

TECNOLOGIAS E PROGRAMAÇÃO DE SISTEMAS DE  
INFORMAÇÃO

# 3 – Shell Script em Linux

Sistemas Operativos e Redes | Eng.º Milton Aguiar

Cofinanciado por:

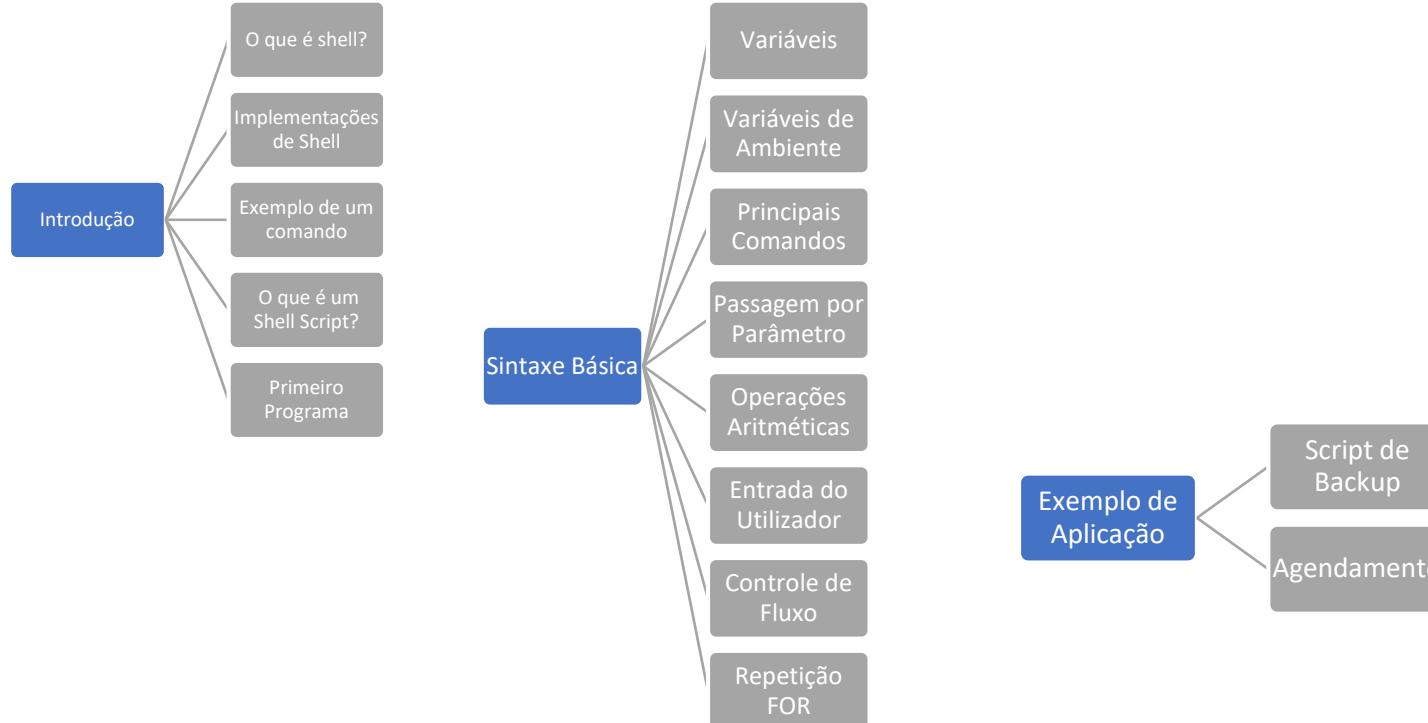


REGIÃO AUTÓNOMA  
DA MADEIRA



UNIÃO EUROPEIA  
Fundo Social Europeu

# Objetivos



# Introdução

O que é Shell?

# Introdução



## O que é Shell?

O Shell é um programa que atua na interface entre o utilizador e o kernel do sistema operativo.

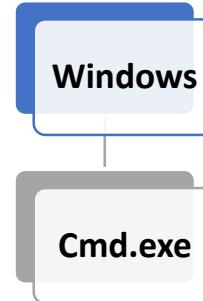
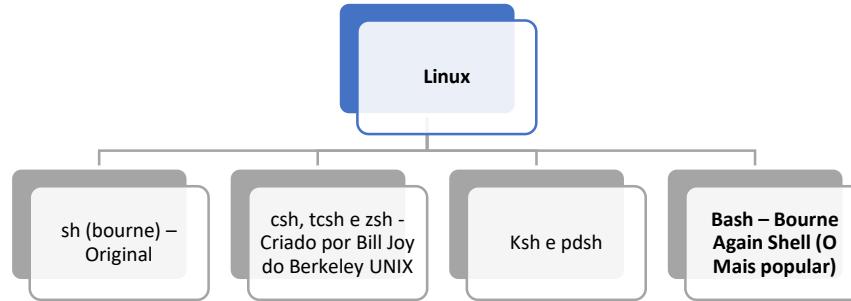
O kernel é quem acessa os equipamentos (hardware) da máquina, como disco rígido, placa de vídeo e modem.



