Shell Script do zero

Índice

Capítulo 1 – Básico do Básico

Apresentação

Este material é dedicado para aqueles que não sabem nada de lógica de programação e Shell Script, nos capítulos a seguir você será capaz de criar scripts básicos a medianos e terá todas as condições de se aprofundar no tema sozinho, procuro sempre dar muitos exemplos para que você entenda de um jeito ou de outro e procure ler devagar para conseguir assimilar o material.

Não se preocupe se os conceitos apresentados até o capítulo 3 ficarem vagos, a partir do capítulo 4 trabalharemos na prática e tudo ficará mais fácil. Faça os exercícios antes de ver a respostas, porque a teoria na programação sem prática deixa muito a desejar.

Esclarecendo que sei pouco, mas juntando este pouco com criatividade e persistência, eu consigo fazer muita coisa.

O que é Shell Script

Script é um arquivo com várias instruções para serem executadas pelo shell que é o interpretador de comandos. Com ele podemos automatizar muitas tarefas no Linux criando grandes facilidades.

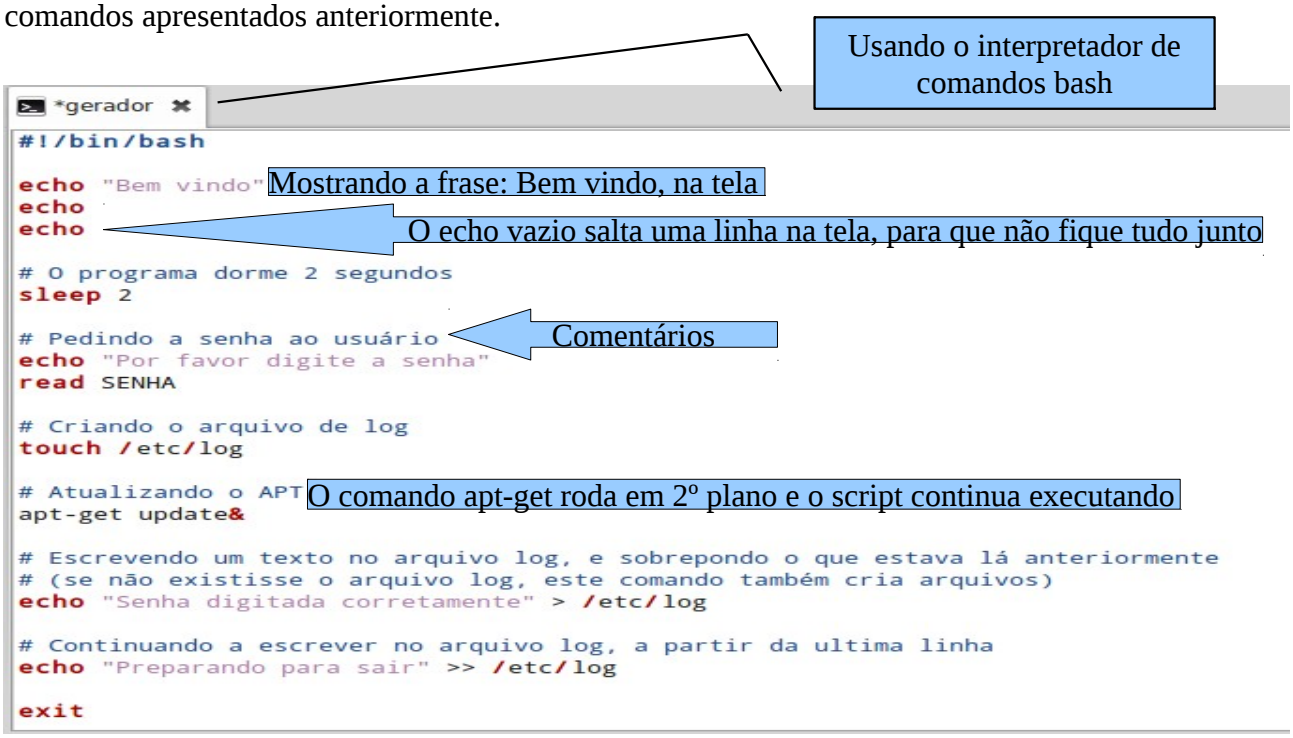
Primeiros Comandos

Podemos usar qualquer comando nos nossos scripts, desde comandos criados por você mesmo (colocando o script dentro de /bin), programas de terceiros e que tem seus comandos no terminal e principalmente alguns comandos do shell que são muito usados em scripts e pouco conhecidos no terminal. Vamos vê-los agora:

Comando	Descrição	Sintaxe
echo	Exibe o texto na tela	echo “texto a ser mostrado”
sleep	Dá um tempo antes de continuar executando	sleep segundos **exemplo:** Sleep 1
read	Recebe o valor de uma variável (veremos ainda)	read variável **exemplo:** read dados
>	Escreve num arquivo-texto *(apagando o que estava lá)*	echo “texto” > /home/luiz/arquivo
>>	Escreve num arquivo-texto *(última linha, não apaga)*	echo “texto” >> /home/luiz/arquivo
&	Roda o comando em 2º plano e continua o script	Comando&
exit	Sai do script	exit
touch	Cria arquivos-texto	touch nome_do_arquivo
#	Comenta tudo depois deste símbolo	# Comentário
* Comentar quer dizer que a linha é um texto e nunca será executada		

Exemplo destes comandos no script

É claro que o script a seguir não tem muito sentido, é só para visualizarmos a aplicação dos comandos apresentados anteriormente.



Comandos mais conhecidos

São os comandos que estamos acostumados a usar no terminal e podemos usá-los também no script, se você não os conhece, vá aprendendo de acordo com a necessidade, pesquise na internet e consulte a tabela resumida abaixo sempre que preciso.

Diretórios		
Comando	**Sintaxe**	**Descrição**
rm -rf	rm -rf +diretório	Deleta arquivos/pastas e tudo que estiver dentro (cuidado)
pwd	pwd	Mostra em qual diretório estamos
chmod	chmod 777 arquivo_ou_pasta	Muda as permissões, 777 = permissão total
chown	chown user:grupo arq_ou_diret.	Muda o proprietário de arquivos e pastas
cd	cd diretório	Entra em diretórios
Usuários		
Comando	**Sintaxe**	**Descrição**
useradd	useradd luiz -g alunos (no grupo)	Adiciona um usuário
userdel	userdel usuário	Deleta usuário e seus arquivos
groupdel	groupdel grupo	Deleta um grupo
groups	groups nome_usuario	Mostra os grupos do usuário
addgroup	addgroup usuario grupo ou addgroup nomedogrupo	Cria um grupo ou adiciona um usuário ao grupo
sudo	sudo comando	Executa comandos como root
whoami	whoami	Identifica com qual usuário você esta logado

Rede		
Comando	**Sintaxe**	**Descrição**
ifconfig	ifconfig	Mostra as interfaces de rede
hostname	hostname	Mostra ou muda o nome de seu computador na rede
ping	Ping ip_desejado	Dispara pacotes para outro pc, para testar conexões etc
Sistema		
Comando	**Sintaxe**	**Descrição**
killall	Killall nome_do_programa	Mata um processo
whatis	Whatis +nome do programa	Descreve o que faz o comando
diff	diff arquivo1 arquivo2	Compara 2 arquivos
ps	ps -elf	Mostra os programas que estão rodando
cat	cat arquivo_texto	Mostra o conteúdo de um arquivo de texto
grep	Comando | grep palavra	Filtra a saída do comando, mostra a linha da palavra pedida
ln	ln -s arquivo_original atalho	Cria atalho
cp	cp arquivo destino	Copia um arquivo ou diretório (-R para diretórios)
apt-get	apt-get nome_programa	Instala aplicativos
find	Find +nome	Procura por arquivos e diretórios

Compactação		
Tar.gz (A melhor em tempo vs compactação)		
Comando	**Função**	**Descrição**
tar -zcf novo.tar.gz pasta_ou_arquivo	Compactar	z=zip c=compact f=file
tar -zxf pasta_ou_arquivo	Descompactar	x=extrair z=zip f=file
Tar (Apenas junta)		
Comando	**Função**	**Descrição**
tar -cf arquivo_novo.tar arquivo	Apenas juntar os arquivos	-c comprimir -f file
tar -xf arquivo.tar	Extrair	-x extrair -f file
Rar		
Comando	**Função**	**Descrição**
rar a novo.rar arquivo	Compacta	a = Adiciona
unrar arquivo.rar	Descompacta	
Tar.xz (Compacta mais)		
Comando	**Função**	**Descrição**
tar -Jcf arquivo_novo.tar.xz arquivo	Compactar	-J = xz -c cria -f files
tar -Jxf arquivo.tar.xz	Descompactar	-x extrai -P preserva permissões

Capítulo 2 – Variáveis

São muitos os valores que lidamos na programação e eles variam muito, por isto é que existem as variáveis, elas podem assumir qualquer valor numéricos ou alfanuméricos a qualquer momento.

Portanto, variável é um nome com um valor dentro dela que fica armazenado na memória para ser usado quando preciso. (seu nome nunca começa com número).

Para confundir menos, é recomendado escrever as variáveis com letras maiúsculas e nem precisa falar que não se pode escreve-las com acento.

Valor:

Numérico → Números armazenados (para fazermos contas)
Alfanuméricos → Podem ser números, textos ou os dois juntos, o importante saber é que sempre será considerado como um texto

Exemplificando:

Posso criar um script que necessite colher o nome de alunos, mas a cada rodada o nome será diferente, então eu posso criar a variável ALUNO que armazenará este valor dentro dela.

Com o simbolo "$" antes da variável, prevalece o valor que esta dentro dela.
Exemplo: Dentro da variável "FAZER" tenho este valor → "mkdir programa"

No script digito uma linha assim: $FAZER
O que vai acontecer ???
O shell vai dar o comando "mkdir programa" criando o diretório pedido
Por causa do simbolo ele considera o conteúdo

Exemplos de como colher os dados para a variável:
Não se preocupe em decorar estes comandos, procure entender o raciocínio.

Você mesmo dá o valor dentro do script

```
ALUNO=$"Jonatan"
```

Aqui ao invés de chamar de variável, podemos chamá-la de constante, já que o valor Jonatan não muda, a não ser que você crie outra linha modificando este valor

Recebendo o valor digitado pelo usuário

```
echo "Digite o nome do aluno"
read ALUNO
```

Onde "read" é o comando para que o usuário digite o valor da variável em questão

O Linux é case sensitive, ou seja, se escreveu determinada variável em maiúscula, você não pode mudar para minuscula no meio do Script, porque ele reconhece como outra palavra.

Pegando o valor de um arquivo-texto

```
ALUNO=$(cat /etc/matricula)
```

O valor da variável será o resultado do comando dentro dos parênteses, neste caso mostra o conteúdo do arquivo matricula.

Neste caso o usuário digita o nome do programa, e a variável PACOTE usa este resultado, exe: se o usuário digitar mplayer, a variável PACOTE terá o Valor: “protecmplayer”.

```
echo "Digite o comando do programa"
read PLAI
PACOTE=$"protec$PLAI"


PING1=$(ping -w 3 192.168.0.130)
echo $PING1 date > /home/log


RODANDO=$(ps -elf $PROG)

echo "Texto
echo "Aperte enter para prosseguir"
read segue
```

Aqui PING1 será o resultado de dois Pings, depois seu valor é gravado no Arquivo /home/log juntamente com a data. (é “-w 3” para sair 2 pings)

É normal usarmos variáveis pegando valores que envolvem outras variáveis, aqui eu uso o comando ps e a variável $PROG para acharmos a linha de algum programa, e o resultado deste comando será o valor de $RODANDO.

Neste caso o valor da variável não importa, pois, sua utilidade é de apenas parar a tela exibida para que o usuário possa ler a mensagem e dê enter para prosseguir.

As variações são muitas, pesquise e teste na medida que precisar.

Capítulo 3 - Operadores lógicos de comparação

Nós vamos ver estes operadores dentro de um determinado contexto, pois é o que necessitamos por enquanto, qualquer função que seja um pouco diferente do descrito aqui abordaremos depois do capítulo 7.

Os operadores lógicos de comparação mostram ao shell uma condição a ser testada, o resultado do teste pode ser **verdadeiro** ou **falso**, este resultado é usado por vários comandos (vamos entender estes comandos na medida do possível), um deles e um dos mais importantes é o IF, que veremos em breve.

Estes operadores são fundamentais em scripts/programação.

O que podemos entender como **verdadeiro** e **falso?**

A frase abaixo, por mais obvia que seja, ela é verdadeira:
4 é menor que 5

Esta também é verdadeira:
O nome camila é diferente do nome Julia

Por fim, esta é falsa:
50 é diferente de 50

O conteúdo destes comparadores podem ser **numéricos** ou **Alfanuméricos** (vai fazer comparação que necessita considerar o número matematicamente, então é numérico o resto é alfanumérico), por exemplo, se eu vou lidar com números de cadastros de funcionários vou considerá-los Alfanuméricos porque além de não ter conta, eu só vou ler como se fosse um texto, já o numérico pode ser quantidade, comparação (1 menor que 2), cálculos etc. Nunca poderemos confundir estas duas categorias.

