Programação em shell

Dulce Domingos
(Setembro de 2015)

Índice

1. Introdução

O objectivo deste documento é apresentar os conceitos subjacentes à programação em *shell*.

A utilização da *shell* como linguagem de programação permite, por exemplo:

- Construir novos comandos;
- Construir comandos que executam tarefas condicionais;
- Construir comandos que repetem outros comandos.

Aos programas escritos em *shell* designamos por *shell-scripts*.

2. Criar um novo comando

A *shell* permite a criação de novos comandos, como sequências ou agrupamentos de comandos existentes.

Exemplo:

Através de um editor de texto foi criado o ficheiro hello.sh.

A primeira linha do *shell-script* indica que o ficheiro deverá ser executado por /bin/sh. Esta é a localização normal da *Bourne shell* nos sistemas Unix. Nos sistemas Linux, normalmente, este ficheiro é um *link* simbólico para a *bash* (ver `ls –la /bin/sh`).

O caracter '#' indica início de comentário, excepto no caso em que "#!" surge na primeira linha dos ficheiros.

```
[so000@falua exerc2]$ cat hello.sh
#!/bin/sh
# This is a comment!
echo Hello World        # This is a comment, too!
```

Em Unix, para que seja possível executar um comando é necessário que ele tenha permissões de execução.

```
[so000@falua exerc2]$ ./hello.sh
-bash: ./hello.sh: Permission denied
[so000@falua exerc2]$ chmod u+x hello.sh
[so000@falua exerc2]$ ./hello.sh
Hello World
```

Se a variável PATH não incluir a directoria corrente, é necessário indicar a localização do ficheiro. Considere o seguinte exemplo:

```
[so000@falua ex]$ echo $PATH
/usr/local/bin:/bin:/usr/bin:/home/falua/so/so000/bin
[so000@falua exerc2]$ hello.sh
bash: hello.sh: command not found
[so000@falua exerc2]$ ./hello.sh
Hello World
```

3. Variáveis

As variáveis são definidas do seguinte modo (sem espaços antes ou depois do caracter '='):

```
NOME_DE_VAR=valor_de_var
```

O valor das variáveis pode ser acedido dos seguintes modos:

- `$NOME_DE_VAR`
- `${NOME_DE_VAR} - esta sintaxe é útil quando o nome da variável é seguido por outro texto.`

Exemplo:

```
[so000@falua so000]$ DIA=5
[so000@falua so000]$ echo $DIA
5
[so000@falua so000]$ echo $DIAa feira
feira
[so000@falua so000]$ echo ${DIA}a feira
5a feira
```

3.1. Regras para nomes de variáveis

Os nomes das variáveis começam por um caracter alfanumérico ou pelo caracter '_'. Os caracteres seguintes são alfanuméricos.

Podem ser definidas variáveis com valor NULL dos seguintes modos:

```
[so000@falua so000]$ NADA1=
[so000@falua so000]$ NADA2=""
[so000@falua so000]$ echo $NADA1

[so000@falua so000]$ echo $NADA2

[so000@falua so000]$
```

3.2. Variáveis especiais

3.2.1 Argumentos da linha de comandos

A variável $0 contém o nome do programa (*shell script*). As variáveis $1, $2, ...$9 contêm os argumentos do programa. Se o programa contiver mais de 9 argumentos, pode-se utilizar o comando `shift` (ver `man shift`).

A variável $# contém o número de argumentos do programa.

As variáveis $* e $@ contêm todos os argumentos do programa. A diferença entre estas variáveis reside no modo como se comportam quando ocorrem entre aspas:

- "$*" comporta-se como "$1 $2 $3 ..."
- "$@" comporta-se como "$1" "$2" "$3" ....

3.2.2 Outras variáveis especiais

$? Valor de retorno do último comando executado. Se o programa foi executado normalmente deve retornar 0. Em caso de erro deve retornar um valor diferente de 0.

