Aula 9: Shell Script

Professor: Emerson Ribeiro de Mello http://docente.ifsc.edu.br/mello

1 O interpretador de comandos

O *shell* consiste em um interpretador de comandos presente em todos os sistemas operacionais variantes dos Unix, que inclui Linux, BSD e MacOS. No linux existem diversos tipos de *shell*, sendo estes: `csh`, `bash`, `ksh` e `zsh`.

No interpretador de comandos é possível invocar comandos isolados ou ainda combinar diversos comandos. Por exemplo, o comando `ls -l` pode ser executado sozinho, porém seria difícil visualizar uma lista grande de arquivos. Assim, o comando `ls` poderia ser combinado com o comando `more` o que permite paginar a saída, tornando a leitura mais fácil. Essa combinação de comandos se dá através do uso do *pipe*, representado pelo símbolo |. Exemplo: `ls -l | more`.

2 Programando em shell

Como dito, o *shell* é um interpretador de comandos e temos a opção de entrar com uma seqüência de comandos sempre que desejarmos realizar uma tarefa ou podemos colocar tal sequência dentro um arquivo e chamar este arquivo sempre que necessário. E assim temos o *shell script* ilustrado pelo Código 1.

Código 1: Meu primeiro shell script

```
1 #!/bin/bash
2
3 echo "Ola mundo!"
```

2.1 Alguns comandos interessantes para shell script

Abaixo um lista com os principais comandos que iremos utilizar em nossos scripts.

- **echo** – tem por objetivo imprimir mensagens no dispositivo de saída padrão, no caso o monitor. Abaixo algumas opções:

  -e Ativa a interpretação de caracteres de escape (\)
  - `\n - nova linha`
  - `\t - tab`
  - `\a alerta (beep)`

  -n Exibe a mensagem sem pular linha

- **read** – Permite que o usuário forneça informações via teclado (é necessário pressionar `ENTER` para finalizar a leitura). Algumas opções:

  -s não exibe os caracteres que estão sendo fornecidos

  -t seg aguarda N segundos para que o usuário entre com algum dado

  -n N Após ler N caracteres o `read` é encerrado sem que precise pressionar `ENTER`

- **expr** – para fazer cálculos, porém só faz operações com inteiros. Exemplo de uso:

```
4  # executando o expr em um terminal
5  expr 2 + 2
6  # executando o expr em um terminal e guardando o resultado na varial 'soma'
7  soma=`expr 2 + 2`
```

- **bc** – trata-se de uma calculadora, ideal para quando necessitamos efetuar cálculos com números reais. Exemplo de uso:

```
8  # executando o bc em um terminal, combinado com o echo
9  echo "scale=2; 1/2" | bc
10
11 # armazendo o resultado da saida do bc na variavel 'resultado'
12 resultado=`echo "scale=2; 1/2" | bc`
```

2.2 Variáveis

Nas linguagens de programação as **variáveis** possuem uma função semelhante com as variáveis da matemática, ou seja, armazenam valores para que possam ser recuperados posteriormente. O Código 2 ilustra algumas formas para atribuir e obter valores em variáveis.

Código 2: Exemplo de definição e uso de variáveis

```
13 #!/bin/bash
14
15 # Isto é um comentário. Todo texto após o caracter # não será interpretado pela shell
16
17 echo "Trabalhando com variaveis"
18
19 a=1
20 b=2
21 c=`expr $a + $b` # a expressao está entre crases
22 d=$((c+a))
23
24 echo "O valor de a e' $a, o valor de b e' $b, o valor de c e' $c e o valor de d e' $d"
25
26 curso="FIC Linux"
27
28 echo "O conteudo de curso e' $curso"
29
30 # outro exemplo
31 versao=$(uname -r)
32
33 echo "A versao do kernel e' $versao"
```

Código 3: Usando variáveis em conjunto com o comando read

```
34 #!/bin/bash
35
36 echo -n "Entre com o seu nome: "
37 read nome
38 echo "Ola $nome!"
```

2.2.1 Variáveis de ambiente

As *variáveis de ambiente* são aquelas que afetam o comportamento do interpretador de comandos e do *shell script*. É importante frisar que cada `processo` possui seu ambiente. Um *script* só pode exportar tuas variáveis para os processos filhos. Um *script* invocado através da linha de comando não pode exportar de volta uma variável para o ambiente da linha de comando.

