Programação em BASH

Sistemas Operativos I
99/00

Orlando Sousa
Departamento de Engenharia Informática
Instituto Superior de Engenharia do Porto

Programação em BASH

A programação da *shell* (neste caso é a BASH) permite que o computador execute uma sequência de comandos de um modo *automático* (sem ser o utilizador a efectuar este processo). A *script* contém os nomes dos comandos que vão ser executados (a *shell* executa esses comandos).

Criação de uma script

Para criar uma *script*, é necessário utilizar um editor de texto onde vamos escrever a sequência de comandos que se pretende executar. Considere que o ficheiro **fich1** contém o seguinte:

```
#!/bin/bash
# Esta script procura em todos os ficheiros do directório actual a
# string Sistemas e mostra o conteúdo de
# cada ficheiro que contenha essa string.

for ficheiro in *
do
      if grep –l Sistemas $ficheiro
      then
            more $ficheiro
      fi
done
```

Os comentários começam com # e continuam até ao fim da linha. Normalmente este símbolo é colocado na primeira coluna de cada linha. O comentário **#!/bin/bash** é um comentário especial, onde **#!** *informa* o sistema que o argumento que se segue é o programa que vai ser utilizado para executar este ficheiro (neste caso é /bin/bash).

Para executar uma *script*, podemos utilizar duas formas diferentes. A primeira é invocar a *shell* com o nome da *script* como parâmetro.

Exemplo:

```
$  /bin/bash fich1
```

A outra maneira de executar *scripts* é escrever apenas o nome da *script* que queremos executar. Para isso o ficheiro deve ter permissão de *execução*.

Exemplo:

```
$ chmod +x fich1

$ fich1
```

Nota: Também é possível obter o mesmo resultado que a *script* anterior em apenas uma linha de comandos. Para isso, pode-se utilizar:

```
$ more `grep –l Sistemas *`
```

ou

```
$ more $(grep –l Sistemas *)
```

Primeiro é feito o **grep –l Sistemas ***, e só depois a *shell* executa o comando **more** sobre o resultado obtido.

Variáveis

Para utilizar variáveis não é necessário declará-las primeiro. As variáveis são criadas quando se utilizam, isto é, quando lhes atribuímos um valor inicial. Por defeito, todas as variáveis são consideradas do tipo *string*, mesmo aquelas

que têm valores numéricos. A *shell* converte a "string numérica" em valores numéricos de modo a fazer o que é necessário. Um factor a ter em conta é que os nomes das variáveis são *case-sensitive*.

Para se ter acesso ao conteúdo de uma variável temos de utilizar o símbolo **$** antes do nome da variável e utilizar o comando **echo** para mostrar esse conteúdo no monitor. Em todas as situações que se utilizam as variáveis é necessário utilizar o símbolo **$**, excepto quando se trata de atribuir um valor a essa variável.

Exemplos:

```
$ valor=Sistemas
$ echo $valor
Sistemas
$ valor="Sistemas Operativos I"
$ echo $valor
Sistemas Operativos I
$ valor=4+5
$ echo $valor
4+5
$
```

Quando se utilizam espaços, é necessário utilizar *aspas* como delimitador da *string*. A funcionalidade das variáveis como **$valor** depende do tipo de aspas que estamos a utilizar. Se a variável está entre aspas (") (ex:"$valor"), então é substituída pelo valor da variável quando a linha for executada. Se a variável está entre aspas (') então não é efectuada nenhuma substituição. Para remover o significado especial de **$** é necessário utilizar antes desse símbolo a barra invertida (\).

Também se pode atribuir um valor a uma variável através do comando **read**.

Exemplo:
A *script*:

```
#!/bin/bash

valor = "Sistemas Operativos I"

echo $valor
echo "$valor"
echo '$valor'
echo  \$valor

echo Introduza texto:
read valor

echo '$valor' foi modificado para $valor
```

dá como resultado:

```
Sistemas Operativos I
Sistemas Operativos I
$valor
$valor
Introduza texto:
Exame de SO1
$valor foi modificado para Exame de SO1
```

Variáveis Ambiente

Quando uma *shell script* é executada, algumas variáveis são inicializadas com os valores do seu *ambiente*. Os nomes das variáveis ambiente são em maiúsculas para se destinguirem das variáveis definidas nas *scripts* (que normalmente são em minúsculas).

