

Básico de Shell Script

I Ultra-Maratona How To de Software Livre

Carlos Ferreira

19 de Julho de 2008

Agenda

- Onde está o Shell ?
- Vantagens e Desvantagens
- Atribuição ou Comando ?
- Tipos de Shell
- Redirecionamentos de Saída, Entrada e Especiais
- Coisas estranhas
- Manipulando arquivos
- Tratamento de cadeias
- Instruções condicionais
- Instruções de laço
- Lendo no Shell
- Funções
- Alguns Comandos
- Créditos Finais

Onde está o Shell ?



Você está aqui

Vantagens e Desvantagens

Linguagem
interpretada
(produtividade)

Nativa do ***n?x | *bsd**
(portabilidade)

Visibilidade do
ambiente
(integração)

Linguagem
interpretada
(performance)

Não acessa ao
hardware (como o C)

Não tem acesso às
GUIs (Graphical User
Interface)

Vantagens e Desvantagens

Linguagem
interpretada
(produtividade)

Nativa do *n?x | *bsd
(portabilidade)

Visibilidade do
ambiente
(integração)

Linguagem
interpretada
(performance)

Não acessa ao
hardware (como o C)

Não tem acesso às
GUIs (Graphical User
Interface)

Atribuição ou Comando ?

- **Atribuição:**

- Nome da variável à esquerda do sinal de igual, formado por letras, números, sublinha (_) e não pode começar por número;
- O Shell não pode ver nenhum espaço em branco em uma atribuição;

Exemplos:

```
Prompt> Var= .....Certo
Prompt> Var=5 .....Certo
Prompt> Var = 5 .....Errado
```

Atribuição ou Comando ?

- **Comando:**

- Os comandos são separados das suas opções e argumentos por espaços em branco;

Exemplos:

```
Prompt> echo $Var
```

```
5
```

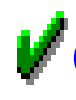
```
Prompt> who ..... Sem opção nem argumento
```

```
Prompt> who -H ..... Só com opção
```

```
Prompt> who am I ..... Só com argumento
```

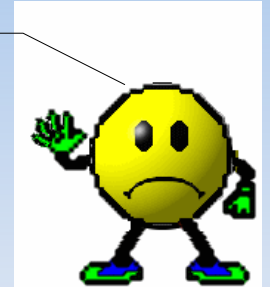
```
Prompt> who am I -H. . Com opção e argumento
```

Atribuição ou Comando ?

- Comando `grep -i $CADEIA * > progs`
 - Identifica o nome do programa `grep`
 - Pesquisa sua existência (PATH) e permissões  OK
 - Identifica opções/parâmetros `-i | $CADEIA *`
 - Identifica redirecionamentos `>`
 - Identifica variáveis `$CADEIA`

Atribuição ou Comando ?

Você devia ter feito:
`sort arq -o arq`



■ Comando:

- Resolução de redirecionamentos: `sort arq > arq`
- Substituição de variáveis: `echo $LOGNAME`
- Substituição de meta caracteres: `echo *`
- Substituição de aliases: `rm='rm -i`

Muito importante

- **Resolve os redirecionamentos;**
- **Substitui as variáveis pelos seus valores;**
- **Resolve e substitui os meta caracteres;**
- **Passa a linha já toda esmiuçada para execução.**

Tipos de Shell

Os principais Shells são:

- sh, bash, ksh
- **Efetivamente o Shell é o programa associado ao último campo de /etc/passwd**

```
$ grep carlos /etc/passwd
```

```
carlos:x:54002:101:::/home/carlos:/bin/bash
```

```
$ grep daemon /etc/passwd
```

```
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
```

Redirecionamentos de Saída

- > Redireciona a saída de um comando para um arquivo, destruindo seu conteúdo.
- >> Redireciona a saída de um comando para um arquivo, mantendo o conteúdo intacto (anexa ao fim do arquivo).
- 2> Redireciona a saída de erros para um arquivo, destruindo seu conteúdo.

```
$ ls JaEra
```

```
ls: JaEra: No such file or directory
```

```
$ ls JaEra 2> ArqErr
```

```
$ cat ArqErr
```

```
ls: JaEra: No such file or directory
```

