Programando em shell-script

1. Primeira parte, uma introdução

Quem usa Linux conhece bem o prompt de comando sh, ou variações como o bash. O ue muita gente não sabe é que o sh ou o bash têm uma "poderosa" linguagem de script embutido nelas mesmas. Diversas pessoas utilizam-se desta linguagem para facilitar a realização de inúmeras tarefas administrativas no Linux, ou até mesmo criar seus próprios programinhas. Patrick Volkerding, criador da distribuição Slackware, utiliza esta linguagem para toda a instalação e configuração de sua distribuição. Você poderá criar scripts para automar as tarefas diárias de um servidor, para efetuar backup automático regularmente, procurar textos, criar formatações, e muito mais. Para você ver como esta linguagem pode ser útil, vamos ver alguns passos introdutórios sobre ela.

Interpretadores de comandos são programas feitos para intermediar o usuário e seu sistema. Através destes interpretadores, o usuário manda um comando, e o interpretador o executa no sistema. Eles são a "Shell" do sistema Linux. Usaremos o interpretador de comandos bash, por ser mais "extenso" que o sh, e para que haja uma melhor compreensão das informações obtidas aqui, é bom ter uma base sobre o conceito de lógica de programação.

Uma das vantagens destes shell scripts é que eles não precisam ser compilados, ou seja, basta apenas criar um arquivo texto qualquer, e inserir comandos à ele. Para dar à este arquivo a definição de "shell script", teremos que incluir uma linha no começo do arquivo (#!/bin/bash) e torná-lo "executável", utilizando o comando chmod. Vamos seguir com um pequeno exemplo de um shell script que mostre na tela: "Nossa! Estou vivo!":

```
#!/bin/bash
echo 'Nossa! Estou vivo!'
```

Fácil, hein? A primeira linha indica que todas as outras linhas abaixo deverão ser executadas pelo bash (que se localiza em /bin/bash), e a segunda linha imprimirá na tela a frase "Nossa! Estou vivo!", utilizando o comando echo, que serve justamente para isto. Como você pôde ver, todos os comandos que você digita diretamente na linha de comando, você poderá incluir no seu shell script, criando uma série de comandos, e é essa combinação de comandos que forma o chamado shell script. Tente também dar o comando 'file arquivo' e veja que a definição dele é de Bourne-Again Shell Script (Bash Script).

Contudo, para o arquivo poder se executável, você tem de atribuir o comando de executável para ele. E como citamos anteriormente, o comando chmod se encarrega disto:

```
$ chmod +x arquivo
```

Pronto, o arquivo poderá ser executado com um simples "./arquivo".

Conceito de Variáveis em shell script

Variáveis são caracteres que armazenam dados, uma espécie de atalho. O bash reconhece uma variável quando ela começa com $, ou seja, a diferença entre 'palavra' e '$palavra' é que a primeira é uma palavra qualquer, e a outra uma variável. Para definir uma variável, utilizamos a seguinte sintaxe:

```
variavel="valor"
```

O 'valor' será atribuído a 'variável '. Valor pode ser uma frase, números, e até outras variáveis e comandos. O valor pode ser expressado entre as aspas (""), apóstrofos ('') ou crases (``). As aspas vão interpretar as variáveis que estiverem dentro do valor, os apóstrofos lerão o valor literalmente, sem interpretar nada, e as crases vão interpretar um comando e retornar a sua saída para a variável. Vejamos exemplos:

```
$ variavel="Eu estou logado como usuário $user"
$ echo $variavel
   Eu estou logado como usuário cla

$ variavel='Eu estou logado como usuário $user'
$ echo $variavel
   Eu estou logado como usuário $user

$ variavel="Meu diretório atual é o `pwd`"
$ echo $variavel
   Meu diretório atual é o /home/cla
```

Se você quiser criar um script em que o usuário deve interagir com ele, é possível que você queira que o próprio usuário defina uma variável, e para isso usamos o comando read, que dará uma pausa no script e ficarará esperando o usuário digitar algum valor e teclar enter. Exemplo:

```
echo "Entre com o valor para a variável: " ; read variavel
(O usuário digita e tecla enter, vamos supor que ele digitou 'Teste de
variável ')
echo $variavel
   Teste de variável
```