Comparadores Numéricos

-lt	Número é menor que (Less Than)
-gt	Número é maior que (Greater Than)
-le	Número é menor ou igual (Less Equal)
-ge	Número é maior ou igual (Greater Equal)
-eq	Número é igual (EQual)
-ne	Número é diferente (Not Equal)

Exemplificando:

Supondo que a variável **A** tem o valor de **30** e **B** tem o valor de **20**, então poderíamos ter as seguintes situações:

Ele não executa o comando, porque a condição não é satisfeita, pois, A não é menor que B.
se [$A **-lt** $B]; então faça (se A é menor que B então faça o comando) Condição Falsa
comando

Neste caso, ele executaria o comando, porque a condição é satisfeita, já que A é diferente de B
se [$A **-ne** $B]; então faça Condição Verdadeira
comando

Comparadores Alfanuméricos

=	Texto é igual
!=	Texto é diferente
-n	Texto não nulo
-z	Texto é nulo

Suponhamos agora que **A** contém a palavra "**noite**" e **B** contém a palavra "**dia**", então:

Aqui ele não executa o comando, porque A não é igual a B.
se [$A = $B]; então faça
comando

Aqui ele executa o comando, já que A é diferente de B.
se [$A **!**= $B]; então faça
comando

Com o "-n" ele executará o comando caso a variável tenha algum valor, já com o "-z", só executará se a variável estiver vazia. Embora esteja na tabela de alfanuméricos eles também funcionam se a variável conter apenas números. Vamos ver como escrevê-los no próximo capítulo.

Capítulo 4 – Condição IF e escrevendo o primeiro script

Condição IF

No inglês “if” significa “se”, (SE a condição for satisfeita eu executo o comando), condição que será testada usando o conceito anterior. Se o teste de comparação acusou verdadeiro ele executa o comando que está dentro do if, o teste acusando falso ele pula este if e segue com o script.

Pense nos operadores de comparação como se fossem chaves e o if sendo a porta, se a chave for “verdadeira” ela abre a porta e executa o que tem lá dentro, se a chave for falsa o shell não consegue abrir a porta e consequentemente não executa. Neste capítulo aprenderemos como escrevê-lo na linguagem do shell e consequentemente o primeiro script.

Estrutura do if

```
if [ condição ];then
   comando
fi
```

- if → Abre o comando
- Condição → Condição comparativa para execução dos comandos dentro do if
- Comando → Qualquer comando do shell e quantos você quiser
- fi → Fecha o if (escrito ao contrário)

Exemplo na linguagem do shell:

→Espaço obrigatório

A palavra julia está entre aspas porque é um texto (numéricos ficam sem aspas), a mesma coisa vale para a variável.

```
if ["$USUARIO"="julia"];then
   mkdir $USUARIO
fi
```

```
#!/bin/bash

if [ "$USUARIO" = "julia" ];then
mkdir $USUARIO
fi
```

No Shell

Tradução ↑
Se o conteúdo da variável USUARIO é igual a palavra julia então execute o comando, que no caso cria um diretório com o nome de julia.

Comando auxiliar Else

Uma função complementar e muito útil no comando if é o else (senão), caso a condição do if não seja verdadeira, ele automaticamente executa o que está no else, seu uso é opcional.

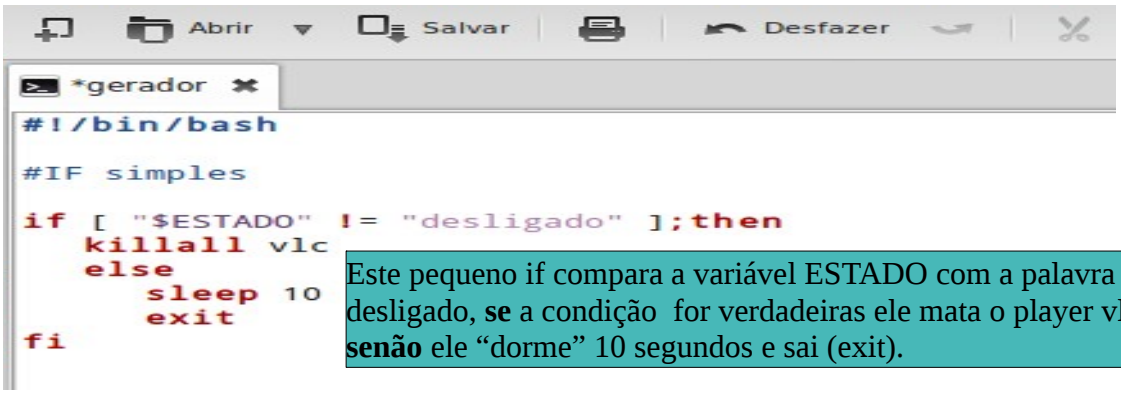
Sentido do comando	Sintaxe no shell
se [condição é verdadeira]; **então execute o comando** mkdir $USUARIO **senão execute este outro comando** mkdir $USUARIO2	if [condição];then comando else comando fi

Abaixo um exemplo escrito no editor de texto:

Valor da variável a seguir

ESTADO=$"operando"

Se a variável ESTADO é diferente de "desligado", então execute o comando killall... **senão** ...



Este pequeno if compara a variável ESTADO com a palavra desligado, **se** a condição for verdadeiras ele mata o player vlc, **senão** ele "dorme" 10 segundos e sai (exit).

Agora que sabemos escrever a condição if, vou mostrar como se escreve os operadores lógicos de comparação "-n" e "-z", prometidos anteriormente, eles são escritos antes da variável a ser comparada como nula ou não nula.


```
if [ -z "$PORTA" ];then
echo "Variavel sem valor"
fi
```

Se a variável PORTA estiver vazia ele mostra o texto na tela: Variável sem valor

```
if [ -n "$PORTA" ];then
echo "A variavel não esta vazia"
fi
```

Ela recebendo algum valor ele mostra:
A variável não esta vazia

Poderíamos dar o "pulo do gato" e colocar este script com apenas um if → **se** a variável não tem valor, mostre na tela: "variável sem valor" **senão** mostre: "A variável não esta vazia"

```
#!/bin/bash

if [ -z "$PORTA" ];then
   echo "Variavel sem valor"
   # Senão quer dizer que ela tem algum valor
   else
      echo "A varivel não esta vazia"
fi
```

Supondo que você necessite rodar um comando apenas se a variável conseguiu colher os dados do comando "ps" por exemplo (para verificar se determinado programa esta rodando), não hesite em usar o -z e o -n.

Antes de fazermos o nosso primeiro script vamos "juntar as peças": vimos os comandos básicos, o conceito de variáveis, os operadores de comparação e por último o comando if. Se juntarmos isto mais as pequenas dicas a seguir seremos capazes de fazer o primeiro script.

↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓ **Nosso primeiro script**

Vamos fazer nosso primeiro script/exercício, temos ele pronto no capítulo de resolução de exercícios, mas é claro que você vai resolvê-lo antes de conferir, não precisa ficar igual, basta funcionar.

Onde escrever os scripts

Uma função interessante em alguns editores de texto no Linux, é que eles facilitam a visualização e escrita de scripts, colorindo os comandos e suas estruturas, recomendo o gedit ou pluma. Para que esta função seja ativada é necessário colocar no começo do script #!/bin/bash ou #!/bin/sh etc, e salvar, esta linha especifica o interpretador de comandos, caso não tenha o shell executa qualquer um. (caso não fique colorido basta modificar a opção que fica no rodapé mudando de "texto sem formatação" para "sh")

Passos a passo de como fazer o script, antes do exercício:

- Abra o editor de texto escolhido
- Escreva na primeira linha #!/bin/bash
- Agora vamos salvar para que as linhas fiquem coloridas
- Tem pessoas que colocam a extensão .sh no nome do arquivo (tanto faz)
- Para não termos problemas futuros, já vamos dar permissão de execução (chmod +x script)
- Agora já podemos escrever o script
- Quando o script estiver pronto entre no diretório em questão e dê o comando ./nome_do_script para executá-lo

Lembretes:

- Preste muita atenção nos espaços, porque se der espaço a mais ou a menos não vai rodar.
- Lembre-se de sempre fechar o if → "fi"
- Os comandos sempre são executados de cima para baixo sequencialmente, ou seja, só executa a próxima linha quando terminar a atual, a não ser que usemos o "&"
- Quando houver erros, o terminal mostra em qual linha o mesmo está, nem sempre é aquela linha
- Você pode saltar quantas linhas quiser

Exercício 1 – Primeiro Script

Função do script → Temos 3 médicos, cada um atende num turno diferente, o usuário informa qual turno deseja se consultar e o programa mostra o nome do médico do turno escolhido, só isso!

Etapas do script:

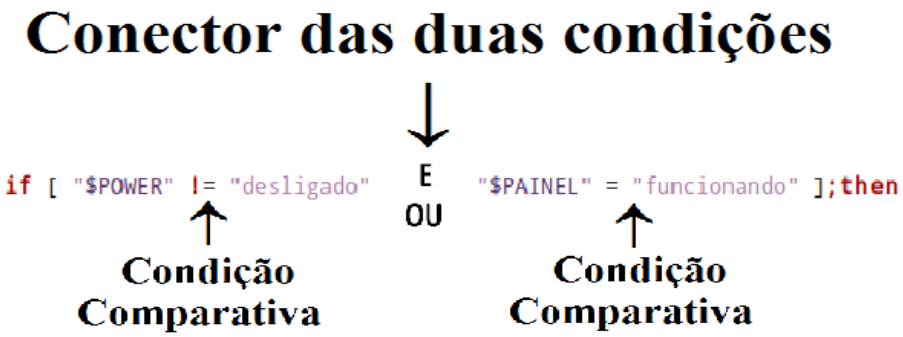
1- Mostra um texto de boas vindas ao usuário
2- Pede que o usuário escolha qual turno que deseja se consultar
3- Mostra o texto nestes moldes: Médico TAL é o único a atender no turno TAL
4- "Dorme" por 2 segundos
5- Informa que a consulta está marcada
6- O programa "dorme" por mais 4 segundos e sai

Resolução

Capítulo 5 – Operadores Lógicos de Conexão

Com estes operadores podemos conectar duas ou mais condições criadas com os operadores de comparação, criando assim mais de um evento a ser testado pelo shell, abaixo aprenderemos sobre os conectores **E** e **OU**.

Vamos visualizar como ficam os conectores para ficar mais claro:



* No exemplo acima temos dois conectores, mas é só exemplo, já que o correto ali seria apenas um conector.

Operador lógico (E)

Entenda **conjunção** sendo a união das condições comparativas feitas pelos conectores.
rasgando o verbo: É tudo que esta dentro dos colchetes do if [ ]

Aqui a **conjunção** é verdadeira se todas as condições de comparação forem verdadeiras, então mostro a tabela para entendermos a lógica.

Nº	Condição 1	Conector	Condição 2	Resultado do teste	Explicação
1º	V	**e**	V	Verdadeiro	Porque as duas condições são verdadeiras
2º	F	**e**	V	Falso	É falso porque apenas uma condição atende
3º	V	**e**	F	Falso	É falso porque apenas uma condição atende
4º	F	**e**	F	Falso	Nenhuma condição atende

Exemplificando:

Vamos considerar esta lógica como se fosse um porteiro e ele libera o acesso de acordo com a situação descrita abaixo:

Só entra na festa casais que o homem se chama César **E** a mulher Juliana:

1º	César	**e**	Juliana	Entrada permitida (as duas condições são verdadeiras)
2º	Paulo	**e**	Juliana	O nome Juliana bate mas o nome Paulo não atende, barrados !
3º	César	**e**	Mônica	O nome César está na lista mas o nome Mônica não, barrados !
4º	Júlio	**e**	Carolina	Nenhum dos dois nomes estão na lista, barrados !

Muitas vezes vamos esbarrar com a necessidade de usar os conectores, vamos supor que eu preciso de um if rodando apenas em duas situações:

Eu fiz um script que diminuía a velocidade dos meus downloads para 100k quando a minha irmã conectava o notebook, mas quando o pc da minha mãe estava ligado simultaneamente, como eu poderia diminuir a velocidade para 50k compensando duas máquinas ligadas? Então eu criei um if para esta situação.

Esta é a parte que identifica as duas máquinas ligadas:

```
if [ “$NOTE” = “ligado” E “$PCMAE” = “ligado” ];then
  Comando (wondershaper eth0 400 que é igual a → 50k )
fi
```

Para o conector **E**, basta lembrar:
se é tudo verdadeiro então executa.

Eu usei vários comandos para chegar no valor ligado antes de ser comparado e para outras velocidades, mas aqui vamos nos prender apenas ao sentido do comando.

