

Programação Shell Script

Wesley Lima

Introdução

- O que é Shell?
 - programa que conecta e interpreta os comandos;
 - linguagem de programação completa interpretada;
 - possui variáveis;
 - construções condicionais e interativas;
 - ambiente adaptável ao usuário;
- O que é Shell Script?
 - Um script é um arquivo que guarda vários comandos e pode ser executado sempre que preciso. Os comandos de um script são exatamente os mesmos que se digita no prompt, ou seja, são comandos que poderiam ser digitados diretamente no terminal e seriam executados

Introdução

- **Produtividade**

- Linguagem interpretada não compilada
- Um programador médio pode duplicar ou triplicar sua produtividade com o uso do Shell
- Comparação de Bruce Cox (pai do Objective C)
 - shell 1 linha de código
 - linguagem orientada a objeto 10 linhas de código
 - linguagem C 100 linhas de código

O primeiro script

```
#!/bin/bash  
echo "Olá Mundo"
```

- Antes de executar o script, altere suas permissões com o comando `chmod`.
 - **Ex:** `chmod 755 ola.sh`
- Para executar:
 - `./ola.sh`
 - `bash ola.sh`

Tipos de Shell

- Existem variantes do shell para os diferentes sistemas Linux/UNIX
 - Bourne Shell – SH
 - Korn Shell – KSH
 - Bourne Again Shell – BASH
 - C Shell – CSH
- Cada shell possui particularidades no seu uso o que impede, em alguns casos, que um script escrito em um shell execute corretamente em outro.

Variáveis

- Os nomes podem conter letras maiúsculas ou minúsculas, algarismos e o símbolo _
- O nome deve iniciar com letras
- As variáveis não precisam ser declaradas

Variáveis especiais

- Variáveis pré-definidas pelo shell que são muito úteis para se obter informações importantes para o script. Algumas delas:
 - \$0 : o nome do script
 - \$n : o n-ésimo argumento da linha de comando
 - \$* : todos os argumentos da linha de comando
 - \$# : número de argumentos
 - \$? : status do último comando executado (status <> 0 indica erro)
 - \$\$: número de processo (PID) do shell que executa o script
- Exercício: Teste o uso destas variáveis criando um script que mostre os valores destas variáveis

Argumentos

- Posição de Argumentos

cmd	arg1	arg2	arg3	arg4	arg5	arg6	...	arg9
\$0	\$1	\$2	\$3	\$4	\$5	\$6	...	\$9

Exercício

- Fazer um script que faça as seguintes tarefas
 - Listar os diretórios
 - Mostrar as partições ativas do sistema
 - Mostrar a memória disponível
 - Mostrar os processos ativos do usuário

Redirecionamento de entrada e saída

- Permite:
 - Criar ou anexar arquivos;
 - Usar arquivos existentes como entrada para o shell;
 - Reunir dois fluxos de saída;
 - Usar a saída de um comando como entrada de outro.
- Fluxos existentes
 - stdin – Entrada padrão(Normalmente teclado)
 - stdout – Saída padrão(Normalmente terminal)
 - stderr – Saída de erro padrão(Normalmente terminal)

Redirecionamento

- Operadores
 - `<arquivo` – Usa *arquivo* como entrada
 - `>arquivo` – Usa *arquivo* como saída
 - `>>arquivo` – Anexa as saída ao final de *arquivo*
 - `2>` - Redireciona *stderr*
 - `>&` - Reúne *stderr* a *stdout*
- Exemplos
 - `sed -e "s/shell/Shell/g"<cap1.txt >novocap1.txt`
 - `ls /xpto 2>out.txt`
 - `ls />>out.txt`
 - `ls /xpto /u* >& out.txt`

Redirecionamento - Pipes

- Conecta a *stdout* de um comando à *stdin* de outro
- Meio de condução para transportar dados de um comando para outro.
- Exemplos:
 - `ls -R|grep ".gz"`
 - `ls -l|grep -v ^d`

Exercícios

- Determine quanto arquivos normais (não diretórios nem links) existem em /usr.
- Qual a sequência de comandos para se obter uma listagem dos usuários da máquina. As informações podem ser obtidas no arquivo /etc/passwd.