$- Opções utilizadas na invocação da *shell*.

$$ Pid do processo corrente

$! Pid do último processo executado em *background*

3.3. Expansão do valor de variáveis

`$VAR`

valor da variável

`${VAR}`

valor da variável

`${VAR-qqcoisa}`

valor da variável se estiver definida, senão qqcoisa

`${VAR:-qqcoisa}`

valor da variável se estiver definida com um valor diferente de NULL, senão qqcoisa (esta variante com o caracter ':' aplica-se também às expansões seguintes)

`${VAR=qqcoisa}`

valor da variável se estiver definida, senão qqcoisa que passa também a ser o valor de VAR

`${VAR?mensagem}`

valor da variável se estiver definida, senão a mensagem é escrita para *stderr*

`${#VAR}`

tamanho de $VAR

4. Controlo de fluxo

De modo a privilegiar uma vertente mais prática, optou-se por apresentar nesta secção o controlo de fluxo. Nas observações de cada exemplo foram inseridas referências para as secções que descrevem assuntos adicionais.

4.1. if...then...else...fi

Sintaxe:

```
if condição
then lista de comandos
{elif condição
 then lista de comandos}
[else lista de comandos]
fi
```

Exemplo1:

```
#!/bin/sh
if [ $1 -lt 0 ]
then echo "$1 e' um numero negativo"
elif [ $1 -eq 0 ]
  then echo "$1 e' o numero zero"
  else echo "$1 e' um numero positivo"
fi
```

Observações:

- A avaliação de expressões (por exemplo:`[ $1 -lt 0 ]`) é descrita na secção 5.

Exemplo2:

```
[so000@falua exemplosShell]$ cat isUser.sh
#!/bin/sh
echo -n "Qual o nome do utilizador? "
read user
if grep $user /etc/passwd; then
  echo "$user tem uma área nesta máquina"
else
  echo "$user não tem uma área nesta máquina"
fi

[so000@falua exemplosShell]$ ./isUser.sh
Qual o nome do utilizador? maria
maria não tem uma área nesta máquina

[so000@falua exemplosShell]$ ./isUser.sh
Qual o nome do utilizador? so000
so000:x:2001:2000::/home/falua/so/so000:/bin/bash
so000 tem uma área nesta máquina
```

Observações:

- Neste exemplo é apresentada outra sintaxe que pode ser aplicada às várias estruturas de controlo de fluxo da *shell*.
- Qualquer comando pode ser utilizado como condição. Se o comando retornar 0, a condição é verdadeira, caso contrário é falsa.

- O resultado do comando `grep` está a ser escrito para `stdout`: "so000:x:2001:2000::/home/falua/so/so000:/bin/bash". Se quisermos omitir este resultado podemos redireccioná-lo para `/dev/null` – ver exemplo seguinte.

Exemplo3:

```
[so000@falua exemplosShell]$ cat isUser2.sh
#!/bin/sh
echo -n "Qual o nome do utilizador? "
read user
if grep $user /etc/passwd > /dev/null; then
  echo "$user tem uma área nesta máquina"
else
  echo "$user não tem uma área nesta máquina"
fi

[so000@falua exemplosShell]$ ./isUser2.sh
Qual o nome do utilizador? so000
so000 tem uma área nesta máquina

[so000@falua exemplosShell]$ ./isUser2.sh
Qual o nome do utilizador? s
s tem uma área nesta máquina
```

Observações:

- De modo a garantir a correcção dos resultados deste comando temos de garantir que a procura é efectuada no início das linhas do ficheiro `/etc/passwd` e delimitada pelo caracter separador dos vários campos que compõem este ficheiro, o caracter ':' – ver exemplo seguinte.

