INSTITUTO POLITÉCNICO
DE VIANA DO CASTELO
Escola Superior
de Tecnologia e Gestão

Trabalho Prático no. 6

Introdução à programação em *shell script*

Objectivos

- Saber identificar uma “shell script”
- Conhecer e utilizar as estruturas de controlo de fluxo “if” e “case”
- Conhecer e utilizar as estruturas de controlo de ciclos “while”, “until” e “for”
- Construir pequenas “shell scripts”.

“Shell scripts”

- Uma “script” é um ficheiro de texto que contém uma sequência de comandos para uma determinada “shell” (interpretador de comandos).
- As “shell scripts” são ferramentas muito poderosas ao permitem automatizar a maior parte das tarefas de administração do sistema e de utilização diária.

Escolha da "shell" a utilizadar

- A primeira linha de uma "script" costuma indicar o nome do interpretador de comandos a utilizar.

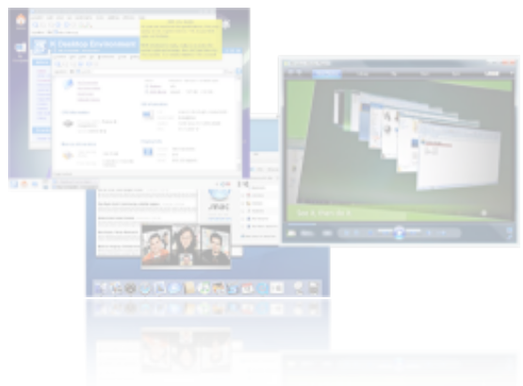
- Por isso, começa com os caracteres **#!** seguidos do nome do executável do interpretador, com endereçamento absoluto.

- No caso da Bourne Again Shell temos:

  #!/bin/bash

Escolha da “shell” a utilizadar (termo)

- Se esta linha for omitida, será usada a “shell” standard do sistema (por omissão).
- O ficheiro ‘**/etc/shells**’ possui uma lista com todas as “shells” instaladas no sistema. A primeira linha indica a “shell” standard.



“Input” e “output”

- Dentro de uma “script” usa-se o comando “echo” para enviar dados para o “standard output”.
- O comando “read” é usado para ler valores do teclado (standard input) e atribuí-los a variáveis.

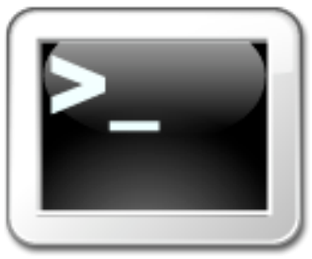
“Input” e “output” (termo)

- Nas “shell scritps” podemos combinar os comandos “echo” e “read” para pedir ao utilizador que introduza um valor e, depois, atribuí-lo a uma variável.

- Por exemplo, seja a “script” **saudar**:

```
#!/bin/bash
echo “Introduza uma saudação”
read SVAR
echo “A saudação que introduziu foi \”$SVAR\””
```

Execução de uma “shell script”



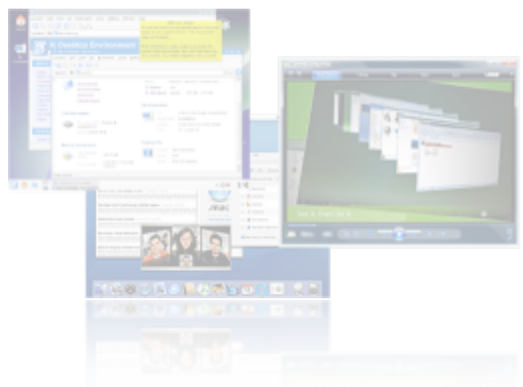
- Para executar a “shell script” **saudar** podemos fazê-lo de 3 formas:

1. Executá-la invocando o nome da “shell”:

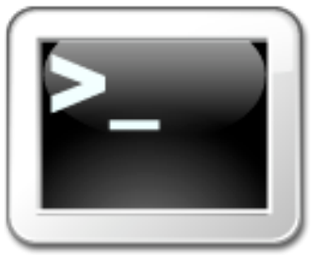
   $ bash saudar

2. Executá-la invocando a “shell standard”:

   $ source saudar



Execução de uma "shell script" (termo)

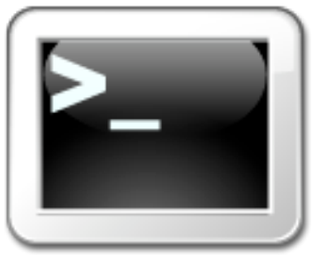


3. Transformá-la num programa executável e executá-la:

$ chmod a+x saudar

$ ./saudar

Passagem de argumentos

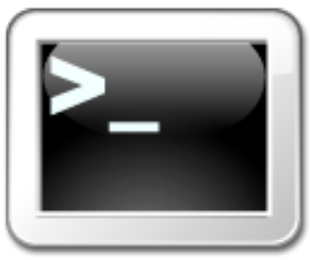


- Uma "shell script" pode ser executada com argumentos. Isto é, podem introduzir-se argumentos na linha de comandos após o nome da "shell script".

