Table of Contents

Programamando em shell−script.....**1**
1. Primeira parte, uma introdução.....1
2. Segunda parte, se aprofundando mais!.....5

Programando em shell−script

Hugo Cisneiros, hugo_arroba_devin_ponto_com_ponto_br
Última atualização em 01/02/2003

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 'eu sou um frutinha')
echo $variavel
   eu sou um frutinha
```

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
!=	Diferente
−gt	Maior
−lt	Menor
−o	Ou

−d	Se for um diretório
−e	Se existir
−z	Se estiver vazio
−f	Se contiver texto
−o	Se o usuário for o dono
−r	Se o arquivo pode ser lido
−w	Se o arquivo pode ser alterado
−x	Se o arquivo pode ser executado

Outras alternativas

Existem inúmeros comandos no Linux, e para explicar todos, teríamos de publicar um verdadeiro livro. Mas existem outras possibilidades de aprendizado desta língua, que também é usado em todas as programações. Primeiro de tudo você pode dar uma olhada na manpage do bash (comando man bash), que disponibilizará os comandos embutidos no interpretador de comandos. Uma das coisas essencias para o aprendizado é sair coletando exemplos de outros scripts e ir estudando−os minuciosamente. Procure sempre comandos e expressões novas em outros scripts e em manpages dos comandos. E por último, mas não o menos importante, praticar bastante!

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

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

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
```

Hugo Cisneiros, hugo_arroba_devin_ponto_com_ponto_br