqui?"
read _INPUT_STRING

if [[ $_INPUT_STRING = 'tchau' ]]; then
continue
else
echo "Você ainda deseja ficar aqui"
fi

done

# Fim do arquivo: 0n4-read-continue.sh
```

.

OBS:
Se for o caso, SAIA COM: Ctrl +C

.

Execute o script abaixo:

```
#!/bin/bash

_INPUT_STRING="Olá"
while :
do
echo "Você deseja ficar aqui?"
read _INPUT_STRING

if [[ $_INPUT_STRING = 'tchau' ]]; then
continue
else
break
fi

done
```

```
# Fim do arquivo: 1n5-read-continue.sh
```

.

OBS:
Se for o caso, SAIA COM: Ctrl +C

.

Executa o script abaixo:

```
#!/bin/bash

_INPUT_STRING="Olá"
while :
do
echo "Você deseja ficar aqui?"
read _INPUT_STRING

if [[ $_INPUT_STRING != 'tchau' ]]; then
continue
else
break
fi

done

# Fim do arquivo: 1n5-continue-break.sh
```

.

Resultado:

```
~ $sh 0n4-read-continue.sh
Você deseja ficar aqui?
Hmhm
Você deseja ficar aqui?
Sim
Você deseja ficar aqui?
Não sei
Você deseja ficar aqui?
Pare!
Você deseja ficar aqui?
tchau
~ $
```

.

Executa o script abaixo:

```
#!/bin/bash

_INPUT_STRING="Olá"
while :
do
```

```
echo "Você deseja ficar aqui?"
read _INPUT_STRING

if [[ $_INPUT_STRING != 'tchau' ]]; then
continue
else
exit 0
fi

done

# Fim do arquivo: 2n-continue-exit.sh
```

.

COMANDO EVAL

O comando eval é perigoso evito usar ele. Aqui um exemplo de uso do eval. Quem desenvolve software é que sabe usar o eval.

.

Execute os comandos abaixo no terminal:

```
A='ls'

B=A

echo $B

echo '$'$B

eval echo '$'$B
```

.

O COMANDO EXEC

O comando exec substitui o processo shell atual pelo comando especificado. O exec cria a segunda parte da junção, o que não seria possível usando o pipe |.

.

APRENDENDO SHELL SCRIPT - MATEMÁTICA EM SHELL SCRIPT

.

Expressões com parênteses são efetuadas em primeiro lugar.

.

*, % e / são efetuados antes de + e -

.

Todo o resto é efetuado da ESQUERDA para a DIREITA.

.

No caso da multiplicação utiza-se *, pois o asterisco é um curinga do Linux.

.

FORMAS DE CÁLCULO:

.

```
echo $((2+2))

echo $((2+2*5))

echo $((2+2*5/2))

