

Introdução ao Bash

Daniel Darlen

`daniel.correa@planejamento.gov.br`

IV SDSL

Brasília – DF

Dezembro de 2004

Introdução

Introdução

Duas formas básicas de interação com o computador:

- Interface Gráfica
- Linha de Comando

Introdução – Shell

Um interpretador de linha de comando (*shell*, em inglês)

é um programa que tem por finalidade receber comandos do usuário e passá-los ao sistema operacional.

Introdução – Prompt

A interação é realizada por meio do interpretador de comandos que aguarda as instruções por meio de um “sinal de pronto” (*prompt*, em inglês).

Introdução – Prompt

Prompt utilizado na família Unix

- \$ – para o usuário comum;
- # – para o super-usuário (*root*).

Introdução – Prompt

Exemplo:

```
$ df -h
```

```
Filesystem      Size  Used Avail Use% Mounted on
/dev/hda1       2.8G  2.5G  370M  87% /
/dev/hdc4       17G   360M   16G   3% /mnt/home
/dev/hdc4       17G   360M   16G   3% /home
/dev/hdd        605M  606M    0 100% /mnt/cdrom
```

```
$
```

Introdução – Prompt

Exemplo:

```
# cat /etc/fstab | grep home
/dev/hdc4    /home      ext3    auto,exec  1    1
#
```

Introdução – Script

Administradores de sistemas GNU/Linux geralmente utilizam ferramentas poderosas para automação e controle das tarefas do sistema, os *scripts*!

Um *script* é uma sequência pré-determinada de comandos.

Introdução - Bash

Bash é o interpretador de comandos mais utilizado nos sistemas GNU/Linux.

O nome Bash vem de "*Bourne Again SHell*", num trocadilho com "*born again shell*", indicando ser uma versão melhorada do antigo *Bourne Shell* para Unix.

Introdução – Bash

Bash é considerado o *shell* mais apropriado para desenvolvimento de scripts. Suas principais características:

- portátil;
- apresenta mais recursos para tratamento de arquivos;
- possui vasta documentação.

Introdução – Bash

À primeira vista, o *Bash*, assim como qualquer interpretador de comandos GNU/Linux/Unix, tem cara de “poucos amigos”. Não se preocupe, se você se dedicar terá o respeito e as respostas que quiser.

Introdução – rumo ao primeiro script

Demanda:

- Verificar a data e hora do sistema, identificar cada *file-system*, além do nome do sistema operacional e sua versão.

Introdução – rumo ao primeiro script

Pode-se começar com os comandos:

```
$ date
```

```
$ df
```

```
$ uname
```

Introdução – rumo ao primeiro script

Após uma busca nos manuais dos comandos, podemos incrementar nossa pesquisa colocando os parâmetros:

```
$ date
```

```
$ df -h
```

```
$ uname -s -r
```

Introdução – rumo ao primeiro script

A demanda pode ser automatizada por um *script*, que será executado a partir de um único comando.

Introdução – o primeiro script

Para escrever o primeiro script, abra um editor de texto de sua preferência (VI, emacs, etc...) e digite as seguintes linhas no arquivo:

```
#!/bin/bash  
date  
df -h  
uname -s -r
```

Introdução – o primeiro script

Salve o arquivo com nome primeiro.sh.

É uma boa prática salvar os scripts com a extensão (.sh). Embora esse procedimento não seja obrigatório, pode ser muito útil na hora de realizar uma busca por rotinas já escritas.

Introdução – o primeiro script

Todo script deve começar com '#!' seguido do nome do programa que o executa. No exemplo, descreveu-se o caminho completo do Bash (/bin/bash), para que o sistema identificasse qual interpretador de comandos deveria ser acionado.

Introdução – o primeiro script

Vá para o diretório onde o arquivo foi salvo e tente executá-lo:

```
$ ./primeiro.sh
```

Introdução – o primeiro script

Vá para o diretório onde o arquivo foi salvo e tente executá-lo:

```
$ ./primeiro.sh
```

```
bash: ./primeiro.sh: Permission denied
```

```
$
```

Ops... o que houve de errado?

Introdução – o primeiro script

É necessário que o sistema reconheça o script como um comando executável. Para isso digite o seguinte comando:

```
$ chmod +x primeiro.sh
```

Introdução – o primeiro script

Finalmente temos:

```
$ ./primeiro.sh
```

```
Sat Dec 4 16:24:25 BRST 2004
```

Filesystem	Size	Used	Avail	Use%	Mounted on
/dev/hda1	2.8G	2.5G	370M	87%	/
/dev/hdc4	17G	359M	16G	3%	/mnt/home
/dev/hdc4	17G	359M	16G	3%	/home
/dev/hdd	605M	606M	0	100%	/mnt/cdrom

```
Linux 2.4.20
```

```
$
```

Introdução – um script básico

Exemplo de um script básico (sistema.sh):

```
#!/bin/bash
# Aprimoramento do primeiro script
echo
echo "Data e Horário:"
date
echo
echo "Uso do disco:"
df
echo
echo "Informações do Sistema:"
uname -s -r
```