Operador lógico (OU)

Aqui a conjunção é verdadeira se uma ou outra condição for verdadeira (sendo as duas verdadeiras também é válido):

Nº	Condição 1	Conector	Condição 2	Resultado do teste	Explicação
1º	V	**ou**	V	Verdadeiro	Porque pelo menos uma condição é verdadeira
2º	F	**ou**	V	Verdadeiro	Temos uma condição verdadeira, e é suficiente
3º	V	**ou**	F	Verdadeiro	Temos uma condição verdadeira, e é suficiente
4º	F	**ou**	F	Falso	Nenhuma condição verdadeira para validarmos

Exemplificando:

Vamos usar o mesmo exemplo da conexão anterior, só que desta vez eu preciso que apenas uma condição seja verdadeira para que ele execute o comando, trocaremos o **e** pelo **ou**.

Só entra na festa casais que o homem se chama César **OU** a mulher Juliana:

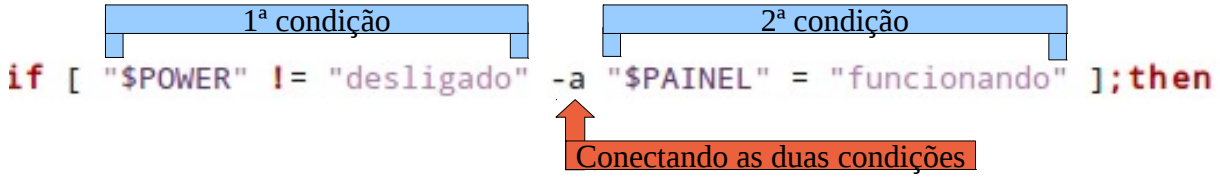
1º	César	**ou**	Juliana	Entrada permitida, as duas condições são verdadeiras
2º	Paulo	**ou**	Juliana	Entrada permitida, pelo menos uma condição verdadeira (juliana)
3º	César	**ou**	Mônica	Entrada permitida, a condição (César) valida a entrada
4º	Júlio	**ou**	Carolina	Nenhum dos dois nomes estão na lista, barrados !

Pegando o exemplo anterior do script que diminui a velocidade da internet, podemos pensar na seguinte situação: E se eu quisesse diminuir a velocidade da internet para 100k independentemente da quantidade de pcs ligados, ou seja, se o pc da minha irmã **OU** o pc da minha mãe estiverem ligados, ou se os dois simultaneamente diminui para 100k e pronto.

```
if [ “$NOTE” = “ligado” OU “$PCMAE” = “ligado” ];then
  Comando (wondershaper eth0 800 que é igual a → 100k)
fi
```

Na linguagem do shell o operador “**e**” é representado como “**-a**”
e o operador “**ou**” é representado por “**-o**” (não confundir com -zero)

Como mostrado abaixo:



Do exemplo acima podemos entender: Se a variável POWER é diferente de desligado **E** a Variável PAINEL é igual a funcionando então faça o comando.

Capítulo 6 – Usando os Conectores

Vamos abordar mais alguns exemplos dos conectores e partiremos para a prática a seguir.

Podemos usar quantos conectores quisermos:

```
if [ "$P1" = "of" -a "$P2" = "of" -a "$P3" = "of" ];then
```

Só todas as condições sendo verdadeiras é que ele executa

```
if [ "$P1" != "of" -o "$P2" != "of" -o "$P3" != "of" ];then
```

Se pelo menos uma for verdadeira ele executa o comando

Exemplo de um script completo, que usa o recurso dos conectores:

Note que os comandos if não estão usando aspas nas variáveis e nem nos valores, justamente por se tratarem de valores numéricos/matemáticos.

saudacao

```
#!/bin/bash

# Este Script dá boa tarde bom dia e boa noite de acordo com o horario

# Escreve as horas na variável DATA
DATA=$(date +%H)

# Agora que identificamos a hora, o programa executará os audios nos horarios corretos

# Se a variável DATA é igual ou maior que 13 e menor que 18 entao execute o arquivo boatarde.mp3
if [ $DATA -ge 13 -a $DATA -lt 18 ] ; then
   mpg123 /usr/share/boatarde.mp3 # Chama o programa para rodar o boatarde.mp3
fi

# se a hora é menor que 13 entao execute o arquivo bomdia.mp3
if [ $DATA -lt 13 ] ; then
   mpg123 /usr/share/bomdia.mp3
fi

# se a hora é maior que 18 entao execute o arquivo boanoite.mp3
if [ $DATA -ge 18 ] ; then
   mpg123 /usr/share/boanoite.mp3
fi
exit
```

Usando o conector

Neste script fui obrigado a usar o conector -a para definir o horário da tarde, já que “estar de tarde” significa ser mais que 13 horas **E** menos que 18 horas.

Exercício 2 - Escrevendo scripts mais elaborados

A função do script é informar a classe do carro conforme pedido pelo usuário

Ferrari e Lamborghini	Celta	Palio e Uno
Classe A	Classe B	Classe C

Etapas do script:

- Informar os carros disponíveis ao usuário
- Pede ao usuário que digite qual carro deseja ver a classe
- Mostra a classe do carro e sai

Aqui não vale opção numerica o usuário deve digitar o nome do carro, para alcançarmos o objetivo do exercicio

Conseguindo fazer o script acima teremos atingido os nossos objetivos até aqui, agora faremos um script que exige um pouco mais de raciocínio, se você não conseguir fazer não se preocupe, pode continuar com os próximos capítulos que o raciocínio lógico mais apurado vem com o tempo.

Exercício 3 – Lógica dos PC1 e PC2

Nós temos 2 pcs, vamos perguntar ao usuário quais pcs estão ligados (um de cada vez), depois do usuário ter digitado “ligado” ou “desligado” para cada pc, o programa mostra a soma na tela (não é soma feita no shell, leia o quadro abaixo para saber).

Conforme os valores abaixo vamos mostrar na tela o valor total referente a soma dos computadores ligados. Supondo que o PC1 e PC2 estejam ligados, então o valor mostrado é 15, se for apenas o PC2 o valor por sua vez será 10 e assim sucessivamente, o desafio será estruturar os ifs para que não apareçam informações demais ou de menos.

PC1	5
PC2	10

Atenção, não usaremos a função matemática de somar, porque nem aprendemos ela ainda, você vai fazer a soma de cabeça e colocar o resultado com o comando echo. O objetivo aqui é raciocinar e fazendo pela matemática iria ficar muito simples e não é nosso objetivo neste exercício

Temos os incrementos abaixo que poderemos adicionar ao exercício opcionalmente oferecendo a você mais algumas atividades para aguçar o raciocínio em Shell Script, faça um backup do script feito acima e adicione as funções pedidas abaixo.

Incrementos:

Se digitar ligado ou desligado errado, avisar o usuário e sair

Se o usuário digitar desligado para os dois, retornar o valor de zero

Resolução

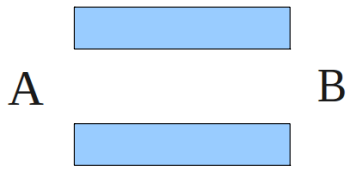
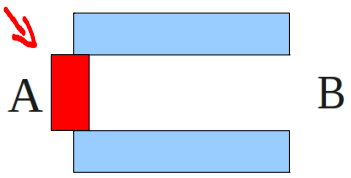
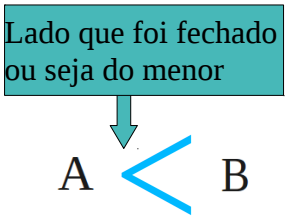
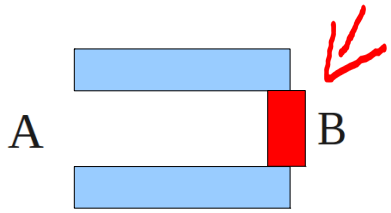
Capítulo 7 – Fazendo loops com o “for” e script remoto

Já parou para pensar como podemos deixar um script independente, que fique ali de prontidão vigiando as situações e executando comandos de acordo com o que acontece, sem precisar de nenhum gatilho humano, o tempo que for, como se fosse uma sentinela.

Isto é possível com o “for”, ele é um dos comandos que possibilita a inserção de loops em scripts, a sua tradução é “para”, ou seja, para determinada condição faça o comando enquanto ela for verdeira.

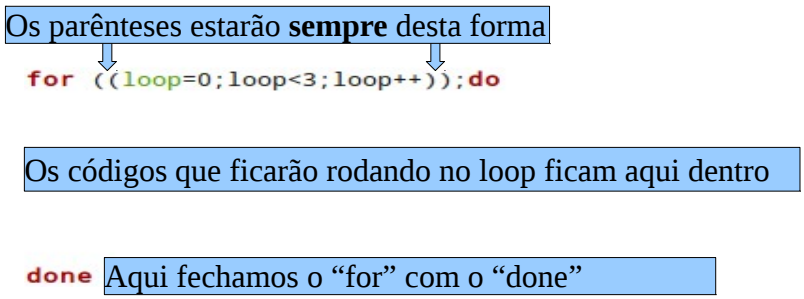
Maior e menor

Ao usar o comando “**for”** temos que saber o sentido destes dois comparadores (“>” “<”), eu uso a seguinte regra para não confundi-los:

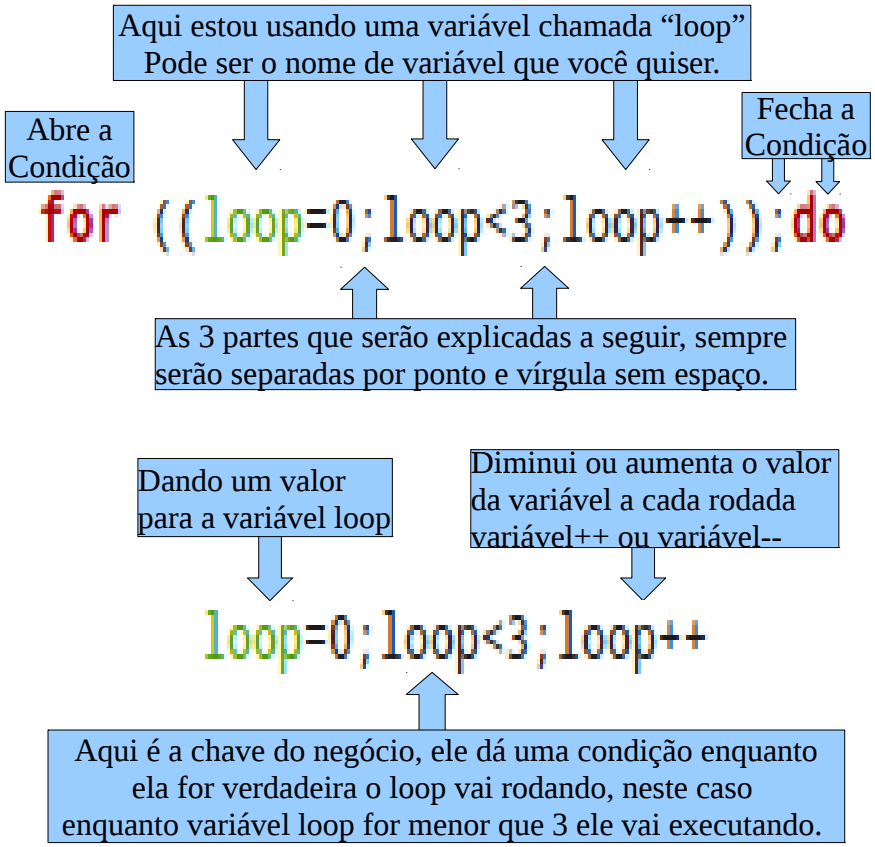
Pense em duas barras 	Eu quero dizer para o shell que **A é menor que B** então eu diminuo o lado da barra em que está o A. 	Ficando assim: 
Agora eu quero dizer que **B é menor que A** (ou A é maior que B). Então é só inverter a lógica. 	Ficando assim: 	Eu posso trocar o B de lugar com o A? Ficaria a mesma coisa!  O comando que apresentarei não podemos inverter os lados.

Entendendo o comando e criando loops limitados

Sintaxe



Continuando com a sintaxe



A tradução do comando “for”:

```
for ((loop=0;loop<3;loop++));do
    1      2      3
```

1 **Para** a variável loop que vale zero
2 Rode os comandos enquanto ela for menor que 3
3 E a cada rodada acrescente + 1.

Atenção

Este loop rodará 3 vezes, nestes valores da variável loop → → → 0, 1 e 2

Acontece isto:

Ele verifica a condição e roda uma vez, $loop passará a valer 1, ele verifica se loop estiver menor que 3 ele roda novamente, depois ele verifica e vê que loop vale 2 e que a condição ainda é verdeira e roda novamente, agora loop vale 3 e não é menor do que 3, então ele deixa de executar o “for” e continua o script, já que a condição passou a ser falsa.

Como enchemos o comando de placas, coloquei o código ao lado sem nenhuma poluição. → → →

```
#!/bin/bash

for ((loop=0;loop<3;loop++));do

   echo $loop

done
```

É interessante você colocar o comando: echo $loop (dentro do “for”) para ir vendo na tela quantos loops dará

A condição deve ser verdadeira ao passar pelo “for” na primeira vez, caso seja falsa passará direto e não voltará para executá-lo, então preste atenção na hora de montar a lógica. Salvo em momentos que você colocar um loop dentro de outro loop e o primeiro rodará até criar condições de cair no segundo.