Variável	Descrição	Variável	Descrição
$BASH	caminho do binário do bash	$$	número do processo do shell
$HOME	diretório *home* do usuário	$HOSTNAME	nome da máquina
$PATH	caminho para os binários	$SECONDS	número de segundos desde quando o script começou a ser executado

2.3 Estruturas de decisão

Antes de apresentar as estruturas de decisão, na tabela 1 são apresentados os operadores relacionais e lógicos que são de grande importância para tais estruturas.

Operadores lógicos e relacionais					
Numéricos		**Cadeia de caracteres**		**Operadores lógicos**	
`-eq`	igual	`=`	igual	`-a`	E lógico (AND)
`-ne`	diferente	`!=`	diferente	&&	E lógico (AND)
`-ge`	maior ou igual	`-n`	não é nula	`-o`	OU lógico (OR)
`-le`	menor ou igual	`-z`	é nula	||	OU lógico (OR)
`-gt`	maior			`!`	negação
`-lt`	menor				

Tabela 1: Operadores relacionais e lógicos

2.3.1 Se... então... senão

Código 4: Estrutura de decisão SE

```
39 #!/bin/bash
40
41 nota=5
42
43 if [ $nota -ge 5 ];
44   then
45
46     echo "nota maior ou igual a 5"
47
48   else
49
50     echo "nota menor que 5"
51
52 fi
```

Código 5: Usando operador lógico E

```
#!/bin/bash

a=3
b=2
c=1

# usando o operador E
if [ $a -gt $b ] && [ $a -gt $c ];
  then
    echo "A e' o maior"
  else
    echo "A nao e' o maior"
fi

# outra forma para usar o operador E
if [ $a -gt $b -a $a -gt $c ];
  then
    echo "A e' o maior"
  else
    echo "A nao e' o maior"
fi
```

2.3.2 Escolha... caso...

Código 6: Estrutura de decisão ESCOLHA

```
#!/bin/bash

echo -n "Entre com um numero de 1 a 5: "
read numero

case $numero in
  1)
    echo "Voce escolheu 1"
    ;;
  2)
    echo "Voce escolheu 2"
    ;;
  3)
    echo "Voce escolheu 3"
    ;;
  4 | 5)
    echo "Voce escolheu 4 ou 5"
    ;;
  *)
    echo "Voce escolheu um numero diferente de 1, 2, 3, 4 ou 5"
    ;;
esac
```

2.4 Estruturas de repetição

2.4.1 Enquanto

Código 7: Estrutura de repetição ENQUANTO

```
97  #!/bin/bash
98
99  num=10
100
101 while [ $num -gt 0 ]; do
102   echo "contando $num"
103   num=$((num-1))
104 done
105
106
107 #----------------------------#
108 #usando o operador de negação '!'
109
110 num=10
111
112 while ! [ $num -eq 0 ]; do
113   echo "contando $num"
114   num=$((num-1))
115 done
```

2.4.2 Para

Código 8: Estrutura de repetição PARA

```
116 #!/bin/bash
117
118 for contador in `seq 1 10`; do
119   echo $contador
120 done
121
122 # percorrendo uma lista de palavras separadas por espaco
123 lista="FIC Linux Redes IFSC"
124
125 for palavra in $lista; do
126   echo "Palavra $palavra"
127 done
128
129 # Listando todos os arquivos de um diretório
130 # lista=`ls -1`
131 for arquivo in `ls -1`; do
132   echo "Nome do arquivo: $arquivo"
133 done
```

3 Exercícios

1. Desenvolva um algoritmo que leia dois números inteiros e exiba a soma destes números.
2. Desenvolva um algoritmo que solicite ao usuário seu nome e exiba uma mensagem de boas vindas utilizando este nome.
3. Desenvolva um algoritmo que leia um número inteiro e determine se este é par ou ímpar.
4. Desenvolva um algoritmo que leia dois números inteiros e exiba qual deles é o maior
5. Desenvolva um algoritmo que leia um número inteiro positivo e imprima a sequência de 0 até este número.

6. Desenvolva um algoritmo que simule a autenticação de usuários. O usuário deve fornecer uma senha e se esta senha for igual a palavra **secreta** deverá exibir a mensagem "Acesso autorizado", caso contrário deverá exibir "Acesso negado". O algoritmo deverá solicitar a senha ao usuário até que este forneça a senha correta ou até que o número de tentativas permitidas seja alcançado. No caso, o número máximo de tentativas é 3.