a=2+2 ; echo $a

declare -i a

a=2+2 ; echo $a

a=2+2 ; echo $a

echo "5/2" | bc

echo "scale=2;5/2" | bc

bc

2+2

scale=2;5/2

quit

expr 2 + 2

bc << calc.txt

dc calc.txt

echo "obase=2;2" | bc

echo "obase=2;54" | bc

expr lenght "Linux"
```

.

GNU/LINUX SHELL SCRIPT – EXPRESSÕES REGULARES E PROGRAMAS GRÁFICOS

.

APRENDENDO SHELL SCRIPT

criar programas graficos e interativos com shell script

.

Podemos criar programas interativos pelo terminal e programas gráficos.

Tem muitas ferramentas de comandos.

Algumas já vem pré-instaladas nas Distros Linux.

Vamos tentar entender duas. Dialog e Yad.

.

Primeiro o Dialog mas saiba que o Yad - Para programas gráficos é a evolução do Zenity. Tem mais opções. O autor do Yad é o ucraniano Victor Ananjevsky.

.

Dialog - Para programas cli (modo texto = cli) mas que cria uma interatividade com o usuário. Dialog cria widgets, menus, avisos, barras de progresso, entre outras coisas que colocamos em Shell Script e aparecem no terminal ao executar o script. Pode as vezes, usar o mouse para clicar nas janelas do Dialog. Existem várias caixas de Dialog que podemos usar. Essas caixas são utilizadas para compor interfaces amigáveis com o usuário, para que ele responda perguntas ou escolha opções.

.

Agora sabemos que o Dialog é um executável e recebe todos os parâmetros via linha de comando, então ele geralmente é usado dentro de um Shell Script. Serve para fazer programas interativos, que o usuário precisa operar durante sua execução. Tarefas comuns feitas com o Dialog são escolher uma opção em um menu, escolher um arquivo, uma data, e digitar frases ou senhas.

.

Instale o Dialog pela central de programas da sua Distro, ou usando o gerenciador de pacotes da sua Distribuição. Verifique se tem o Yad instalado também.

.

OBS:

O kdialog é bem interessante também.

.

Para criar uma tela simples execute os comandos abaixo no terminal:

dialog --msgbox 'É a tua primeira tela' 5 40

dialog --msgbox 'Vamos usar Dialog?' 5 25

.

OBS:

dialog (É o comando.)

--msgbox (É o tipo de diálogo/widget.)

'Vamos usar Dialog?' (É a mensagem.)

5 = Altura (Linhas)

25 = Largura (Colunas)

.

Digito o texto abaixo no terminal e eperte a tecla Enter:

dialog --msgbox 'Shell Script Primeiro Programa com Dialog' 5 50

.

Aperte: Enter

Digito: clear

.

Então o comando dialog utiliza parâmetros de linha de comando para determinar que tipo de widget de janela deve ser criada.

Um widget é um tipo de elemento de janela.

.

ABAIXO ALGUNS DOS TIPOS DE WIDGETS SUPORTADOS PELO DIALOG:

.

1) gauge (Mostra uma barra de progresso)

2) calendar (Vê um calendário e escolhe uma data)

3) checklist (Vê uma lista de opções e escolhe várias)

4) infobox (Mostra uma mensagem sem esperar por uma resposta)

5) inputmenu (Fornece um menu editável)

6) menu (Mostra uma lista de seleções para escolha)

7) msgbox (Mostra uma mensagem e pede que o usuário pressione um botão OK)

8) infobox (Vê uma mensagem sem botões)

9) passwordbox (Mostra uma caixa de texto simples que esconde o texto digitado)

10) radiolist (Fornece um grupo de itens de menu onde apenas um item pode ser selecionado)

11) tailbox (Mostra o texto de um arquivo em uma janela com rolagem usando o comando tail)

12) textbox (Mostra o conteúdo de um arquivo em uma janela com rolagem)

13) timebox (Fornece uma janela para selecionarmos uma hora, minuto e segundo)

14) yesno (Fornece uma mensagem simples com botões Yes e No.)

.

Para especificarmos um widget na linha de comandos, usamos a sintaxe:

dialog --widget parâmetros

.

YAD - CRIAR PROGRAMAS GRÁFICOS E INTERATIVOS COM SHELL SCRIPT

.

Yad - Para programas gráficos. Evolução de um programa chamado Zenity. Tem mais opções. O autor do Yad é o ucraniano Victor Ananjevsky. Instale o Yad. Após instalar, pode testar com o comando abaixo:

yad

.

A SINTAXE BÁSICA É:

yad [--tipo-dialogo] [--options]

.

TIPOS DE DIÁLOGO:

.

1) --calendar (calendário)

2) --color (paleta de cores)

3) --entry (entrada de dados)

4) --icons (mostra uma caixa com ícones de atalho para aplicações)

5) --file (diálogo para selecionar arquivos)

6) --font (diálogo para seleção de fontes)

7) --form (formulários)

8) --list (diálogo para gráficos --list)

8) --list (diálogo com itens em lista)

9) --notification (mostra um ícone da barra de notificação do sistema)

10) --progress (diálogo de progresso)

11) --text-info (mostra o conteúdo de um arquivo texto)

12) --scale (diálogo para seleção de valor, usando uma escala)

.

Para cada um dos exemplos podemos colocar o script e depois algumas imagens da sua execução.

--calendar:

Mostra um calendário permitindo selecionar a data e envia o valor para a saída padrão

.

EXEMPLO:

.