Exemplo4:

```
[so000@falua exemplosShell]$ cat isUser3.sh
#!/bin/sh
echo -n "Qual o nome do utilizador? "
read user
if grep ^$user: /etc/passwd > /dev/null; then
  echo "$user tem uma área nesta máquina"
else
  echo "$user não tem uma área nest máquina"
fi

[so000@falua exemplosShell]$ ./isUser3.sh
Qual o nome do utilizador? s
s não tem uma área nesta máquina
```

4.2. for...do...done

Sintaxe1:

```
for variável in lista
do
  lista de comandos
done
```

Exemplos:

```
[so000@falua exemplosShell]$ cat forLista.sh
#!/bin/sh
echo "argumentos da linha de comandos"
for i in $*  #idêntico a "for i"
do
  echo $i
done
```

```
[so000@falua exemplosShell]$ ./forLista.sh
argumentos da linha de comandos

[so000@falua exemplosShell]$ ./forLista.sh aa bb cc
argumentos da linha de comandos
aa
bb
cc
```

```
[so000@falua exemplosShell]$ cat tabuada1.sh
#!/bin/sh
echo -n "qual o número ? "
read n
for i in 1 2 3 4 5 6 7 8 9 10
do
  echo "$n * $i = `expr $i \* $n`"
done

[so000@falua exemplosShell]$ ./tabuada1.sh
qual o número ? 5
5 * 1 = 5
5 * 2 = 10
5 * 3 = 15
5 * 4 = 20
5 * 5 = 25
5 * 6 = 30
5 * 7 = 35
5 * 8 = 40
5 * 9 = 45
5 * 10 = 50
```

Observações:

- o exemplo seguinte melhora esta solução para o problema da tabuada,
- o comando `expr` é apresentado na secção 6. Nesta secção é também justificada a utilização de "*"
- O caracter especial "` ' utilizado em `` `expr $i * $n` `` é apresentado na secção 7.
- O comando `read` lê uma linha de `stdin` – ver `man bash` e secção 10.

Sintaxe2:

```
for ((expr1; expr2; expr3))
do
  lista de comandos
done
```

Exemplo:

```
[so000@falua exemplosShell]$ cat tabuada2.sh
#!/bin/sh
echo -n "qual o número ? "
read n
for ((i = 1; i<= 10; i++ ))
do
  echo "$n * $i = `expr $i \* $n`"
done

[so000@falua exemplosShell]$ ./tabuada2.sh
qual o número ? 7
```

```
7 * 1 = 7
7 * 2 = 14
7 * 3 = 21
7 * 4 = 28
7 * 5 = 35
7 * 6 = 42
7 * 7 = 49
7 * 8 = 56
7 * 9 = 63
7 * 10 = 70
```

4.3. while...do...done

Sintaxe:

```
while condição
do
  lista de comandos
done
```

Observação: O ciclo é efectuado enquanto a condição for verdadeira.

Exemplo:

```
[so000@falua exemplosShell]$ cat tabuada3.sh
#!/bin/sh
echo -n "qual o número ? "
read n
i=1
while test $i -le 10
do
  echo "$n * $i = `expr $i \* $n`"
  let i=$i+1
done

[so000@falua exemplosShell]$ ./tabuada3.sh
qual o número ? 9
9 * 1 = 9
9 * 2 = 18
9 * 3 = 27
9 * 4 = 36
9 * 5 = 45
9 * 6 = 54
9 * 7 = 63
9 * 8 = 72
9 * 9 = 81
9 * 10 = 90
```

Observações:

- o comando `let` permite avaliar expressões aritméticas – ver `man bash`.

4.4. until…do…done

Sintaxe:

```
until condição
do
   lista de commandos
done
```

Observação: O ciclo é efectuado enquanto a condição for falsa.