Controle de fluxo com o if

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". Vamos dar um exemplo:

```
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 que usei 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

Na tabela a seguir, você pode encontrar uma listagem de comandos para usar em sua shell script:

-x	Se o arquivo pode ser executado
echo	Imprime texto na tela
read	Captura dados do usuário e coloca numa variável
exit	Finaliza o script
sleep	Dá uma pause de segundos no script
clear	Limpa a tela
stty	Configura o terminal temporariamente
Tput	Altera o modo de exibição
IF	Controle de fluxo que testa uma ou mais expressões
Case	Controle de fluxo que testa várias expressões ao mesmo tempo
For	Controle de fluxo que testa uma ou mais expressões
while	Controle de fluxo que testa uma ou mais expressões

Controle de fluxo que testa uma ou mais expressões

E assim seja, crie seus próprios scripts e facilite de uma vez só parte de sua vida no Linux!

2. Segunda parte, se aprofundando mais!

Falamos sobre o conceito da programação em Shell Script, e demos o primeiro passo para construir nossos próprios scripts. Agora vamos nos aprofundar nos comandos mais complicados, aprendendo a fazer programas ainda mais úteis. Nestes comandos estão inclusos o case e os laços for, while e until. Além disso, vamos falar de funções e, por último, teremos um programa em shell script.

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. Vejamos um exemplo:

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

Aqui 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.

Você pode ver que, com o case fica muito mais fácil criar uma espécie de "menu" para o shell script do que com o if. Vamos demonstrar a mesma função anterior, mas agora usando o if:

```
if [ -z $1 ]; then
   echo "Você tem de entrar com um parâmetro válido"
   exit
elif [ $1 = "parametro1" ]; then
   comando1
   comando2
elif [ $1 = "parametro2" ]; then
   comando3
   comando4
else
   echo "Você tem de entrar com um parâmetro válido"
fi
```

Veja a diferença. É muito mais prático usar o case! A vantagem do if é que ele pode testar várias expressões que o case não pode. O case é mais prático, mas o if pode substituí-lo e ainda abrange mais funções. Note que, no exemplo com o if, citamos um "comando" não visto antes: o elif - que é uma combinação de else e if. Ao invés de fechar o if para criar outro, usamos o elif para testar uma expressão no mesmo comando if.

For

O laço for vai substituindo uma variável por um valor, e vai executando os comandos pedidos. Veja o exemplo:

```
for i in *
do
   cp $i $i.backup
   mv $i.backup /usr/backup
done
```

Primeiramente o laço for atribuiu o valor de retorno do comando "*" (que é equivalente a um ls sem nenhum parâmetro) para a variável $i, depois executou o bloco de comandos. Em seguida ele atribui outro valor do comando "*" para a variável $1 e reexecutou os comandos. Isso se repete até que não sobrem valores de retorno do comando "*". Outro exemplo:

```
for original in *; do
   resultado=`echo $original |
              tr '[:upper:]' '[:lower:]'`
   if [ ! -e $resultado ]; then
      mv $original $resultado
   fi
done
```

Aqui, o que ocorre é a transformação de letras maiúsculas para minúsculas. Para cada arquivo que o laço lê, uma variável chamada $resultado irá conter o arquivo em letras minúsculas. Para transformar em letras minúsculas, usei o comando tr. Caso não exista um arquivo igual e com letras minúsculas, o arquivo é renomeado para o valor da variável $resultado, de mesmo nome, mas com letras minúsculas.

Como os exemplos ilustram, o laço for pode ser bem útil no tratamento de múltiplos arquivos. Você pode deixá-los todos com letras minúsculas ou maiúsculas sem precisar renomear cada um manualmente, pode organizar dados, fazer backup, entre outras coisas.

While

O while testa continuamente uma expressão, até que ela se torne falsa. Exemplo:

```
variavel="valor"
```

```
while [ $variavel = "valor" ]; do
   comando1
   comando2
done
```

O que acontece aqui é o seguinte: enquanto a "$variavel" for igual a "valor", o while ficará executando os comandos 1 e 2, até que a "$variavel" não seja mais igual a "valor". Se no bloco dos comandos a "$variavel" mudasse, o while iria parar de executar os comandos quando chegasse em done, pois agora a expressão $variavel = "valor" não seria mais verdadeira.

Until

Tem as mesmas características do while, a única diferença é que ele faz o contrário. Veja o exemplo abaixo:

```
variavel="naovalor"
until [ $variavel = "valor" ]; do
   comando1
   comando2
done
```

Ao invés de executar o bloco de comandos (comando1 e comando2) até que a expressão se torne falsa, o until testa a expressão e executa o bloco de comandos até que a expressão se torne verdadeira. No exemplo, o bloco de comandos será executado desde que a expressão $variavel = "valor" não seja verdadeira. Se no bloco de comandos a variável for definida como "valor", o until pára de executar os comandos quando chega ao done.

Vejamos um exemplo para o until que, sintaticamente invertido, serve para o while também:

```
var=1
count=0
until [ $var = "0" ]; do
   comando1
   comando2
   if [ $count = 9 ]; then
      var=0
   fi
   count=`expr $count + 1`
done
```

Primeiro, atribuímos à variável "$var" o valor "1". A variável "$count" será uma contagem para quantas vezes quisermos executar o bloco de comandos. O until executa os comandos 1 e 2, enquanto a variável "$var" for igual a "0". Então usamos um if para atribuir o valor 0 para a variável "$var", se a variável "$count" for igual a 9. Se a variável "$count" não for igual a 0, soma-se 1 a ela. Isso cria um laço que executa o comando 10 vezes, porque cada vez que o comando do bloco de comandos é executado, soma-se 1 à variável "$count", e quando

chega em 9, a variável "$var" é igualada a zero, quebrando assim o laço until.

Usando vários scripts em um só

Pode-se precisar criar vários scripts shell que fazem funções diferentes, mas, e se você precisar executar em um script shell um outro script externo para que este faça alguma função e não precisar reescrever todo o código? É simples, você só precisa incluir o seguinte comando no seu script shell:

```
. bashscript2
```

Isso executará o script shell "bashscript2" durante a execução do seu script shell. Neste caso ele será executado na mesma script shell em que está sendo usado o comando. Para utilizar outra shell, você simplesmente substitui o "." pelo executável da shell, assim:

```
sh script2
tcsh script3
```

Nessas linhas o script2 será executado com a shell sh, e o script3 com a shell tcsh.

Variáveis especiais

$0	Nome do script que está sendo executado
$1-$9	Parâmetros passados à linha de comando
$#	Número de parâmetros passados
$?	Valor de retorno do último comando ou de todo o shell script. (o comando “exit 1” retorna o valor 1)
$$	Número do PID (Process ID)

Você também encontra muitas variáveis, já predefinidas, na página de manual do bash (comando "man bash", seção Shell Variables).

Funções

Funções são blocos de comandos que podem ser definidos para uso posterior em qualquer parte do código. Praticamente todas as linguagens usam funções que ajudam a organizar o código. Vejamos a sintaxe de uma função:

```
funcao() {
   comando1
   comando2
   ...
}
```

Fácil de entender, não? A função funcionará como um simples comando próprio. Você executa a função em qualquer lugar do script shell, e os comandos 1, 2 e outros serão executados. A flexibilidade das funções permite facilitar a vida do programador, como no exemplo final.

Exemplo Final

Agora vamos dar um exemplo de um programa que utilize o que aprendemos com os artigos.

```
#!/bin/bash
# Exemplo Final de Script Shell
Principal() {
   echo "Exemplo Final sobre o uso de scripts shell"
   echo "-----------------------------------------"
   echo "Opções:"
   echo
   echo "1. Trasformar nomes de arquivos"
   echo "2. Adicionar um usuário no sistema"
   echo "3. Deletar um usuário no sistema"
   echo "4. Fazer backup dos arquivos do /etc"
   echo "5. Sair do exemplo"
   echo
   echo -n "Qual a opção desejada? "
   read opcao
   case $opcao in
      1) Transformar ;;
      2) Adicionar ;;
      3) Deletar ;;
      4) Backup ;;
      5) exit ;;
      *) "Opção desconhecida." ; echo ; Principal ;;
   esac
}
Transformar() {
   echo -n "Para Maiúsculo ou minúsculo? [M/m] "
   read var
   if [ $var = "M" ]; then
      echo -n "Que diretório? "
      read dir
      for x in `/bin/ls` $dir; do
         y=`echo $x | tr '[:lower:]' '[:upper:]'`
         if [ ! -e $y ]; then
            mv $x $y
         fi
      done
   elif [ $var = "m" ]; then
      echo -n "Que diretório? "
      read dir
      for x in `/bin/ls` $dir; do
         y=`echo $x | tr '[:upper:]' '[:lower:]'`
         if [ ! -e $y ]; then
            mv $x $y
```