```
#!/bin/bash

# usando o yad com --calendar
# mostra um calendário iniciando no dia 20/03/2018
# guarda o valor selecionar na variável $DATA

DATA=$(\
yad --calendar \
--day=20 \
--month=3 \
--year=2018 \
--date-format=%d\/%m\/%Y \
--title=Calendario \
--center \ # disposição do diálogo na tela
)

# mostra um diálogo informando a $DATA selecionada
yad --title="AVISO" \ --text="Você selecionou a data $DATA"
#.EOF

# Fim do 09c-yad-script.sh
```

.

Salve com o nome de "09c-yad-script.sh", dê permissão de execução:

```
chmod a+x 09c-yad-script.sh
```

.

Execute:

```
sh 09c-yad-script.sh
```

.

EXEMPLO COM YAD --COLOR

yad --color:

Diálogo de seleção de cores, permite selecionar uma determinada cor usando a paleta de cores, editando diretamente por código, etc. Bom para saber o código de uma cor específica.

.

EXEMPLO A SER EXECUTADO:

```
!/bin/bash

# uso do yad - com color
# permite selecionar determinada cor numa paleta de cores
# e envia o valor para a saída padrão, no caso armazenei na variável $COR

COR=$(\
yad --color \
--init-color="#FFFFFF" \ #cor que inicialmente fica selecionada na paleta de cores.
--palette \
)

yad --title="YAD COM COLOR" \
--text="Você selecionou a cor $COR"
#.EOF

# Fim do 4j-script-color.sh
```

.

Salve com o nome de "4j-script-color.sh", dê permissão de execução:

```
chmod a+x 4j-script-color.sh
```

.

Execute:

```
bash 4j-script-color.sh
```

SITE PARA APRENDER COMANDOS:

http://explainshell.com/

(leio, releio e leio de novo)

LINUX SHELL SCRIPT, ENTENDENDO EXPRESSÕES REGULARES

Para trabalhar com as expressões regulares, usarei os meta-caracteres, ou seja, caracteres que representam um conjunto de outros caracteres ou que estipulem certas regras para a busca. E para que tudo fique mais poderoso, saiba que é possível combinar texto comum com meta-caracteres. Portanto, vamos ver o que faz cada um deles e, em seguida, como usá-los na prática.

METACARACTERES
1) ^ (circunflexo): representa o começo da linha
2) $ (cifrão): representa o fim da linha
3) . (ponto): casa com um caractere qualquer
4) .* (curinga): casa qualquer coisa, é tudo ou nada
5) a+ (mais): casa a letra "a" uma ou mais vezes
6) a* (asterisco): casa a letra "a" zero ou mais vezes
7) a? (opcional): casa a letra "a" zero ou mais vezes
8) a{2} (chaves): casa a letra "a" duas vezes
9) a{2,} (chaves): casa a letra "a" no mínimo duas vezes
10) [abc] (lista): casa as letras "a" ou "b" ou "c"
11) [^abc] (lista): casa qualquer caractere, exceto "a", "b", e "c"
12) (isso|aquilo) (Ou): casa as strings "isso" ou "aquilo"

Compreendendo cada um destes metacaracteres acima, pode juntar eles. Os "metacaracteres" juntos formarão uma "Expressão Regular" que vai resolver algum problema que apareça pelo caminho. E assim, é que pegamos os metacaracteres transformamos em expressões regulares e colocamos em shell scripts e quem tá de fora quando vê aquela coisa estupenda, tem a certeza de que aquilo pode até fazer alguma coisa, porém é absolutamente uma completa loucura (o que não é verdade). Dá uma olhada nesta serpente abaixo e me diga o que pensa/sente:

```
egrep "\b[a-zA-Z0-9.-]+@[a-zA-Z0-9.-]+\.[a-zA-Z0-9.-]+\b" arquivo.txt
```

A EXPRESSÃO REGULAR acima, é para casar qualquer e-mail dentro de um arquivo.

LEIO O TEXTO ABAIXO QUE EXPLICA MAIS UM POUCO SOBRE META CARACTERES:

LISTA DE META-CARACTERES:
1) . (Qualquer letra)

2) ^ (início da linha)
3) $ (final da linha)
4) [xyz] (Qualquer das letras dentro dos colchetes)
5) [^xyz] (Qualquer letra fora as dentro dos colchetes)
6) [t-z] (Qualquer das letras entre t e z)
7) z* (Letra z zero ou mais vezes)
8) z+ (Letra z uma ou mais vezes)
9) ?{0,1} (Pode aparecer ou não (opcional)
10) *{0,} (Pode aparecer em qualquer quantidade)
11) +{1,} (Deve aparecer no mínimo uma vez)
12) a{2} (Casa a letra 'a' duas vezes)
13) a{2,4} (Casa a letra 'a' de duas a quatro vezes)
14) a{2,} (Casa a letra 'a' no mínimo duas vezes)
15) .* (Casa qualquer coisa, é o tudo e o nada)
16) ^ (início da linha)
17) $ (final da linha)
18) [abc] (casa com os caracteres a, b e c)
19) [a-c] (casa com os caracteres a, b e c)
20) [^abd] (não casa com os caracteres a, b e d)
21) (um|dois) (casa com as palavras um e dois)

LISTA DE META-CARACTERES(Repetições)
1) a{2} (casa com a letra “a” duas vezes)
2) a{2,5} (casa com a letra “a” duas a cinco vezes)
3) a{2,} (casa com a letra “a” duas vezes ou mais)
4) a? (casa com “a” letra a zero vezes ou uma)
5) a* (casa com a letra “a” zeros vezes ou mais)
6) a+ (casa com a letra “a” uma vez ou mais)

LISTA DE META-CARACTERES(Curingas)
1) . (casa com qualquer caracter uma vez)
2) .* (casa com qualquer caracter várias vezes)
3) (esse|aquele) (casa as palavras 'esse' ou 'aquele')

Exemplos

1) Procura a palavra seu-usuario-whoami (qual é seu usuário?) no arquivo /etc/passwd

```
grep seu-usuario-whoami /etc/passwd
```

2) Procura todas as linhas começadas pela letra u no arquivo /etc/passwd:

```
grep '^u' /etc/passwd
```

3) Procura todas as linhas terminadas pela palavra false no arquivo /etc/passwd:

```
grep 'false$' /etc/passwd
```

grep '[a-z]$' /etc/passwd

4) Procura todas as linhas começadas pelas vogais no arquivo /etc/passwd:

grep '^[aeiou]' /etc/passwd

5) Procura todas as linhas começadas por qualquer caracter e segundo caracter seja qualquer vogal no arquivo /etc/passwd:

grep '^.[aeiou]' /etc/passwd

6) Procura todas as linhas que contenham uma sequência de 4 números consecutivos:

grep '[0-9][0-9][0-9][0-9]' /etc/passwd

7) Comando para encontrar linhas em branco:

grep '^$' /etc/passwd

8) Encontrar e contar linhas em branco:

grep '^$' /etc/passwd | wc -l

9) Encontrar mesmo nome, porém com letra inicial minúscula e maiúscula:

grep '[Mm]arcos' /etc/passwd

10) Encontrar 27 sequência^de 27 caracteres:

egrep '^.{27}$' passwd

(Acima foi usado o egrep e não o grep. Porque as chaves fazem parte de um conjunto avançado de Expressões Regulares (“extended”), então o egrep lida melhor com elas. Se fosse para usar o grep normal, teria que “escapar” as chaves.)

---

grep '^.\{27\}$' /etc/passwd

11) Para procurar por linhas que tenham de 20 a 40 caracteres:

egrep '^.{20,40}$' /etc/passwd

12) Para obter as linhas que possuem 40 caracteres ou mais:

egrep '^.{40,}$' /etc/passwd

13) Econtrar números com 3 dígitos (de 0 a 9) ou mais:

egrep '[0123456789]{3,}' /etc/passwd

14) Econtrar linhas que começam com ‘vogal minúscula’ e terminam com a palavra ‘bash’, usa-se o curinga “.*” para significar “qualquer coisa”(não confunda com “qualquer caracterer” somente “.”):