O comando sed

- **SED** - Edita e altera dados.
- Sintaxe:
`sed [opções] stdin > stdout`
- Opções:
- `-e "script"` : edita e altera dados
Onde script pode ser:
`"s/string antiga/string nova/g"`
`"/string pertencente à linha a ser deletada/d"`
`"s/string a ser deletada//g"`
- Exemplo
– `sed -e "s/:/+/g" /etc/passwd`

Exercício

- Leia o arquivo `/etc/passwd` e crie um novo arquivo chamado `/home/senacti/usuarios` ordenado pelo nome do usuário

Tabela de comandos

TIPO	COMANDO
Diretório	cd ls pwd
	mkdir rmdir
Arquivo	cat cp csplit
	ln mv rm split
Seleção	awk
	cut
	grep
	head
	line
	sed
	tail
Junção	uniq
	wc
	cat
	join
Ordenação	paste
	sort
Transformação	sed
	tr
	dd
Impressão	cat
	echo
Segurança	chmod
Leitura	\$<
	touch
	sleep
	exit

Fonte: Slides SENAC

Capturando a saída de um comando

- A saída de um comando pode ser capturada e armazenada em uma variável
- Para isso usa-se a crase.
- Exemplo
 - `LISTAGEM=`ls -l``
 - `echo $LISTAGEM`

Comando `expr`

- Os argumentos são usados como elementos
- A expressão é avaliada
- O resultado é escrito na saída padrão
- Exemplo
 - `expr 17 + 61`
 - `78`
- Use `expr` para criar um contador
 - `index=`expr $index + 1``
- Expressões matemáticas podem ser calculadas também da seguinte maneira:
 - `index=$((index+1))`

Códigos de Retorno

- Exemplo
- `grep root /etc/passwd`
- `echo $?`
- 0
- `grep john /etc/passwd`
- `echo $?`
- 1
- `echo $?`
- 0

Códigos de Retorno

- `true`
- `echo $?`
- 0
- `false`
- `echo $?`
- 1
- `echo $?`
- 0

Comando IF

```
if cmd
then
    cmda
    cmdb
    ....
fi
```

```
if cmd
then
    cmda
    cmdb
else
    cmdx
fi
```

Comando IF

```
if
    grep $1 /etc/passwd > /dev/null
then
    echo $1 Encontrado
    exit 0
else
    echo $1 Não encontrado
    exit 1
fi
```

Comando IF

- Execução
- `finder root`
 - Root Encontrado
- `echo $?`
 - 0
- `finder Fernando`
 - Fernando Não Encontrado
- `Echo $?`
 - 1

Comando IF

- Vamos escrever um *script* que nos diga se uma determinada pessoa, que será passada por parâmetro, fez *login* no seu computador:

```
$ cat talogado
#
# Verifica se determinado usuario esta "logado"
#
if who | grep $1
then
    echo $1 esta logado
else
    echo $1 nao esta logado
fi
```

Comando IF

- Repare que a execução do *talogado* gerou uma linha correspondente à saída do *who*, que não traz proveito algum para o programa;
- Existe um *buraco negro* do UNIX que todas as coisas que lá são colocadas desaparecem sem deixar vestígio: */dev/null*

Comando IF

```
$ cat talogado
#
# Verifica se determinado usuario esta "logado" -
# versao 2
#
if who | grep $1 > /dev/null # /dev/null redirecionando a saída
do who
then
    echo $1 esta logado
else
    echo $1 nao esta logado
fi
$ talogado alex
alex esta logado      #Não apareceu a linha indesejada
```

Comando IF

- Faça um exemplo que leia dois parâmetros e verifique se esses parâmetros são usuários válidos na máquina. As saídas possíveis são
 - \$1 e \$2 São ambos válidos
 - Somente \$1 foi encontrado
 - Somente \$2 foi encontrado
 - Nenhum dos dois foi encontrado

Comando IF

- Para saber se o conteúdo de uma variável é numérico ou não, poderíamos fazer:

```
if expr $1 + 1 > /dev/null 2> /dev/null
  #Resultado de expr e erro que irão para /dev/null
then
  echo É um numero
else
  echo Não é um numero
fi
```

Se o resultado de qualquer operação feita pelo `expr` resultar em zero, seu *código de retorno* será diferente de zero, então no exemplo acima, caso o conteúdo da variável fosse -1, o resultado do `if` seria inválido. Experimente fazer `expr -1 + 1` ou `expr 0 + 0` e em seguida teste o código de retorno.

