Programação para Redes de Computadores 17/03/2014

Aula 5 e 6: Introdução ao shell

Professor: Jorge H. B. Casagrande casagrande@ifsc.edu.br
Notas de aula adaptada da original do prof. Emerson R. de 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 até o 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 seqüência dentro um arquivo e chamar este arquivo sempre que necessário. E assim temos o *shell script* ilustrado pela Listagem 1.

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

Figura 1: Meu primeiro shell script

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 *seg* 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:

```
# executando o expr em um terminal
expr 2 + 2
# executando o expr em um terminal e guardando o resultado na varial 'soma'
soma=`expr 2 + 2` # expr entre crases
```

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

```
# executando o bc em um terminal, combinado com o echo
echo "scale=2; 1/2" | bc

# armazendo o resultado da saida do bc na variavel 'resultado'
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. A Listagem 2 ilustra algumas formas para atribuir e obter valores em variáveis.

```
#!/bin/bash
# Isto e' um comentario. Todo texto apos o caracter # sera' ignorado
echo "Trabalhando com variaveis"
a=1
b=2
c=`expr $a + $b` # a expressao esta' entre crases
d=$((c+a))
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"
curso="Aula de PRC"
echo "O conteudo de curso e' $curso"

# outro exemplo
versao=$(uname -r)

echo "A versao do kernel e' $versao"
```

Figura 2: Exemplo de definição e uso de variáveis

```
#!/bin/bash

echo -n "Entre com o seu nome: "
read nome
echo "Ola $nome!"
```

Figura 3: Usando variáveis em conjunto com o comando read

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 *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

```
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
```

Figura 4: Estrutura de decisão SE

2.3.2 Escolha... caso...

```
#!/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
```

Figura 5: Usando operador lógico E

```
#!/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
```

Figura 6: Estrutura de decisão ESCOLHA

2.4 Estruturas de repetição

2.4.1 Enquanto

```
#!/bin/bash

num=10
while [ $num -gt 0 ]; do
   echo "contando $num"
   num=$((num-1))
done
#---------------------------#
#usando o operador de negacao '!'
num=10
while ! [ $num -eq 0 ]; do
   echo "contando $num"
   num=$((num-1))
done
```

Figura 7: Estrutura de repetição ENQUANTO

2.4.2 Para

```
#!/bin/bash

# maneira tradicional do shell
for contador in `seq 1 10`; do
   echo $contador
done

# maneira semelhante a linguagem C e Java
for ((contador=1; contador < 11; contador++)); do
   echo $contador
done

# percorrendo uma lista de palavras separadas por espaco
lista="Cabeamento Gerencia Programacao Equipamentos"
for disciplina in $lista; do
   echo "Disciplina $disciplina"
done
# Dica: que tal listar os arquivos de um diretorio?
# lista=`ls` # o ls esta' entre crases
```

Figura 8: Estrutura de repetição PARA

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 três números inteiros e determine qual é o maior e o menor

6. Desenvolva um algoritmo que leia um número inteiro positivo e imprima a sequência de 0 até este número. O programa deverá tratar caso o usuário forneça um número menor que zero.

7. 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.

8. Desenvolva um algoritmo que leia um número inteiro e calcule seu fatorial.

9. Desenvolva um algoritmo que leia um número inteiro positivo e imprima todos os números do intervalo de 1 até este número.