```
egrep '^[aeiou].*bash$' /etc/passwd
```

Script com validações de Tel,E-mail,CEP,IP,Data…

```
#!/bin/bash

# Script com validações de Tel,E-mail,CEP,IP,Data...

# VALIDAR TELEFONE formato: (99)9999-9999 #

echo 'Informe o número de Telefone.Formato: (99)9999-9999';
read TELEFONE
echo $TELEFONE | egrep '^[(][0-9]{2}[)][0-9]{4}+-[0-9]{4}$' && echo -e
'\033[01;32m Número válido! \033[0m' || echo -e '\033[01;31m NÃO é válido
esse número.\033[0m'

############## VALIDAR IP ##############

echo 'Informe o número de IP';
read IP
echo $IP | egrep '^[0-9]{1,3}[.]{1}[0-9]{1,3}[.]{1}[0-9]{1,3}[.]{1}[0-9]
{1,3}$' && echo -e '\033[01;32m IP válido! \033[0m' || echo -e
'\033[01;31m NÃO é válido esse IP.\033[0m'

############## VALIDAR CEP ##############

echo 'Informe o CEP';
read CEP
echo $CEP | egrep '^[0-9]{5}[-][0-9]{3}$' && echo -e '\033[01;32m Número
válido! \033[0m' || echo -e '\033[01;31m NÃO é válido esse número.\033[0m'

#### VALIDAR DATA formato dd/mm/aaaa ####

echo 'Informe a Data.Formato dd/mm/aaaa';
read DATA
echo $DATA | egrep '^[0-3]{1}[0-9]{1}[/][0-1]{1}[0-9]{1}[/][0-9]{4}$' &&
echo -e '\033[01;32m Data válida! \033[0m' || echo -e '\033[01;31m NÃO é
válida essa Data.\033[0m'

############## VALIDAR E-MAIL ###########

echo 'Informe o E-mail';
read EMAIL
echo $EMAIL | egrep '^([a-zA-Z0-9_-.])+@[0-9a-zA-Z.-]+\.[a-z]{2,3}$' &&
echo -e '\033[01;32m E-mail válido! \033[0m' || echo -e '\033[01;31m NÃO é
válido esse E-mail.\033[0m'