Criando loops infinitos

Trazendo a lógica que vimos acima, para criarmos loops infinitos, basta fazer uma condição que jamais deixará de ser verdadeira. Exemplo:

```
for ((loop=2;loop>1;loop++));do
```

Conforme mostrado acima, você deve concordar comigo que:

- Loop tem o valor de 2
- Para rodar o “for” loop deve ser maior que 1 (e a condição já é verdadeira)
- E o valor de loop nunca vai diminuir, sempre vai aumentar ++

Ou seja, o loop nunca será menor que 1 para que a condição seja falsa, então ele rodará “para sempre”.

Se criarmos um loop com comandos rápidos, o script dará milhares de voltas em poucos segundos, é recomendado colocar um sleep dentro do “for” limitando assim o tempo das rodadas.

Tenho mais duas formas de escrever o “for” que funciona um pouco diferente do que expliquei anteriormente, como eu nunca usei na prática vou apenas apresentar.

```
for cor in azul vermelho amarelo verde
do
   echo $cor
done
```

Aqui ele rodará 4 vezes, uma para cada parâmetro, sendo que cada vez que ele rodar, a variável “cor” terá o valor de um parâmetro, que são: azul, vermelho ...

```
for PAR in $*
do
   echo $PAR
done
```

Usamos o mesmo conceito neste “for”, a diferença é que o usuário digita os parâmetros (quantos quiser). Ele digita o comando do script, espaço, depois os parâmetros. ./script parâmetro1 parâmetro2 parâmetro3 ...

Veremos o conceito de parâmetros futuramente, ele tem utilidade no script como um todo e não somente no “for”.

Imagine que você está saindo de casa faltando 20 minutos para terminar aquele download e você precisa ficar presente estes 20 minutos para depois abrir outro programa de download, torrent por exemplo já que ele atrapalharia o download atual, depois disto seria necessário desligar o pc após 2 horas, converter aquele vídeo pesado ou dar um comando demorado etc. Mas como fazer isto sem estar presente?

Seja bem-vindo ao nosso exercício 4 – script remoto

O script remoto ficará ao seu dispor para executar qualquer comando, usaremos o Dropbox, assim pelo celular ou qualquer dispositivo com internet você poderá comandar o script, mas fique à vontade de não usar o Dropbox e colocar os arquivos em qualquer diretório do pc, já que a nossa intenção é o aprendizado e testar nosso script.

Para o script apontar corretamente a home de qualquer usuário que o execute, basta usar uma das duas opções abaixo:

$HOME ou /home/$USER

Exemplo: echo “teste” > $HOME/arquivo se transformará em: **/home/luiz** por exemplo

Funcionamento

- O script rodará 1 vez a cada minuto
- A cada rodada ele lê o arquivo-texto
- No arquivo-texto escreveremos o comando desejado
- Ele executa o comando digitado e limpa o arquivo-texto

Avisos

- Zere ou crie o arquivo “comando” usando o → echo "" > $HOME/comando (fora do loop)
- Você deverá usar um certo recurso para o script não ficar agarrado no comando (cap. 1)
- Use → echo $loop dentro do “for”, senão vai parecer que deu pau
- Na construção do script diminua o tempo de loop para fazer os testes

Se o script for desligar o computador ou qualquer outro comando que exija privilégios, rode ele como root, caso contrário execute como usuário normal que é bem melhor já que abrindo um programa como root as configurações estarão diferentes do normal.

Alguma duvida de como o script funciona ?
Veja o vídeo com ele rodando, no link abaixo

http://www.mediafire.com/download/7ceth5ukcc4c5vq/video_exercicio_4.ogv

Incremento

- Se escrever → vivo? - ele responde → “sim vivo, esperando o comando”, a reposta estará num arquivo de leitura (avisos).

Resolução

Capítulo 8 – While e Until

Digamos que os dois comandos descritos aqui tem um pouco do “if” e um pouco do “for”, um pouco do “for” porque eles também tem como característica rodar em loop e um pouco do “if” porque tem a mesma estrutura e necessitam de uma condição para rodar.

While

While em inglês significa “enquanto”, ou seja, enquanto a condição for verdadeira faça o comando, no “if” se a condição é verdadeira ele roda uma vez, aqui ele vai rodando **enquanto** ela for verdadeira (fica “agarrado” dentro dela até mudar para falsa).

Podemos usar um valor numérico para determinar quantos loops teremos, podemos dar opção do usuário digitar se quer tentar novamente etc. O importante é usar uma variável para “pescar” o while ou until.

Sintaxe

```
while [ Condição ];do
  comando
done
```

Da mesma forma que o if, no while e until usaremos os operadores lógicos de comparação ensinados no cap. 3

Tanto no while quanto no until o valor da variável que ele usará como referência para rodar deve estar definido antes de chegar no comando em questão.

O loop do while pode usar para rodar:

Números

Usando o mesmo conceito do “for” podemos ver que o script abaixo dará cinco voltas: 0, 1, 2, 3 e 4. Eu não ensinei a lidar com matemática, mas basta seguir o conceito abaixo, não coloque aspas na soma e respeite os espaços visíveis (a mesma coisa vale para a condição comparativa).

```
#!/bin/bash

VOLTA=$"0"

while [ $VOLTA -lt 5 ];do

echo $VOLTA

VOLTA=$[ $VOLTA + 1 ]

done
```

Mesmo sendo um valor numérico nós damos este valor usando aspas

Condição: **Enquanto** VOLTA for menor que 5 faça o comando

Comando dentro do loop

Aqui eu somo +1 para contar mais um loop dado: variável $VOLTA é igual ao valor de $VOLTA +1

Fechando o while

Este script diz:

- Volta é igual a zero
- Enquanto $VOLTA for menor que 5 faça o comando
- Adicione +1 na variável VOLTA

Um exemplo prático foi o que usei no proteccontinuo, eu perguntava ao usuário quantos players ele desejava adicionar no programa e quando o script fosse criar os players ele pegava a variável que usei com o usuário, então se o usuário respondeu 3 o while dava 3 loops, repetindo o comando dentro do while 3 vezes e consequentemente criando o proteccontinuo para 3 players.

Texto

Podermos pegar textos para dar voltas no while usando a saída de um comando por exemplo, eu faço um programa de verificação e **enquanto** a saída deste comando for determinada palavra o while vai rodando até que ela mude e saia do loop. As variações são infinitas, mais para frente aprenderemos a filtrar textos/saídas de uma forma que possamos resumi-los a uma linha, uma palavra etc.

A seguir usaremos um loop que dará voltas e voltas até que o usuário decida passar adiante.

```
#!/bin/bash

VOLTA=$"sim"

while [ "$VOLTA" = "sim" ];do

echo "Digite sim se deseja tentar novamente ou con para continuar"
read VOLTA

done
```

Garantindo que entrará no loop, poderíamos perguntar isto ao usuário também

O script poderá fazer algo que só o usuário pode decidir quando prosseguir, então ele responde continuar e o while automaticamente sairá do loop.

É importantíssimo entender que o “movimento” para tirar o while ou until do loop deve estar dentro dele, senão rodará “para sempre”.

Until

Until significa até → até que determinada condição seja verdadeira execute, ou seja, ele executa se a condição for falsa e só termina quando ela for verdadeira. Então eu posso usar uma variável de valor 5 e **até** que ela se torne 10 vai rodando e dentro do until usaremos a mesma sintaxe de soma que usamos no while. O dado que determina o loop também pode ser texto, mas não acho interessante, o importante usá-lo quando achar necessário executar o “**até”**.

Sintaxe

```
until [ Condição ];do
  comando
done
```

Podemos ter mais de uma condição tanto no while quanto no until usando os conectores “E” e “OU” apresentados no cap. 5

Ele é idêntico ao while, tendo apenas a sua **lógica invertida,** tudo que abordamos com o while anteriormente vale para o until, então não explicarei novamente, vou dar um exemplo:

```
#!/bin/bash

VOLTA=$"0"

until [ $VOLTA -gt 5 ];do

echo $VOLTA

VOLTA=$[ $VOLTA + 1 ]

done
```

Até que $VOLTA seja maior que 5 então faça o comando. Ele vai rodando até que a condição se torne verdadeira, ou seja, **até** $VOLTA ser maior que 5.

As suas voltas serão: 0, 1, 2, 3, 4 e 5

Exercício 5 – Pedindo senha ao usuário

O que o script faz:

- Dá boas vindas
- Pede senha ao usuário

A senha pode ser 123

- A senha sendo correta ele executa o comando → echo “senha correta” e sai
- Caso contrário o usuário tenta novamente sem sair do script (loop)
- São até 5 tentativas e depois das 5 o script mostra “tentativas esgotadas” e sai

O desafio aqui é a lógica e saber quais comandos colocar dentro do loop.

Este script terá utilidade quando aprendermos sobre funções, já que ela é um atalho que nos joga para determinada parte do script, então supondo que façamos um programa de código fechado, o usuário só conseguiria executar determinada parte do script se digitasse a senha corretamente e o atalho estaria dentro do while ou until.

Resolução

Capítulo 9 – Operações matemáticas e inicialização de scripts

Já vimos um pouco de matemática quando somamos o valor da variável +1 para contarmos o loop:

```
VOLTA=$[ $VOLTA + 1 ]
```

Ou usando o sinal de menos ↓↓↓ (esquece multiplicação e divisão nesta sintaxe)

```
VOLTA=$[ $VOLTA - 1 ]
```

Quando o script precisar literalmente fazer contas igual uma calculadora, usamos a sintaxe abaixo, deixando a conta sempre entre os dois parênteses e associando o resultado a uma variável, neste comando não faz diferença escrever com ou sem espaço.

```
#!/bin/bash

CONTA=$((10 * 2))

echo $CONTA
```

Os símbolos matemáticos usados no shell são os que estamos acostumados

\+

\-

*

/

Acima a variável CONTA é igual ao resultado da operação entre parênteses e no echo eu mostro o resultado.

Uma vez usei este conceito para redundância, se o comando retornasse 64 eu executaria o comando 2, só que apenas uma verificação poderia ter falhas, então coloquei três verificações somei todas elas usando suas variáveis e na lógica de comparação eu dizia que poderia executar o comando **se** o valor fosse igual ou maior que 64, ou seja, se até duas verificações falhassem eu conseguiria o valor 64 o que indicava que pelo menos uma estava funcionando. Abaixo uma aplicação das operações matemáticas usando variáveis.

```
#!/bin/bash

N1=$"3"
N2=$"2"

CONTA=$(($N1 * $N2))
```

Infelizmente o shell só mostra resultados inteiros se a resposta for fracionária ele mostra como inteiro ou as vezes não funciona.

É difícil dar exemplos práticos, eu usei pouquíssimas vezes contas em script, a não ser que o seu programa seja mais voltado para matemática.

Devemos salientar que se o valor for considerado matemático devemos ter cuidado com as aspas no if, while etc.

Aqui fazemos o comando considerando a condição como um valor matemático	E aqui como um texto, colocando aspas na variável e item a ser comparado
`if [ $CONTA -eq 1 ];then`	`if [ "$CONTA" = "1" ];then`

Podemos ter outras variações que podem dar certo, você sabendo estas 2, vamos complicar pra quê?

Inicializando scripts com o sistema e programando-os

Daqui para baixo não se preocupe em decorar nada, use para consultas futuras.

Além das opções abaixo você também pode colocar o script nos Aplicativos de Sessão

Inicialização

Rc2.d

Com o diretório rc2.d o script será uma das últimas coisas a serem executadas como root, basta seguir o passo a passo abaixo:

coloque o script em → /etc/init.d
coloque seu atalho em → /etc/rc2.d/

Atalhos começando com nome S99 são os últimos executados, podendo ficar assim:
/etc/rc2.d/S99meus_cript

Se você quiser trabalhar melhor a ordem de execução, é só dar um "ls /etc/rc2.d" e verificar os scripts que constam lá.

Rc.local

Basta colocar o endereço do script no arquivo-texto → etc/rc.local

Exemplo:
/etc/meu_script

Inicializando com usuário especifico:
sudo -u usuário /etc/script

Também podemos programar o script para executar em determinados dias e horários.

Sempre deixe a ultima linha do crontab vazia

Cron

/etc/crontab
service cron restart

Editamos o arquivo /etc/crontab levando em consideração os campos abaixo.

Campo	Função	Preenchimento
1º	Minuto	0-59
2º	Hora	0-23
3º	Dia do Mês	1-31
4º	Mês	1-12
5º	Dia da Semana	0 Domingo, 1 Segunda ...
6º	usuário	root luiz etc.
7º	Programa para execução	Comando

*O 6º campo pode ser omitido, mais evite fazer isto, porque costuma dar pau

Ficando assim → 20 10 2 12 2 luiz /home/luiz/meu_script

Do comando gerado acima podemos entender → aos 20 minutos das 10 horas do dia 2 de dezembro numa terça feira o usuário luiz executará o "meu_script" que está em /home/luiz

Referência do Cron: http://www.hardware.com.br/dicas/cron.html

Capítulo 10 – Comando Case

É muito mais fácil criarmos menus com um comando próprio, na programação nós temos o "case" que facilita em muito a construção destas estruturas, com ele podemos colher dados de uma variável que indicará qual opção o case deve executar.