Introdução – um script básico

Comentários

- caracter "#" é utilizado para iniciar um comentário
- comando `echo` é utilizado para exibir mensagens na tela

Caracteres Especiais

Caracteres Especiais

#	declaração de comentários
;	separação de comandos - permite descrever mais de um comando ou comandos internos (if, while, etc.), numa mesma linha
;;	terminador especial do comando case
.	executa um script dentro de outro
"	delimitador de string - preserva o significado da maioria dos caracteres especiais
' (apóstrofo)	delimitador de string - cancela o significado da maioria dos caracteres especiais
\	cancela o significado especial do próximo caracter
` (crase)	delimitador de substituição de comandos - substitui um comando pela saída gerada pela sua execução
{xxx,yyy,...}	descreve valores possíveis para uma expansão -- ex.: ls *.{txt,html} lista todos os arquivos cujo nome termina em .txt ou .html
>, < e	redirecionadores de fluxo de dados (, >, <, > , > >, >&, > >&)

Variáveis e Parâmetros

Variáveis e Parâmetros

O Bash não exige identificação dos tipos de variáveis a serem utilizadas, nem uma seção especial no arquivo para sua definição.

- variáveis não inicializadas assumem valor 0 para operações aritméticas e “vazio” para uma string.

Variáveis e Parâmetros

- o uso de variáveis em Bash podem transformar números em strings e vice-versa;
- uma variável é normalmente associada pelo uso do sinal "=";
- o conteúdo de uma variável é acessado pelo uso do sinal "\$" antes do nome da variável;
- uma variável pode armazenar o resultado de um comando.

Variáveis e Parâmetros

Exemplos:

```
a="primeira associação"
```

```
b=20
```

```
c=$b
```

```
quemsou=$(whoami)
```

Variáveis e Parâmetros

Os parâmetros passados para um script Bash podem ser recuperados usando-se as variáveis '\$0', '\$1', '\$2', etc.

- \$0 – nome do script chamado
- \$1 – primeiro parâmetro
- \$2 – segundo parâmetro
- ...
- \$10 – décimo parâmetro

Variáveis e Parâmetros

Script: parametros.sh

```
#!/bin/bash
# Autor:
echo "O programa foi chamado com o nome \"$0\"
echo "Os $# parametros da chamada sao: "
echo "\"$*\
echo "O primeiro parametro da chamada foi \"$1\"
echo "O segundo foi \"$2\"
shift
echo "O terceiro foi \"$2\""
```

Estruturas de controle de fluxo

Controle de Fluxo

O Bash possui estruturas para se testar condições e implementar *loops*.

As mais utilizadas:

- if
- for
- while

Controle de Fluxo

```
if COMANDO
then
    comandos
else
    comandos
fi
```

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

```
while COMANDO
do
    comandos
done
```

Controle de Fluxo

If

- O if testa um comando e não uma condição
- Normalmente é utilizado em conjunto com o comando `test`

Controle de Fluxo

Script: testa10.sh

```
#!/bin/bash
# Autor:
echo "Digite um número: "
read numero
if test $numero -gt 10
then
    echo "$numero é maior que 10"
else
    echo "$numero não é maior que 10"
fi
```

Controle de Fluxo

Pode-se utilizar `[` ao invés de `test` no exemplo anterior ficaria:

```
if [ $numero -gt 10 ]
then
    echo "$numero é maior que 10"
else
    echo "$numero não é maior que 10"
fi
```

Controle de Fluxo

Comando test para variáveis

-z	a string passada é vazia
-n	a string passada não é vazia
-eq	os valores são iguais
-ne	os valores são diferentes
-gt	o primeiro é maior que o segundo
-ge	o primeiro é maior ou igual ao segundo
-lt	o primeiro é menor que o segundo
-le	o primeiro é menor ou igual ao segundo
= ou ==	os valores são iguais
!=	os valores são diferentes.

Controle de Fluxo

Comando test para arquivos

- e o arquivo existe
- d o arquivo é um diretório
- f o arquivo é um arquivo comum
- s o tamanho do arquivo é maior que zero
- r é possível ler o conteúdo do arquivo
- w é possível alterar o conteúdo do arquivo
- x é possível executar o arquivo
- nt o arquivo é mais recente
- ot o arquivo é mais antigo
- ef o arquivo é o mesmo

Controle de Fluxo

While

- serve para repetir comandos enquanto uma determinada condição for satisfeita.

Controle de Fluxo

Script: contador_while.sh

```
#!/bin/bash
# Autor:
i=0
while test $i -lt 10
do
    let "i = $i + 1"
    echo "Contando: $i"
done
```

Controle de Fluxo

For

- percorre uma lista de argumentos pegando um por vez

Controle de Fluxo

Script: exemplo_for.sh

```
#!/bin/bash
# Autor:
for x in a b c 1 2 3 abc
do
    echo "$x "
done
echo
```

Controle de Fluxo

Script: for_contador.sh

```
#!/bin/bash
# Autor:
for (( i = 1; $i <= 10; i++ ))
do
    echo "$i "
done
echo
```

Operações Aritméticas

Operações Aritméticas

É possível realizar operações aritméticas em Bash utilizando o comando `let`.

