Curso do Unix

O Script do Shell

Apostila 3

Agosto de 2016

Prof. Me Lázaro Aparecido da Silva Pinto

Versão 1.9 Ago 5, 2016

1. O Script do Shell

Shell é o nome genérico dado ao interpretador de comandos do UNIX. Além de executar comandos solicitados pelo usuário, o shell também possui uma linguagem de programação interpretada.

A programação shell consiste na criação de arquivos que contém uma lista de comandos. Estes arquivos são denominados **scripts**. Scripts shell são utilizados para automação de tarefas realizadas freqüentemente bem como customizar seu ambiente de trabalho.

A sintaxe dos comandos de um script varia de acordo com o shell utilizado. Nossa discussão será restrita a sintaxe do Korn shell. Antes porém, vejamos os seus caracteres especiais:

Caracter		Descrição
\	->	a barra invertida nega o significado do caracter posterior a ele.
'....'	->	anula o significado especial de todos os caracteres.
"..."	->	anula o significado especial de todos os caracteres, com exceção dos símbolos: ` ` $ \
`...`	->	permite a execução de um comando dentre de um outro comando.
?	->	qualquer caracter único.
*	->	qualquer caracter.

2. Criação de Scripts

Para a criação de scripts devemos utilizar um editor de textos (vi ou edit) para a definição de um arquivo e nele incluirmos comandos do shell. Abaixo descrevemos um exemplo de script:

cat > modelo1.ksh ou vi modelo1.ksh

```
# Este script posiciona o usuário em seu diretório home
# e lista o nome e o conteúdo deste diretório
#
cd
pwd
ls
```

3. Execução de Scripts

Para executar um script basta digitarmos o nome do arquivo. Se o usuário não especificar em que diretório o arquivo se encontra, o shell procura no diretório definido pela variável path. Os diretórios onde estão armazenados scripts e programas escritos pelo próprio usuário, devem ser adicionados à variável path, para que possam ser localizados automaticamente.

Exemplo:

```
cd
pwd
/usr/users/maria
./modelo1.ksh
ksh: modelo1.ksh: not found
print $PATH
/usr/bin:/sbin/usr/sbin
PATH=$HOME:$PATH
print $PATH
/usr/users/maria:/usr/bin:/sbin:/usr/sbin
./modelo1.ksk
/usr/users/maria
livros  cartas  memorandos.doc  mala.dir
```

Obs:

(1) Para executar qualquer script, primeiro é necessário habilitar o arquivo para execução: **chmod u+x modelo1.ksh –v** (resultado r w x r w – r - -)

(2) Para executar o programa acima fazer: **./modelo1.ksh**

4. Variáveis

Como qualquer outra linguagem de programação, o shell utiliza variáveis para armazenar e recuperar informações. Começa com uma letra ou sublinhado (_).

A definição de variáveis podem ser feitas de duas formas:

Variável=valor_correspondente
ou
typeset [opções] variável

Onde as opções podem ser:

Opção	Descrição
-u	indica que a variável possuirá caracteres maiúsculos
-l	indica que a variável possuirá caracteres minúsculos
-i	indica que a variável é do tipo inteiro
-r	define a variável como sendo read-only
-x	atribui a variável a característica **export**, ou seja, é uma variável global
-Ln	indica que a variável é alinhada pela esquerda, e que possui o tamanho de n caracteres
-Rn	indica que a variável é alinhada pela direita, e que possui o tamanho de n caracteres

A tabela abaixo mostra alguns exemplos:

Exemplo	**Função**
salario=media	atribui a variável *salario* o valor da variável *media*
quantidade=120	assinala um valor numérico à variável *quantidade*. Por default todas as variáveis são strings.
Vazio=	cria a variável vazio com o valor nulo (null)
typeset –i quantidade=60	declara *quantidade* como sendo uma variável do tipo inteiro e atribui a ela o valor de 60
integer y=100	tornar *y* um valor inteiro
set	lista todas as variáveis e seus correspondentes valores
typeset	lista todas as variáveis e seu tipo de dado
unset salario	elimina o conteúdo da variável *salario*

Obs.1: a declaração integer também pode ser: int y=100 x=200 k=1.
Obs.2: para armazenar ou atribuir valor a uma variável, basta colocar o nome da variável, um sinal igual (=) colado ao nome escolhido e, colado ao sinal de igual, o valor estipulado.