Estrutura do case

Supondo que criamos um menu com 3 opções para o usuário escolher uma, o case executa o comando de acordo com o valor recebido pela variável.

```
case $VARIAVEL in
```

Caso o valor da variável esteja **em** → 1, 2 ou 3 faça os comandos correspondente a seu valor

```
1)
Comando;;
```

O número ou palavra da opção fica antes dos parênteses e do lado ou abaixo ficam os comandos a serem executados (1 em cada linha).

```
2)
Comando;;

3)
Comando;;
```

Depois de adicionar todas as linhas daquela opção devemos colocar no final do ultimo comando ponto e virgula duas vezes para fechá-la.

```
esac
```

Aqui fechamos o case escrevendo ele ao contrario

Exemplo na prática

```
echo "Qual é o comando do seu Screensaver?"
echo
echo "1- mate-screensaver"
echo "2- gnome-screensaver"
echo "3- XScreensaver"
echo "4- Outro"
```

Mostrando o menu para o usuário

```
read distro
echo
case $distro in
```

Colhendo o valor da variável e abrindo o case

```
1)
SCREN=$"mate-screensaver";;

2)
SCREN=$"gnome-screensaver";;

3)
SCREN=$"xscreensaver";;

4)
echo "Digite o comando
read SCREN;;
```

Repare que na opção 4 temos dois comandos, o ponto e virgula só consta no ultimo comando

```
*)
echo "Opção invalida"
exit;;
```

Aqui eu coloquei a opção "*", ou seja, se o usuário digitar algo que não consta nas opções ele cai aqui

```
esac
```

Não existem limites na quantidade de linhas que colocaremos nas opções, eu já cheguei a colocar verdadeiros scripts dentro de cada uma.

É isto, não temos muito o que falar do case, com ele basta criarmos condições para que a variável receba algum valor antes de executá-lo e sempre repetir sua sintaxe:

- Abertura do comando → case variável_usada in
- Opções que ficam antes dos parênteses → número_ou_texto)
- Terminando os comandos da opção, colocar ponto e vírgula 2 vezes no final do último comando
- Fechá-lo escrevendo case ao contrário → esac

Podemos precisar mostrar na tela textos usando o echo que contenham acentos ou símbolos, neste caso é obrigatório colocá-los entre aspas, senão é erro atrás de erro, eu sempre coloco entre aspas assim é menos uma coisa para se preocupar (as variáveis que ficarão dentro de aspas fazem a sua função de mostrar seu conteúdo normalmente)

Exercício 6 – Pequeno desafio de lógica

A variável recebe um número digitado pelo usuário

São 3 rodadas no total e em cada uma delas o script multiplica o número dado pelo usuário com o número indicado abaixo:

1º rodada → 10
2º rodada → 20
3º rodada → 30

Então se o usuário digitou 3, serão 3x10 3x20 e 3x30 mostrando na tela um resultado a cada loop, o desafio aqui é fazer uma multiplicação diferente a cada rodada já que não podemos prever o número que será digitado pelo usuário.

Exemplo de como deve aparecer na tela:

```
luiz@linuxmint ~ $ ./6
Por favor digite o número a ser processado

30
Rodada 1 - 30 multiplicado por 10 é igual a 300
Rodada 2 - 30 multiplicado por 20 é igual a 600
Rodada 3 - 30 multiplicado por 30 é igual a 900
```

Existem várias formas de se fazer este pequeno script, aqui usaremos o **while** e **case** para alcançarmos os nossos objetivos. Se você tiver dificuldades saiba que usaremos 90 % do tempo para pensar e 10 % para escrever o script.

Resolução

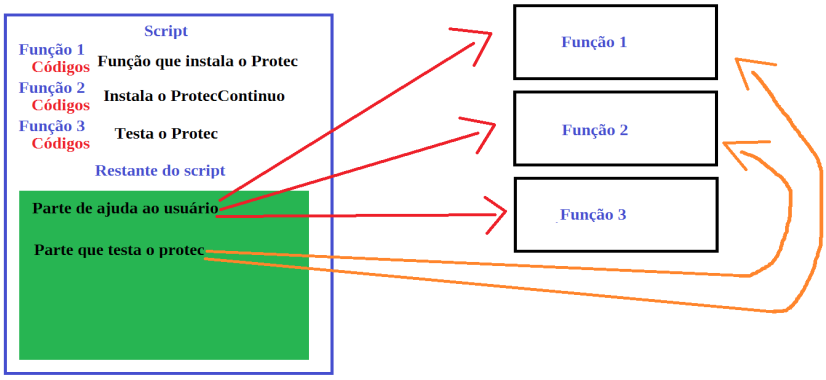
Capítulo 11 – Função e Parâmetro

Função

O papel da função é segmentar vários códigos no scripts tornando-os acessíveis a qualquer momento e em qualquer parte, usar a função é como colocar vários scripts num só script. Um exemplo prático disto é o protec onde eu tinha muitas opções a serem executadas de várias partes do script (o case não resolveria).

Para quem não conhece o protec (não deixa o protetor de tela ficar entrando nos nossos vídeos), é só baixá-lo no link → http://www.mediafire.com/download/rrdrwl0ceicz8y9/protec.zip

No desenho ao lado podemos reparar que temos 3 funções e elas são acessadas dentro dos códigos de ajuda e teste.



Não seria viável fazê-lo com o case porque eu teria que repetir os códigos em várias partes do script e o que eram quase 1000 linhas se tornariam 1500 ou mais.

Na parte de ajuda em determinado momento eu perguntarei ao usuário se ele deseja instalar o Protec, ProtecContinuo ou testar, usando o valor da variável vou comparar no if ou case executando assim a função que desejamos. Então no desenho apresentado podemos visualizar com as setas para onde o shell pode ir e da mesma forma poderíamos criar acessos de vários pontos para vários pontos.

Entenda que podemos fazer uma “salada mista” com os comandos que vimos até aqui, while dentro de while, if dentro de if e dentro deste if um while etc. Não fique com receio de fazer estes tipos de códigos porque muitas das vezes é esta “bagunça” que deixa o nosso script mais fácil de montar

Sintaxe

A sintaxe da função será sempre a mesma, como mostrado abaixo, primeiro o nome dela (no meu exemplo “instala”) depois dois parênteses sem espaço dentro e uma chave apontando para esquerda. Nas linhas abaixo você coloca os códigos a serem executados na função e quando terminar coloque outra chave apontando para direita fechando a função.

```
instala(){

# Aqui dentro os codigos a serem executados na função

}
```

Depois de declarar a função ao shell vamos chamá-la, para fazer isto basta escrever o nome da função na linha desejada, no exemplo abaixo o usuário já instalou o Protec e se desejar instalá-lo para outro player deve digitar a letra “i”, assim cai no if que tem uma linha chamando a função de nome “instala” o que faz ele “pular” de onde está indo para outra parte do script.

```
echo "Digite 'i' para instalar em outro player ou 's' para sair"
read prox

if [ "$prox" = "i" ];then
   instala
fi
```

Chama a função instala

A função deve ser mostrada ao shell antes de ser chamada, vamos supor, eu tenho uma função no meio do script e resolvo chamá-la no começo, não vai funcionar porque o shell não leu ainda, eu sempre coloco as funções no começo do script assim dá menos trabalho

Parâmetros

Os parâmetros são valores armazenados em variáveis pré-determinadas, estes valores são fornecidos quando o usuário digita o comando que chama o programa/script pelo terminal, depois de digitar o “chamador” damos espaço e digitamos o primeiro parâmetro e assim sucessivamente conforme mostrado abaixo.

```
luiz@linuxmint ~ $ ./script parametro1 parametro2
```

Ou na prática → abaixo eu tenho o script “back” que em seu código dá um cp (copiar) no primeiro parâmetro e cola no segundo.

```
luiz@linuxmint ~ $ ./back /home/luiz /Dropbox
```

Sem saber usamos os parâmetros no terminal, o comando apt-get install firefox, apt-get é o comando do programa, install e firefox são os parâmetros, internamente o apt compara o valor do primeiro parâmetro para saber se instala, remove etc. E depois ele pega o segundo parâmetro que será o objeto a ser tratado.

É claro que o apt é muito mais do que isso e temos comandos gigantescos com vários parâmetros que eu não faço ideia da explicação, porém se você pegou a noção de parâmetros já é o suficiente.

Se o usuário não digitar os parâmetros o script vai rodar, o que pode acontecer é o mesmo dar pau por falta de dados

Endereço com vários sites tutoriais de Shell Script

http://aurelio.net/shell/

Sintaxe

Nós temos os parâmetros $1 $2 $3 $4 $5 $6 $7 $8 $9, então colocamos ele no script não para receber valor, mas já usando os valores que serão digitados no terminal.

```
#!/bin/bash

echo $1 $2 $3 $4 $5 $6 $7 $8 $9
```

O script anterior demonstra o funcionamento dos parâmetros, se o usuário digitar o comando do script e ao lado colocar → Linux, então será executado o comando echo mostrando a palavra Linux, se ele digitar “Linux e livre” serão 3 parâmetros e o shell executará o echo até o terceiro parâmetros, e assim sucessivamente.

Para mais de 9 parâmetros usaremos a dica do Laércio de Sousa do G+

Para declarar mais de 10 parâmetros no shell, basta colocar sua expansão entre chaves:

${10}
${11}
${12}

E assim sucessivamente

As aplicações são infinitas, eu poderia usar o valor de um parâmetro para determinar a quantidade de loops, para colher um nome de usuário, para receber valores a serem calculados etc. O script abaixo por exemplo, dorme com o tempo determinado pelo usuário e depois mata o programa também com a indicação do usuário.

```
#!/bin/bash

sleep $1

killall $2
```

O usuário daria um comando parecido com este → ./script 100 firefox

Tratamos os parâmetros como as variáveis, só tendo em mente que o seu valor será dado pelo usuário antes de iniciar o script

Exercício 7 – Função e parâmetro no mesmo script

- O usuário entra no programa utilizando o seu nome como parâmetro
- Aparecerá a tela de boas vindas constando o nome deste usuário
- Ele faz um cadastro (poucos dados só para simular, cpf é obrigatório)
- Ele é jogado para a segunda parte (de compra), nesta parte é pedido o cpf novamente e se o mesmo estiver errado (de acordo com o que foi cadastrado), o script volta para cadastrar o cpf novamente

Bons estudos e procure entender a lógica de programação mais do que decorar comandos

Resolução

Capítulo 12 – Comandos sed, cut, pipeline, tr e grep

Os comandos apresentados aqui deixam o script mais “inteligente” ajudando-o a “ler”, escrever, filtrar, colocar mais de um comando na mesma linha etc.

O texto abaixo será usado de exemplo nas próximas ferramentas mostradas aqui

L1 - Contribuir com o software livre é contribuir consigo mesmo,
L2 - porque hoje você usa a tecnologia criada por alguém no passado
L3 - e amanhã outros poderão desfrutar da sua contribuição iniciada hoje.

Sed

O comando sed tem várias funcionalidades relacionadas a filtragem, como exibir linhas com determinada palavra, extrair um trecho do texto, trocar linhas de lugar etc. Como o material do mesmo é vasto vamos aprender apenas a filtrar textos especificando por linha, o que nos ajudará nas filtragens.

Para mostrarmos somente determinada linha damos o comando:

```
sed -n '2p'
```

Linha 2

Mas isto não é o suficiente, devemos executar algum comando para que ele filtre sua saída, seja o ls, cat, ps etc. Abaixo eu usei o comando cat (exibe o conteúdo de arquivos-texto) no nosso texto exemplo e filtrei com o sed.

```
cat /home/luiz/texto | sed -n '2p'
```

Agora sim, a saída do comando cat são 3 linhas, o sed pega este resultado faz a filtragem e mostra apenas a linha 2. Veremos a barra antes do sed nesta aula, por enquanto basta saber que faz o sed processar o resultado do primeiro comando.

Com o comando acima a saída no texto de exemplo seria esta:

L2 - porque hoje você usa a tecnologia criada por alguém no passado

Para “eliminarmos” determinada linha:

Usamos a sintaxe abaixo indicando a linha a ser ocultada

```
cat /home/luiz/texto | sed 1d
```

Saindo assim o nosso texto exemplo:

L2 - porque hoje você usa a tecnologia criada por alguém no passado
L3 - e amanhã outros poderão desfrutar da sua contribuição iniciada hoje.

Ou posso eliminar mais de uma linha:

```
cat /home/luiz/texto | sed -e 1d -e 3d
```

Como são 3 linhas poderíamos usar o comando de incluir e chegaríamos no mesmo resultado, mas é só exemplo.

Saindo assim:

L2 - porque hoje você usa a tecnologia criada por alguém no passado

Como somos usuários iniciantes no shell apresentei apenas o básico e na sua caminhada você poderá conhecer os vários recursos que este comando tem.

