

# Um passeio pelo Shell Script

Por: Rudson R. Alves

Bacharel em Física - UFES

Mestre em Física - UNICAMP

Usuário de UNIX deste 1991

Slackware GNU/Linux deste 1994

Fundador de Grupo Guar - 2003

Fundador do GUSES: <http://www.guses.com.br>

Colaborador do projeto Slack.Sarava: mkbuild,  
createpkg, ... <http://slack.sarava.org/>

# Apresentação

- Shell Script
- Echo
- Variáveis
- Estruturas do Bash (testes)
- Passando Parâmetros
- Função
- Redirecionamento
- Algumas ferramentas do Shell

# Vários Shells

O Shell é um interpretador de comandos. Mais que uma camada entre o sistema operacional e o usuário, ele também é uma poderosa linguagem de programação.

- Bsh – Bourne Shell (Steve Bourne 7ª versão do UNIX Bell Labs)
- Ksh - Korn Shell (David G. Korn at AT&T Bell Labs)
- Csh – C like shell
- Bash - GNU Bourne-Again SHell

# Shell Script

Um programa em shell, geralmente é chamado de *Script*, ou *Script Shell*

*Shell Script* = seqüência de chamadas a programas, binários compilados e todo o repertório de comandos do UNIX + estruturas de programação do shell (Ifs, WHILEs, FORs, testes, ...)

“...Como o Shell é poderoso, com o tempo mais e mais tarefas começam a ser desempenhadas por Scripts e estes começam a crescer, ficar lentos, complicados, difíceis de manter. ... É normal também escrever Scripts rápidos, de poucas linhas, que é usado por alguns dias e depois simplesmente esquecido... (Aurélio Marinho Jargas)”

# Shell Script

## Recomendações para a construção de bons Scripts

- Cabeçalho inicial com detalhes sobre o funcionamento
- Código alinhado (indent) e bem espaçado verticalmente
- Comentários explicativos e esclarecedores
- Nomes descritivos de Funções e Variáveis
- Controle de alterações e Versões
- Estrutura interna coesa e padronizada

# Shell Script

## O primeiro Script

```
01 #!/bin/sh
02 #
03 # Script para limpar a tela
04 echo "A tela será apagada em 10s"
05 sleep 10
06 clear
07
```

`#!/bin/sh` - Shell a executar o script  
(bash, zsh, ash, perl, ...)

`#` - comentário

# Shell Script

## Executando o Script

```
$ sh ./meu_script
```

```
$ . ./meu_script
```

```
$ source meu_script
```

```
$ chmod +x meu_script
```

```
$ ./meu_script
```

ou

```
$ meu_script
```

Se o diretório corrente (".")  
estiver no PATH

# Imprimindo

## Comando echo

Sintaxe: `echo [opções] [string]`

Imprime uma string na saída padrão

Algumas opções:

- n        imprime sem alimentação de linha
- e        habilita interpretação de texto
- \a - beep
- \n - nova linha
- \c - não alimenta linha
- \t - tabelamento horizontal
- \0NNN - caracter de código ASCII NNN (octal)
- ...

# Imprimindo

## Exemplos com echo

```
$ echo "Linux"  
Linux
```

Entra com um nome e o imprime

```
#!/bin/bash  
# Recebe um nome e imprime  
#
```

```
$ echo -e "\nLinux"  
  
Linux
```

```
clear  
echo -n "Entre com seu nome: "  
read NOME  
echo "Seu nome é $NOME"
```

```
$ echo -e "\033[31m Linux \033[0m"  
Linux
```

# Imprimindo

## Aurélio + echo

```
rudson@khelben:xx$ ./cores.sh
```

	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

```
rudson@khelben:xx$ more cores.sh
```

# Variáveis

## Manipulando Strings

<code>\${var:-texto}</code>	Se var não está definida, retorna 'texto'
<code>\${#var}</code>	Retorna o tamanho da string
<code>\${!var}</code>	Executa o conteúdo de \$var (igual a: eval "\$var")
<code>\${var:N}</code>	Retorna o texto à partir da posição 'N'
<code>\${var:N:tam}</code>	Retorna 'tam' caracteres à partir da posição 'N'
<code>\${var#texto}</code>	Corta 'texto' do início da string
<code>\${var%texto}</code>	Corta 'texto' do final da string
<code>\${var/texto/novo}</code>	Substitui 'texto' por 'novo', uma vez
<code>\${var//texto/novo}</code>	Substitui 'texto' por 'novo', sempre

