

Introdução ao Shell Script

Alexandre Mulatinho

<http://mulat0z.blogspot.com>
alex@mulatinho.net

15 de agosto de 2009

- Shell e shell-script são termos diferentes.

Introdução

Visão dos principais conceitos

- Shell e shell-script são termos diferentes.
- Shell é um software criado para interagir entre o usuário e o sistema operacional, aceitando comandos digitados pelo usuário e os interpretando/transformando numa forma em que o PC possa entender.

Introdução

Visão dos principais conceitos

- Shell e shell-script são termos diferentes.
- Shell é um software criado para interagir entre o usuário e o sistema operacional, aceitando comandos digitados pelo usuário e os interpretando/transformando numa forma em que o PC possa entender.
- Shell-script é um conjunto de palavras específicas que fazem com que a shell transforme-se numa linguagem de programação interpretada. Exemplos dessas palavras chaves são: IF, WHILE, CASE, FOR, etc.

- Shell e shell-script são termos diferentes.
- Shell é um software criado para interagir entre o usuário e o sistema operacional, aceitando comandos digitados pelo usuário e os interpretando/transformando numa forma em que o PC possa entender.
- Shell-script é um conjunto de palavras específicas que fazem com que a shell transforme-se numa linguagem de programação interpretada. Exemplos dessas palavras chaves são: IF, WHILE, CASE, FOR, etc.
- Há varias shell's diferentes como: SH, BASH, CSH, KSH, etc. A que usaremos é a mais comum nos dias atuais, o BASH. Conhecida como Bourne-Again Shell criada por Brian Fox e Chet Ramey da Free Software Foundation.
- Apesar de ser parte importante no Unix, a shell não faz parte do código interno do Kernel.

Diferenças entre tipos de linguagens.

Linguagem interpretada e compilada.

- Linguagem interpretada é aquela que não precisa ser transformada em binário de forma que o computador possa entender cada linha do programa, ela é processada por um software, no nosso caso a shell, e aí sim transformada numa forma em que o PC entenda.

Diferenças entre tipos de linguagens.

Linguagem interpretada e compilada.

- Linguagem interpretada é aquela que não precisa ser transformada em binário de forma que o computador possa entender cada linha do programa, ela é processada por um software, no nosso caso a shell, e aí sim transformada numa forma em que o PC entenda.
- Linguagem compilada é aquela que é lida linha por linha por um software compilador que aí então a transforma num binário para ser executado e entendido diretamente pelo sistema operacional.

Diferenças entre tipos de linguagens.

Linguagem interpretada e compilada.

- Linguagem interpretada é aquela que não precisa ser transformada em binário de forma que o computador possa entender cada linha do programa, ela é processada por um software, no nosso caso a shell, e aí sim transformada numa forma em que o PC entenda.
- Exemplos: Python, Shell-Script, PHP, Perl, etc.
- Linguagem compilada é aquela que é lida linha por linha por um software compilador que aí então a transforma num binário para ser executado e entendido diretamente pelo sistema operacional.
- Exemplos: Assembly, C, Pascal, etc.

- Todo programa a ser rodado pela shell deve ter permissão de 'execução' (modo +x) ou caso seja um script ser chamado por uma das shell's do sistema operacional. (Ex.: bash script.sh)
- A shell interage no meio, servindo figuramente como uma ponte entre o usuário e o sistema operacional, ela não só é responsável por executar programas mas também por interpretar uma série de comandos e 'coringas' em conjunto. (Ex: ls a* && xargs rm -f)
- A shell apenas utiliza **os comandos que foram escritos pro Unix** e os processa em conjunto com o que o usuário deseja de um modo que ele seja compreendido pelo sistema operacional.

- A shell retorna 0, verdadeiro, caso um programa seja executado com sucesso e diferente de 0, falso, caso tenha havido algum problema ou erro.
- A shell difere o uso de ' (aspas simples), "(aspas dupla) e ' (crase) no uso da linha de comando e especialmente em variáveis: aspas simples para indicar textos puros, aspas duplas para indicar textos complexos com ou sem variáveis e a crase para execução de comandos em processos filhos.
- Todo script shell precisa de um interpretador para os seus comandos, o que afirma que um script shell não depende do sufixo '.sh' para ser considerado um shell.