Caso queira se aprofundar no sed acesse o link:
http://thobias.org/doc/sosed.html

Cut

Com o cut limitamos a saída dos comandos em campos, este texto que escrevo podemos considerar que seus campos são determinados pelo carácter "espaço", então se eu pedir ao cut que me mostre o conteúdo do campo 3 ele me retornará a palavra "cut", porque as palavras são os campos e os espaços mostram ao shell onde é a delimitação dos mesmos, nada te impede de usar a letra "a" por exemplo como delimitador, sairá um resultado muito louco, com isto digo que é você quem determinar o que será o delimitador e consequentemente os campos.

No exemplo anterior considerei o texto não tendo nenhum enter, ou seja se temos um arquivo com várias linhas que são exibidas separadamente, o cut mostrará o campo X de cada linha.

O texto abaixo por exemplo, podemos usar o delimitador "dois pontos"

mouse:azul:novo carro:branco:usado radio:preto:novo

Então usando o comando cut para mostrar o campo 2, seria exibido isto:

azul branco preto

Se quisermos mostrar somente a palavra "preto" é só usar o cut e sed		
cat texto |	"cut" pedindo o 2º campo	| sed -n '3p' mostrou a linha 3
mouse:azul:novo carro:branco:usado radio:preto:novo	azul branco preto	preto

O importante é sempre procurar padrões que lhes sirvam de base para chegar no conteúdo que alimente corretamente o script. No comando ps podemos filtrá-lo de uma forma que cheguemos somente no pid, o problema é que o pid passando de certa numeração ganha mais um carácter numérico diminuindo um carácter espaço, ou seja o seu comando que funcionava perfeitamente, simplesmente não vai servir mais (tem maneiras de contornar isto, mas não abordarei aqui).

Então preste atenção em todas as variações possíveis para evitar erros no script que nem sempre são fáceis de enxergar.

Sintaxe

Comando	**-f mais número do campo a ser mostrado**	**Aqui mostro o carácter que determina o campo "espaço".**
`cut`	`-f1`	`-d' '`

Da mesma forma que o sed, o cut também precisa de usar o resultado de outro comando:

```
cat /home/luiz/texto | cut -f5 -d' '
```

Resultado do comando acima dado no texto exemplo (somente na primeira linha e sem o L1):

```
luiz@linuxmint ~ $ cat /home/luiz/texto | cut -f5 -d' '
livre
```

É isto que tínhamos sobre o cut e nunca esqueça que um dia você precisará de uma informação tão especifica para o seu script e o cut + sed serão a solução.

Pipeline

Muitas vezes precisamos pegar a saída do comando A para usarmos no comando B e chegar no resultado final com o comando C, D etc. Além disto podemos diminuir o tamanho do script juntando vários comandos na mesma linha, é exatamente isto que o pipeline faz, ele é representado pelo símbolo "|" que está ao lado da letra "z" bastando digitá-lo com o shift.

Exemplo simples:

Pipeline

```
echo 1 > /prog/.LOOP1.txt | echo 1 > /prog/.LOOP2.txt | echo 1 > /prog/.LOOP3.txt | echo 1 > /prog/.LOOP4.txt
```

No exemplo acima eu precisava zerar os arquivos-texto responsáveis pelo loop e não fazia muito sentido usar várias linhas para isto, então coloquei todos concatenados na mesma linha usando o pipeline. (mais tarde eu descobri que não precisava usar arquivo-texto para o loop rsrs)

Exemplo mais avançado:

```
ting=$(ping -w 2 192.168.0.160 | cut -f1 -d' ' | sed -n '4p' )
```

Acima dou 3 pings, limito a saída para o campo 1 (determinado por espaço) e o resultado disto eu ainda pego e mostro apenas a linha 4 usando o sed.

Para usar o pipeline basta ter criatividade, seja para chegar num resultado que seria mais difícil de encontrar sem o pipe ou apenas para compactar o script, o importante é sempre testar, porque nem tudo que parece funcionar vai funcionar.

Tr

Com este comando é possível substituir caracteres de um texto por outros, é claro que podemos incluir como texto um script ou um arquivo de configuração por exemplo.

Sintaxe

Comando	Palavra ou carácter a ser substituído	O que vai entrar no lugar
tr	['a,e']	['4,3']

É importante não deixar espaços dentro dos colchetes para funcionar

Executando o comando abaixo em nosso texto exemplo, ele substituirá a letra “a” pelo número “4” e a letra “e” pelo número 3.

```
cat /home/luiz/texto | tr ['a,e'] ['4,3']
```

Ficando assim:

Contribuir com o softw4r3 livr3 é contribuir consigo m3smo,
porqu3 hoj3 você us4 4 t3cnologi4 cri4d4 por 4lguém no p4ss4do
3 4m4nhã outros pod3rão d3sfrut4r d4 su4 contribuição inici4d4 hoj3.

Lembrando que este foi um resultado mostrado na tela e se precisarmos de uma ação definitiva necessitaremos direcionar a saída para um arquivo-texto (pode ser o mesmo)

Uma função muito útil no comando tr é a de transformar caracteres minúsculos em maiúsculos e vice-versa, assim se pedirmos ao usuário que digite algo e que este valor seja obrigatório em maiúsculo ou minúsculo podemos transformar este valor com o tr garantindo assim o funcionamento do script.

Transforma tudo em maiúscula	**Tudo minúscula**
cat texto | tr a-z A-Z	cat texto | tr A-Z a-z
Elimina o que tem entre aspas do texto (aqui elimina espaços)	
cat texto | tr -d " "	

O código abaixo é um exemplo de como podemos pedir um valor ao usuário e transformá-lo do jeito que quisermos, aqui passo tudo que esta maiúsculo para minúsculo. Se o usuário digitar “CASA” então o valor da senha passará a ser “casa”, se ele digitar “CAsa” então teremos “casa” e assim por diante.

```
read senha

senha=$(echo $senha | tr A-Z a-z)
```

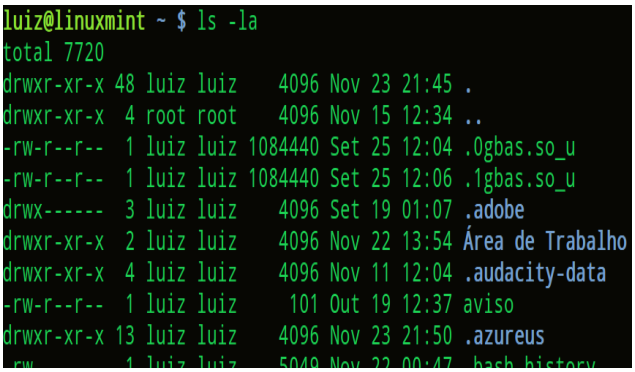
Grep

Com o comando grep também filtramos saídas, mas aqui é por palavra e não por número de linha, este comando mostra todas as linhas onde haja a incidência da palavra especificada, ele é muito usado para filtrar saídas extensas no terminal.

Sintaxe

Comando a ser filtrado e pipeline para concatenar	**Comando grep**	**Palavra a ser filtrada**
`ls |`	`grep`	`vlc`

No comando acima eu listo os diretórios e peço o grep para mostrar todos que tenham a palavra vlc.

Vamos supor que vou dar o comando “ls -la” no “/home” o resultado será bem grande, mas eu desejo localizar os diretórios contendo o nome “gnome”, então …	
Sem grep	**Com grep**
 E muito mais resultado para baixo	 Mostrando apenas as linhas onde constam a palavra gnome

Se houver qualquer dificuldade com este comando basta treiná-lo no terminal filtrando as saídas dos comandos que você está acostumado usar.

Exercício 8 – Usando os comandos

Usando a saída do comando “ls -la” no /home

Faça um comando que exiba:

Somente o mês de novembro, apenas uma vez e mais nada.

A resposta será somente → Nov

Pode ser o mês que desejar, este não vai ter resposta porque de um jeito ou de outro você conseguirá chegar no resultado esperado.

Capítulo 13 – Últimos comandos

Contando as linhas com o WC

Além de linhas o wc conta bytes, caracteres, palavras etc. Vide manual (man wc), ele pode ser muito útil já que em determinadas situações precisaremos saber o número de linhas totais para rodar um loop por exemplo.

Sintaxe

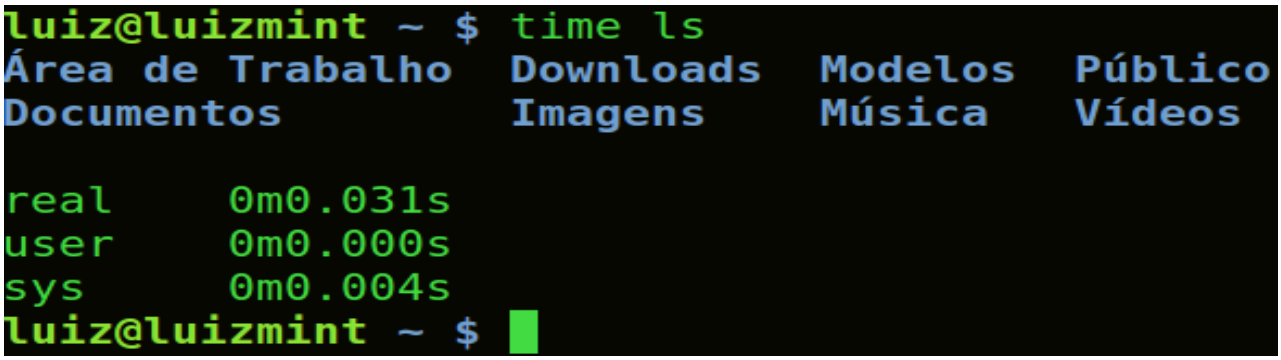
Comando que colhe os dados + pipeline	Comando wc contando as linhas
cat texto |	wc -l

Aqui o cat mostraria todo o texto como temos o wc ele pega a saída conta quantas linhas e mostra somente a contagem na tela. Muito simples já que você conhece o funcionamento de comandos que usam a saída de outros comandos.

> Caso precise usar o texto além de sua contagem, o recomendado é direcionar a saída do cat para um arquivo-texto “>” e em outra linha usar o comando wc neste arquivo.

Time

Podemos cronometrar o tempo de execução dos comandos executados no shell, para isto basta colocar o “time” antes do comando assim:



No momento eu não faço ideia de uma aplicação prática deste comando que não seja a curiosidade, mas nunca se sabe quando criaremos um programa que necessite colher os dados do tempo de execução de determinado comando.

Sort

Com o sort podemos ordenar as saídas dos comandos alfabética e numericamente

Sem nenhuma opção ele classificará as saídas alfabeticamente:

```
luizmint luiz # ls | sort
Área de Trabalho
Documentos
Downloads
Imagens
Modelos
Música
Público
Vídeos
```

Usando a opção -n temos a classificação numérica:

```
luizmint luiz # ls -la | cut -f3 -d ' ' | sort -n

luiz
luiz
1
1
1
1
1
1
1
1
1
1
2
2
2
2
2
2
2
2
2
2
2
2
3
```

Eliminando a saída dos comandos

Muitas vezes precisamos que o script execute determinado comando e que não seja mostrado nada na tela para o usuário, para isto direcionamos a saída para um “buraco negro” no Linux.

Direcionamos para o “> /dev/null”

Exemplo:

```
wondershaper eth0 260 40 > /dev/null
mpg123 /prog/nextel.mp3 > /dev/null
```

Acima dois comandos de um script onde era melhor não confundir o usuário com a saída dos comandos.

Figlet

Com o figlet podemos exibir as famosas escritas feitas no terminal, Dando assim uma incrementada no seu visual:

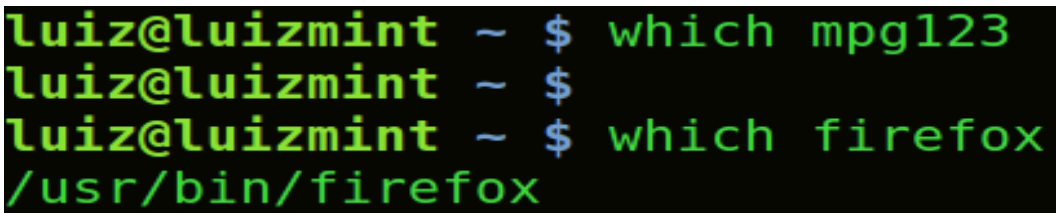
Sintaxe:

figlet palavra

É provável que o figlet não venha instalado por padrão, por isto dependendo do contexto você pode usar o próximo comando.

Which

Com este comando verificamos se determinado programa esta instalado, como mostrado abaixo o programa mpg123 não estava instalado por isto o valor retornado foi vazio enquanto que o firefox já estava e ele mostrou onde o executável se encontrava.



Sintaxe

which programa

Abaixo eu criei uma variável que recebe o valor do comando which, se ela estiver vazia eu pergunto ao usuário se ele deseja instalar o mpg123.

```
WIT=$(which mpg123)
```

Id -u (verificando se é root)

Algumas vezes o script roda da maneira esperada apenas como root ou como usuário normal, para fazer esta identificação usamos o comando “id -u”, quando executado ele pode retornar o valor “0” (zero) o que significa que estamos logados com usuário detentor de privilégios root caso contrário indica que o mesmo não tem privilégios.