```
         fi
      done
   fi
}
Adicionar() {
   clear
   echo -n "Qual o nome do usuário a se adicionar? "
   read nome
   adduser nome
   Principal
}
Deletar() {
   clear
   echo -n "Qual o nome do usuário a deletar? "
   read nome
   userdel nome
   Principal
}
Backup() {
   for x in `/bin/ls` /etc; do
      cp -R /etc/$x /etc/$x.bck
      mv /etc/$x.bck /usr/backup
   done
}
Principal
```

3. Terceira parte, janelas graficas

Nos dois topicos anteriores, vimos algumas coisas básicas e lógicas de programação em shell no Linux. Agora para completar, darei aqui dicas de como usar janelas gráficas em seus shell-scripts. Isso mesmo, janelas que rodam no ambiente gráfico, utilizadas facilmente em seu shell-script. Com esse recurso, vai ser possível deixar o seu programa bem amigável.

Não vai ser preciso saber muita coisa de programação em shell, pois é muito simples. Através do programa *Xdialog*, você poderá criar os mais variados tipos de janelas para o programa: caixas de texto, seleçÕes, radios, etc. O Xdialog é uma idéia que vem do *dialog/cdialog*, um programa para console que gera "janelas" no console (aquelas da instalação do Slackware) usando a biblioteca *ncurses*. O Xdialog ao invés de usar a biblioteca ncurses, usa a *Xlib* para criar as janelas no ambiente gráfico.

Primeiro de tudo será necessário você obter o Xdialog no seu sistema. Não é comum o Xdialog estar incluso nas distribuições, então você terá de pegar e compilar o programa. Obtenha o programa no CD da Revista ou visite o endereço oficial do Xdialog, que é http://xdialog.free.fr. Aqui eu peguei o arquivo *Xdialog-1.4.5.tar.bz2*, e agora vamos aos passos básicos para instalar ele. Primeiro descompacte-o com o comando *bunzip2 Xdialog-1.4.5.tar.bz2*, e logo em seguida *tar xpvf Xdialog-1.4.5.tar*. Um diretório chamado Xdialog-1.4.5 será criado, e entrando nele você estará pronto para compilá-lo e instalá-lo. Para fazer isso use os comandos *./configure*, depois *make* e por último *make install*. No

passo do *make install*, o binário do Xdialog será colocado em */usr/local/bin*. Pronto, agora você já poderá utilizar o Xdialog através de qualquer shell-script.

E agora vamos à ação! Como aprendemos nos artigos anteriores, em shell-script é só colocar o comando dentro do arquivo que ele já vai ser executado quando o script for executado. Então só o que temos de aprender aqui é como usar o Xdialog. Vamos ver um primeiro exemplo:

```
#!/bin/bash

Xdialog --title "Exemplo número 1!" --center --stdout --yesno \
"Isto é legal?" \
0 0

echo $?
```

Como você pôde ver, o programa Xdialog gerou uma janela com título "Exemplo número 1!", perguntando "Isto é legal?" e com opções de Sim e Não. Note que a \ (barra) serve para indicar à shell para continuar o comando da linha seguinte, então estas três linhas são que nem uma só. Como último comando do exemplo dado, temos o *echo $?*, que eu coloquei apenas para indicar qual foi o retorno da pergunta. Caso o usuário apertou em Sim, o retorno vai ser *0*, e se apertou em Não, vai ser *1*. Podemos usar este retorno para controlar o que o usuário escolher. Vejamos um exemplo:

```
Xdialog --title "Exemplo número 2!" --center --stdout --yesno \
"Neste exemplo, vamos ver o que você quer fazer. Você deseja continuar
com o programa?" \
0 0

if [ $? = "0" ]; then
   echo "Que bom! Você continuou o programa! Parabéns!"
elif [ $? = "1" ]; then
   echo "Você saiu do programa..."
fi
```

Viu como funciona? Agora vamos ver outros recursos que o Xdialog pode oferecer. Eu vou dar vários exemplos aqui e sair comentando cada opção. Você precisará praticar bastante e conhecer as várias opções. Primeiro vamos gerar uma simples mensagem pro usuário ver:

```
Xdialog --title "Aviso" --center --stdout --msgbox \
"Este programa é apenas um exemplo para você ver como o Xdialog \
\nfunciona. Apropósito, se você praticar bastante pode criar \
\nprogra mas incríveis e facilmente, que daria muito mais \
\ntrabalho fazendo em outras linguagens." \
0 0
```

O usuário aperta Ok e o shell-script continua normalmente. No primeiro exemplo eu usei a opção *--yesno* que gerava o sim e não. Agora usei o *--msgbox*. Mas e se você quiser que o usuário digite algo e isto seja gravado em um arquivo por exemplo? Vamos ver este exemplo:

```
Xdialog --title "Que Anime que você mais gosta?" --center --inputbox \
"Se você sabe o que é Anime, e gosta, qual o seu preferido?\n \
 Sua resposta será gravada no arquivo resposta." \
0 0 2> resposta
```

Depois que o usuário preenche o campo e dá Ok, a resposta que este usuário digitou será gravada no arquvio *resposta*. Isto ocorreu pelo fato de eu ter colocado o direcionador *2>* para o arquivo resposta. Se eu colocasse a opção --stdout na linha de comando do Xdialog, a resposta do usuário apareceria na tela. Tente você.

Vamos ver agora seleção de itens, que é uma das coisas mais importantes num programa. Desta vez usaremos a opção *--menubox* para gerar um menu com os itens a serem selecionados. Mais uma vez, vamos ao exemplo:

```
Xdialog --title "Exemplo de Menu" --center --stdout --menubox \
"Qual sua distribuição Linux favorita?" \
20 50 0 \
1 "Slackware" \
2 "Debian" \
3 "Red Hat" \
4 "Conectiva Linux" \
5 "Eu tenho minha própria distribuição"
```

Viu como é fácil? O que o usuário escolher vai aparecer como resultado no console (por causa da opção *--stdout*). Se eu colocasse o redirecionador *2>*, poderia ir para um arquivo como no exemplo anterior. Vamos esclarecer uma coisa aqui também... Depois do texto "Qual sua distribuição Linux favorita?", há 2 números. Estes dois números correspondem à altura e comprimento da janela. Nos exemplos anteriores eu coloquei *"0 0"* pois aí o Xdialog dimensiona automaticamente o tamanho da janela. Então já sabe, se quiser mudar o tamanho da janela, é só mudar estes dois números.

Agora como último exemplo, vamos criar uma janela em que o usuário poderá escolher uma ou mais opções. Isto é chamado de *checklist*, e pode ser visto no exemplo a seguir:

```
Xdialog --title "Último exemplo - checklist" --center --checklist \
"Como se pronuncia Linux?" \
0 0 0 \
"Opção 1" "Láinucs" off \
"Opção 2" "Lenocs" off \
"Opção 3" "Linúcs" off \
"Opçào 4" "Línucs" on \
"Opção 5" "GNUUU/Linux" off
```

Veja agora a diferença entre esta checklist e o menu do exemplo anterior. Verifique que depois de cada opção há o *on* e o *off*. O *on* indica que esta opção deverá já estar marcada, e o *off* que não deverá estar marcada. Se o usuário escolher 3 opções, as mesmas serão o resultado.

Bem fácil criar uma interface amigável para suas shell-scripts. Se você pensa que isto é tudo, ainda tem muito mais por trás do Xdialog. Para não ter que ficar comentando cada opção, vou dar uma lista de parâmetros e a descrição de suas funções. O que você deverá fazer é sair testando todas as opções e se impressionar :)

–yesno	Uma janela com opções de “Sim” ou “Não”
–msgbox	Apenas mostra um texto informativo
–infobox	Mesmo que –msgbox, só que desaparece automaticamente em um determinado tempo
–inputbox	O usuário preenche um campo
–rangebox	Escolhe um número entre X e Y, com uma barra de rolagem
–textbox	Mostra o conteúdo de um arquivo numa caixa de texto
–editbox	Edita o conteúdo de um arquivo numa caixa de texto
–menubox	Cria um Menu de opções, onde se seleciona um item
–checklist	Mesmo que –menubox, só que pode-se selecionar vários itens
–radiolist	Mesmo que –menubox, mas agora em outro estilo
–treeview	Opções organizadas em forma de “árvore” (interessante)
–gauge	Um indicador de processo
–tailbox	Mostra o conteúdo de um arquivo
–fselect	Abre uma janela de seleção de um arquivo
–dselect	Abre uma janela de seleção de um diretório
–calendar	Mostra um calendário para escolher uma data
–timebox	Mostra uma janela para edição de horário

Você também pode encontrar alguns exemplos no diretório *samples* que vem junto com o código-fonte do programa. Se você fizer algum programa legal em shell-script, sinta-se a vontade para me mandar um e-mail. Outra coisa, como o Xdialog é uma idéia tirada do dialog/cdialog (existe também o whiptail que é parecido), você pode usar a mesma

sintaxe para criar estas "janelas" no modo console. Espero que estas informações foram úteis a você e até a próxima :)

Autor do Documento: Hugo Cisneiros