Comando Test

- A essa altura dos acontecimentos você irá me perguntar: ora, se o comando `if` só testa o conteúdo da variável `$?`, o que fazer para testar condições?
- Para isso o *Shell* tem o comando `test`, que na sua forma geral obedece a seguinte sintaxe:
 - `test <expressão>`
- Veja o exemplo a seguir

Comando Test

```
$ cat pedi
#
#  Testa a resposta a um pedido. Deve ser (S)im ou (N)ao
#

resp=$1
if test $resp = N    Saída do test é entrada do if
then
    echo Ela nao deixa...
else
    if test $resp = S    Um if dentro do outro dificulta a legibilidade
    then
        echo Oba, ela deixou!!!
    else
        echo Acho que ela esta na duvida.
    fi
fi
```

Comando Test

- `$ pedi s`
- Oba, ela deixou!!!
- `$ pedi N`
- Ela nao deixa...
- `$ pedi A`
- Acho que ela esta na duvida.
- `$ pedi Xiii`, *esqueci de passar o parâmetro...*
- `pedi[6]: test: argument expected` *Ué, o que houve?*
- `pedi[10]: test: argument expected` *Ídem...*
- Acho que ela esta na duvida.

Operações Comparação

- Comparação

Comparação Numérica	
-lt	É menor que (LessThan)
-gt	É maior que (GreaterThan)
-le	É menor igual (LessEqual)
-ge	É maior igual (GreaterEqual)
-eq	É igual (Equal)
-ne	É diferente (NotEqual)

Comparação de Strings	
=	É igual
!=	É diferente
-n	É não nula
-z	É nula

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

Operações Comparação

Testes em arquivos	
-b	É um dispositivo de bloco
-c	É um dispositivo de caractere
-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 é um link simbólico
-o	O dono do arquivo é o usuário atual
-r	O arquivo tem permissão de leitura
-s	O tamanho do arquivo é maior que zero
-S	O arquivo é um socket
-w	O arquivo tem permissão de escrita
-x	O arquivo tem permissão de execução

Exemplo

```
#!/bin/bash
if test $UID = 0
then
    adduser $1
    echo $2 | passwd $1 --stdin
    echo "Usuário $1 adicionado com sucesso"
else
    echo "Pra criar usuários requer poderes de root."
fi
```

Exemplo

```
#!/bin/bash
mkdir /tmp/teste
if [ $? -eq 0 ]; then
    echo $?
    echo "comando executado com sucesso!"
else
    echo $?
    echo "falha na execução do comando."
    exit
fi
```

1- Exercício

- Faça um script que execute a seguinte função:
 - executa o comando 'cd'. se NÃO CONSEGUIU executar 'mkdir'
 - cd algumdiretorio || mkdir alagumdiretorio
 - Retornar a configuração de Rede
 - Endereço IP
 - NetMask
 - MAC
 - Gateway

2 -Exercício

- Recebe dois números como parâmetro e mostra a relação entre eles. Exemplo:

```
prompt$ ./relacao.sh 3 5  
3 é menor 5
```

- Recebe um número como parâmetro e o diminui até chegar a zero, mostrando na tela cada passo, numa mesma linha. Exemplo:

```
prompt$ ./zerador.sh 5  
5 4 3 2 1 0  
prompt$ ./zerador.sh 10  
10 9 8 7 6 5 4 3 2 1 0
```

3 -Exercício

- Recebe duas palavras como parâmetro e checa se a primeira palavra está contida dentro da segunda. Só mostra mensagem informativa em caso de sucesso, do contrário não mostra nada. Exemplo:

```
prompt$ ./substring.sh ana banana  
ana está contida em banana  
prompt$ ./substring.sh banana maria  
prompt$ ./substring.sh banana  
prompt$ ./substring.sh  
prompt$
```

4 - Exercício

- Do arquivo `/etc/passwd`, mostra todos os shells (último campo) que os usuários usam. Não mostrar linhas repetidas.

- Exemplo:

```
prompt$ ./shells.sh  
/bin/bash  
/bin/false  
/bin/sync  
/sbin/halt  
/sbin/shutdown
```

5 - Exercício

- Mostra na tela todos os parâmetros recebidos na linha de comando, contando-os.

