Capítulo 10 - Shell Script Parte 1

10.1. Objetivos:

Entender como funciona e como criar scripts em Shell

10.2. Introdução

Os sistemas Unix-like, como o GNU/Linux, possuem camadas. Estas camadas são o hardware, o kernel, os programas/comandos e o shell. O shell é a camada mais externa de um sistema Unix-like.

Script é um arquivo de texto que possui uma seqüência de instruções e comandos que são executados linha a linha. A vantagem de se programar em Shell Script é automatizar tarefas rotineiras, como backup, instalação ou remoção de programas.

Bourne Again Shell: Este é o shell desenvolvido para o projeto GNU usado pelo GNU/Linux, é muito usado pois o sistema que o porta (GNU/Linux) evolui e é adotado rapidamente.

Funções do Shell :

- *Analisar dados a partir do prompt (dados de entrada);*
- *Interpretar comandos;*
- *Controlar ambiente Unix-like (console);*
- *Fazer redirecionamento de entrada e saída;*
- *Execução de programas;*
- *Linguagem de programação interpretada.*

10.3. Conhecimentos básicos

O uso da tralha (#). A tralha ou jogo da velha (#) representa, em várias linguagens de programação, um comentário, o mesmo acontece com o Shell Script. Um script em Shell é iniciado com a seguinte linha:

```
#!/bin/SHELL_EM_USO
```

Para o GNU/Linux:

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

Esta linha acima indica o caminho (path) para o interpretador que será usado no script.

10.3.1. Crases

As crases são usadas para dar prioridade a um comando, veja um exemplo:

```
$ echo “A versão do kernel do `uname -o` é `uname -r`”
```

Saída:

```
A versão do kernel deste GNU/Linux é 2.6.15
```

Se você tirar as crases, veja a saída:

```
A versão do kernel deste uname -o é uname -r
```

10.3.2. Variáveis

A variável representa um espaço em memória para que sejam armazenados dados. Uma variável é representada por $ (cifrão). Exemplo de variável:

```
# guarda_roupa=camiseta
# echo $guarda_roupa
```

Saída: camiseta

Depois de editar um novo script, é necessário modificar a permissão deste arquivo, senão este não poderá ser executado, veja o porquê:

```
$ ls -l
-rw-r--r-- 1 leo users 0 2006-05-20 13:20 backup.sh

$ chmod +x backup.sh

$ ls -l
-rwxr-xr-x 1 leo users 0 2006-05-20 13:20 backup.sh

Para executar:
$ ./backup.sh
```

10.3.3. Cálculos em Shell Script - expr

```
Operadores aritméticos:
+ Soma
- Subtração
* Produto
/ Divisão
% Resto da divisão
```

Para fazer um cálculo é necessário usar o comando expr.

Exemplos:

```
$ expr 20 + 05
$ expr 20 - 05
$ expr 20 \* 05
$ expr 20 / 05
$ expr 20 % 05
```

10.3.4. Trabalhando com parâmetros

Um parâmetro é representado por $n, onde n é a posição do caractere ou conjunto de caracteres.

$1 é o primeiro caractere ou primeiro conjunto de caracteres;
$2 é o segundo caractere ou o segundo conjunto de caracteres e assim por diante.

Na prática, veja como é fácil:

Supondo um programa chamado "monte_nome" que exibe o nome montado após receber letra por letra. Veja o script:

```
#
# Script para montar nomes
# Este script recebe nove parâmetros
#
#!/bin/bash
echo $1 $2 $3 $4 $5 $6 $7 $8 $9
#
# Fim do script
#
$./monte_nome M A R I A (Note que entre cada parâmetro há um espaço)
Saída: MARIA
```

E o parâmetro $0? Qual é o seu conteúdo?