# Variáveis

## Exemplos

```
$ VAR="E assim, quando mais tarde me procure, quem sabe a morte,  
angústia de quem vive..."
```

```
$ echo "Esta frase possui ${#VAR} caracteres"  
Esta frase possui 82 caracteres
```

```
$ echo ${VAR//quem/QUEM}  
E assim, quando mais tarde me procure, QUEM sabe a morte,  
angústia de QUEM vive...
```

```
$ echo ${VAR#E assim, quando mais tarde me procure, }  
quem sabe a morte, angústia de quem vive...
```

```
$ echo ${VAR%, quem sabe a morte, angústia de quem vive...}  
E assim, quando mais tarde me procure
```

```
$ echo ${VAR:2:5}  
assim
```

# Variáveis

## Operações Matemáticas

```
#!/bin/sh
# Script para calcular a idade
#
echo -n "Entre com o seu ano de nascimento: "
read RESP
IDADE=$(( $(date +%Y) - $RESP ))
echo "Sua idade é de $IDADE anos"
```

# Estruturas do Bash

## Sintaxe do comando if

```
if CONDIÇÃO; then
    comandos
elif CONDIÇÃO; then
    comandos
else
    comandos
fi
```

# Como testar?

## Testes

### Expressões numéricas

- lt é menor que (Less Than)
- gt é maior que (Greater Than)
- le é igual ou menor (Less Equal)
- ge é maior ou igual (Greater Equal)
- eq é igual (EQual)
- ne é diferente (Not Equal)

### Strings

- = é igual
- != é diferente
- ne não é nula
- z é nula

### Condicionais

- ! NÃO lógico (NOT)
- a E lógico (AND)
- o OU lógico (OR)

### Exemplos

- [ 5 -lt 3 ] 5 é menor que 3?
- [ ! 5 -lt 3 ] 5 não é menor que 3?
- [ "a" = "b" ] "a" é igual a "b"?
- [ -z \$VALOR ] \$VALOR está vazia?

# Como testar?

## Mais testes

### Testes de arquivo/diretório

- d é um diretório
- e o arquivo existe
- f é um arquivo normal
- G o grupo do arquivo é o do usuário atual
- L o arquivo/diretório é um link simbólico
- O o dono do arquivo é o usuário atual
- p o arquivo é um named pipe
- r o arquivo/diretório tem permissão de leitura
- s o tamanho do arquivo é maior que zero
- N o arquivo foi modificado desde a última leitura
- w o arquivo/diretório tem permissão de escrita
- x o arquivo/diretório tem permissão de execução
- nt o arquivo é mais recente (Newer Than)
- ot o arquivo é mais antigo (Older Than)
- ef o arquivo é o mesmo (Equal File)

# Estruturas do Bash

if na inicialização do sistema

```
#!/bin/sh
#
# rc.M      This file is executed by init(8) when the system is being
...
# Tell the viewers what's going to happen.
echo "Going multiuser..."
...
# Start networking daemons:
if [ -x /etc/rc.d/rc.inet2 ]; then
    . /etc/rc.d/rc.inet2
fi
...
# Start the print spooling system.  This will usually be LPRng (lpd) or CUPS.
if [ -x /etc/rc.d/rc.cups ]; then
    # Start CUPS:
    /etc/rc.d/rc.cups start
elif [ -x /etc/rc.d/rc.lprng ]; then
    # Start LPRng (lpd):
    . /etc/rc.d/rc.lprng start
fi
...