Exemplos diversos:
Exemplo1: para exibir o valor de uma variável, devemos preceder o seu nome por um cifrão ($).

```
Variável=qqcoisa
Contador=0
Vazio=
Echo Variável=$Variavel, Contador=$Contador, Vazio=$Vazio
```

Exemplo2: Mostrar variável numérica com mais zeros
Echo 0$Contador

Exemplo3: Concatenar variáveis
Echo $Variavel-jilo

4.1. Variáveis Globais

Quando um usuário estabelece uma sessão, o sistema inicializa um único processo pai. O mesmo usuário pode, no entanto, disparar vários processos. A criação de um novo processo pelo processo corrente é chamado "forking", e o novo processo é denominado processo filho (child process). A qualquer momento um processo pode criar outro, que por sua vez pode criar outros processos e assim por diante. Vários processos (pais e filhos) podem ser executados concorrentemente. Para realizarmos um forking, ou seja, criarmos um processo filho, podemos utilizar o comando ksh.

Quando uma variável é definida, seu valor só está disponível no shell corrente. Para torná-la uma variável global, ou seja, ficar disponível também à processos filhos do processo corrente, utilizamos o comando export após a definição da variável.

Exemplo:
CLUBE=Flamengo
print "Sou torcedor do $CLUBE"
Sou torcedor do Flamengo!
Ksh
print "Vou assistir o jogo do $CLUBE"
Vou assistir o jogo do
[Ctrl/D]
export CLUBE
ksh
print "Vou assistir o jogo do $CLUBE"
Vou assistir o jogo do Flamengo

Uma outra forma de executarmos comandos em um subprocesso (subshell) é colocando os comandos entre parêntesis.

Exemplo:
pwd; date; (cd /etc; ls –l | grep ^d); pwd

O resultado será:
/usr/users/luiz
Wed Apr 13 15:22:46 1994
drwxr-xr-x 6 root system 2048 Jan 12 08:22
drwxr-xr-x 9 root system 1024 Jan 13 18:22 netman

No exemplo acima, os comandos entre parêntesis foram executados em um subprocesso.

4.2. Variáveis do Korn Shell

O shell possui algumas variáveis reservadas que tem por finalidade caracterizar o ambiente do usuário. O comando set exibe este conjunto de variáveis. Entre estas variáveis podemos destacar:

Variável	Significado
USER	nome do usuário (username)
PATH	definição de paths
HOME	diretório base do usuário
SHELL	shell corrente
TERM	tipo do terminal (ex. VT100)
OLDPWD	o diretório em que se estava posicionado anteriormente
PPID	o PID do processo pai (parent process)
SECONDS	o número de segundos desde que o Korn Shell foi acionado
$	o processo corrente
!	status do último comando ou programa executado. O status 0 indica sucesso, em caso contrário houve erro.
HISTFILE	indica o nome do arquivo de histório de comandos (log)

5. Gravação (impressão) e Leitura de dados no terminal

5.1. Gravação (impressão) de dados no terminal

Para gravarmos dados, cadeia de caracteres ou variáveis, no terminal utilizamos o comando **print**.

```
print "mensagem"
print $variavel
print "mensagem" $variável
```

Obs: print "" – gera uma linha vazia.

5.2. Leitura de dados do termina

A leitura de dados via terminal é feita através do comando **read**.

```
read variavel
```

Obs: Toda vez que o programa encontrar um read, o programa pára esperando que o usuário digite alguma variável ou um Ctrl/C para cancelar o programa.

6. Controle do Fluxo de Execução

6.0. Informações Gerais

Todas as linguagens de programação se utilizam fartamente dos comandos condicionais. Por que o Shell seria diferente? Ele também usa estes comandos, porém de uma forma não muito ortodoxa. Enquanto as outras linguagens testam direto uma condição, o Shell testa o código de retorno do comando que o segue.

O código de retorno de uma instrução está associado à variável $?, de forma que sempre que um comando é executado com sucesso, $? É igual a zero, caso contrário o valor retornado será diferente de zero.

$ ls –l diretorio1
$ echo $?
Será 0 se o diretório1 existir.
Será diferente de 0, se diretório1 não existir.

Quando se executa diversos comandos encadeados em um pipe (|), o return code dado por echo $? Refere apenas o resultado de saída do último comando executado no pipe. O array PIPESTATUS, por sua vez, armazena em cada elemento o resultado respectivo de cada um dos comandos do pipe. ${PIPESTATUS=[0]} tem o return code do primeiro comando, S{PIPESTATUS=[1]} contém o return code do segundo, e assim por diante.

6.1. Desvio condicional usando if

Quando criamos um script mais elaborado, muitas vezes é necessário, após verificarmos o valor de certas variáveis, desviarmos o fluxo da execução do arquivo ou mesmo gerarmos uma estrutura de repetição (loop).

