

Condicionais

João Marcelo Uchôa de Alencar

Universidade Federal do Ceará - Quixadá

6 de Setembro de 2019

Estruturas de Controle

Enquanto outras linguagens testam **condições**, o *Shell* testa o código de retorno do comando que o segue.

- ▶ O código de retorno está associado à variável \$?
- ▶ Sucesso (verdadeiro) equivale à 0.
- ▶ Erro (falso) equivale à 1.

No caso de um `|`, é retornado uma vetor PIPESTATUS, cada posição equivale ao código de retorno do comando na posição equivalente do *pipe*.

- ▶ `${PIPESTATUS[0]}` é o código de retorno do primeiro comando.
- ▶ `${PIPESTATUS[*]}` exibe todo o vetor.

A estrutura if/then/else

```
if <comando>  
then  
    <comando1>  
    <comando2>  
    <...>  
else  
    <comando3>  
    <comando4>  
    <...>  
fi
```

Exemplo

Como seria um *script* chamado **numeroOuNao.sh** que recebe um parâmetro e afirma se o mesmo é um número ou não.

O comando *test*

Como fazer para analisar condições? $a > b$, $a = b$, etc?

```
$ test <expressao>
```

Sendo **expressao** a condição a ser testada.

Ao usar o **test** é bom colocar as variáveis entre aspas. Há diferença entre comparar a string vazia e uma variável não existente.

Exemplo

Como seria um *script* chamado *param.sh* que verifica se o usuário passou parâmetros ou não. Caso não tenha passado parâmetros, deve imprimir apenas o nome do *script*. Caso tenha passado parâmetros, deve imprimir todos.

Testando Arquivos

-r arquivo	Permissão de Leitura
-w arquivo	Permissão de Escrita
-x arquivo	Permissão de Gravação
-f arquivo	Arquivo Regular
-d arquivo	Diretório
-s arquivo	Tamanho maior que zero

Testando Cadeia de Caracteres

-z cad1	Se o tamanho é zero
-n cad1	Se o tamanho é diferente de zero
cad1 = cad2	Se são idênticas
cad1	Se é não nula

Testando Números

int1 -eq int2	Igualdade
int1 -ne int2	Desigualdade
int1 -gt int2	Maior
int1 -ge int2	Maior ou igual
int1 -lt int2	Menor que
int1 -le int2	Menor ou igual

Simplificando a escrita do *test*

No lugar de:

```
$ test <expressao>
```

Podemos escrever:

```
$ [ <expressao> ]
```

Importante notar que há um espaço entre os colchetes e a expressão.

Se houver um sinal de ! antes da expressão, ela está negada. Mas lembre, tudo separado por **espaços!!!**

O **e** lógico pode ser criado usando **-a** para separar as expressões.

O **ou** lógico pode ser criado usando **-o** para separar as expressões.

Você pode usar parênteses, mas eles devem ser precedidos da barra invertida para o shell não interpretá-los:

```
\( $a = $b -o $c = $d \) -a \( $a -gt $d \)
```

Usando o `&&` e o `||`

Executar o segundo comando apenas se o primeiro for sucesso:

```
comando1 && comando2
```

Executar o primeiro comando, e caso este falhe, executar o segundo:

```
comando1 || comando2
```

Exemplo

Vamos definir dois comandos:

- ▶ Caso um diretório chamado **novoDir** não exista, deve ser criado.
- ▶ Mover todos os arquivos *.txt* para um diretório chamados **textos** e em seguida compacte o diretório.

Testando Padrões

Você pode testar se uma cadeia casa com um padrão usando:

`[[expressão]]`

Sendo que expressão obedece:

<code>cadeia == padrão</code>	Casamento
<code>cadeia != padrão</code>	Incompatibilidade
<code>cadeia1 < cadeia2</code>	Ordem alfabética
<code>cadeia1 > cadeia2</code>	Ordem alfabética
<code>expr1 && expr2</code>	Lógica
<code>expr1 expr2</code>	Lógica

Expressões Regulares

Além de padrões simples, podemos casar expressões regulares:

```
[[ $cadeia =~ expressaoregular ]]
```

Se a expressão regular usar parênteses, o vetor BASH_REMATCH armazena as subcadeias que casaram com cada subexpressão.

- ▶ \${BASH_REMATCH[0]} é cadeia inteira
- ▶ \${BASH_REMATCH[1]} é a subcadeia que casou com a subexpressão no primeiro parênteses.

No caso das expressões, os parênteses não precisam ser escritos precedidos por barras.

```
$ num=100
```

```
$ [[ $num =~ ([0-9])[0-9][0-9] ]]
```

Exemplo

Escrever um comando que preencha o vetor com os octetos de um endereço IP.

Por exemplo, para o IP 200.19.190.1:

`${BASH_REMATCH[1]}` = 200

`${BASH_REMATCH[2]}` = 19

`${BASH_REMATCH[3]}` = 190

`${BASH_REMATCH[4]}` = 1

O comando case

```
case <valor> in
    padr1)
        <comando1>
        <comando2>
        <...>
        ;;
    padr2)
        <comando1>
        <comando2>
        <...>
        ;;
    ...
    padrN)
        <comando1>
        <comando2>
        <...>
        ;;
*)
    <comando1>
    <comando2>
    <...>
esac
```

Os padrões só usam um conjunto básico de meta-caracteres, não são expressões regulares completas.

Atividade - Parte I

Na pasta *atividades/atividade05*, faça um *script* chamado *isfile.sh* que receba um parâmetro e verifique se é o nome de um arquivo ou diretório e informe se você tem permissão de escrita e leitura. Por exemplo:

```
$ ./testFile.sh /etc/hosts
```

É um arquivo.

Tem permissão de leitura.

Não tem permissão de escrita.

Atividade - Parte II

Na pasta *atividades/atividade05*, faça um *script* chamado *maiorDe3.sh* que receba três números como parâmetros e retorne o maior. Não pode utilizar o comando **sort**. Por exemplo:

```
$ ./maiorDe3.sh 4 6 5  
6
```

Atividade - Parte III

Na pasta *atividades/atividade05*, faça um *script* chamado *maiorDe3Verificado.sh* que receba três números como parâmetros e retorne o maior. Não pode utilizar o comando **sort**. Esse *script* tem que reclamar caso um dos parâmetros não seja número. Por exemplo:

```
$ ./maiorDe3Verificado.sh 4 6 5
```

```
6
```

```
$ ./maiorDe3Verificado.sh casa 10 11
```

```
Opa!!! casa não é número.
```

Atividade - Parte IV

Na pasta *atividades/atividade05*, faça um *script* chamado *infoDir.sh* que receba como um parâmetro um caminho. Se for um diretório, deve informar o tamanho do mesmo e quantos arquivos ou subdiretórios possui. Caso contrário, deve exibir uma mensagem reclamando.

```
$ ./inforDir.sh /etc
```

```
0 diretório /etc ocupa 1035 kilobytes e tem 45 itens.
```

```
$ ./inforDir.sh /etc/passwd
```

```
/etc/passwd não é um diretório!!!
```

Atividade - Parte V

De volta ao diretório *atividades/atividade05*, escreva um *script* chamado **servico.sh**. Esse script deve receber um parâmetro.

- ▶ Se o parâmetro for *start*, deve imprimir “Iniciando Serviço”.
- ▶ Se o parâmetro for *stop*, deve imprimir “Parando Serviço”.
- ▶ Se o parâmetro for *restart*, deve imprimir “Reiniciando Serviço”.
- ▶ Se o parâmetro for qualquer outra coisa, deve imprimir “Uso: servico.sh (start|stop|restart) ”.