Redirecionamentos de Entrada

- < Avisa ao Shell que a entrada não será feita pelo teclado, mas sim por um arquivo

```
$ mail usuario < ArqMala
```

- << Indica ao Shell que o escopo do comando começa na linha seguinte e termina quando encontrar um rótulo (*label*) definido.

```
$ mail carlos << FimMail
```

```
> SP, `date`
```

```
> Hoje o conteudo do directorio era `ls -l`
```

```
> FimMail
```

Prompts de continuação (PS2)

Redirecionamentos Especiais

- | Passa a saída de um comando para a entrada de outro. Conhecido como “*pipe*”.
- tee** Passa a saída de um comando para a saída padrão (tela) e também para um arquivo.

```
$ ls c* | tee arq.tee
```

```
calculadora.sh
```

```
cores.sh
```

```
cripta.sed
```

```
$ cat arq.tee
```

```
calculadora.sh
```

```
cores.sh
```

```
cripta.sed
```

Coisas Estranhas

Caracteres usados para impedir interpretação do Shell

- Aspas (") – Entre elas, o Shell "vê" as crases, contrabarras e cifrão.
- Apóstrofes ou plicas (') (normalmente abaixo das aspas no teclado)
- Contrabarra (\) - "Esconde" o caractere seguinte

```
$ alias rm='rm -i'
$ > arq
$ ls arq
arq
$ rm arq
rm: apagar arquivo comum vazio `arq'? n
$ \rm arq
$ ls arq
ls: arq: No such file or directory
```

Coisas Estranhas

O acento grave, que chamamos de crase (`) e no teclado normalmente fica ao lado do "P" (com *shift*), serve para dar precedência na execução de um comando. Este tipo de construção vem sendo substituída por outra da forma `$(cmd)`.

```
$ echo "Existem `who | wc -l` usuários conectados"
```

```
$ echo "Existem $(who | wc -l) usuários conectados"
```

Manipulando Arquivos

- Simples
- Imediato
- Eficiente
- Principais ferramentas:
 - redirecionamentos
 - `sed`
 - `awk`
 - `fgrep`; `grep`; `egrep` (ou `grep -e`)
 - `find`

Manipulando Arquivos

- -d Se for um diretório
- -e Se existir
- -z Se estiver vazio
- -f Se contiver texto
- -o Se o usuário for o dono
- -r Se o arquivo pode ser lido
- -w Se o arquivo pode ser alterado
- -x Se o arquivo pode ser executado

Variáveis Especiais

- \$0 Nome do script que está sendo executado
- \$1-\$9 Parâmetros passados à linha de comando
- \$# Número de parâmetros passados
- \$? Valor de retorno do último comando ou de todo o shell script. (o comando "exit 1" retorna o valor 1)
- \$\$ Número do PID (Process ID)

Tratamento de Cadeias

- `cut` – Corta um pedaço ou um campo de uma cadeia
- `paste` – O oposto do `cut`. Cola cadeias de caracteres
- `expr`
 - `length` – Devolve o tamanho de uma cadeia
 - `substr` – Extrai uma subcadeia de uma cadeia
 - `index` – Devolve a posição de uma subcadeia
- `tr` – Transforma `<dos-caracteres>` `<para-os-caracteres>`

```
$ echo sou lindo e modesto | tr a-z A-Z
```

```
$ who # Este comando lista usuários logados
```

```
$ who | cut -f1 -d" " | sort | uniq
```

```
$ who | cut -f1 -d" "
```

Instruções Condicionais

Os quatro mandamentos do `if`:

- Não testa condições
- Testa instruções
- O comando `test` testa condições
- Para testar condições use o `if` junto com o `test`

```
$ if ls -ld diret | grep diret; then
>     cd diret
> else
>     mkdir diret
>     cd diret
> fi
$ test -d diret
$ echo $?
```

