



Aula 01: Shell Script

14/06/2020

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 sequência 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 [Listagem 1](#).

Listagem 1: Meu primeiro shell script

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



Para executar um *bash script* digite: `bash nome-do-arquivo.sh` ou deixe o arquivo com permissão para execução, com o comando `chmod +x nome-do-arquivo.sh` e executá-lo da seguinte forma: `./nome-do-script.sh`

2.1 Alguns comandos interessantes para shell script

Abaixo uma 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:

```
1 # executando o expr em um terminal
2 expr 2 + 2
3 # executando o expr em um terminal e guardando o resultado na variável 'soma'
4 soma=`expr 2 + 2`
```

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

```
1 # executando o bc em um terminal, combinado com o echo
2 echo "scale=2; 1/2" | bc
3
4 # armazenando o resultado da saída do bc na variável 'resultado'
5 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 [Listagem 2](#) ilustra algumas formas para atribuir e obter valores em variáveis.

Listagem 2: Exemplo de definição e uso de variáveis

```
1 #!/bin/bash
2
3 # Isto é um comentário. Todo texto após o caracter # não será interpretado pela shell
4
5 echo "Trabalhando com variaveis"
6
7 a=1
8 b=2
9 c=`expr $a + $b` # a expressao está entre crases
10 d=$((c+a))
11
12 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"
13
14 curso="FIC Linux"
15
16 echo "O conteudo de curso e' $curso"
17
18 # outro exemplo
19 versao=$(uname -r)
20
21 echo "A versao do kernel e' $versao"
```

Listagem 3: Usando variáveis em conjunto com o comando read

```
1 #!/bin/bash
2
3 echo -n "Entre com o seu nome: "
4 read nome
5 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 suas 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 <i>home</i> 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

Listagem 4: Estrutura de decisão SE

```
1 #!/bin/bash
2
3 nota=5
4
5 if [ $nota -ge 5 ];
6 then
7
8     echo "nota maior ou igual a 5"
9
10 else
11
12     echo "nota menor que 5"
13
14 fi
```

Listagem 5: Usando operador lógico E

```
1 #!/bin/bash
2
3 a=3
4 b=2
5 c=1
6
7 # usando o operador E
8 if [ $a -gt $b ] && [ $a -gt $c ];
9     then
10         echo "A e' o maior"
11     else
12         echo "A nao e' o maior"
13     fi
14
15 # outra forma para usar o operador E
16 if [ $a -gt $b -a $a -gt $c ];
17     then
18         echo "A e' o maior"
19     else
20         echo "A nao e' o maior"
21     fi
```

2.3.2 Escolha...caso...

Listagem 6: Estrutura de decisão ESCOLHA

```
1 #!/bin/bash
2
3 echo -n "Entre com um numero de 1 a 5: "
4 read numero
5
6 case $numero in
7     1)
8         echo "Voce escolheu 1"
9         ;;
10    2)
11        echo "Voce escolheu 2"
12        ;;
13    3)
14        echo "Voce escolheu 3"
15        ;;
16    4 | 5)
17        echo "Voce escolheu 4 ou 5"
18        ;;
19    *)
20        echo "Voce escolheu um numero diferente de 1, 2, 3, 4 ou 5"
21        ;;
22 esac
```

2.4 Estruturas de repetição

2.4.1 Enquanto

Listagem 7: Estrutura de repetição ENQUANTO

```
1 #!/bin/bash
2
3 num=10
4
5 while [ $num -gt 0 ]; do
6     echo "contando $num"
7     num=$((num-1))
8 done
9
10
11 #-----#
12 #usando o operador de negação '!'
13
14 num=10
15
16 while ! [ $num -eq 0 ]; do
17     echo "contando $num"
18     num=$((num-1))
19 done
```

2.4.2 Para

Listagem 8: Estrutura de repetição PARA

```
1 #!/bin/bash
2
3 for contador in `seq 1 10`; do
4     echo $contador
5 done
6
7 # percorrendo uma lista de palavras separadas por espaco
8 lista="FIC Linux Redes IFSC"
9
10 for palavra in $lista; do
11     echo "Palavra $palavra"
12 done
13
14 # Listando todos os arquivos de um diretório
15 # lista=`ls -l`
16 for arquivo in `ls -l`; do
17     echo "Nome do arquivo: $arquivo"
18 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.