4.5. case...in...esac

Sintaxe:

```
case $nome_de_variável  in
   padrão1) commandos;;
   padrão2) comandos;;
   padrão3) comandos;;
   ...
    *) comandos por omissão;;
esac
```

Exemplo:

```
[so000@falua exemplosShell]$ cat case.sh
#!/bin/sh

echo "exemplo de utilização do case"
echo -n "quer continuar? Sim|Não "
read resp
case $resp in
        Sim|sim|S|s)    echo "para ver o shell-script utilize o comando cat";;
        Nao|nao|N|n)    echo "fim";;
        *)              echo "resposta inválida";;
esac

[so000@falua exemplosShell]$ ./case.sh
exemplo de utilização do case
quer continuar? Sim|Não s
para ver o shell-script utilize o comando cat

[so000@falua exemplosShell]$ ./case.sh
exemplo de utilização do case
quer continuar? Sim|Não n
fim

[so000@falua exemplosShell]$ ./case.sh
exemplo de utilização do case
quer continuar? Sim|Não g
resposta inválida
```

5. Avaliação de expressões : Comando `test` ou `[expressão]`

A avaliação de expressões pode ser efectuada de duas formas:

- `test expressão`
- [`expressão`]

Se a expressão for **verdadeira** é retornado o **valor zero**, caso a expressão seja falsa é retornado um valor diferente de zero.

O comando `test` ou `[expressão]` podem ser utilizados com:

- Inteiros
- Strings
- Ficheiros

Operadores para inteiros:

Operador	**Descrição**	**test expressão**	**[expressão]**
-eq	Igual	if test 5 –eq 6	if [5 -eq 6]
-ne	Diferente	if test 5 -ne 6	if [5 -ne 6]

-lt	Menor	if test 5 -lt 6	if [5 -lt 6]
-le	Menor ou igual	if test 5 -le 6	if [5 -le 6]
-gt	Maior	if test 5 -gt 6	if [5 -gt 6]
-ge	Maior ou igual	if test 5 -ge 6	if [5 -ge 6]

Operadores para Strings:

Operador	**Descrição**
string1 = string2	Igual
string1 != string2	Diferente
string1	String1 não é NULL ou não está definida
-n string1	String1 não é NULL e existe
-z string1	String1 é NULL e existe

Operadores para ficheiros:

Teste	**Descrição**
-a file	Verdadeiro se ficheiro existe
-d file	Verdadeiro se ficheiro existe e é uma directoria
-e file	Verdadeiro se ficheiro existe
-r file	Verdadeiro se ficheiro existe e tem permissões de leitura
-s file	Verdadeiro se ficheiro existe e não está vazio
-w file	Verdadeiro se ficheiro existe e tem permissões de escrita
-x file	Verdadeiro se ficheiro existe e tem permissões de execução
file1 -nt file2	Verdadeiro de file 1 é mais recente (de acordo com a data de alteração) do que o file2 ou se o file1 existe e o file2 não existe.

Para obter mais informações sobre outros testes que podem ser efectuados a ficheiros ver `man bash`, secção CONDITIONAL EXPRESSIONS.

Os operadores lógicos podem ser utilizados para juntar duas ou mais condições:

Operador	Significado
! expr	NOT lógico
exp1 –a expr2	AND lógico
exp1 –o expr2	OR lógico

6. Comando expr

Sintaxe:

```
expr expressão
```

Escreve para `stdout` o valor da expressão. A expressão pode ser:

`arg1 op arg2`, onde op pode ser <, >, <=, >=, !=, +, -, *, /

As comparações são aritméticas se ambos os argumentos forem números, caso contrário são lexicográficas.

Exemplos:

```
[so000@falua exemplosShell]$ expr 1 + 2
3
[so000@falua exemplosShell]$ expr 1 - 2
-1
[so000@falua exemplosShell]$ expr 1 * 2
expr: syntax error
[so000@falua exemplosShell]$ expr 1 \* 2
2
[so000@falua exemplosShell]$ expr 1 / 2
0
[so000@falua exemplosShell]$ expr 1 % 2
1
```

Observações:

- O caracter "*" é um metacaracter interpretado pela *shell* – ver metacaracteres no documento de introdução ao sistema operativo unix.

7. Outros caracteres especiais: &&, ||, ``

Os caracteres especiais "&&" e "||" são interpretados da seguinte forma:

- comando1 && comando2

  o comando2 apenas é executado se o comando1 retornar verdadeiro

- comando1 | | comando2

  o comando2 apenas é executado se o comando1 retornar falso

Observação:

- o comando2 não é executado se o valor de retorno do comando1 for suficiente para determinar o valor da sequência dos dois comandos.
- em *shell*, um resultado com valor 0 é considerado verdadeiro, um resultado com valor diferente de zero é considerado falso.