Instruções Condicionais

- O comando `test` pode ser substituído por um par de colchetes (`[]`), separados por espaços em branco dos argumentos.
- Aumentará enormemente a legibilidade

```
$ if [ -d diret ]; then  
>     cd diret  
> else  
>     mkdir diret  
>     cd diret  
> fi
```

Instruções de Laço

- O comando `case` suporta uso de metacaracteres:

```
case var in
```

```
    padr1) comandos ; ;
```

```
    padrn) comandos ; ;
```

```
    *) comandos ; ;
```

```
esac
```

Comparações Numéricas

[**\$var1 -eq \$var2**] Verdadeiro se os valores forem (numericamente) iguais.

[**\$var1 -ne \$var2**] Verdadeiro se os valores forem diferentes.

[**\$var1 -gt \$var2**] Verdadeiro se o valor \$var1 fôr maior do que o valor \$var2.

[**\$var1 -ge \$var2**] Verdadeiro se o valor \$var1 fôr maior ou igual do que o valor \$var2.

[**\$var1 -lt \$var2**] Verdadeiro se o valor \$var1 fôr menor do que o valor \$var2.

[**\$var1 -le \$var2**] Verdadeiro se o valor \$var1 fôr menor ou igual do que o valor \$var2.

Comparações Numéricas

cadeia == padrão

cadeia1 = padrao

Verdadeiro se cadeia1 casa com padrão

cadeia1 != padrao
padrao.

Verdadeiro se cadeia1 não casa com

cadeia1 < cadeia1

Verdadeiro se cadeia1 vem antes de

cadeia1 alfabeticamente.

cadeia1 > cadeia1

Verdadeiro se cadeia1 vem depois de

cadeia1 alfabeticamente

expr1 && expr2

"E" lógico, verdadeiro se ambos expr1 e

expr2 são verdadeiros

Instruções de Laço

- Comando `for`

Recebe cadeias de caractere como parâmetro

- Comando `while`

Assim como o `if`, este comando testa instruções

- Comando `until`

Testa instruções também

Lendo no Shell

- Posicionamento
 - Comando `tput`
- Leitura na tela
 - Comando `read`
- Leitura de arquivos
 - Comando `read`
- Otimizando a captura dos dados
 - Formas otimizadas do `read`

Lendo no Shell

- Posicionamento

Comando tput	Ação
<code>tput cup 0 0</code>	Posiciona na linha 0 coluna 0
<code>tput lines</code>	Devolve a quantidade de linhas
<code>tput cols</code>	Devolve a quantidade de colunas
<code>tput bold</code>	Coloca o terminal em modo de ênfase
<code>tput sgr0</code>	Coloca o terminal no modo <i>default</i>
<code>tput el</code>	Apaga a linha a partir do cursor
<code>tput ed</code>	Apaga a tela a partir do cursor

Lendo no Shell

- Leitura na tela
 - Comando `read`

```
$ read var1 var2 var3
```

```
Escrevi uma frase!
```

```
$ echo -e "$var1\n$var2\n$var3"
```

```
Escrevi
```

```
uma
```

```
frase!
```

Lendo no Shell

Leitura de arquivos

- Comando `read`

```
# 2º Processo
$ cat teste_read.txt | while read linha
> do
> echo Linha=$linha
> done
```

Lendo no Shell

- Otimizando a captura dos dados
 - Formas otimizadas do comando `read`
 - `read -p` # Prompt
 - `read -n` # Número de caracteres
 - `read -t` # Tempo
 - `read -s` # Silenciosamente

Lendo no Shell

- Otimizando a captura dos dados
 - Formas otimizadas do comando `read`

- `read -p` # Prompt

```
$ read -p "Digite seu nome: " Nom  
Digite seu nome: Botelho  
$ echo O nome digitado foi: $Nom  
O nome digitado foi: Botelho
```