# Fonte: http://terminalroot.com.br/2015/01/shell-script-
validandotele.html

################## FIM ##################
```

.

LINUX EXPRESSOES REGULARES

Expressões Regulares: o que são e para que servem?

Muitas vezes precisamos buscar por determinadas palavras, nomes ou até mesmo trechos de código em um arquivo e, a partir dessa procura, realizar algumas alterações. As expressões regulares surgem como uma maneira de especificar um padrão existente entre essas palavras, para que não seja necessário procurar cada uma separadamente.

Metacaracteres para toda ocasião

Para trabalhar com as expressões regulares, usaremos os metacaracteres, ou seja, caracteres que representam um conjunto de outros caracteres ou que estipulem certas regras para a busca. E para que tudo fique mais poderoso, saiba que é possível combinar texto comum com metacaracteres.

Para estipular que nossa busca deve ser realizada no início de uma linha. Para isso, usamos o caractere ^. Caso a busca deva casar com uma expressão no fim da linha, usamos o cifrão ($).

Existem os caracteres que trabalham com o texto em si. Podemos, por exemplo, estabelecer uma lista de letras a serem buscadas, colocando-as entre colchetes: [asd], por exemplo, busca pelas letras "a" ou "s" ou "d". Um intervalo pode ser definido com a ajuda do hífen: [f-h] busca pelas letras "f", "g" ou "h".

O acento circunflexo dentro dos colchetes muda de função e passa a representar negação: [^vxz] busca por qualquer caractere, com exceção de "v", "x" ou "z". E se a procura for por palavras, podemos colocá-las entre parênteses e separadas pela barra vertical, também chamada de pipe (cano, em inglês): (Escritor|Livros), por exemplo, busca pelas palavras "Escritor" ou "Livros".

---

para especificar uma repetição, para procurar por um erro de digitação muito comum: duas letras "a" seguidas. Para isso, usaríamos o número entre colchetes, logo depois da letra: a{2}. Se a repetição fosse de duas a cinco vezes, a expressão ficaria a{2,5}. Para estipular que a repetição da letra "a" deve ocorrer pelo menos duas vezes, use a{ 2,}. Não se esqueça de trocar a letra e o número de acordo com as suas necessidades.

Ainda sobre a quantidade com que uma letra ou expressão pode ocorrer durante a busca, existem os metacaracteres ?, * e +, que simbolizam, respectivamente, zero ou uma vez, zero ou mais vezes, uma ou mais vezes. Exemplos: a?, a* ou a+.

Existem caracteres curingas. O ponto final, por exemplo, casa com um caractere qualquer, enquanto o asterisco casa com qualquer coisa. A partir disso, é possível combinar esses operadores de diversas formas e com a extensão que você precisar. Não há limites de número de caracteres para a criatividade ou necessidade.

GREP: UTILITÁRIO DE BUSCA DE REGEX

A maioria (se não for todas) das distribuições Linux incluem o grep, que é um buscador de regex.

A sintaxe é bem simples: grep 'padrao' entrada. Se a entrada não for especificada, usa-se a entrada padrão (stdin). Você também pode usar alguns parâmetros, como o “-v”, que mostra o inverso do padrão (ou seja, tudo aquilo que não bate com o padrão passado), e o “-i”, que torna a busca insensível à casa (não diferencia maiúsculas de minúsculas).

Note que usei aspas simples delimitando o padrão. Nem sempre elas são necessárias, mas em alguns casos, um metacaracter pode acabar sendo traduzido pelo shell, causando resultados errôneos, e as aspas simples evitam isto. Não entendeu? Relaxa, tá explicado aí embaixo…

Entenda que há caracteres e metacaracteres numa regex. Os caracteres são literais, ou seja, as letras (a-z, A-Z) e números (0-9), além de alguns símbolos e acentos. E os metacaracteres são caracteres que têm um significado especial, como o “^”, que indica começo de linha, e o “$”, que representa final de linha. Se você quer que um símbolo seja tratado literalmente, isto é, sem que seja tratado como um metacaracter, você precisa escapá-lo, colocando uma barra invertida (\ ) antes do dito cujo. Um exemplo de uso disto é para o ponto (.), que é um metacaracter que representa qualquer caracter, e você pode querer tratá-lo como sendo apenas um ponto mesmo.

Aí vai de novo uma listinha com alguns dos metacaracteres mais usados:

1) ^
começa com

2) $
término de linha

3) .

qualquer caracter

4) []
relação de valores possíveis para um caracter. Você pode especificar uma lista ([abcde]), uma faixa ([0-9]), ou várias faixas ([a-zA-Z0-9]).

5) \{\}
especifica quantas vezes o caracter pode se repetir. Por exemplo: “{2}” (duas vezes), “{2,5}” (duas a cinco vezes), “{2,}” (duas ou mais vezes).

6) |
operador lógico ou.

7) .*
operador lógico e.

Olhe esta “cobra”, por exemplo: [0-9]\{3\}\.[0-9]\{3\}\.[0-9]\{3\}-[0-9]\{2\}. Adivinha com o que ela casa? Com um número de CPF que esteja formatado com pontos e traço. Leia a expressão com calma que você vai enxergar isto =)

E onde eu uso este tipo de coisa?

Se você só navega na internet e papeia pelo Whatsapp ou algo parecido, aprender este tipo de coisa pode parecer desnecessário. Mas se costuma ler os logs do sistema, ou se trabalha com arquivos texto, ou se é um admin de sistemas Linux, ou se alguma vez se deparar com a necessidade de achar uma agulha no meio do palheiro, você vai usar regex.

METACARACTERES MAIS USADOS:

1) * ^ : começa com

2) * $: término de linha

3) * . : qualquer caracter

4) * [] : relação de valores possíveis para um caracter. Você pode especificar uma lista ([abcde]), uma faixa ([0-9]), ou várias faixas ([a-zA-Z0-9]).

5) * {} : especifica quantas vezes o caracter pode se repetir. Por exemplo: “{2}” (duas vezes), “{2,5}” (duas a cinco vezes), “{2,}” (duas ou mais vezes).

6) * | : operador lógico ou

7) * .* : operador lógico e

Existe um outro utilitário, o egrep, que é uma versão extendida do grep. A sintaxe de uso é a mesma. Uma coisa legal dele é dispensar o escape para certos metacaracteres, como o “{}”, o que tornaria esta mesma expressão um pouquinho mais curta: [0-9]{3}.[0-9]{3}.[0-9]{3}-[0-9]{2}. Note que o ponto ainda precisou ser escapado, pois a intenção é tratá-lo apenas como ponto mesmo

---

.

REGRAS DO CÓDIGO LIMPO SHELL SCRIPT

.

ANOTAÇÕES LINUX, SHELLSCRIPT, TERMINAL LINUX BASH, VIM
REGRAS DO CÓDIGO LIMPO

.

Acho que talvez o cabeçalho para o shell script ficar mais claro é assim:

.