- Para os referenciar utiliza-se o operador $ seguido do número que indica a sua posição na linha de comandos:

```
$0 - nome próprio da script
$1 - primeiro argumento
$2 - segundo argumento
…
$9 - nono argumento
${10} - décimo argumento
```

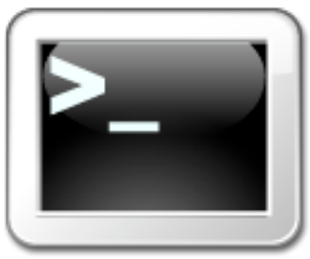
Passagem de argumentos (termo)



- Outras características da linha de comandos podem ser vistas com:

```
$* - todos os argumentos da linha de comandos
$# - número de argumentos da linha de comandos
```

Operações aritméticas



- O comando "**let**" permite realizar operações com valores aritméticos, sem ser necessário o uso do carácter $ para indicar o valor da variável:

```
let 2*7                ; imprimirá 14
let "RES = 2 * 7"      ; atribuirá à variável RES o valor 14
let "RES = RES + 1"    ; incrementará a variável RES
let "RES < 6"          ; verifica se o valor da variável RES é inferior a 6
```

Operações aritméticas (termo)

- Outra forma de realizar operações aritméticas é utilizar os comandos "**$[**" e "**]**" a limitar as expressões matemáticas.
- Neste caso, as variáveis usam-se com o carácter **$**:

```
RES=$[ 2 * 7 ]      ; atribuirá à variável RES o valor 14
RES=$[$RES + 1]     ; incrementará a variável RES
```

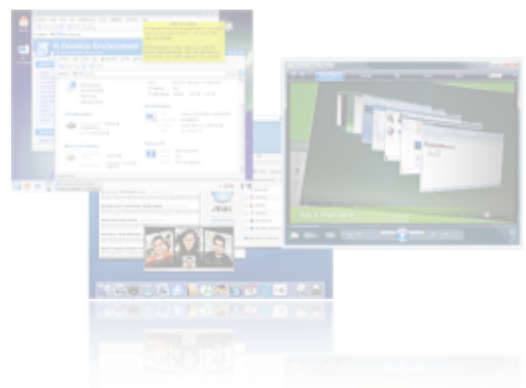
O comando "*test*"

- Com o comando "*test*" podemos comparar valores inteiros, strings e realizar operações lógicas. A sua sintaxe é:

```
test valor1 -option valor2
test string1 operador string2
```

- Em vez da palavra "*test*" podemos utilizar os parêntesis rectos:

```
test $NUM -eq 10   pode ser escrito como:   [ $NUM -eq 10 ]
```



O comando "*test*" (cont.)

- É de notar que, tem de se colocar sempre espaços antes e depois dos parêntesis rectos.
- As operações válidas são:

Comparações inteiras	Função
n1 –**gt** *n2*	verdadeiro se *n1* superior a *n2*
n1 –**lt** *n2*	verdadeiro se *n1* inferior a *n2*
n1 –**ge** *n2*	verdadeiro se *n1* igual ou superior a *n2*
n1 –**le** *n2*	verdadeiro se *n1* inferior ou igual a *n2*
n1 –**eq** *n2*	verdadeiro se *n1* igual a *n2*
n1 –**ne** *n2*	verdadeiro se *n1* diferente de *n2*

O comando "*test*" (termo)

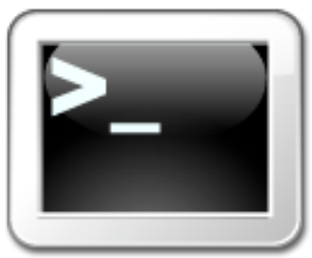
Comparação de strings	
-**z** *str1*	verdadeiro se string *str1* vazio
-**n** *str1*	verdadeiro se string *str1* não é vazio
str1 = *str2*	verdadeiro se string *str1* igual a string *str2*
str1 **!=** *str2*	verdadeiro se string *str1* diferente de string *str2*
str *str1*	verdadeiro se string *str1* não é nulo