# Introdução



## .Implementações de Shell



# Introdução



## .Exemplo de Comando

Listagem de diretório – Comando ls (Linux)

```
[guga@davinci AulaSockets]$ ls
cliente1    cliente6      cliente_multicast    servidor2.cpp    servidor6.cpp
cliente1.cpp  cliente6_2    cliente_multicast.cpp  servidor3.cpp    servidor7
cliente2    cliente6_2.cpp  pegasdata          servidor3.cpp    servidor7_2
cliente2.cpp  cliente6_3    pegasdata.cpp        servidor4.cpp    servidor7_2.cpp
cliente3    cliente6_3.cpp  peganome           servidor4.cpp    servidor7.cpp
cliente3.cpp  cliente6.cpp  peganome.cpp        servidor5.cpp    servidor_multicast
cliente4    cliente7      server_socket1       servidor5.cpp    servidor_multicast.cpp
cliente4.cpp  cliente7_2    servidor1          servidor6
cliente5    cliente7_2.cpp  servidor1.cpp        servidor6_3
cliente5.cpp  cliente7.cpp  servidor2          servidor6_3.cpp
[guga@davinci AulaSockets]$
```

Listagem de diretório – Comando dir (Windows)

```
C:\WINDOWS\system32\cmd.exe
C:\WINDOWS\NLDR>dir
0 volume na unidade C n&oacute;o tem nome.
0 n&umero de s&erie do volume & 24EB-8E9B
 Pasta de C:\WINDOWS\NLDR
23/07/2008  01:16    <DIR>    -
23/07/2008  01:16    <DIR>    001
23/07/2008  01:24    <DIR>    001
          0 arquivo(s)          0 bytes
          3 pasta(s)  32.035.709.056 bytes disponiveis
C:\WINDOWS\NLDR>
```

# Introdução

O que é Shell Script?

# Introdução



## .Shell Script

**Shell script** é uma linguagem de programação interpretada usada em vários sistemas operativos.

De outra maneira, é uma seqüência de comandos armazenados em um ficheiro.

Ficheiro que pode ser executado.

# Introdução



.Exemplos de Aplicações

Backups  
Automáticos

Compilar uma  
série de  
ficheiros

Criar  
utilizadores  
do sistema

# Primeiro Programa!

# Primeiro Programa

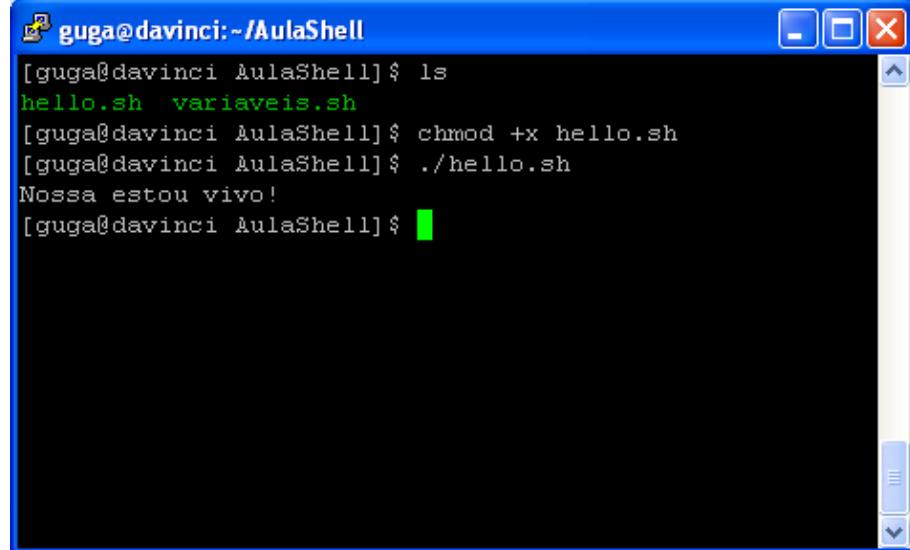
Basta criar um arquivo texto com os comandos!

```
1 #!/bin/bash
2 echo 'Nossa! Estou vivo!'
3
```

A primeira linha deve ser  
**#!/bin/bash**

Torne-o executável com o comando **chmod**

Pronto!