Variável Ambiente	Descrição
$HOME	Directório *home* do utilizador
$PATH	Lista de directórios separados por **:**
$PS1	Prompt (normalmente é **$**)
$PS2	Prompt secundário (normalmente é >)
$IFS	Input Field Separator. Lista de caracteres utilizada para separar palavras.
$0	Nome da *shell script*
$#	Número de parâmetros da *shell script*
$$	PID (*Process IDentification*) da *shell script*

Parâmetros

Se a *script* é invocada com parâmetros, são criadas algumas variáveis adicionais. Mesmo que não se *passem* parâmetros, a variável **$#** continua a existir, mas obviamente com o valor **0**.

As variáveis que permitem *trabalhar* os parâmetros são:

Variável	Descrição
$1, $2, $3, …	Os parâmetros da *script*
$*	Lista com todos os parâmetros, numa única variável, separados pelo primeiro caracter da variável ambiente **IFS**
$@	Semelhante ao **$***, só que não utiliza a variável ambiente **IFS**

Exemplo (considere que o nome da *script* é **script_variaveis)** :

```
#!/bin/bash

valor="Sistemas"
echo $valor
echo "O programa  $0 está a ser executado"
echo "O segundo parâmetro é $2"
echo "O primeiro parâmetro é $1"
echo "O terceiro parâmetro é $3"
echo "A lista de parâmetros é $*"
```

Se executar a *script*, obtém o seguinte:

```
$ script_variaveis exame sistemas operativos
Sistemas
O programa script_variaveis está a ser executado
O segundo parâmetro é sistemas
O primeiro parâmetro é exame
O terceiro parâmetro é operativos
A lista de parâmetros é exame sistemas operativos
```

Execução de Comandos

Para executar comandos utiliza-se **$(comando)** ou **`comando`**. O resultado é a *saída* do respectivo comando (não é o estado do comando).

Considere a *script*:

```
#!/bin/bash

echo A variável PATH é $PATH
echo Os utilizadores que estão no sistema são $(who)
```

Esta *script* mostra o conteúdo da variável PATH, bem como os utilizadores que estão actualmente no sistema. Tenha em atenção que foi necessário executar o comando **who**.

O conceito de colocar o resultado da execução de um comando numa variável é muito poderoso.

Condições

Um dos factores essenciais em todas as linguagens de programação é a possibilidade de testar condições e fazer determinadas *acções* em função dessas condições.

O comando *test* e o comando *[ ]*

Estes dois comandos servem para testar condições e são equivalentes.

Para perceber a funcionalidade do comando **test**, vamos verificar se o ficheiro **fich.c** existe, e se existe apresentar o seu conteúdo. O comando para verificar essa condição é **test –f <ficheiro>** , portanto na *script* teremos:

```
#!/bin/bash
if test –f fich.c
then
      more fich.c
fi
```

Também podemos utilizar o comando **[ ]** para obter a mesma funcionalidade:

```
#!/bin/bash
if [ –f fich.c ]
then
      more fich.c
fi
```

Nota: Na utilização do comando **[ ]** é necessário existir um espaço depois de **[** , e um espaço antes de **]**.

Comparação de *strings*

Comparação	Resultado
String	**Verdade**, se a *string* não é vazia
String1 = string2	**Verdade**, se as *strings* são iguais
String1 != string2	**Verdade**, se as *strings* são diferentes
-n string	**Verdade**, se a *string* não é **nula**
-z string	**Verdade**, se a *string* é nula

Comparações Aritméticas:

Comparação	Resultado
Expressão1 –eq expressão2	**Verdade**, se forem iguais
Expressão1 –ne expressão2	**Verdade**, se as expressões são diferentes
Expressão1 –gt expressão2	**Verdade**, se expressão1 > expressão2
Expressão1 –ge expressão2	**Verdade**, se expressão1 ≥ expressão2
Expressão1 –lt expressão2	**Verdade**, se expressão1 < expressão2
Expressão1 –le expressão2	**Verdade**, se expressão1 ≤ expressão2
!expressão	Nega a expressão. *Retorna* **Verdade** se a expressão é **falsa**

Condições em ficheiros:

Comparação	Resultado
-d ficheiro	**Verdade**, se o directório existe
-f ficheiro	**Verdade**, se o ficheiro existe
-r ficheiro	**Verdade**, se é possível ler o ficheiro
-s ficheiro	**Verdade**, se o ficheiro tem um tamanho > 0
-w ficheiro	**Verdade**, se é possível *escrever* no ficheiro
-x ficheiro	**Verdade**, se é possível *executar* o ficheiro

Estruturas de Controlo

if - testa o resultado de um comando e executa condicionalmente um grupo de comandos.

```
if condição
then
      comando1
      comando2
      …
      comandon
else
      comando1
      …
      comandon
fi
```

Nota: Para utilizar o **then** na mesma linha do **if** é necessário acrescentar **;** depois da condição.

Considere o seguinte exemplo, que faz uma decisão baseado numa resposta:

```
#!/bin/bash

echo “Passou no exame? “
read resposta

if [ $resposta = “sim” ]; then
      echo “Parabens!”
else
      echo “Não estudou !!!”
fi
```

elif - A *script* anterior tem um problema – aceita qualquer resposta, excepto a resposta **sim** para escrever **Não estudou**. Para resolver esta situação podemos utilizar o comando **elif**, que permite testar uma segunda condição quando o **else** é executado.

```
#!/bin/bash

echo “Passou no exame? “
read resposta

if [ $resposta = “sim” ]; then
      echo “Parabéns!”
elif [ $resposta = “não” ]; then
      echo “Não estudou !!!”
else
      echo “Não conheço a resposta $resposta. Introduza sim ou não!”
fi
```

for – executa um ciclo um determinado número de vezes (em função de um conjunto de valores). Esses valores podem ser especificados na *script*, ou serem o resultado da *expansão* de comandos.

```
for variável in valores
do
      comando 1
      …
      comando n
done
```

considere o seguinte exemplo:

```
#!/bin/bash

for valor in exame de sistemas “SO1 – teste” operativos
do
      echo $valor
done
```

dá como resultado:

```
exame
de
sistemas
SO1 – teste
operativos
```

Considere o seguinte exemplo;

```
#!/bin/bash

for valor in $(ls so[123].txt)
do
      more $valor
done
```

Este exemplo mostra o conteúdo dos ficheiros que são o resultado de executar **ls so[123].txt**, isto é, mostra o conteúdo dos ficheiros so1.txt, so2.txt so3.txt se existirem.

O ciclo **for** funciona bem quando se trata de situações em que temos um conjunto de *strings*. Quando é necessário executar um grupo de comandos um número *variável* de vezes, este comando não é o mais aconselhado.

Considere a *script*:

```
#!/bin/bash

for valor in 1 2 3 4 5 6 7 8 9 10
do
      echo “Sistemas Operativos”
done
```

Esta *script* escreve **dez** vezes "Sistemas Operativos". Também obteríamos o mesmo resultado se depois de **in** tivéssemos qualquer valor **dez** vezes seguidas (ex: a a a a a a a a a a).

while - o ciclo **while** é útil nas situações em que não existe um número **fixo** de vezes para executar um determinado grupo de comandos.

Estrutura do comando **while**:

```
while condição do
      comando 1
      …
      comando n
done
```

Considere a *script*:

```
#!/bin/bash

echo "Introduza um nome: "
read nome

while [ "$nome" != "Sistemas" ]; do
      echo “Não acertou no nome - tente de novo !”
      read nome
```

```
done
```

Esta *script* só termina quando o utilizador introduzir o nome correcto. Enquanto introduzir o nome errado, vai ter que introduzir um novo nome.