O caracter especial ‘`’ é interpretado da seguinte forma:

- `comando`

  a *shell* substitui `comando` pelos resultados escritos pelo comando em *stdout*

  Exemplos:

```
[so000@falua exemplosShell]$ a=2+3
[so000@falua exemplosShell]$ echo $a
2+3
[so000@falua exemplosShell]$ a=`expr 2 + 3`
[so000@falua exemplosShell]$ echo $a
5
[so000@falua exemplosShell]$ a=ls
[so000@falua exemplosShell]$ echo $a
ls
[so000@falua exemplosShell]$ a=`ls`
[so000@falua exemplosShell]$ echo $a
case.sh   forLista.sh   isUser2.sh   isUser3.sh   isUser.sh   tabuada1.sh
tabuada2.sh tabuada3.sh while.sh
```

8. Funções

Uma função é definida do seguinte modo:

```
Nome_de_função () {
  Lista de comandos
}
```

E é invocada da seguinte forma

```
Nome_de_função args
```

Os argumentos da função podem ser acedidos através de $1, $2,....

Exemplos:

```
[so000@falua ex]$ cat exemplof1
# funcao sum2
# objectivo: soma dois numeros inteiros
# parametros de entrada: dois numeros inteiros

sum2() {
x=`expr $1 + $2`
echo $x
}

sum2 2 3
sum2 10 12
[so000@falua ex]$ ./exemplof1
5
22
```

```
[so000@falua ex]$ cat exemplof2
# funcao sumN
# objectivo: calcula a soma dos numeros inteiros passados como argumentos
# parametros de entrada: numeros inteiros
# obs: se não forem passados números inteiros à função é apresentado em stdout o
# valor zero

sumN() {
x=0
for i in $*
do
  x=`expr $x + $i`
done
echo $x
}

sumN
sumN 10
sumN 10 12
sumN 1 3 9 5 9

[so000@falua ex]$ ./exemplof2
0
10
22
27
```

9. Debug de shell scripts

As opções –v e –x da *shell* podem ser utilizadas para se efectuar *debug* de *shell-scripts*.

Exemplos:

```
[so000@falua ex]$sh -v tabuada1.sh
#!/bin/sh
echo -n "qual o número ? "
qual o número ? read n
5
for i in 1 2 3 4 5 6 7 8 9 10
do
  echo "$n * $i = `expr $i \* $n`"
done
expr $i \* $n
5 * 1 = 5
expr $i \* $n
5 * 2 = 10
expr $i \* $n
5 * 3 = 15
expr $i \* $n
5 * 4 = 20
expr $i \* $n
5 * 5 = 25
expr $i \* $n
5 * 6 = 30
expr $i \* $n
5 * 7 = 35
expr $i \* $n
5 * 8 = 40
expr $i \* $n
5 * 9 = 45
expr $i \* $n
5 * 10 = 50
```

```
[so000@falua ex]$sh -x tabuada1.sh
+ echo -n 'qual o número ? '
qual o número ? + read n
7
++ expr 1 '*' 7
+ echo '7 * 1 = 7'
7 * 1 = 7
++ expr 2 '*' 7
+ echo '7 * 2 = 14'
7 * 2 = 14
++ expr 3 '*' 7
+ echo '7 * 3 = 21'
7 * 3 = 21
++ expr 4 '*' 7
+ echo '7 * 4 = 28'
7 * 4 = 28
++ expr 5 '*' 7
+ echo '7 * 5 = 35'
7 * 5 = 35
++ expr 6 '*' 7
+ echo '7 * 6 = 42'
7 * 6 = 42
++ expr 7 '*' 7
+ echo '7 * 7 = 49'
7 * 7 = 49
++ expr 8 '*' 7
+ echo '7 * 8 = 56'
7 * 8 = 56
++ expr 9 '*' 7
+ echo '7 * 9 = 63'
7 * 9 = 63
++ expr 10 '*' 7
+ echo '7 * 10 = 70'
7 * 10 = 70
```

