

Programação Shell Script: como dominar seu terminal

Renê de Souza Pinto

14 de Outubro de 2008

- 1 Um pouco de BASH
 - Teclas de Atalho
 - Jobs
 - Redirecionamento de Entrada e Saída
 - Personalizando
- 2 Programando
 - Criando um script
 - Variáveis
 - Argumentos
 - Laços condicionais
 - Laços de repetição
- 3 Praticando...
- 4 Bibliografia

Um pouco de BASH

- O BASH é o shell padrão na grande maioria das distribuições Linux
- Possui bastante recursos
- Robusto

Teclas de atalho

Podemos poupar horas de trabalho e digitação através de atalhos de teclado:

Ctrl + a	Vai para o começo da linha (mesmo que Home)
Ctrl + e	Vai para o final da linha (mesmo que End)
Ctrl + l	Limpa a tela
Ctrl + u	Limpa conteúdo da linha até a posição cursor
Ctrl + k	Limpa conteúdo da linha depois da posição do cursor
Ctrl + w	Apaga a ultima palavra
Ctrl + r	Busca reversa
Ctrl + t	Inverte os dois ultimos caracteres antes do cursor
Setas ↑ e ↓	Acessa histórico de comandos

Teclas de atalho

Podemos poupar horas de trabalho e digitação através de atalhos de teclado:

Alt + f	Avança para próxima palavra da linha
Alt + b	Volta para a palavra anterior da linha
Esc + t	Troca as duas ultimas palavras antes do cursor
Tab	Auto-completa um comando
Ctrl + c	Envia um sinal de interrupção para o processo em execução
Ctrl + z	Suspende o processo em execução
Ctrl + d	Sai do shell atual

Jobs

- Quando um processo é iniciado o BASH o inicia em foreground, ou seja, o terminal é travado até que o programa seja finalizado (ou interrompido com Ctrl+c, Ctrl+z, etc).
- Ctrl+z interrompe o programa, para retornar a execução:
 - fg n - Retorna em foreground (travando o terminal)
 - bg n - Retorna em background (deixando o terminal disponível)
 - n - Número do Job
- Para exibir os trabalhos do usuário:
jobs

Jobs

Exemplo:

```
$ find /usr > /dev/null  
(Ctrl+z)  
[1]+ Stopped find /usr > /dev/null  
$ jobs  
[1]+ Stopped find /usr > /dev/null  
$ bg  
[1]+ find /usr > /dev/null &  
$ jobs  
[1]+ Running find /usr > /dev/null &
```

Redirecionamento de Entrada e Saída

- Recurso extremamente útil
- O S.O. possui 3 descritores de arquivos padrão:
 - 0: corresponde a entrada padrão (teclado, por exemplo)
 - 1: corresponde a saída padrão (monitor, por exemplo)
 - 2: corresponde a saída de erros padrão (monitor ou arquivo de log, por exemplo)
- Podemos redirecionar estas saídas

Redirecionamento de Entrada e Saída

Redirecionamento de saída

>	Redireciona a saída de um comando para um arquivo especificado, inicializando-o caso não exista ou destruindo seu conteúdo anterior.
>>	Redireciona a saída de um comando para um arquivo especificado, anexando-o ao seu fim. Caso este arquivo não exista, será criado.
2 >	Redireciona os erros gerados por um comando para o arquivo especificado. Mesmo que não ocorra erro na execução do comando, o arquivo será criado.

Fonte: [1]

Redirecionamento de Entrada e Saída

Redirecionamento de entrada

<	Avisa ao Shell que a entrada padrão não será o telado, mas sim o arquivo especificado.
<<	Também chamado de here document. Serve para indicar ao Shell que o espço de um comando começa na linha seguinte e termina quando encontra uma linha cujo conteúdo seja unicamente o label que segue o sinal <<.

Fonte: [1]

Redirecionamento de Entrada e Saída

Redirecionamentos especiais

	Este é o famoso pipe, e serve para direcionar a saída de um comando para a entrada de outro. É utilíssimo; não tenha parcimônia em usá-los, pois, normalmente otimiza a execução do comando.
<code>tee</code>	Captura a saída de um comando com pipe, copiando o que está entrando no tee para a saída padrão e outro comando ou arquivo.