```

# Estruturas do Bash

## Script para resolver equação de 2º grau

```
#!/bin/bash
# Equação de segundo grau
# Por Rudson R. Alves
# Versão 1.1
if [ -z $3 ]; then
    echo -e "Entre: segrau <A> <B> <C>\npara: AX^2 + BX + C = 0"
    exit
fi

echo "$1 X^2 + $2 X + $3 = 0"
DELTA=$( echo "scale=10; $2^2 - 4*$1*$3" | bc -l )

if [ "$DELTA" -lt "0" ]; then
    echo "Não possui raizes reais"
else
    echo "Raizes reais:"
    SQRT=$( echo "scale=10; sqrt($DELTA )" | bc -l )
    X1=$( echo "scale=10; (((-1)*$2) + $SQRT)/(2*$1)" | bc -l )
    X2=$( echo "scale=10; (((-1)*$2) - $SQRT)/(2*$1)" | bc -l )
    echo -e "R1 = $X1\nR2 = $X2"
fi
```

# Estruturas do Bash

## Um pouco de diversão

```
$ echo "abacaxi" > t1
$ echo "abobora" > t2
$ mkdir t3
$ ln -s t1 t4
$ chmod 770 t1
$ ls -l
total 8
-rw-r--r-- 1 rudson users 8 2006-04-22 12:17 t1
-rw-r--r-- 1 rudson users 8 2006-04-22 12:17 t2
drwxr-xr-x 2 rudson users 72 2006-04-22 12:17 t3/
lrwxrwxrwx 1 rudson users 2 2006-04-22 12:17 t4 -> t1
```

# Estruturas do Bash

## Identificando um arquivo com o if

```
#!/bin/bash
# Script para identificar o arquivo passado como diretório, arquivo,
# link, ...
#
if [ -z $1 ]; then
    echo "Você deve entrar com um arquivo"
    echo "$0 [arquivo/diretório/...]"
    exit
fi

if [ ! -e $1 ]; then
    echo "$1 não existe"
elif [ -L $1 ]; then
    echo "$1 é um link"
elif [ -f $1 ]; then
    echo "$1 é um arquivo"
elif [ -d $1 ]; then
    echo "$1 é um diretório"
else
    echo "$1 não identificado"
fi
```

# Estruturas do Bash

## Outra forma de testar

```
[ TESTE ] && CONDIÇÃO VERDADEIRA || CONDIÇÃO FALSA
```

```
[ -x t1 ] && echo "t1 é executável" || echo "t1 não é executável"
```

```
[ -f t1 ] && echo "t1 é um arquivo" || echo "t1 não é um arquivo"
```

```
[ -d t1 ] && echo "t1 é um diretório" || echo "t1 não é um diretório"
```

```
[ t1 -nt t2 ] && echo "t1 é mais novo que t2" || echo "t1 é mais velho que t2"
```

# Estruturas do Bash

Sintaxe do comando for

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

# Estruturas do Bash

## Comando for

```
$ for FILE in *; do mv $FILE $FILE.dsk; done
```

```
$ for FILE in *.wav; do play $FILE; done
```

```
$ LISTA="1 banana carlos 4 0011"
```

```
$ for N in $LISTA; do echo ">$N<"; done
```

```
>1<
```

```
>banana<
```

```
>carlos<
```

```
>4<
```

```
>0011<
```

```
$ for i in $(seq 1 10); do echo $i; done
```

```
$ for ((i=1;i<11;i++)); do echo $i; done
```

```
$ for FIG in *.jpg; do convert $FIG ${FIG/.jpg/.png}; done
```

# Estruturas do Bash

## Exemplo com o for

```
#!/bin/bash
# gauge.sh - barra de progresso usando caracteres de controle
# 2003/07/23 Aurelio Marinho Jargas
#
#      [.....]          0%
#      [=====.....]   50%
#      [=====]        100%
#
# barra vazia
echo -n "[>.....] 0%"
passo="=>"
for i in $(seq 1 100); do
    sleep .1
    pos=$((i/2+1))           # calcula a posição atual da barra
    echo -ne "\033[G"       # vai para o começo da linha
    echo -ne "\033[${pos}C" # vai para a posição atual da barra
    echo -n "$passo"        # preenche mais um passo
    echo -ne "\033[55G"     # vai para a posição da porcentagem
    v="  $i"
    echo -n "${v:${#v}-3}"   # mostra a porcentagem
    echo -ne "\033[53G"
done
echo
```

# Estruturas do Bash

## Sintaxe do comando while

```
while CONDIÇÃO;  
do  
    comandos  
done
```

# Estruturas do Bash

## Comando while

```
$ while [ "$ANS" != "y" ]; do read ANS; done
```

```
$ while read LINE; do echo $LINE; done < /etc/fstab
```

```
$ more /etc/fstab | while read LINE; do echo "-->$LINE";  
done
```