Uma condição é normalmente uma expressão relacional em que o valor de um ou mais itens são comparados. Para isto, dispomos do comando **if**, cuja sintaxe é:

```
if ((condição))
then
        Comandos verdade
else
        Comandos falsidade
fi
```

Quando a condição é entre variáveis numéricas elas são colocadas entre dois parêntesis (( )), e quando é entre variáveis do tipo string são colocadas entre dois colchetes [[ ]]. Quando a condição é feita com comandos do Unix, não precisa ser colocado nada.

Exemplo:

$ cat > talogado.txt

```
#
# Verifica se determinado Usuário está logado no sistema
#
if who | grep $1
        then
                echo $1 está logado
        else
                echo $1 não está logado
fi
```

Ao executar:
./talogado.txt fsa33333

O comando test.

test <expressão>

Sendo <expressão> a condição que se deseja testar. O comando test avalia <expressão> e, se o resultado for verdadeiro, gera o código de retorno ($?) igual a zero; caso contrário, será gerado um código de retorno diferente de zero.

Para definirmos a condição utilizamos os seguintes operadores relacionais e lógicos:

Operadores Relacionais	**Descrição**
=	igual (utilizado para variáveis não numéricas)
==	igual (utilizado para variáveis numéricas)
!=	diferente
>	maior que
<	menor que
>=	maior ou igual
<=	menor ou igual

Lógicos

||	ou
&&	e

Informações importantes quando usar comandos if:

a. Todo if deve ser terminado pelo comando fi
b. Todo if deve ter a expressão then logo após a condição
c. Se um if tiver apenas o then, então ele será executado sempre que a condição for verdadeira. Se for falsa, o comando seguinte ao fi será executado.
d. É possível fazer encadeamento de IF Ex: 1

```
if condição 1
        then
                condição 2
        else
                if condição 3
                        then
                                condição 4
                        else
                                condição 5
                fi
fi
```

Ex: 2

```
if condição 1
        then
        if condição 2
                then
                if condição 3
                        then
                                condição 4
                        else
                                condição 5
                fi
        else
                condição 6
        fi
fi
```

6.2. Estrutura de repetição com for

- Conceito de bloco de programa.

O agrupamento de instruções compreendido entre um sinal de abre chaves ({) e um de fecha chaves (}).

```
[ -d Diretório  ] ||        * O operador lógico || obriga a execução da instrução
{                           * seguinte,caso a anterior tenha sido mal sucedida.
    mkdir Diretorio         * Então, no caso de não existir o diretório contido na
    cd Diretorio            * variável Diretorio, o mesmo será criado e, então,
}                           * faremos um cd para dentro dele.
```

Veja o seguinte código:

```
RegUser=`grep "^$1:" / etc/passwd` ||       * Repare que desta forma será
{                                           * atribuído valor à variável $RegUser,
    echo "ERRO: Não existe usuário [$1], cadastrado !!!"    * caso o 1º Parâmetro
    exit 3                                  * ($1) seja encontrado no  início de um
}                                           * registro contrário, o bloco de comandos
                        * será de /etc/passwd, caso executado, dando uma
                        * mensagem de erro e abortando a a execução do
                        * programa.
```

Um bloco de programa também pode ser aberto por um do, por um if, por um else ou por um case e fechado por um done, um else, um fi ou um esac.

Comando For – Controle de decisão:

Sintaxe:

```
for     var in valor1, valor2,...., valorN
do
        <comando1>
        <comando2>
        < ……   >
        <comandoM>
done
```

Para implementarmos estruturas de repetição podemos utilizar o **for**:

```
for n in 1 2 3 4 5; do
        Comandos 1
        Comandos 2
        Comandos 3
done
```

Especifica uma lista de valores para uma variável.
A variável n recebe o numero 1 e executa os comandos abaixo, depois o numero 2, até o número 5. Após o número 5, o programa seguirá para a próxima instrução após o done.

Ex:

```
$ cat bronze
#
# Lista múltiplos de onze a partir de 11 até 99
#
for i in 1 2 3 4 5 6 7 8 9
do
        echo $i$i
done
```

Comando para gerar uma seqüência numérica:

Sintaxe:

seq ultimo

seq primeiro ultimo

seq primeiro incremento ultimo

No primeiro caso, seria gerada uma seqüência numérica de todos os reais começando em **1** e terminando em **último**.

No segundo caso, seria gerada uma seqüência numérica de todos os reais começando em **primeiro** e terminando em **último**.

No terceiro caso, seria gerada uma seqüência numérica de todos os reais começando em **primeiro** e terminando em **último**, porém os reais viriam espaçados de **incremento**.

```
seq –s " " 10
1 2 3 4 5 6 7 8 9 10
seq –s " " 10 15
10 11 12 13 14 15
seq –s " " 5 2 15
5 7 9 11 13 15
```

Obs. está sendo usado a opção –s " " para que o separador entre os números gerados fossem um espaço em branco.