- Exemplo:

```
prompt$ ./parametros.sh a b c d e f
```

- Parâmetro 1: a
- Parâmetro 2: b
- Parâmetro 3: c
- Parâmetro 4: d
- Parâmetro 5: e
- Parâmetro 6: f

While

```
while [ condition ]  
do  
    command1  
    command2  
    command3  
    ..  
    ....  
done
```

While

- Exemplo

```
while true#Loop de leitura  
do  
    read isto  
    if [ ! "$isto" ]#Se o campo estiver vazio.  
    then  
        exit  
    else  
        continue  
    fi  
done
```

For

- `for var in lista_de_palavras do`
 - no corpo do loop, temos `$var` = a próxima palavra da lista
- `done`

For

- Exemplo
`for i in segunda terca quarta quinta sexta`
`do`
`echo "os dias da semana sao $i"`
`done`
- `./dias`
 - os dias da semana sao segunda
 - os dias da semana sao terca
 - os dias da semana sao quarta
 - os dias da semana sao quinta
 - os dias da semana sao sexta

Exemplo

```
#!/bin/bash
i=1
teste=`echo $1 | cut -c1`
if [ "$teste" = / ]
then
    echo dir1: /
    i=2
fi
for x in `echo $1 | tr / '\012'`
do
    echo "dir$i: $x"
    let i++
done
```

For

- O comando for é ótimo para ler um arquivo seqüencial. Veja o exemplo abaixo

```
arquivo=`cat /etc/group`
for i in $arquivo
do
    echo $i
    echo -----
done
```

6 -Exercícios

- Faça o exemplo anterior apresentando apenas o nome do grupo
- Faça um script que informe que o root se logou na máquina
- Faça um script de um relógio

Case

```
case $variable-name in
  pattern1) command
    ...
    command;;
  pattern2) command
    ...
    command;;
  patternN) command
    ...
    command;;
  *) command
    ...
    command;;
esac
```


Exemplo

```
#!/bin/bash
while true
do
clear
cat <<!
```

```
Menu de Usuario
```

```
1 - ls
2 - ps -f
3 - who
4 - exit
```

```
Digite sua Opcao :
```

```
!
```

```
while
tput cup 11 34 ; read opt
[ "$opt" = "" ]
do
: #Comando NULL(Apenas para o loop
ocorrer
done
case $opt in
1) clear
ls ;;
2) clear
ps -f ;;
3) clear
who ;;
4) clear
echo Adios Amigo
exit 0 ;;
*) clear
echo opcao invalida
for x in 1 2 3 4 5 6 7 8
do
echo "\a\c"
sleep 1
done
;;
esac
echo Tecle Enter para continuar "\c"
read
done
```

7 - Exercício

- O que faz o script abaixo?

```
#!/bin/bash
val=`expr ${1:-0} + ${2:-0} + ${3:-0}`
echo $val
```

- `${var:-texto}` Se `var` não está definida, retorna '`texto`'

8 -Exercício

- Faça um script que monte um menu, como mostra abaixo
 - Script de Administracao
 - 1 – Ver processos ativos
 - 2 – Mostra dos files systems da máquina
 - 3 – Mostra a quanto tempo a máquina está no ar
 - 4 – Usuários ativos na máquina
 - 5 – Versão do kernel
 - 6 – Lista de usuários da máquina
 - 7 – Sair do sistema

Exercícios

- 3. O que faz o script abaixo?


```
echo O primeiro argumento é: $1
echo O segundo argumento é: $2
echo O terceiro argumento é: $3
echo O quarto argumento é: $4
echo " n n n"
shift
echo "Shiftou... n"
echo Agora o primeiro argumento é: $1
echo Agora o segundo argumento é: $2
echo Agora o terceiro argumento é: $3
echo Agora o quarto argumento é: $4
```

Exercícios

```
echo "Escreva o nome do arquivo e a palavra a ser
      pesquisada:"
read file word
if grep $word $file > /dev/null
then
    echo "A palavra $word existe no arquivo $file."
fi
```

Exercícios

```
DIA=$1
MES=$2
ANO=$3
DIA=`expr $DIA - 1`
if [ $DIA -eq 0 ]; then
    MES=`expr $MES - 1`
    if [ $MES -eq 0 ]; then
        MES=12
        ANO=`expr $ANO - 1`
    fi
    DIA=`cal $MES $ANO`
    DIA=`echo $DIA | awk '{ print $NF }'`
fi
echo $DIA $MES $ANO
```