Nota: A utilização de aspas (") em **["$nome" != "Sistemas"]** permite salvaguardar a situação em que o utilizador utiliza o *Enter* sem introduzir mais nada (nesta situação a condição de teste ficaria **[!= "Sistemas"]**, que não é uma condição válida). Com a utilização de aspas o problema é resolvido, pois a condição de teste será **["" != "Sistemas]**.

Considere a *script*:

```
#!/bin/bash

valor=1

while [ $valor -le 10 ]
do
      echo “Sistemas Operativos”
      valor=$(($valor + 1))
done
```

Esta *script* escreve "Sistemas Operativos" **dez** vezes. Para isso utiliza a variável **valor** que vai sendo incrementada.

until - é semelhante ao ciclo **while**. A única diferença é que o teste da condição é feito no fim, isto é, o ciclo continua até que a condição seja **verdade**.

Estrutura do comando **until**

```
until condição
do
      comando 1
      …
      comando n
done
```

Considere a *script*:

```
#!/bin/bash

until who | grep "$1" >/dev/null
do
      sleep 10
done

echo *** O utilizador $1 entrou no sistema ! ***
```

Esta *script* verifica se um determinado utilizador entrou no sistema, isto é, de 10 em 10 segundos verifica se o utilizador está no sistema. Quando o utilizador entrar no sistema a *script* termina.

case - permite verificar o conteúdo de uma variável em relação a vários *padrões*, executando depois os respectivos comandos.

Estrutura do comando **case**:

```
case variável in
      padrão [| padrão …]) comandos;;
      padrão [| padrão …]) comandos;;
      …
esac
```

Considere a *script*:

```
#!/bin/bash

echo “Passou no exame? “
read resposta
```

```
case "$resposta" in
      "sim") echo "Parabéns!" ;;
      "não") echo "Não estudou !!!" ;;
      "s"  ) echo "Parabéns!" ;;
      "n"  ) echo "Não estudou !!!" ;;
      *    ) echo “Não conheço a resposta $resposta!” ;;
esac
```

A *script* compara o conteúdo de **resposta** com todos os *padrões* (quando se *verifica* um dos padrões o comando **case** termina a *procura*). O asterisco (*) pode ser utilizado para expandir *strings.* Neste exemplo, o asterisco faz concordância (*matching*) de todas as *strings*, permitindo assim executar uma acção por defeito (quando nenhum dos outros padrões se verificou).

Obtemos a mesma funcionalidade com a *script*:

```
#!/bin/bash

echo “Passou no exame? “
read resposta

case "$resposta" in
      "sim" | "s" ) echo "Parabéns!" ;;
      "não" | "n" ) echo "Não estudou !!!" ;;
      *    ) echo “Não conheço a resposta $resposta!” ;;
esac
```

Listas de Comandos

Para executar uma lista de comandos em que só é necessário executar o comando seguinte se o comando anterior foi bem sucedido, faz-se o seguinte:

```
comando1 && comando2 && comando3 && …
```

O **comando2** só é executado se o **comando1** teve sucesso; o **comando3** só é executado se o **comando2** teve sucesso, etc.

Para executar-mos uma série de comandos até que um tenha sucesso, faz-se o seguinte:

```
comando1 || comando2 || comando3 || …
```

Se o **comando1** tem sucesso, já não é executado mais nenhum comando da lista. Se o **comando1** falhou, então é executado o **comando2**; Se o **comando2** tem sucesso então termina; Se o **comando2** falhou então é executado o **comando3**, etc.

Nota: Para se utilizar um grupo de comandos em situações em que só um comando é permitido (como é o caso das *listas de comandos*), temos de agrupar esses comandos entre **{ }**.