# Estruturas do Bash

## Sintaxe do comando case

```
case $VAR in
    caso1)
        comandos
        ;;
    caso2)
        comandos
        ;;
    . . .
esac
```

# Estruturas do Bash

Script de inicialização do serviço ssh (/etc/rc.d/rc.sshd)

```
#!/bin/sh
# Start/stop/restart the secure shell server:

...

case "$1" in
'start')
    sshd_start
    ;;
'stop')
    sshd_stop
    ;;
'restart')
    sshd_restart
    ;;
*)
    echo "usage $0 start|stop|restart"
esac
```

# Passando Parâmetros

## Passando parâmetros

- \$0 Parâmetro número 0 (nome do comando ou função)
- \$1 Parâmetro número 1
- ...
- \$9 Parâmetro número 9
- \$10 Parâmetro número 10
- ...
- \$# Número de parâmetros
- \$\* Todos os parâmetros numa única string
- @\$ Todos os parâmetros em strings separadas
- \$\$ Número PID do processo
- \$? Código de retorno do último comando

# Passando Parâmetros

## Script parametros.sh

```
#!/bin/sh
#
# Apresenta os parâmetros enviados ao script,
# enumerando-os, um por linha.
#
# Entrada: Parâmetros quaisquer
#

echo "Número de parâmetros passados $#"
```

```
j=1
for i in $*; do
    echo "Parâmetro $j: $i"
    let j++
done
```

# Função

## Sintaxe de função

```
function nome_da_função()  
{  
    comandos  
}
```

```
nome_da_função()  
{  
    comandos  
}
```

# Função

## Funções: funcao.sh

```
#!/bin/bash
#
# Script para teste de funções
#
function func()
{
    echo "Número de parâmetros passados $#"
```

```
    j=1
    for i in $*; do
        echo "Parâmetro $j: $i"
```

```
        let j++
    done
}
#-----
echo "Teste de funções"
```

```
LISTA=$*
func $LISTA
```

# Redirecionamento

## Redirecionamentos

- > redireciona a saída padrão (STDOUT)
- < redireciona a entrada padrão (STDIN)
- >> redireciona a saída padrão (anexando)
- 2> redireciona a saída de erro (STDERR)
- 2>> redireciona a saída de erro (anexando)
- 2>&1 conecta a saída de erro a saída padrão
- >&2 conecta a saída padrão a saída de erro
- >&- fecha a saída padrão
- 2>&- fecha a saída de erro
- | conecta a saída padrão com a entrada padrão do comando seguinte

# Redirecionamento

## Exemplos de redirecionamento

```
$ ls -la > LISTA.TXT
```

```
$ startx 2> ERROS-XORG
```

```
$ ls -la /dev/ | grep "^b"
```

```
$ echo -e '\n echo "Iniciando o automount..." \n /etc/rc.d/rc.autofs start \n' >> /etc/rc.d/rc.local
```

# Cat

## Comando cat

Sintaxe: `cat [opções] [arquivo]`

Apresenta o conteúdo de um arquivo.

Algumas opções:

<code>-n, --number</code>	enumera as linhas
<code>-s, --squeeze-blank</code>	remove linhas em branco
<code>-E, --show-ends</code>	coloca um \$ no fim de cada linha

Exemplos:

```
$ cat /etc/fstab
```

```
...
```

```
$ cat -n /etc/fstab
```

```
1 ...
```

# Cut

## Comando cut

Sintaxe: `cut [opções] [arquivo]`

Extrai seções específicas de uma linha.

Algumas opções:

`-d, --delimiter=DELIM`

utiliza um delimitador específico, ao invés do TAB

`-f, --fields=LIST`

seleciona apenas os campos especificados

`-c, --characters=LIST`

seleciona apenas estes caracteres

Exemplos:

```
$ echo "01:02:03:04:05:06:07:08:09" | cut -c 5-8
```

```
2:03
```

```
$ echo "01:02:03:04:05:06:07:08:09" | cut -d: -f3
```

```
03
```

```
$ echo "01:02:03:04:05:06:07:08:09" | cut -d: -f3,6-8
```

```
03:06:07:08
```

# Grep

## Comando grep

Sintaxe: `grep [opções] FILTRO [arquivo]`

Filtra linhas em um arquivo que possuem um dado padrão (FILTRO).