O parâmetro $0 representa o nome do próprio programa. Mas pense neste programa , é péssimo pois está limitado a nove caracteres apenas. É simples arrumar isso, veja:

```
#
# Script para montar nomes
# Este script recebe "n" parâmetros
#
#!/bin/bash
echo $*
echo Foram passados $# parametros
#
# Fim do script
#

$ ./monta_nome M A R I A D A S I L V A

Saída: MARIADASILVA
```

O $* recebe todos os parâmetros passados. O Linux não se importa com a extensão de arquivos. Foi colocada a extensão .sh para ajudar a lembrar que esse arquivo é um script.

É através da permissão que ele sabe se o arquivo é executável ou não!

```
# vi usuarios.sh
```

A primeira linha do script deve ser essa aqui:

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

Depois que defini-se essa linha, já pode colocar os comandos que deseja-se executar. O objetivo é obter uma lista dos usuários cadastrados no sistema.

```
echo “Lista completa dos usuários do sistema”
cut -f1 -d : /etc/passwd | sort | more
```

A primeira opção que temos dele é a f1 que está pedindo para o cut cortar o primeiro campo (field) do arquivo /etc/passwd. Só que além de falar para mostrar o primeiro campo, é preciso dizer para o cut qual é o meu delimitador, ou seja, o que separa cada informação. E no caso, o delimitador (opção -d) é o :

```
 echo "O Sistema possui `cat /etc/passwd | wc -l` usuarios"
```

A novidade nessa terceira linha além das crases é o comando wc.

wc significa word count. É um comando usado para contar palavras. Mas com a opção -l (é l de limão!), ele conta as linhas. Então, agora é só executar o script. Para isso, basta salvar o arquivo, sair e dar permissão de execução:

```
# chmod +x usuarios.sh
```

Então vamos executar:

```
./usuarios.sh
```

10.4. Estruturas Condicionais

Uma linguagem de programação não pode sobreviver sem estruturas condicionais. As estruturas condicionais são usadas em tarefas muito corriqueiras. Para compreender a estrutura condicional do Shell Script, primeiro é necessário saber como ele testa se uma condição é falsa ou verdadeira.

Para isso, o shell script trabalha com um código de retorno. Esse código de retorno fica guardado dentro de uma variável que é representada por $? (cifrão + ponto de interrogação). De modo que se um comando no shell for executado com sucesso, o código de retorno será igual a ZERO.

Se um comando no shell falhar, o valor retornado será DIFERENTE DE ZERO.

Listando um arquivo que existe:

```
$ touch testando.txt
$ ls testando.txt
testando.txt
$ echo $?
0
```

Listando um arquivo que NÃO existe:

```
$ ls nada
ls: nada: Arquivo ou diretório não encontrado
$ echo $?
2
```

O código de retorno foi diferente de zero, então o comando NÃO foi executado com sucesso. O comando if do Shell, no seu formato normal, não testa uma condição! Ele testa se uma instrução foi executada com sucesso ou não, ou seja, se seu código de retorno é igual ou diferente a zero. É possível testar condições também, isto será mostrado mais a frente.

Sintaxe do if:
if [<expressão>]
then
comando(s)
else
comando(s)
fi

Exemplo:

```
$ vi pedido.sh
# Pergunta ao usuário se ele quer listar o diretório corrente
#!/bin/bash
resposta=$1
if test $resposta = S
then
echo "O conteúdo do diretório corrente é:"
ls
else
if test $resposta = N
then
echo "Não quer listar!"
fi
fi
```

Para cada if tenho um fi

Salve e dê permissão de execução no arquivo.

```
$ ./pedido.sh S
O conteúdo do diretório corrente é:
documentos
arquivo.txt

$ ./pedido.sh N
Não quer listar!
```

Reescrevendo o código de maneira mais legível:

```
$ vi pedido.sh
#
# Pergunta ao usuário se ele quer listar o diretório corrente
#
#!/bin/bash
resposta=$1
if [ $resposta = S ]
then
echo "O conteúdo do diretório corrente é:"
ls
else
if [ $resposta = N ]
then
echo "Não quer listar!"
fi
fi
```