A sintaxe é simplesmente “id -u”, abaixo temos as primeiras linhas do protec onde o mesmo não pode ser executado como root senão temos uma série de erros, então a primeira coisa que o programa faz é a verificação e caso o usuário seja root é mostrado uma mensagem e exit.

```
#Verifica se não é usuário root
ROT=$(id -u)
if [ "$ROT" = "0" ];then
   echo "O programa não funciona em modo root, por favor entre novamente"
   sleep 1
   exit
fi
```

Exercício 9 – Configurando o Samba

Vamos criar um script que automatiza a configuração do samba (programa que compartilha arquivos entre linux-windows linux-linux), facilitando assim a vida do usuário.

Linhas do Samba

Com as linhas abaixo gravadas no /etc/samba/smb.conf já são suficientes para compartilhar arquivos na rede.

```
[global]
workgroup = admin # Nome da rede
name resolve order = lmhosts wins bcast host # Evita erros de mapeamento

[arquivos] # Nome que será exibido para o diretório em questão
path = /home/luiz # Pasta compartilhada
read only = no # Dá permissão de escrita (não é default)
public = yes # Permissão para convidados (é necessário que o usuário esteja adicionado no samba)
```

Acima os itens em azul serão dados pelos usuários (através de variáveis) e o restante serão fixos

Características do programa

Atenção, quando for verificar se o samba está instalado você deverá se referir a ele como smbd

- Instala o samba se necessário
- Explica que o programa cria um compartilhamento publico.
- Menu → 1- Nova configuração 2- Adicionar novo diretório 3- Adicionar novo usuário
- Pergunta ao usuário → nome da rede, diretório a ser compartilhado e o nome que aparecerá
- Renomeia o smb.conf atual criando um backup
- Escreve o novo smb.conf usando as informações dadas pelo usuário
- Pergunta e adiciona usuários de outras máquinas **(#adduser usuário e #smbpasswd -a usuário)**
- E pergunta e adiciona o usuário local ao samba **(#smbpasswd -a usuário)**
- Reinicia o samba (**sudo service nmbd restart** e **sudo service smbd restart**)
- Avisa que em breve o compartilhamento já estará ativo e sai

Visualize a imagem abaixo para não errar na hora de imprimir os dados no smb.conf:

```
echo "texto de configuração do samba." > smb.conf
echo "As variaveis não serão impressas, mas sim o seu conteúdo" >> smb.conf
echo "Bastando deixar tudo dentro de aspas, como no exemplo abaixo" >> smb.conf
echo "texto texto $VARIAVEL texto $VARIAVEL texto etc" >> smb.conf
```

Resolução

Capítulo 14 – Indentação

Indentação é um recuo no texto, na programação usamos para deixar os códigos mais legíveis facilitando assim a compreensão das vastas linhas de um script. No shell ela não é obrigatória, mas se vamos escrever centenas de linhas é recomendável usá-la.

Como não domino o tema vou dar a minha contribuição da forma que uso, então sinta-se à vontade em aprofundar no tema.

O conceito é simples se temos um código dentro de outro damos 3 espaços e assim por diante.

```
#!/bin/bash

while [ "$WILE1" = "n" ];do
   if [ "$prox" = "i" ];then
      instala
      else
         exit
   fi
done
```

Acima temos o if dentro do while, por isto ele está 3 espaços a frente e dentro do if temos uma chamada de função “instala” e também está 3 espaços a frente, a situação do “else” é a mesma do “instala” consequentemente estão na mesma coluna e como o exit está dentro do “else” consequentemente ele recebe os espaços.

Repare os comandos que fecham o while e if eles estão rentes aos comandos que os abrem (olhe os pontilhados), assim mesmo se tivermos um número grande de ifs, whiles etc. no script não dará tanta confusão já que fica mais fácil de visualizar onde começa e onde termina as estruturas e quais comandos estão dentro de quais comandos.

Olhe a diferença dos códigos sem organização:

```
#!/bin/bash

while [ "$WILE1" = "n" ];do
if [ "$prox" = "i" ];then
instala
else
exit
fi
done
```

Outra coisa que recomendo é usar comentários para dividir seções no script:

```
#####################################################################
###################### CHAMA AJUDA ##################################
#####################################################################
Códigos
#####################################################################
###################### DESINSTALA ###################################
#####################################################################
Códigos
```

De qualquer forma você entendendo seus códigos é o que importa.

Capítulo 15 – Script com janelas

Script com janelas

As janelas são uma espécie de interface gráfica para os nossos scripts que interagem com o clicar do mouse, usando elas melhoramos em muito sua aparência e facilitamos a interação com o usuário. Nós poderemos recolher senhas, respostas, dar simples avisos etc. Vai depender do **tipo** que usaremos com o comando.

Apenas o --tipo será modificado na sintaxe abaixo, o resto é padrão

dialog --tipo 'texto' altura largura

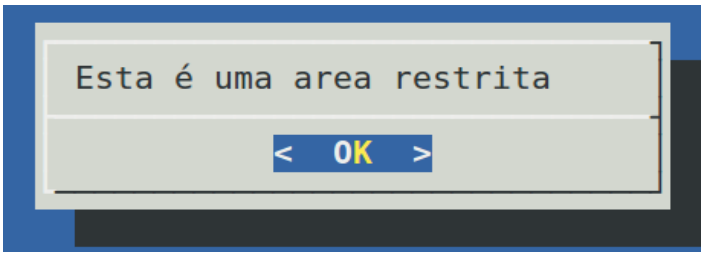
A seguir vamos conhecer alguns tipos de janelas

msgbox (caixa de mensagem)

Este tipo possibilita mostrar uma mensagem na tela bastando dar ok depois de lida para prosseguir no script.

Sintaxe/Exemplo
dialog --msgbox 'Esta é uma area restrita' 7 40

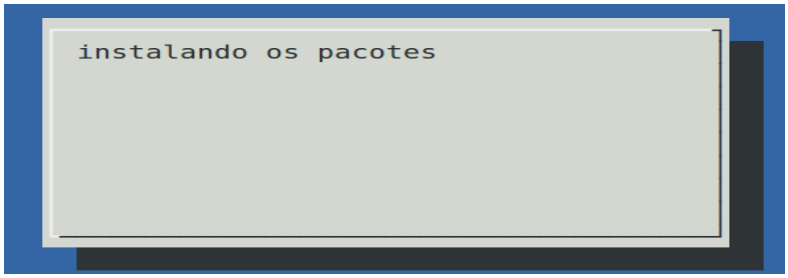
Saindo assim no terminal:



Infobox (caixa de informação)

Aqui mostramos uma mensagem e logo em seguida o script continua executando sem pedir confirmação.

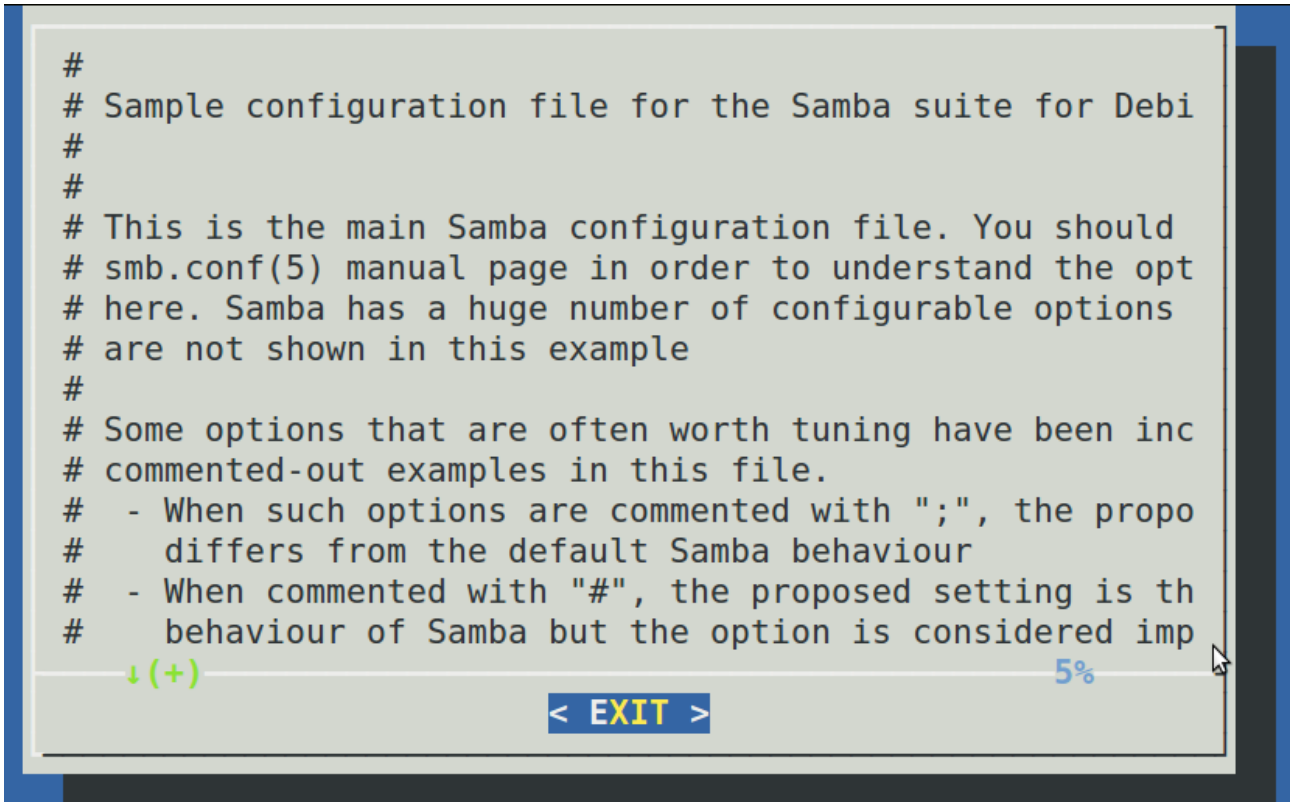
Sintaxe/Exemplo
dialog --infobox 'instalando os pacotes' 10 40



Textbox (caixa de texto)

Com este tipo podemos exibir um arquivo-texto no terminal como instruções, configurações etc. abaixo peço para o shell mostrar o arquivo de configuração do samba:

Sintaxe/Exemplo
dialog --textbox /etc/samba/smb.conf 40 60

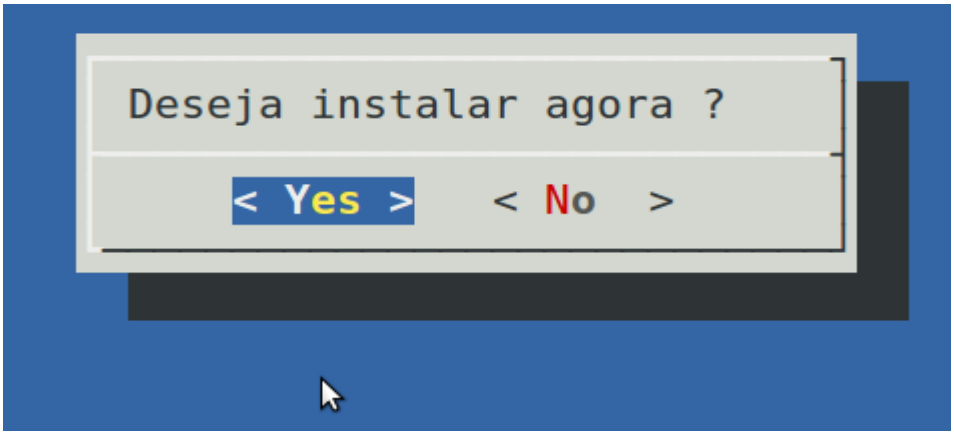


Onde o usuário clica em + para subir o texto e temos a porcentagem do que já foi exibido.