6.3. Estrutura de repetição com while

Para implementarmos estruturas de repetição podemos utilizar o **while**:

```
while ((condição))
do
        Comandos
        Comandos
done
```

Executa os comandos especificados enquanto a condição for verdadeira.

6.4. Estrutura de repetição com until

Para implementarmos estruturas de repetição podemos utilizar o **until**:

```
until ((condição))
do
        Comandos
        Comandos
done
```

É o oposto do while, ou seja, opera com uma condição falsa.

7. Passagem de parâmetros

Podemos passar parâmetros para um script bastando para isso digitar, após o nome do arquivo, cada argumento separando-os por um espaço. Os argumentos são então associados as variáveis $1, $2, $3, ..., $n respectivamente. Dispomos ainda das variáveis $#, número de parâmetros passados, e $0, nome do arquivo de script.

```
cat parametros.ksh
#!/bin/ksh
#
if (( s# != 3 ))
then
        print "Você deve passar três parâmetros !!!"
else
        print "Seu nome é $1 $2 $3"
fi
```

Forma 1:
Executando o programa como parametros.ksh
Resultado: Você deve passar três parâmetros !!!
Forma 2:
Executando o programa como parametros.ksh lua sol terra
Resultado: Seu nome é lua sol terra

8. Executando com comando case

O comando case permite que se estruture o programa de forma a executar subrotinas de forma racionalmente. A sintaxe do case é:

```
case $variavel in
        1) função; comandos; ;;
        2) função; comandos; ;;
        3) função; comandos; ;;
        4) função; comandos; ;;
esac
```

Como utilizar o comando case:

a) Ao criar uma programa com o comando case, você deve usar rotinas fechadas, ou seja:

O programa ps01.ksh é feito assim:

```
#!/bin/ksh
menu()
{
    echo "Entre com uma opção:"
    echo
    echo "1. Exibir ......."
    ëcho "2. Exibir ......."
    echo "3. Calcular ..... "
    echo "4. Sair do ... "
    echo –n "Digite opção: "
    read opcao
    função_opção
}
Função_opção()
{
case $opcao in
    1) ..... ;;
    2) ..... ;;
    3) ..... ;;
    4) ..... ;;
esac
}
menu
```

9. Informações gerais

a) Quando **atribuir valores a variáveis**

Certo: quantidade=30 Errado: quantidade = 30

b) Quando usar **condições com strings**

Certo: if ((campo=”VAGO”)) Errado: if ((campo == “VAGO”))

Certo: if ((campo = ”VAGO”)) Errado: if ((campo == “VAGO”))

c) Quando usar **condições com números**

Certo: if ((valor==30)) Errado: if ((valor = 30))

Certo: if ((valor == 30)) Errado: if ((valor = 30))

d) Quando usar **referência a variável**

Certo: numero=$numero+1 Errado: numero=numero+1

Certo: numero=$numero+1 Errado: numero = numero+1

e) Quando usar **comando echo ou print**

Certo: echo “informe o valor” Errado: echo”informe o valor”

Certo: print “O valor é $valor” Errado: print“O valor é valor”

f) Quando usar **vários comandos na mesma linha**

Certo: date; print “Valor $x”; exit Errado: date, print “Valor $x”, exit

g) Quando usar **definir variáveis com tipos**

Certo: integer y=10 x=20 k=30 Errado: integer y=10,x=20,k=30

Certo: int y=10 x=20 k=30 Errado: int y=10,x=20,k=30

h) Quando usar **definir variáveis com tipos múltiplos**

Certo: area –xi Errado: area –x, -i

Certo: area –x –i Errado: area –x, -i

i) Quando usar **utilizar expressões aritméticas**

Certo: valor=$numero*16/3+$outronumero*7/4

Errado: valor = $numero * 16/3 + $outronumero * 7 / 4

j) Quando usar **várias funções num mesmo programa**

Certo: Colocar o nome da primeira função a ser executada no final do programa

Exemplo:

```
Menu()
{
    comandos
    comandos
}
Função_escolha()
{
    comandos
    comandos
}
Função_a()
{
    comandos
}
Função_b()
{
    comandos
}
menu
```

Obs. Ao executar o programa, o interpretador do shell vai saltar todas as funções e, ao achar a palavra menu, vai executar a função menu.

k) Quando usar **if encadeado**

```
Certo:    if (( a==3))
          then
                if (( b==4))
                then
                      condição verdade
                else
                      condição falsa
                fi
          fi
```

"A espécie de felicidade de que preciso não é fazer o que quero, mas não fazer o que não quero."

Jean-Jacques Rousseau