Exercícios

- Crie um arquivo com o seguinte conteúdo(nome: sitevistado)

```
jamiesob mucus.slime.com
tonsloye xboys.funnet.com.fr
tonsloye sweet.dreams.com
root sniffer.gov.au
jamiesob marvin.ls.tc.hk
jamiesob never.land.nz
jamiesob guppy.pond.cqu.edu.au
tonsloye xboys.funnet.com.fr
tonsloye www.sony.com
janesk horseland.org.uk
root www.nasa.gov
tonsloye warez.under.gr
tonsloye mucus.slime.com
root ftp.ns.gov.au
tonsloye xboys.funnet.com.fr
```

Exercícios

- O que faz o script abaixo? Faça-o funcionar

```
while read linha
do
  usuario=`echo $linha | cut -d" " -f1`
  site=`echo $linha | cut -d" " -f2`
  if [ "$usuario" = "$1" ]
  then
    echo "$usuario visitou $site"
  fi
done < sitevisitado
```

exercícios

- Crie um arquivo com o seguinte conteúdo(nome: siteproibido)

```
mucus.slime.com
xboys.funnet.com.fr
warez.under.gr
crackz.city.bmr.au
www.hotwarez.com.br
```

Exercícios

O que faz esse script

```
for verifUsuario in $*
do
  while read linha
  do
    while read verifSite
    do
      usuario=`echo $linha | cut -d" " -f1`
      site=`echo $linha | cut -d" " -f2`
      if [ "$usuario" = "$verifUsuario" -a "$site" =
"$verifSite" ]
      then
        echo "$usuario visitou o site proibido $site"
      fi
    done < siteproibido
  done < sitevisitado
done
```

Exercícios

```
#!/bin/bash
# ARQUIVO: list
#
numLinha=1
while read linha
do
    echo "$numLinha $linha"    numLinha=`expr
    $numLinha + 1`
done < $1
```

Exercícios

- A partir do script anterior, crie um arquivo chamado newarq2 com as linhas numeradas

Exercícios

- Fazer um script que dê bom dia se for manhã, boa tarde, se for tarde ou boa noite de for noite
- Crie um script para informar que o root se logou na máquina.
- Crie um script para fazer ordenar 3 números
- Faça um script que avise quando o processamento da máquina ultrapassar 50%

Sinais e Traps

- **Sinais são enviados para processos de várias formas**
 - Pelo kernel, quando um processo faz besteira
 - Pelo usuário, usando o teclado (^C, ^\, encerrando a sessão)
 - Usando o comando kill
- **Ação normal: o processo morre**
 - Porém, um processo pode ignorar os sinais ...
 - ... ou captura-los para fazer algo

Interrupções

- Os principais sinais de interrupção mascaráveis são:

- **1 = SIGHUP / HANGUP**

Sinal enviado ao processo pelo sistema operacional quando a shell ou sessão a partir da qual o script foi executado é finalizada.

- **2 = SIGINT / INTERRUPT**

Sinal enviado pelo sistema operacional ao processo em execução em uma sessão interativa, quando o usuário pressiona CTRL+C.

- **15 = SIGTERM / TERMINATE**

Sinal padrão enviado pelo comando kill para informar que o processo deve terminar.

Sinais e Traps

- 0 Fim do shell
- 1 Hangup
- 2 Interrupt (^C)
- 3 Quit(^)
- 4 Illegal Instruction
- 5 Trace trap
- 6 IOT instruction
- 7 EMT instruction
- 8 Floating point exception(bug de programa)
- 9 Morte certa (kill -9)
- 10 Bus error(bug de programa)
- 11 Violação de segmentação(bug de programa)
- 12 Bad argument
- 13 Pipe write error
- 14 Alarm
- 15 Software termination signal (kill sem argumentos)

Quiz

- O que faz o trap nesse exemplo?

```
#!/bin/bash
```

```
tempfile=/tmp/temp.$$
```

```
trap "rm -f $tempfile" 0 1 2
```

```
ls -la > $tempfile
```

```
cp $tempfile /tmp/resultado
```

```
for i in $*
```

```
do
```

```
    echo $i
```

```
done
```