- Um interpretador é um software que será responsável por lêr todas as linhas de código do script e processá-las.

- Um interpretador é um software que será responsável por lêr todas as linhas de código do script e processá-las.
- Na linguagem dos programadores **she-bang** significa os caracteres `#!` que são responsáveis por informar a shell qual o interpretador ela deve utilizar para processar o código, este deve vir na primeira linha do script.

- Um interpretador é um software que será responsável por lêr todas as linhas de código do script e processá-las.
- Na linguagem dos programadores **she-bang** significa os caracteres **#!** que são responsáveis por informar a shell qual o interpretador ela deve utilizar para processar o código, este deve vir na primeira linha do script.
- Por padrão não é necessário colocar o she-bang em um script shell, porém é essencial informar ao script qual interpretador shell ou software ele deve utilizar para processar o código, **senão será utilizado o shell atual.**

- A shell por padrão interpreta expressões regulares e coringas/escapes.

- A shell por padrão interpreta expressões regulares e coringas/escapes.
- Um coringa geralmente é descrito por um caracter não alfa-numérico.

- A shell por padrão interpreta expressões regulares e coringas/escapes.
- Um coringa geralmente é descrito por um caracter não alfa-numérico.
- Os coringas mais conhecidos são: *, [], {}, ' e ". Porém há coringas bem mais complexos que veremos mais a frente e que são extremamente úteis.

- A shell por padrão interpreta expressões regulares e coringas/escapes.
- Um coringa geralmente é descrito por um caracter não alfa-numérico.
- Os coringas mais conhecidos são: *, [], {}, ' e ". Porém há coringas bem mais complexos que veremos mais a frente e que são extremamente úteis.

Exemplos

```
$ ls
```

```
alex.txt alessandra.txt adalberto.txt flavia.txt
```

```
$ ls al*
```

- A shell por padrão interpreta expressões regulares e coringas/escapes.
- Um coringa geralmente é descrito por um caracter não alfa-numérico.
- Os coringas mais conhecidos são: *, [], {}, ' e ". Porém há coringas bem mais complexos que veremos mais a frente e que são extremamente úteis.

Exemplos

```
$ ls  
alex.txt alessandra.txt adalberto.txt flavia.txt  
$ ls al*  
alex.txt alessandra.txt  
$ ls *{a,o}.txt
```

- A shell por padrão interpreta expressões regulares e coringas/escapes.
- Um coringa geralmente é descrito por um caracter não alfa-numérico.
- Os coringas mais conhecidos são: *, [], {}, ' e ". Porém há coringas bem mais complexos que veremos mais a frente e que são extremamente úteis.

Exemplos

```
$ ls  
alex.txt alessandra.txt adalberto.txt flavia.txt  
$ ls al*  
alex.txt alessandra.txt  
$ ls *{a,o}.txt  
alessandra.txt adalberto.txt
```

- Operadores são responsáveis pelas atribuições de condições a execução, controle de processos, redirecionamento de descritores, operadores de testes e contas aritméticas em geral. Resumindo, são muitos operadores mas todos são extremamente úteis.

Operadores condicionais de execução

Operador	Função Principal
&&	AND condicional. Executa o comando da esquerda, e se verdadeiro, executa o da direita.
	OR condicional. Executa o comando da direita apenas se o comando da esquerda não retornar verdadeiro.

Exemplos

Operadores condicionais de execução

Operador	Função Principal
&&	AND condicional. Executa o comando da esquerda, e se verdadeiro, executa o da direita.
	OR condicional. Executa o comando da direita apenas se o comando da esquerda não retornar verdadeiro.

Exemplos

```
$ ls
```

```
auditores.txt advogados.txt clientes.txt funcionarios.txt
```

```
$ ls a* && rm -f a* && ls
```

Operadores condicionais de execução

Operador	Função Principal
&&	AND condicional. Executa o comando da esquerda, e se verdadeiro, executa o da direita.
	OR condicional. Executa o comando da direita apenas se o comando da esquerda não retornar verdadeiro.