Lendo no Shell

- Otimizando a captura dos dados
 - Formas otimizadas do comando `read`
 - `read -n` # Número de caracteres

```
$ read -n5 -p"Digite 5 numeros: " Num  
Digite 5 nnumeros: 12345  
$ echo Os numeros digitados foram: $Num  
Os numeros digitados foram: 12345
```

Lendo no Shell

- Otimizando a captura dos dados
 - Formas otimizadas do comando `read`

- `read -t` # Tempo

```
if read -t 2 -p "Digite seu nome: " Opc
; then
    echo Seu nome e: $Opc
else
    echo "Voce é muito mole"
    exit 1
fi
```

Funções

- para poupar tempo de programação

```
function nome_da_funcao ()  
{  
    código da função  
}
```

```
funcao_teste ()  
{  
    echo Parabens pela funcao  
}
```

Alguns Comandos

Comando eval

Este comando é usado quando se deseja dar duas passadas em uma instrução a ele associada. Na 1ª a instrução é resolvida e na 2ª é executada.

```
$ var1=3
$ var2=var1
$ echo `$echo $var2`
$var1
$ eval echo `$echo $var2`
3
```

Alguns Comandos

Comando trap

Este comando é usado para monitorar sinais capturados pelo Shell.

Principais sinais:

- **0** – Fim normal
- **2** – ctrl + c

```
trap "rm -f /tmp/lixo ; exit" 0 1 2 3 15
```

Alguns Comandos

Comando `source`(ou `.`)

Este comando é usado para executar um *script* no shell corrente, isto é, não é criado um subshell (ou shell filho) para sua execução.

```
$ cat script_source  
cd  
ls  
$ pwd  
$ script_source  
$ pwd  
$ source script_source  
$ pwd
```

Créditos Finais

- Grupo SL-RJ
- Já fizemos o DFD(março/2008) e o FLISOL RJ(abril/2008)
- GNUGRAF(Agosto/2008) - www.gnugraf.org
- Apoio a Ultra Maratona How To de Software Livre(Julho/2008) e a PyConBrasil 2008(setembro/2008)
- Estamos preparando um evento maior chamado Hack'nRio(abril/2009)



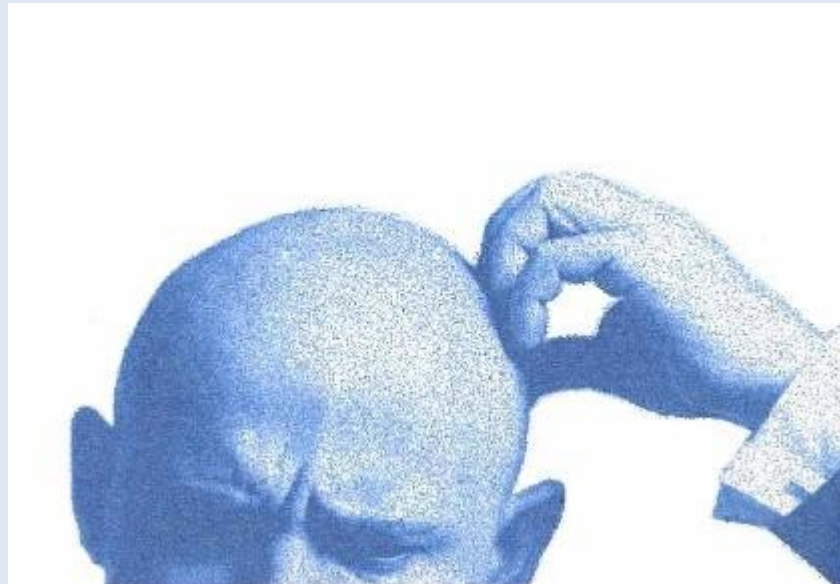
Dúvidas ?

Lista de Shell -

<http://br.groups.yahoo.com/group/shell-script>

Site do Julio Neves -

<http://twiki.softwarelivre.org/bin/view/TWikiBar/WebHome>



Obrigado



Carlos Ferreira

chlferreira@gmail.com

<http://carnivorosemdentes.blogspot.com/>