```
#!/bin/bash

########################################
# Nome do Script:
#
# Descrição:
#
#
#
#
# Autor:
#
# Email:
#
#
########################################
# Sobre este script:
#
#
#
# Exemplo:
#
#
#
# Histórico de modificações:
#
#
#
# Comentário:
#
```

```
"
#
#
# Regras do código limpo:
#
# 1) Colocar apenas um comando por linha
#
# 2) Alinhar verticalmente comandos de um mesmo bloco
#
# 3) Deslocar o alinhamento a direita a cada novo bloco
#
# 4) Usar linhas em branco para separar trechos
#
# 5) Não ultrapassar o limite de 80 colunas por linha
#
#
# Comentários especiais:
#
#
# TODO - indica uma tarefa a ser feita
#
#
# FIXME - indica um bug conhecido que precisa ser arrumado
#
#
# XXX - Notícia, chama a atenção
#
#
########################################

kdialog \
--title "Bem vindo(a)!!" \
--msgbox "Aprendendo Shell Script" \
10 40

kdialog \
--title "Listar diretórios?" \
--yesno "Para listar diretórios e arquivos: ls -R" \

echo

ls -R

echo

# Fim do script

################## FIM #################

.

ESTE ARTIGO ACABA AQUI. TUDO DE GRAÇA.

DE GRAÇA É MAIS GOSTOSO!
```

Saiba que você foi longe e depois deste artigo, se houver necessidade de reconhecimento, pode se considerar um USUÁRIO/LEITOR AUTO-DIDATA AVANÇADO DE GNU/Linux. Porque as pessoas já escreveram trilhões de palavras, textos e livros, mas só quem pode aprender/entender é você. Depende da tua vontade.

Pratico o aprendido de maneira positiva.

Sempre estudante. Existe muito mais a saber sobre shell script. Mas se você for um usuário comum como eu, este conhecimento adquirido aqui aliado a necessidade e criatividade, é ferramenta para fazer muita coisa útil e se não for criador, vai mesmo assim, entender bastante quando encontrar shell scripts pelo caminho.

É um bom começo!