Exemplos

```
$ ls  
auditores.txt advogados.txt clientes.txt funcionarios.txt  
$ ls a* && rm -f a* && ls clientes.txt funcionarios.txt  
$ ls a* || rm -f {c,f}*  
$ ls
```

Operadores de controle

Operador	Função Principal
&	Envia jobs para background.
	Envia saída da esquerda pra direita.

Exemplos

```
$ ./meu_server &  
$ ps ax | less
```

Operadores de redirecionamento

Operador	Função Principal
>	Redireciona a saída ao descritor da direita, Sobrescrevendo o descritor caso haja algum.
<	Redireciona a entrada ao descritor da esquerda.
«	Redireciona a entrada ao descritor da esquerda.
»	Redireciona a entrada ao descritor da direita, Adicionando os dados ao final, sem apagar o descritor.

Exemplos

```
$ echo "E ae meu querido»" > saudacao.txt
```

```
$ echo "Tudo Beleza?»" >> saudacao.txt
```

```
$ tail -n 1 < saudacao.txt
```

Operadores de Testes

Op1	Op2	Função Principal
-gt	>	Maior que. 7 é maior que 5.
-ge	>=	Maior ou igual. 7 é Maior ou igual a 7.
-lt	<	Menor que.
-le	<=	Menor ou igual.
-eq	=	Igual a. 8 é igual a 8.
-ne	!=	Não igual a. 8 não é igual a 7.
Opção	Significado do teste	
-f	testa se arquivo existe	
-x	testa se arquivo é executavel	
-d	testa se existe diretorio	

Operadores Aritméticos

Operador	Função Principal
+	Soma. $5 + 7$.
-	Subtração. $7 - 2$.
*	Multiplicação. $5 * 9$.
/	Divisão. $15 / 5$.
%	Módulo. $12 \% 2$.

```
$ expr 5 - 2
```

```
3
```

```
$ expr 8 * 4
```

```
expr: syntax error
```

```
$ echo $((5*20))
```

```
100
```

Variáveis

Criando e modificando variáveis simples e complexas

- Na shell variáveis são alfa-numéricas (letras e números). Então não há uma pré-definição do que a variável irá se tornar, ela pode ser qualquer coisa, obviamente fazer contas com variáveis alfas acarretam em problemas.
- Podemos atribuir à variáveis não só outras variáveis como também saída de comandos simples e complexos.
- As variáveis também tem coringas especiais que fazem manipulação de strings.

Variáveis

Criando e modificando variáveis simples e complexas

Algumas variáveis especiais da shell

Variável	O que significa
HOME	Indica a pasta do usuário atual.
SHELL	Indica a shell que o usuário está usando.
PWD	Indica a pasta em que o usuário se encontra.
PATH	Pastas as quais o usuário tem acesso aos softwares.

```
$ echo $SHELL  
/bin/bash
```

Para ver mais variáveis do sistema basta digitar na shell o comando **env** que mostra as variáveis locais e o comando **export** que mostra as variáveis globais.

Variáveis

Criando e modificando variáveis simples e complexas

```
$ VAR=Hello World
```

```
$ echo $VAR
```

```
Hello
```

```
$ TESTE = Hello World
```

```
bash: VAR: command not found
```

```
$ echo TESTE
```

Variáveis

Criando e modificando variáveis simples e complexas

```
$ VAR='Hello World'
```

```
$ echo $VAR
```

```
Hello World
```

```
$ VAR2=456
```

```
$ echo $VAR2
```

```
456
```

```
$ NOVAVAR='VAR VAR2' && echo $NOVAVAR
```

```
VAR VAR2
```

```
$ NOVAVAR='$VAR $VAR2' && echo $NOVAVAR
```

```
$VAR $VAR2
```

```
$ NOVAVAR="$VAR $VAR2" && echo $NOVAVAR
```

```
Hello World 456
```