Funções

As funções têm a seguinte estrutura:

```
nome_da_função () {
      comando1
      …
      comandon

}
```

Considere a *script*:

```
#!/bin/bash

escreve () {
      echo "A função está a ser executada "
}

echo "Início da script"
escreve
echo "Fim da script"
```

Embora a definição da função esteja no princípio da *script*, a *script* só começa a executar os comandos que estão depois da definição da função. Quando se *chama* uma função, a *shell* executa-a e depois *retorna* para a linha seguinte à da função. É necessário ter em atenção que é necessário definir a função antes de utilizá-la, isto é, o *código* das funções deve ser colocado no princípio da *script*. Quando uma função é *invocada*, os parâmetros da *script* **$***, **$@**, **$#**, **$1**, **$2**, etc. são substituídos pelos parâmetros da função. Quando a função termina a sua execução, os parâmetros são restaurados.
Para que a função *retorne* um valor numérico, é necessário utilizar o comando **return**. A única maneira de *retornar strings* é utilizar uma variável *global*, de modo a ser possível utilizá-la quando a função terminar a sua execução. Para declarar variáveis *locais* à função, utiliza-se a palavra **local** antes da variável.

Considere a *script*:

```
#!/bin/bash

texto="Variável global"

escreve () {
      local texto="Variável local"
      echo "A função está a ser executada"
      echo $texto
}

echo "Início da script"
echo $texto
escreve
echo $texto
echo "Fim da script"
```

A *script* dá o seguinte resultado:

```
Início da script
Variável global
A função está a ser executada
Variável local
Variável global
Fim da script
```

Quando não se utiliza o comando **return** na função, a função *retorna* o estado do último comando que foi executado.

Considere a *script* ***teste*** :

```
#!/bin/bash

pergunta() {
      echo "Os parâmetros da função são $*"
      while true
            echo -n "sim ou não"
            read resposta
            case "$resposta" in
                  s | sim ) return 0;;
                  n | não ) return 1;;
                  * )       echo "Responda sim ou não"
            esac
      done
}
```

```
echo "Os parâmetros da script são $*"

if pergunta "O nome é $1 ?"
then
      echo "Olá $1"
else
      echo "Engano"
fi
```

Esta *script passa* parâmetros para a função. A função retorna valores numéricos.

Exemplo de utilização da *script* anterior:

```
$ teste Orlando Sousa

Os parâmetros da script são Orlando Sousa
Os parâmetros da função são O nome é Orlando Sousa ?
sim ou não
não
Engano
```

Comandos

break - é utilizado para *saír* de um ciclo **for**, **while** ou **until**.

exemplo:

```
#!/bin/bash

for ficheiro in so*
do
      if [ -d "$ficheiro" ]; then
            break;
      fi
done

echo O primeiro directório com iniciais so é $ficheiro
```

Esta *script* mostra o nome do primeiro directório com iniciais **so**.

continue - Avança para a próxima iteração do ciclo **for**, **while** ou **until**.

Exemplo:

```
#!/bin/bash

for ficheiro in so*
do
      if [ -d "$ficheiro" ]; then
            continue
      fi
      echo $ficheiro
done
```

Esta *script* apenas mostra os nomes de ficheiros que tenham como iniciais **so** (não mostra os directórios).

echo - mostra o conteúdo de uma *string* (seguido de *newline*). Para evitar o *newline* é necessário utilizar a opção **-n**.

eval - serve para *avaliar* argumentos.

A *script*:

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

```
valor=5
x=valor
y='$'$x
echo $y
```
dá como resultado **$valor**.

Mas a *script*:
```
#!/bin/bash

valor=5
x=valor
eval y='$'$x
echo $y
```
dá como resultado **5**, isto é, dá o valor do valor da variável.

export - faz que uma variável fique *visível*, isto é, cria uma variável ambiente.

Considere as seguintes *scripts*:

teste2:
```
#!/bin/bash

echo $valor
echo $resposta
```

teste1:
```
#!/bin/bash

valor="Variável que não utiliza export"
export resposta="Variável que utiliza export"

teste2
```

se executarmos a *script* **teste1**, dá:

```
Variável que utiliza export
```

Como a *script* **teste1** *chama* a *script* **teste2**, apenas é visível a variável **resposta** na *script* **teste2**.

expr - *avalia* argumentos de uma expressão. É normalmente utilizado para cálculos aritméticos.