Fonte: [1]

Redirecionamento de Entrada e Saída

Exemplos:

```
$ ls -l | wc -l
$ cat /proc/cpuinfo > my_cpu
$ cat /proc/devices >> my_cpu
$ cat /proc/cpuinfo | tee my_cpu2
$
$ cat > poema << FIM
O Shell é legal!
O Shell é maneiro!
Meu amigo companheiro,
Sou teu bom velho shelleiro,
Nunca me deixas em devaneio!
FIM
$
```

Personalizando seu BASH

- As configurações pessoais de cada usuário ficam guardadas no arquivo **.bashrc** presente no diretório home do usuário.
- O terminal aceita alguns caracteres especiais que funcionam como comandos, podendo mudar a posição do cursor, limpar tela e mudar a cor dos caracteres.
- A variável PS1 contém o formato da string que antecede o cursor na linha de comando, por exemplo PS1="`\u@\h \$`" diz que a string conterá o nome do usuário, seguido de um arroba (@), seguido do nome da máquina.

Personalizando seu BASH

Algumas opções:

<code>\h</code>	Nome da máquina sem o domínio
<code>\H</code>	Nome completo da máquina
<code>\j</code>	Número de jobs ativos
<code>\s</code>	Nome do shell
<code>\t</code>	Horário no formato 24 horas HH:MM:SS
<code>\u</code>	Login do usuário corrente
<code>\v</code>	Versão do Bash
<code>\w</code>	Diretório corrente, caminho completo
<code>\W</code>	Diretório corrente, somente o último

Personalizando seu BASH

Para colocar cores utilize o formato:

- `\[\e[XX;XX;Xm\]STRING\[\e[0m\]`
- STRING é a string para PS1 (Ex: `"\u@\h \$ "`)
- XX;XX;X deve ser substituído pelo código da cor desejada

Personalizando seu BASH

Códigos de cores do BASH:

	41;30	42;30	43;30	44;30	45;30	46;30	47;30
40;30;1	41;30;1	42;30;1	43;30;1	44;30;1	45;30;1	46;30;1	47;30;1
40;31	42;31	43;31	44;31	45;31	46;31	47;31	
40;31;1	41;31;1	42;31;1	43;31;1	44;31;1	45;31;1	46;31;1	47;31;1
40;32	41;32	43;32	44;32	45;32	46;32	47;32	
40;32;1	41;32;1	42;32;1	43;32;1	44;32;1	45;32;1	46;32;1	47;32;1
40;33	41;33	42;33	44;33	45;33	46;33	47;33	
40;33;1	41;33;1	42;33;1	43;33;1	44;33;1	45;33;1	46;33;1	47;33;1
40;34	41;34	42;34	43;34	45;34	46;34	47;34	
40;34;1	41;34;1	42;34;1	43;34;1	44;34;1	45;34;1	46;34;1	47;34;1
40;35	41;35	42;35	43;35	44;35	46;35	47;35	
40;35;1	41;35;1	42;35;1	43;35;1	44;35;1	45;35;1	46;35;1	47;35;1
40;36	41;36	42;36	43;36	44;36	45;36	47;36	
40;36;1	41;36;1	42;36;1	43;36;1	44;36;1	45;36;1	46;36;1	47;36;1
40;37	41;37	42;37	43;37	44;37	45;37	46;37	
40;37;1	41;37;1	42;37;1	43;37;1	44;37;1	45;37;1	46;37;1	47;37;1

Personalizando seu BASH

Exemplos:

```
$ # usuario@maquina - Amarelo
$ export PS1="\[\e[40;33;1m\]\u@\h\[\e[0m\] \$ "
$
$ # usuario@maquina - Verde e Amarelo
$ export PS1="\[\e[40;33;1m\]\u@\[\e[40;32;1m\]\h\[\e[0m\] \$
"
$
```

Programando

Ufa! Quanta coisa, vamos programar!