Variáveis

Criando e modificando variáveis simples e complexas

```
$ A=4 && B=5
```

```
$ echo $A; echo $B
```

```
4
```

```
5
```

```
$ VAR='echo $PWD | od -x' && echo $VAR
```

```
0000000 682f 6d6f 2f65 7263 6475 2f6f 6574 7473
```

```
0000020 000a
```

```
0000021
```

```
$ VARJOIA="'hostname' é a maquina do $LOGNAME"
```

```
$ echo $VARJOIA
```

```
ininitus.home é a maquina do crudo
```

Exercícios

1. Digitando o comando `env` procure a variável que informa sua pasta, após isso crie uma variavel com o nome 'VAR' que diga exatamente isso: "**Minha pasta atual é: /home/usuario**" onde o diretório será substituído pela variavel que vc encontrou no `env`.
2. Crie uma nova variavel com nome qualquer que contenha a quantidade de caracteres da sua pasta inicial, para isso utilize o comando `wc -c` que conta o numero de caracteres de um texto, lembre-se de usar o operador que passa um comando da direita (que sera a impressao da sua pasta) para o comando da esquerda (que sera o `wc -c`)

Laços Condicionais

O IF-THEN-ELSE no Shell-Script

- Deve-se sempre pensar no algoritmo em si, abstrair-se do mundo real e pensar como um computador, IF = se, ELIF = ainda se, ELSE = senão, evite utilizar IF's desnecessários.
- Deve-se sempre primar pelo código perfeito, utilize os operadores lógicos quando achar que deve testar mais de uma condição ao mesmo tempo.

```
Se, usuario tiver menos que 18 anos,  
    entao ele não poderá entrar nessa opção.  
Senão, se for ao contrário,  
    entre com a opção desejada.  
Fim-se
```

Laços Condicionais

O IF-THEN-ELSE no Shell-Script

```
if [ teste ] ; then
    execute isto..
fi
```

```
if (( teste mais aritmetico ))
then
    execute isto filhote
fi
```

```
if [ $user_age -lt 18 ] ; then
    echo Voce nao tem idade suficiente
    exit
else
    echo Ja es um homem!
    do_option;
fi
```

```
if [ $user_age -lt 18 ] ; then
    echo És muito jovem para isto!
    exit
elif (( $user_age >= 60 )) ; then
    echo Já não passou a tua idade?
    exit
else
    echo Estas na faixa certo
    continua_programa;
fi
```

Exercícios

1. O que tem de errado com o exemplo anterior?
2. Crie duas variáveis com valores aleatorios que serão idades, e faça condições usando IF-THEN-ELSE comparando a idade de um com o outro e diga quem é mais velho e quem é o mais novo.

- A palavra **case** trata variáveis de forma bastante semelhante ao IF-THEN-ELSE porém separando em casos, como o próprio nome diz, a sintaxe é simples mas deve-se ter bastante atenção ao separador de opções.

```
case $VARIABEL in
  OPCA01)
    Faça algo
    ;;
  OPCA02)
    Faça outra coisa
    ;;
  *)
    Já que não fez nada, faça isso
esac
```

Laços Condicionais

O CASE no Shell-Script

```
SEX=Homem
case $SEX in
  Homem)
    echo Sexo do individuo é Masculino
    ;;
  Mulher)
    echo Sexo do individuo é Feminino
    ;;
  *)
    echo Sexo do indiviudo é Desconhecido,
    echo Será que está indeciso?
esac
```

Loops no shell script

O **WHILE** faz algo até que uma condição seja atingida

- A palavra **while** testa uma condição até que ela se torne verdadeira, enquanto a condição for falsa o loop executará o que estiver dentro do bloco de instrução criada pelo programador.

```
Enquanto, a variavel ou o teste nao for o que eu desejo,  
  faça isso e depois aquilo  
  faça mais aquilo e depois isso  
Volte pro Enquanto.
```

Loops no shell script

O **WHILE** faz algo até que uma condição seja atingida

```
i=0
while [ $i -lt 4 ] ; do
    echo "loop numero: $i"
    i=$((i+1))
done
```

```
VAR=0
while [ $VAR -ne 2 ] ; do
    echo 0 arquivo nao tem 2 linhas
    echo "teste" >> arquivo.txt
    VAR=`wc -l arquivo.txt | cut -d' ' -f1`
done
```