Algumas opções:

- i, --ignore-case ignora distinção de letras maiúsculas/minúsculas
- v, --invert-match seleciona linhas que não possuem o FILTRO
- r, --recursive busca recursivamente em todos os diretórios e sub-dir
- n, --line-number coloca número de linha
- s, --no-messages omite mensagens de erro
- l, --files-with-matches apresenta apenas o nome do arquivo que possui o FILTRO

# Grep

## Comando grep

```
$ grep "wireless" /usr/doc/Linux-HOWTOs/
```

...

Apresenta as linhas dos arquivos que possuem a palavra 'wireless'

```
$ grep -r -l -s "wireless" /usr/doc/Linux-HOWTOs/
```

...

Apresenta o nome dos arquivos (-l) que possuem a palavra 'wireless', contidos no diretório e sub-diretórios (-r) /var/log/ e omite mensagens de erro (-s)

# Sort

## Comando sort

Sintaxe: `sort [opções] [arquivo]`

Ordena as linhas de um arquivo.

Algumas opções:

<code>-f, --ignore-case</code>	ignora maiúsculas/minúsculas
<code>-r, --reverse</code>	ordem reversa

Exemplos:

```
$ sort /etc/passwd
```

...

```
$ sort -r /etc/passwd
```

# Find

## Comando find

Sintaxe: `find [-L] [-P] [caminho] [expressão]`

Busca de arquivos

Opções:

- P nunca segue links simbólicos
- L sempre segue links simbólicos

Expressões:

- name <nome> especifica nome ou parte dele
- iname <nome> ignora diferenças entre letras maiúsculas/minúsculas
- type <tipo> especifica o tipo do arquivo (d - diretório, f - arquivo, l - link )
- user <usuário> arquivos pertencentes ao <usuário>
- exec <comando> executa um comando com os arquivos encontrados
- size <tam> arquivos com o tamanho <tam>

# Find

Exemplos:

```
$ find /usr/lib -name libcairo*
```

...

```
$ find . -type d -exec du -sh {} +
```

... apresenta o tamanho de cada subdiretório em ., através do comando du

## Outros comandos

users, wc, tr, date, head, tail, xargs, ...

# Sed

## Comando sed

Sintaxe: `sed 'comandos' [arquivo]`

O comando `sed` é um editor de linha

Criar um arquivo “test.txt” com 20 linhas com um número aleatório em cada linha:

```
$ for i in $(seq 1 20); do SENHA=$RANDOM; echo "$i $SENHA" >> test.txt; done
```

Remover a 5ª linha:

```
$ sed '5 d' test.txt
```

Substituir a primeira ocorrências de “2” por “-DOIS-”:

```
$ sed 's/2/-DOIS-/' test.txt
```

Remover a linha contendo “7 “:

```
$ sed '/7 / d' test.txt
```

Substituir todas as ocorrências de “2” por “-DOIS-”:

```
$ sed 's/2/-DOIS-/g' test.txt
```

Remover todas as linhas que possua os números 6 e 5:

```
$ sed '/6/ d; /5/ d' test.txt
```

# Awk

## Comando awk

Sintaxe: `awk [ opções ] ...`

Uma linguagem de propósitos gerais, designada para processar dados em formato texto

Apresenta o grupo e o nome do dispositivo em `/dev/`

```
ls -l /dev | awk '{ print $4 "\t" $9 }'
```

Soma os tamanhos em bytes dos arquivos (coluna 5 do `ls -l`)

```
ls -l /dev/ | awk '{ sum += $5 } END { print sum }'
```

Somas os tamanhos em bytes dos arquivos pertencentes ao grupo `disk`

```
ls -l /dev/ | awk '$4 == "disk" { sum += $5 } END { print sum }'
```



# Bibliografia

- Lista de Shell-Script  
<http://brgroups.yahoo.com.br/group/shell-script/>
- *Programação Profissional em Shell-Script*, do Aurélio Marinho Jargas, <http://aurelio.net>
- Páginas do manual (man) do sistema GNU/Linux
- *Advanced Bash-Scripting Guide*, Mendel Cooper  
<http://www.tldp.org/LDP/abs/html/>