Yesno (Sim ou não)

Agora podemos fazer uma pergunta ao usuário e esperar que o mesmo responda yes ou no

Sintaxe/Exemplo
dialog --yesno 'Deseja instalar agora ?' 5 30



Para usar a resposta no script devemos utilizar a variável “$?” (leia abaixo)

Exatamente deste jeito, bastando mudar os comandos internos:

```
if [ $? = 0 ]; then
   apt-get install firefox
   else
      echo "Você pode instalar o navegador mais tarde"
      exit
fi
```

Zero é corresponde a sim

Foram apresentadas as janelas mais simples se você deseja usar outros tipos, abaixo temos uma apresentação breve para que você possa pesquisar sua utilização:

gauge	Barra de porcentagem
checklist	Mostra um menu para escolhas multiplas
menu	Menu para a escolha de apenas 1 item
calendar	Mostra um calendário para o usuário escolher uma data
fselect	Digita ou seleciona um arquivo
passwordbox	Pede uma senha

Exercício Resolvidos

Exercício 1 – Primeiro script

```
#!/bin/bash

# Este começo tem apenas a parte escrita na tela
echo "Bem vindo ao sistema de marcação online"
echo
echo

# Se dou opção numérica é melhor, assim evitamos do usuário digitar errado ou com acentos
echo "Por favor digite a opção do turno desejado"
echo
echo "1 - Manhã"
echo "2 - Tarde"
echo "3 - Noite"
echo # Espaço para não ficar embolado com a resposta

# Colhendo os dados do turno
read TURNO

# Estou colocando como alfanumérico porque não vou fazer cálculos e não quero ter problemas
if [ "$TURNO" = "1" ];then
   echo "Apenas o Doutor Wagner atende no turno da manhã"
fi

if [ "$TURNO" = "2" ];then
   echo "Apenas a doutora Camila atende no turno da tarde"
fi

if [ "$TURNO" = "3" ];then
   echo "Apenas o Doutor Cesar atende no turno noite"
fi

# Fazendo o programa dormir por 2 segundos
sleep 2
echo
echo "Sua consulta foi marcada com sucesso!"

# Dormindo para dar tempo de ler a mensagem e depois sai
sleep 4

# Saindo
exit
```

Exercício 2 – Script mais elaborado

```
#!/bin/bash

echo "Olá, abaixo os carros disponíveis neste mês"
echo
echo "Ferrari, Palio, Celta, Lamborghini e Uno"
echo

# Recebendo o valor do carro desejado
echo "Qual carro deseja saber a classe? (digite em minúsculo)"
read CARRO
echo

# Se carro é igual a lamborghini ou a ferrari, então só pode ser classe A
# Coloquei a variável no texto, assim o que ele digitar vai aparecer no texto
if [ "$CARRO" = "lamborghini" -o "$CARRO" = "ferrari" ];then
   echo "$CARRO pertence a classe A"
fi

if [ "$CARRO" = "celta" ];then
   echo "$CARRO pertence a classe B"
fi

if [ "$CARRO" = "palio" -o "$CARRO" = "uno" ];then
   echo "$CARRO pertence a classe C"
fi

exit
```

Exercício 3 - Lógica dos PC1 e PC2

```
#!/bin/bash
echo "Por favor, digite ligado ou desligado para informar o estado dos computadores abaixo"
echo
echo "PC1"
read PC1
echo

echo "PC2"
read PC2
echo

# INCREMENTO, Se o usuário digitou algo que não seja ligado e desligado, então cai neste if
if [ "$PC1" != "ligado" -a "$PC1" != "desligado" ];then
  echo "Estado invalido para PC1, tente novamente"
  sleep 3
  exit
fi

# if de erro para o PC2
if [ "$PC2" != "ligado" -a "$PC2" != "desligado" ];then
  echo "Estado invalido para PC2, tente novamente"
  sleep 3
  exit
fi

# Posso usar tanto "PC2 = desligado" quanto "PC2 != ligado" a segunda opção vai no resto do script
# Se pc 1 estiver ligado e o 2 estiver desligado faça o comando
if [ "$PC1" = "ligado" -a "$PC2" = "desligado" ];then
  echo "Valor da amostra é igual a 5"
fi

# Acima, se eu não incluir o PC2 desligado vai virar uma bagunça e cair naquele if mesmo quando
# PC2 estiver ligado, ou seja, mostrará mais de um resultado

# Não corre o risco do "!= ligado" ser outra palavra que não seja desligado, já que o primeiro
# Incremento se encarregou de eliminar outras escritas
if [ "$PC2" = "ligado" -a "$PC1" != "ligado" ];then # Se PC2 estiver ligado ...
  echo "Valor da amostra é igual a 10"
fi

if [ "$PC1" = "ligado" -a "$PC2" = "ligado" ];then # Se PC1 e 2 estiverem ligados cai aqui
  echo "Valor da amostra é igual a 15"
fi

# INCREMENTO, mostrando o resultado quando dois pcs estiverem DESLIGADOS
if [ "$PC1" != "ligado" -a "$PC2" != "ligado" ];then
  echo "Valor da amostra é igual a zero"
fi

exit
```

Exercício 4 - Remoto

```
#!/bin/bash

# Criando ou zerando os arquivos-texto
echo "" > $HOME/comando
echo "" > $HOME/aviso

# Loop infinito
for ((loop=2;loop>1;loop++));do

  # Colocando a velocidade do loop em 1 minuto
  sleep 60
  echo $loop # Mostrando a rodada na tela

  # Lendo o arquivo-texto e passando o valor para a variável
  COMANDO=$(cat $HOME/comando)

  # INCREMENTO, responde que esta rodando, caso perguntado
  if [ "$COMANDO" = "vivo?" ];then
    echo "sim vivo, esperando o comando, loop $loop" >> $HOME/aviso
    COMANDO=$"" # zerando a variável para não cair no próximo if
    echo > $HOME/comando # Limpando o arquivo-texto
  fi

  # Se a variável COMANDO não estiver vazia, então vou executar o comando dentro dela
  if [ -n "$COMANDO" ];then
    $COMANDO& # Coloquei o & para o script não ficar agarrado aqui
    echo > $HOME/comando # Limpando o arquivo com o comando
    # Escrevendo o comando executado no arquivo avisos
    echo "Executei" $COMANDO "loop $loop" >> $HOME/aviso
  fi

done

# Ele nunca vai sair do loop infinito então não justifica colocar o exit
```

Exercício 5 - Senha

```
#!/bin/bash

echo
echo "                    Seja Bem-vindo"
echo
echo "Por favor digite a senha para logar no sistema"
echo
read SENHA

# zerando a variavel de loop
x=$"0"

# Enquanto x for menor que 4, faça (contei com a primeira tentativa acima)
while [ $x -lt 4 ];do

  # Se a senha for igual a 123 então faça o comando
  if [ "$SENHA" = "123" ];then
    echo
    echo "Senha correta, logado no sistema"
    sleep 2
    exit
  fi

  # Se ele não cair no if, pede a senha novamente
  echo
  echo "Senha incorreta tente novamente"
  read SENHA

  # colocando +1 para contarmos as rodadas
  x=$[$x+1]

done

# Fora do loop eu coloco estes comandos, então assim que as tentativas se esgotarem
# Ele sai do loop e executa aqui
echo
echo "Número de tentativas esgotado, tente mais tarde"
sleep 2

exit
```

Exercício 6 – Pequeno desafio de lógica

```
#!/bin/bash

# Aqui eu pego o valor da variável para ser multiplicado
echo "Por favor digite o número a ser processado"
echo
read NUM

# Determinando um valor para a variável do while
VOLTA=$"1"

# Criando o loop, assim vamos rodar o case 3 vezes

while [ $VOLTA -lt 4 ];do # Enquanto $VOLTA for menor que 4 faça o comando (roda 3 vezes)

# Estou usando a variável de loop no case, assim a cada rodada uma opção diferente será executada
  case $VOLTA in

# Pego o valor dado pelo usuário e multiplico pelo valor pré-determinado no exercício
    1) CONTA=$(( $NUM * 10 ))
      echo "Rodada 1 - $NUM multiplicado por 10 é igual a $CONTA";;

    2) CONTA=$(( $NUM * 20 ))
      echo "Rodada 2 - $NUM multiplicado por 20 é igual a $CONTA";;
# A cada rodada a variável CONTA receberá um valor diferente, então não precisamos criar
#uma variável diferente para cada opção do menu

    3) CONTA=$(( $NUM * 30 ))
      echo "Rodada 3 - $NUM multiplicado por 30 é igual a $CONTA";;

  esac

# Somando os números de voltas do while, fora do case logicamente
  VOLTA=$[ $VOLTA + 1 ]

done
```

Exercício 7 - Função e parâmetro no mesmo script

```
#!/bin/bash
# eu não fiz os códigos nesta ordem a toa, algumas caiam numa "sinuca de bico"
echo
echo "Bem vindo $1"

# Colhendo os dados do cadastro
echo
echo "Digite o nome da sua cidade"
read CIDADE
echo
echo "Digite seu CEP"
read CEP
echo

# Função que cadastra o CPF
CADASTRO(){
  echo "Digite seu CPF"
  read CPF
  echo
  echo "Cadastro finalizado com sucesso"
  LOGANDO # Agora chamamos a função de logar
}

LOGANDO(){

  echo
  echo "Digite seu CPF para logarmos"
  read CPF2
  echo

  # Aqui comparo os CPFs como texto
  if [ "$CPF" = "$CPF2" ];then # Se CPF digitado anteriormente é igual ao agora, então logue
    echo "logado com sucesso"
    sleep 5
    exit # Depois de dormir 5 segundos ele sai porque já alcançamos os nossos objetivos
    else # Senão for igual durma e chame CADASTRO, para recebermos o cpf novamente
      echo "CPF incorreto, porfavor tente novamente"
      sleep 3 # Dando uma dormida para dar tempo do usuário ler a mensagem de erro
      CADASTRO
  fi
}

# O shell vai ler as duas funções, depois passa por aqui chamando a função CADASTRO
CADASTRO

# A única forma do usuário sair do script é digitando o CPF corretamente, se fosse um script
# para empresa é claro que colocaríamos outras opções
```

Exercício 9 – Configurando o Samba

```
#!/bin/bash

WIT=$(which smbd)
# Se o samba não estiver instalado cai aqui
if [ -z "$WIT" ];then
  echo "O samba não está instalado, deseja instalá-lo agora? s/n"
  read INSTALA
    if [ "$INSTALA" = "s" ];then
      apt-get install samba
      else
        exit
    fi
fi

echo
echo "Seja Bem-vindo, a seguir vamos configurar um compartilhamento publico"
echo "Escolha a opção desejada"
echo
echo "1- Para nova configuração"
echo "2- Para adicionar um diretório ao compartilhamento já configurado"
echo "3- Adicionar um novo usuário"

read MENU
case $MENU in

1)
  echo
  echo
  echo "Digite o nome da sua rede de compartilhamento"
  read REDE
  echo
  echo "Digite o endereço do diretório a ser compartilhado"
  read DIRETORIO
  echo
  echo "Qual o nome será exibido na rede para este diretório?"
  read NOME_DIR

  # Renomeando o arquivo de configuração para escrevermos um novo
  mv /etc/samba/smb.conf /etc/samba/BKsmb.conf

  # Escrevendo o novo smb.conf
  echo "[global]" > /etc/samba/smb.conf # Zerando ou criando o arquivo
  echo "workgroup = $REDE" >> /etc/samba/smb.conf
  echo "name resolve order = lmhosts wins bcast hos" >> /etc/samba/smb.conf

  echo "" >> /etc/samba/smb.conf # Escrevendo uma linha de espaço

  echo "[$NOME_DIR]" >> /etc/samba/smb.conf
  echo "path = $DIRETORIO" >> /etc/samba/smb.conf
  echo "read only = no" >> /etc/samba/smb.conf
```

```
  echo "public = yes" >> /etc/samba/smb.conf

  # Loop que adiciona usuários de outro PC
  echo
  echo "Algum usuário de outra máquina vai usar este compartilhamento? s/n"
  read MAIS
  while [ "$MAIS" = "s" ];do
    echo

    echo "Qual?"
    read USU_LA

    adduser $USU_LA
    smbpasswd -a $USU_LA

    echo
    echo "mais algum usuário para adicionar? s/n"
    read MAIS
  done

  echo
  echo "Agora informe o nome do seu usuário para que eu possa adicioná-lo"
  read USU
  smbpasswd -a $USU

  service nmbd restart > /dev/null
  service smbd restart > /dev/null

  echo
  echo "Configuração concluída, aguarde um momento"
  echo "De Linux para Linux demora um pouco a iniciar"

  exit;;

2)
  # Loop que adiciona mais diretórios
  MAIS1=$"s" # Garantindo o primeiro loop
  while [ "$MAIS1" = "s" ];do
    echo
    echo "Digite o endereço do novo diretório a ser compartilhado"
    read DIRETORIO
    echo
    echo "Qual o nome será exibido na rede para este diretório?"
    read NOME_DIR

    echo "" >> /etc/samba/smb.conf # Escrevendo uma linha de espaço

    echo "[$NOME_DIR]" >> /etc/samba/smb.conf
    echo "path = $DIRETORIO" >> /etc/samba/smb.conf
    echo "read only = no" >> /etc/samba/smb.conf
    echo "public = yes" >> /etc/samba/smb.conf
```

```
    echo
    echo "mais algum diretório? s/n"
    read MAIS1
  done

  service nmbd restart > /dev/null
  service smbd restart > /dev/null

  echo
  echo "Configuração concluída, aguarde um momento"
  exit;;

3)
  # Loop que adiciona usuários de outro PC
  MAIS2=$"s" # Garantindo o primeiro loop
  while [ "$MAIS2" = "s" ];do
    echo "Digite o nome de usuário"
    read USU_LA

    adduser $USU_LA
    smbpasswd -a $USU_LA

    echo "mais algum usuário para adicionar? s/n"
    read MAIS2
  done

  service nmbd restart > /dev/null
  service smbd restart > /dev/null

  echo
  echo "Configuração concluída, aguarde um momento"
  exit;;
esac
```