- Importante: `let` não trabalha com números reais. O resultado de uma divisão será sempre a parte inteira.

Operações Aritméticas

Operações do comando `let`

<code>+</code>	soma
<code>-</code>	subtração
<code>*</code>	multiplicação
<code>/</code>	divisão (parte inteira)
<code>**</code>	exponenciação (para expoente positivo)
<code>%</code>	módulo (resto de divisão inteira)
<code>+=</code>	incremento
<code>-=</code>	decremento
<code>*=</code>	multiplicação seguida de atribuição
<code>/=</code>	divisão seguida de atribuição
<code>%=</code>	módulo seguido de atribuição

Operações Aritméticas

Script: comando_let.sh

```
#!/bin/bash
# Autor:
let "x = 2 ** 10"
echo "2 ** 10 = $x" # 1024
y=$x
let "x %= 10" # x = x % 10
echo "$y % 10 = $x" # 4
y="abc"
let "x = $y + 7" # Nao da erro, x e' tido como zero
echo "$y + 7 = $x" # 2
let "y = $x * 3 / 6"
echo "$x*3/6=$y" # 3
```

Redirecionamentos de Entrada e Saída

É possível redirecionar a saída dos programas ou comandos do Bash para arquivos específicos.

- > redireciona a saída para um arquivo;
se o arquivo já existe ele é sobrescrito.
- >> redireciona a saída para um arquivo;
se o arquivo não existe ele será criado, caso
exista a saída é colocada ao final do arquivo,
preservando o conteúdo anterior.

Redirecionamentos de Entrada e Saída

Exemplos:

```
$ echo abc > arquivo.txt
```

```
$ cat arquivo
```

```
abc
```

```
$ echo def > arquivo.txt
```

```
$ cat arquivo
```

```
def
```

```
$ echo xyz >> arquivo.txt
```

```
$ cat arquivo
```

```
def
```

```
xyz
```

```
$
```

Interação com o usuário

É possível melhorar a forma de interação com o usuário por meio do comando `select`, que implementa um menu com listas numeradas.

Exemplo de interação com o usuário

```
#!/bin/bash
#Autor:
PS3="Digite o numero de sua opcao: "
echo "0 que voce deseja fazer?"
select opcao in "Listar a data e hora corrente."\
               "Listar file-systems."\
               "Listar informações do sistema operacional."\
               "Sair"
do
    echo Voce selecionou: $opcao
    case $REPLY in
        "1") date ;;
        "2") df -h ;;
        "3") uname -a ;;
        "4") break ;;
        *) echo "Opcao invalida!" ;;
    esac
done
```

Operadores `&&` e `||`

O comando `test` pode apresentar ainda dois operadores em sua implementação.

- O operador lógico `"&&"`, só executa o segundo comando caso o primeiro tenha sido OK. O operador inverso é o `"||"`.

Operadores && e ||

Script: sistema2.sh

```
#!/bin/bash
#Autor:
echo "Deseja exibir os dados do sistema? [sn] "
read resposta
test "$resposta" = "n" && exit

echo "Data e Horário:"
date
echo
echo "Uso do disco:"
df
echo
echo "Informações do Sistema:"
uname -s -r
```

Exemplos Práticos

Demanda:

- fazer um script "testa-arquivos", que pede ao usuário para digitar um arquivo e testa se este arquivo existe. Se sim, diz se é um arquivo ou um diretório.

```
#!/bin/bash
echo -n "Digite o arquivo: "; read ARQUIVO
[ -d "$ARQUIVO" ] && echo "$ARQUIVO é um diretório"
[ -f "$ARQUIVO" ] && echo "$ARQUIVO é um arquivo"
[ -f "$ARQUIVO" -o -d "$ARQUIVO" ] ||
    echo "O arquivo '$ARQUIVO' não foi encontrado"
echo

#
# Este exercício foi desenvolvido por Aurélio Marinho Jargas.
# http://aurelio.net.
```

Demanda:

- Elaborar um script que recebe dois números como parâmetro e realiza comparação entre eles, indicando se o primeiro é maior, menor ou igual ao segundo.

```
#!/bin/bash
if [ $1 -eq $2 ]; then
    echo "$1 é igual $2"
elif [ $1 -lt $2 ]; then
    echo "$1 é menor $2"
else
    echo "$1 é maior $2"
fi

#
# Este exercício foi desenvolvido por Aurélio Marinho Jargas.
# http://aurelio.net.
```

Referências

- Programação Shell Linux, 4ª Edição
Julio Cezar Neves
- BASH - Guia de Consulta Rápida
Joel Saade
- Apostilas do Aurélio Marinho Jargas
<http://aurelio.net>