Operações lógicas	
! *expr*	NOT lógico - verdadeiro se *expr* falso
expr1 –**a** *expr2*	AND lógico - verdadeiro se ambas as expressões (*expr1* e *expr2*) verdadeiras
expr1 –**o** *expr2*	OR lógico - verdadeiro se uma das expressões (*expr1* ou *expr2*) verdadeira

Teste a ficheiros	
-**f** *filename*	verdadeiro se *filename* existe e é ficheiro
-**r** *filename*	verdadeiro se *filename* tem permissão de leitura
-**w** *filename*	verdadeiro se *filename* tem permissão de escrita
-**x** *filename*	verdadeiro se *filename* tem permissão de execução
-**d** *filename*	verdadeiro se *filename* existe e é directoria
-**h** *filename*	verdadeiro se *filename* existe e é *link*

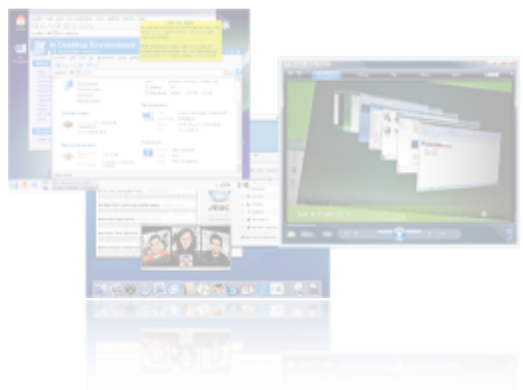
Objectivos

- Saber identificar uma “shell script”
- Conhecer e utilizar as estruturas de controlo de fluxo “if” e “case”
- Conhecer e utilizar as estruturas de controlo de ciclos “while”, “until” e “for”
- Construir pequenas “shell scripts”.



Estruturas de controlo

- As estruturas de controlo utilizam-se para repetir comandos ou decidir sobre que comandos devem ser executados.
- Uma estrutura de controlo é composta por 2 componentes: teste e comandos.
- Se o teste for verdadeiro então os comandos são executados.



Estruturas de controlo (cont.)

- As três estruturas de controlo de ciclos da bash são: "while", "until" e "for".
- As duas estruturas de condição são: "if" e "case".
- As estruturas "if", "while" e "until" têm, como teste, a execução de um comando. Todos os comandos devolvem um estado de retorno (exit status) após a sua execução.

Estruturas de controlo (cont.)

- Se o comando for bem sucedido retorna "0" (zero), senão retorna um valor positivo.
- No caso das estruturas "if" e "while", se o "exit status" do comando for "0", então o comando foi bem sucedido e os comandos da estrutura são executados:

```
while condição; do
    comandos
done
```

```
if [ condition ]
then
        action
elif [ condition2 ]
then
        action2
.
.
.
elif [ condition3 ]
then

else
        actionx
fi
```

Estruturas de controlo (cont.)

- No caso da estrutura "until" enquanto o "exit status" do comando não for "0" os comandos da estrutura são executados:

```
until condição; do
    comandos
done
```

- A estrutura "case" é uma forma restrita da condição "if" e é normalmente utilizada para implementar menus:

Estruturas de controlo (cont.)

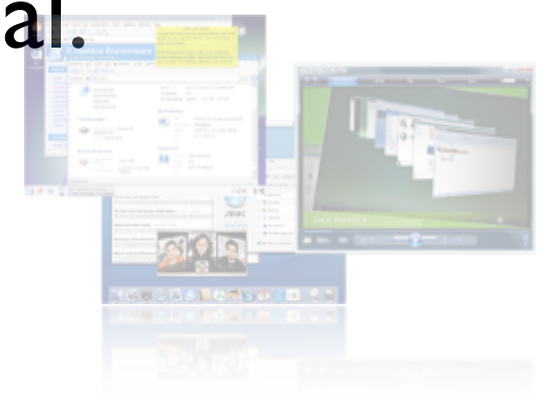
```
case expressão in
   padrão1)
          comandos
          ;;
   padrão2)
          comandos2
          ;;
   ...
   *)
          comandosN
          ;;
esac
```

- A estrutura for é um tipo limitado de ciclo. Corre através de uma lista de valores, atribuindo um novo valor a uma variável em cada iteração:

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

Estruturas de controlo (termo)

- Os comandos "break" e "continue" permitem alterar a sequência de comandos de um ciclo:
  - "break" – sai do ciclo "for", "while" e "until"
  - "continue" – salta o resto dos comandos do ciclo e reinicia o ciclo para uma nova iteração, testando a condição inicial.



Sumário

- Saber identificar uma "shell script"
- Conhecer e utilizar as estruturas de controlo de fluxo "if" e "case"
- Conhecer e utilizar as estruturas de controlo de ciclos "while", "until" e "for"
- Construir pequenas "shell scripts"

O segredo, a saber!