10. Comandos úteis

shift

desloca à esquerda os valores dos argumentos que são dados na linha de comandos ($n)

read

lê valores de *stdin*

echo

escreve para *stdout*

basename

isola o nome do ficheiro, retirando o sufixo correspondente a directorias

dirname

isola o sufixo correspondente a directorias, retirando o nome do ficheiro

getopt

faz *parsing* da linha de comandos

sleep

efectua uma pausa de n segundos

touch

altera o tempo de acesso e de modificação de ficheiros

hostname

apresenta o nome da máquina

id

apresenta informação sobre o utilizador corrente

nohup

executa um comando ignorando sinais de *hangup* – ver `man -s 7 signal`

printf

formata e escreve dados para *stdout*

awk

este comando é utilizado para efectuar tarefas de processamento de texto. No exemplo seguinte, são lidas as linhas do ficheiro /etc/passwd e são escritos para *stdout* os nomes e os uid dos utilizadores. A opção –F permite definir o separador dos campos.

Exemplo:

```
[so000@falua exemplosShell]$ awk -F: '{print $1, $3}' /etc/passwd
```

sed este comando é utilizado para efectuar alterações em texto, por exemplo, para efectuar substituição de strings. O exemplo seguinte substitui todas as ocorrências de "yes" por "sim" e escreve o resultado em *stdout*.

Exemplo:

```
[so000@falua exemplosShell]$ sed -e 's/yes/sim/g'
```

find procura ficheiros em directorias

sintaxe: `find [<directorias>] [opções] [testes] [acções]`

Alguns exemplos de testes mais utilizados:

- `name padrão` – procura ficheiros cujo nome corresponde ao padrão
- `iname padrão` – idêntico ao teste "`name`" mas não diferencia letras minúsculas de letras maiúsculas

Alguns exemplos de acções mais utilizadas:

- `print` – escreve para `stdout` o nome completo do ficheiro
- `exec comando` – executa o comando – ver exemplos e `man find`
- `printf formato` – escreve para `stdout` a informação definida no formato – ver exemplos
- `ls` – executa o comando `ls` sobre o ficheiro

Exemplos:

```
[so000@falua aula2]$ find . -name TRAB1 -print
./so001/TRAB1
./so002/TRAB1
./so004/TRAB1
./so005/TRAB1
./so008/TRAB1
```

```
[so000@falua aula2]$ find . -name fichaEntrega -ls
1055016 4 -rw-r--r-- 1 so000 so 6 Sep 8 12:42 ./so001/TRAB1/fichaEntrega
1055105 4 -rw-r--r-- 1 so000 so 15 Sep 8 12:42 ./so002/TRAB1/fichaEntrega
1055125 4 -rw-r--r-- 1 so000 so 15 Sep 8 12:42 ./so004/TRAB1/fichaEntrega
1055126 4 -rw-r--r-- 1 so000 so 4 Sep 8 12:42 ./so005/TRAB1/fichaEntrega
1055127 4 -rw-r--r-- 1 so000 so 6 Sep 8 12:42 ./so008/TRAB1/fichaEntrega
```

```
[so000@falua aula2]$ find . -name fichaEntrega -printf "%s "
6 15 15 4 6
```

```
[so000@falua aula2]$ find . -name fichaEntrega -exec cat {} \;
ficha
ficha de so002
ficha de so004
ola
hello
```

```
[so000@falua aula2]$ find . -name fichaEntrega -print -exec cat {} \;
./so001/TRAB1/fichaEntrega
ficha
./so002/TRAB1/fichaEntrega
ficha de so002
./so004/TRAB1/fichaEntrega
ficha de so004
./so005/TRAB1/fichaEntrega
ola
./so008/TRAB1/fichaEntrega
hello
```