Programando

Um script nada mais é do que um arquivo contendo comandos para serem executados no Shell. Por exemplo:

```
#!/bin/sh  
  
echo "Olá Mundo do Shell!"  
echo
```

Programando

- Comentários são indicados por #
- Primeira linha contém um comentário funcional. Diz qual interpretador deverá executar o script
- Para executar o script:

```
$ chmod u+x ola.sh  
$ ./ola.sh  
$
```

Variáveis

- Sintaxe: **var=valor**

Exemplos:

```
#!/bin/sh
```

```
## Arquivo teste.sh
```

```
nome="Rene S. Pinto"
```

```
idade=23
```

```
sexo=masculino
```

```
echo "Nome: $nome"
```

```
echo "Idade: $idade"
```

```
echo "Sexo: $sexo"
```

```
echo
```

Variáveis

Resultado:

```
$ ./teste.sh  
Nome: Rene S. Pinto  
Idade: 23  
Sexo: masculino
```

Variáveis

Ler dados: **read**

```
#!/bin/sh

# Arquivo getname.sh

echo -n "Digite seu nome"
read nome

echo
echo "Ola $nome, como vai voce?"
echo
```

Argumentos

- \$0 - Nome do arquivo de script
- \$* - Todos os argumentos
- \$n - n-ésimo argumento passado
- \$# - Número total de argumentos
- \$? - Valor de retorno do ultimo comando executado

If

Estrutura:

```
if <comando>  
then  
  <comandos>  
else  
  <comandos>  
fi
```

If

Exemplos:

```
#!/bin/sh

resp=$1
if test $resp = S
then
echo "Posso ir"
else
echo "NAO posso ir"
fi
```

If

Outro jeito:

```
#!/bin/sh

resp=$1
if [ $resp = S ]; then
echo "Posso ir"
else
echo "NAO posso ir"
fi
```

If

Testes em arquivos		Testes em variáveis	
-b	É um dispositivo de bloco		
-c	É um dispositivo de caractere		Comparação Numérica
-d	É um diretório	-lt	É menor que (LessThan)
-e	O arquivo existe	-gt	É maior que (GreaterThan)
-f	É um arquivo normal	-le	É menor igual (LessEqual)
-g	O bit SGID está ativado	-ge	É maior igual (GreaterEqual)
-G	O grupo do arquivo é o do usuário atual	-eq	É igual (Equal)
-k	O sticky-bit está ativado	-ne	É diferente (NotEqual)
-L	O arquivo é um link simbólico		
-O	O dono do arquivo é o usuário atual		Comparação de Strings
-p	O arquivo é um named pipe	=	É igual
-r	O arquivo tem permissão de leitura	!=	É diferente
-s	O tamanho do arquivo é maior que zero	-n	É não nula
-S	O arquivo é um socket	-z	É nula
-t	O descritor de arquivos N é um terminal		
-u	O bit SUID está ativado		Operadores Lógicos
-w	O arquivo tem permissão de escrita	!	NÃO lógico (NOT)
-x	O arquivo tem permissão de execução	-a	E lógico (AND)
-nt	O arquivo é mais recente (NewerThan)	-o	OU lógico (OR)
-ot	O arquivo é mais antigo (OlderThan)		
-ef	O arquivo é o mesmo (EqualFile)		

Case

Estrutura:

```
case $VAR in
txt1) ... ;;
txt2) ... ;;
txtN) ... ;;
*) ... ;;
esac
```

If

Exemplo:

```
#!/bin/sh

for par; do
case $par in
--help-h) echo "Ajuda" ;;
--version-v)
echo "0.001" ;;
*)
echo "Comando desconhecido: $par"
exit 1
;;
esac
done
```

for

Estrutura:

```
for VAR in LISTA  
do  
<comandos>  
done
```

ou

```
for (( exp1; exp2; exp3 ))  
do  
<comandos>  
done
```

while

Estrutura:

```
while COMANDO  
do  
<comandos>  
done
```

- Dialog, Kdialog, ...
- CGI
- Exercícios

Bibliografia

-  Neves, Julio Cezar. Programação SHELL LINUX - 6ª edição. Brasport, 2006.
-  Jargas, Aurélio M. Introdução ao Shell Script.
<http://aurelio.net>
-  Bash, manual pages.

Até Logo!



¹“Programação Shell Script: dominando seu terminal”, por Renê de Souza Pinto, é licenciado sob a Creative Commons Atribuição-Uso Não-Comercial 2.5 Brasil License.