- A Variável “Exit Status”

Noção de “True” e “False”

- Quando um programa/comando UNIX termina, implicitamente (e não explicitamente como iremos ver) retorna um valor ao programa que o lançou (normalmente, a shell) informando-o se foi executado com sucesso ou não.
- Esse valor é um número e é chamado o “exit status” do programa/comando.

Noção de “True” e “False” (cont.)

- Esse valor, o “exit status”, é normalmente ignorado, quer pela shell quer pelo utilizador.
- No entanto, na construção de “shell scripts” este valor é muito importante.
- Normalmente, se o valor do “exit status” for igual a 0 significa que o programa foi executado com sucesso, enquanto que se for diferente de 0 significa que ocorreu um erro.

Noção de "True" e "False" (cont.)

- Então coloca-se a seguinte questão: como examinar o valor do "exit status"?
- O valor do "exit status" do último programa/ comando executado é gravado na variável "?" e pode ser consultada a qualquer momento através do comando:

```
$ echo "?"
```

Noção de “True” e “False” (cont.)

- É de notar que, o valor desta variável é sistematicamente actualizada cada vez que um comando é executado (incluindo o comando *echo*).
- Isto é, façamos:

```
$ ls

$ echo $?
```

Noção de “True” e “False” (cont.)

```
$ ls dddd

$ echo $?

$ echo $?
```

- Deste modo, torna-se útil, na construção de “shell scripts”, pensar que se o valor do “exit status” for 0 temos equivalente ao termo lógivo “true” e se for diferente temos equivalente ao termo lógico “false”.

Noção de “True” e “False” (cont.)

- É de notar que, esta convenção aqui usada é exactamente o oposto ao que estamos habituados com outras linguagens de programação, como por exemplo em “C”.
- Mais ainda, até existem comandos em UNIX chamados “true” e “false” que nos indicam exactamente esta nova convenção usada com o “exit status”. Façamos:

```
$ true
```

Noção de “True” e “False” (termo)

$ echo $?

$ false

$ echo $?

Executar um comando condicionalmente

- É sempre possível especificar em que condições um determinado comando numa "scritp" deve ser exucutado.

- Tais condições são sempre explicitamente expressas em termos do "exit status" de outro programa. Isto é:

  $ comando1 && comando2

  significa que, o comando2 só será executado

Executar um comando condicionalmente (cont.)

se o comando1 for executado com sucesso com o valor do "exit status" igual a 0.

comando3 || comando4

significa que, o comando4 só será executado se o comando3 não executado com sucesso, sendo valor do "exit status" diferente de 0.

Executar um comando condicionalmente (cont.)

- Por exemplo:

  $ ls file1 && cp file1 /tmp

  $ cp abc xyz && echo O ficheiro foi copiado com sucesso

  $ diff fileA fileB || echo Os ficheiros são diferentes

Executar um comando condicionalmente (termo)

- É de notar que este tipo de condições são muito limitadas:
  - Só pode executar um comando se a condição se verificar (apesar de ser possível agrupar comandos).
  - Não se consegue especificar um segundo comando alternativo caso a condição não se verificar.

A solução com “if”

- Muito mais poderoso e de mais fácil leitura:

  $ if comando1

  > then

  > comando2

  > comando3

  > fi

A solução com "if" (cont.)

- Por exemplo:

```
$ if diff file1  file2 > /dev/null
> then
> echo Os ficheiros são iguais
> rm file2
> fi
```

Utilizando o comando “test”

- A maior parte das linguagens de programação suportam a noção de “comparar” dois valores, duas variáveis ou uma variável e um valor.
- Os valores podem ser comparados por forma a verificarmos se são iguais, diferentes, qual o maior ou o menor, etc.
- A bash, nativamente, não suporta tais comparações, mas existe um comando/programa da shell que o faz: “test”.

Utilizando o comando “test” (cont.)

- O comando “test” é usado da seguinte forma:

```
euler:~ ferreira$ var1=10
euler:~ ferreira$ test $var1 = 20
euler:~ ferreira$ echo $?
1
euler:~ ferreira$
```

- A única finalidade do comando “test” é devolver o valor do “exit status” de acordo com a condição testada.
- Esse valor do “exit status” devolvido é consistente com a noção de “true” e “false”.

Utilizando o comando “test” (cont.)

- Por outras palavras, no exemplo dado, temos uma condição falsa.
- Podemos assim usar o comando “test” com a expressão condicional “if” da seguinte forma:

```
if test $var1 -gt $max
then
        echo Este valor é demasiado grande
fi
```

Dúvidas

(i) Sobre os trabalhos práticos?

(ii) Avaliação prática?