Loops no shell script

Com o **FOR** não tem aperreio, ou voce controla ou deixa se controlar

- A palavra **for** pode criar uma variavel no bloco de código ou utilizar uma variável anterior para começar o loop, esta fica localizada no seu primeiro argumento, o segundo argumento é uma condição que será testada até que se torne verdadeira (enquanto não se tornar o loop continua), e por fim o último argumento indica geralmente uma iteração ou execução dependendo da função do loop.

```
a cada X que esta dentro do GRUPOX; faça  
alguma coisa em especifica  
alguma coisa especial  
volte ao inicio ateh que o X chegue ao fim
```

Loops no shell script

Com o **FOR** não tem apertinho, ou voce controla ou deixa se controlar

```
VARIAVEL="/home /etc /tmp"
LOOP=1
for x in $VARIABLE; do
    echo "loop numero:$LOOP e o valor: $x"
    echo "o tamanho de $x é: 'du -hs $x'"
    LOOP=$((LOOP+1))
done
```

```
for (( i=0;$i<5;i=$((i+1)) )); do
    echo $i
done
```

Loops no shell script

A hora da verdade: exercicios..

1. Escreva um programa que utilize o while e incremente uma variavel de 0 a 10 imprimindo seu resultado.
2. Escreva um programa com o for que processe a variavel `VAR="/var /home"` e copie seus dados (`cp -Rf`) para o diretorio de backup `/tmp`.

- Parâmetros são na verdade palavras/argumentos que colocamos depois de uma chamada de uma script, isso é bem usado quando um script precisa receber um dado para ser executado.

Exemplo de uso de parametros

```
$ cat script.sh
#!/bin/bash
echo arg0: $0
echo arg1: $1
echo arg2: $2 arg3: $3
```

- Parâmetros são na verdade palavras/argumentos que colocamos depois de uma chamada de uma script, isso é bem usado quando um script precisa receber um dado para ser executado.

Exemplo de uso de parametros

```
$ ./script.sh Alex
echo arg0: script.sh
echo arg1: Alex
echo arg2: arg3:
$ ./script.sh Alex Mostrando Shell-Script
echo arg0: script.sh
echo arg1: Alex
echo arg2: Mostrando arg3: Shell-Script
```

- Status de saída é o que diz ao script se ele foi executado com sucesso, com problemas ou com erro total.

Tabela simples de status de saída

Valor: \$?	Significado
0	Sucesso, verdadeiro.
1	Maioria dos erros comuns dos programas.
126	Sem permissão de execução.
127	Comando não encontrado.

```
$ ls > /dev/null; echo $?
```

```
0
```

```
$ lsss > /dev/null; echo $?
```

```
127
```

- Deve-se ter cuidado antes de fazer scripts pois como são programas executáveis podem danificar o sistema se houver falhas ou erros humanos.
- Deve-se dobrar o cuidado quando scripts forem executados como root (super-usuário) pois os mesmos poderão executar em todo o sistema.

Erro humano

```
#!/bin/bash
DIRS="/tmp /home/zigfrido / pst"
for i in $DIRS; do
    echo executando limpeza em "$i"..
    rm -f $i/*
done
```

Exercícios

1. Crie um script que receba como argumento dois numeros faça um loop nas quatro operações basicas da aritmética (soma, subtração, divisao e multiplicação) e teste o valor total com o segundo numero passado como argumento para ver se ele é maior ou menor e tambem se eh igual ou diferente

2. Faça um script shell que reuna as seguintes informações:

- Nome da maquina (hostname) e Endereço IP (ifconfig).
- Processador (/proc/cpuinfo) e memória da máquina (/proc/meminfo)
- Espaço livre ocupado das partições (df -Th).
- Lista de dispositivos PCI (lspci)

Gere um arquivo que contenha o nome

"Relatorio-MES-DIA.txt" e guarde todos os dados acima para