Exemplo: valor = `expr $valor + 1`

Este exemplo coloca em **valor** o resultado de executar o comando **expr $valor + 1.**

Expressão	**Descrição**
Expressão1 | expressão2	Expressão1, se é diferente de zero; senão expressão2
Expressão1 & expressão2	Zero, se uma ou ambas as expressões são zero
Expressão1 = expressão2	Igualdade
Expressão1 != expressão2	Diferentes
Expressão1 > expressão2	
Expressão1 ≥ expressão2	
Expressão1 < expressão2	
Expressão1 ≤ expressão1	
Expressão1 + expressão2	Adição
Expressão1 - expressão2	Subtracção
Expressão1 * expressão2	Multiplicação
Expressão1 / expressão2	Divisão inteira
Expressão1 % expressão2	Resto da divisão

Nota: Em substituição do comando **expr** normalmente utiliza-se **$((…))** , que é mais eficiente (também se pode utilizar **$[…]**).

printf - é utilizado para *formatar* a *saída*.

A sintaxe para este comando é:

```
printf "formato da string" parâmetro1 parâmetro2 …
```

O formato da *string* é semelhante ao formato utilizado na linguagem C, com algumas restrições (Só suporta valores inteiros, pois a *shell* faz todas as suas operações sobre valores inteiros).

set - permite configurar as variáveis da *shell*. É útil como meio de usar *campos* nos comandos que dão como resultado valores separados por *espaço*.

Considere a *script*:

```
#!/bin/bash

echo A data é $(date)
set $(date)
echo O mês da data é $2
```

Como o resultado de executar o comando **date**, dá uma *string* (ex: Mon Jan 17:22:57 MET 1999), apenas o segundo campo (que contém o mês) é apresentado no segundo **echo**.

shift - o comando **shift** retira um parâmetro aos parâmetros da *script* (ex: **$2** torna-se o **$1**, o **$3** torna-se o **$2**, etc). O **shift** é utilizado para pesquisar os parâmetros.

```
#!/bin/bash

while [ "$1" != ""]; do
      echo $1
      shift
done
```

Esta *script* mostra todos os parâmetros introduzidos.

Expansão de Parâmetros

A expansão de parâmetros é muito útil na manuseamento de partes desses parâmetros.

Suponha que precisa de uma *script* que processe o ficheiro **1.tmp** e o **2.tmp**.

A *script* que estaria tentado a fazer possivelmente seria:

```
#!/bin/bash

for valor in 1 2
do
      processa $i.tmp
done
```

Esta *script* não funciona, pois o que a *shell* está a tentar fazer é substituir o valor da variável **$i.tmp**, que não existe. Para proteger a expansão da variável é necessário que o **i** entre **{ }**.

A *script* correcta é:

```
#!/bin/bash

for valor in 1 2
do
      processa ${i}.tmp
done
```

Em cada iteração o valor de **i** é substituído por **${i}** .

Expansão de Parâmetros	Descrição
${parâmetro:-valor}	Se **parâmetro** é nulo então dá como resultado **valor**
${#parâmetro}	Tamanho do parâmetro
${parâmetro%palavra}	Do fim, remove a parte mais pequena que contenha **palavra** e *retorna* o resto.
${parâmetro%%palavra}	Do fim, remove a parte mais longa que contenha **palavra** e *retorna* o resto.
${parâmetro#palavra}	Do inicio, remove a parte mais pequena que contenha **palavra** e *retorna* o resto.
${parâmetro##palavra}	Do inicio, remove a parte mais longa que contenha **palavra** e *retorna* o resto.

Exemplo:

```
#!/bin/bash

echo ${valor:-Vazio}
valor=Cheio
echo ${valor:-Vazio}

valor=/usr/bin/X11/startx
echo ${valor#*/}
echo ${valor##*/}

valor=/usr/local/etc/local/networks
echo ${valor%local*}
echo ${valor%%local*}
```

dá como resultado:

```
Vazio
Cheio
usr/bin/X11/startx
startx
/usr/local/etc
/usr/
```

A *script* seguinte muda todos os ficheiros com extensão **.txt** para **.doc**:

```
#!/bin/bash
for ficheiro in *.txt
do
      mv $ficheiro ${ficheiro%txt}doc
done
```