The terminal window shows the following session:

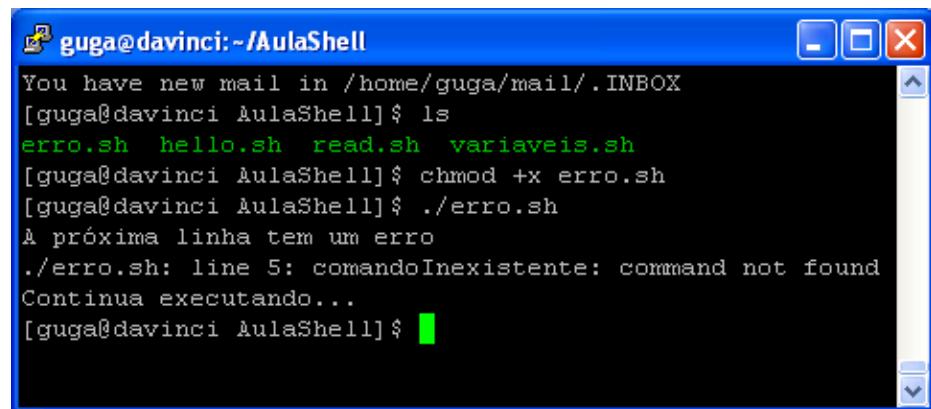
```
guga@davinci:~/AulaShell
[guga@davinci AulaShell]$ ls
hello.sh variaveis.sh
[guga@davinci AulaShell]$ chmod +x hello.sh
[guga@davinci AulaShell]$ ./hello.sh
Nossa estou vivo!
[guga@davinci AulaShell]$
```

# Primeiro Programa

Se um erro ocorrer, o script segue a execução dos comandos seguintes!

Comentários no código são iniciados pelo caracter #

```
1 #!/bin/bash
2
3 echo "À próxima linha tem um erro"
4 #Esse comando não existe!
5 comandoInexistente "erro"
6 echo "Continua executando..."
7
```



The screenshot shows a terminal window titled 'guga@davinci:~/AulaShell'. The user has new mail in their inbox. They run 'ls' to list files: 'erro.sh', 'hello.sh', 'read.sh', and 'variaveis.sh'. Then they chmod +x to 'erro.sh' and run it. The script contains an error: 'comandoInexistente' is undefined. Instead of stopping, it continues to execute the next command, 'Continua executando...'. The terminal window has scroll bars on the right.

```
You have new mail in /home/guga/mail/.INBOX
[guga@davinci AulaShell]$ ls
erro.sh hello.sh read.sh variaveis.sh
[guga@davinci AulaShell]$ chmod +x erro.sh
[guga@davinci AulaShell]$ ./erro.sh
À próxima linha tem um erro
./erro.sh: line 5: comandoInexistente: command not found
Continua executando...
[guga@davinci AulaShell]$
```

# Sintaxe Básica Shell

Sintaxe Básica Shell



## .Variáveis

Não existe a obrigatoriedade de se declarar uma variável

Não é preciso definir o tipo da variável

Valor pode ser uma frase, números, e até outras variáveis e comandos

Ao referenciar uma variável deve-se colocar \$ antes do seu nome identificador

```
1 #!/bin/bash
2 variavel="Eu estou logado como usuário $USER"
3 echo $variavel
4 variavel='Eu estou logado como usuário $USER'
5 echo $variavel
6 variavel="Meu diretório atual é o `pwd`"
7 echo $variavel
8 #Não podem haver espaços ao redor do igual "="
```

Áspas duplas -> variável interpretada

Áspas simples -> valor literal

Acento grave -> interpreta comando

The screenshot shows a terminal window titled "guga@davinci:~/AulaShell". The user runs the command "ls" to list files "hello.sh" and "variaveis.sh". Then, they run "chmod +x variaveis.sh" to make it executable. Finally, they run "./variaveis.sh", which outputs the contents of the script: "Eu estou logado como usuário guga", "Eu estou logado como usuário \$USER", and "Meu diretório atual é o /home/guga/AulaShell".

```
[guga@davinci AulaShell] $ ls
hello.sh variaveis.sh
[guga@davinci AulaShell] $ chmod +x variaveis.sh
[guga@davinci AulaShell] $ ./variaveis.sh
Eu estou logado como usuário guga
Eu estou logado como usuário $USER
Meu diretório atual é o /home/guga/AulaShell
[guga@davinci AulaShell] $
```



## .Variáveis de Ambiente

Quando o script inicia algumas variáveis de ambiente são inicializadas

Para distinguir das variáveis criadas pelo utilizador, as variáveis de ambiente são representadas com letras maiúsculas

Para ter uma lista completa das variáveis de ambiente basta digitar o comando **env**

Variável	Descrição
<b>\$HOME</b>	O diretório HOME do utilizador corrente.
<b>\$PATH</b>	Lista de diretórios separados por ponto e vírgula (;) onde serão procurados os comandos.
<b>\$USER</b>	O utilizador.
<b>\$PWD</b>	O diretório corrente.



## .Principais Comandos

Comando	Descrição
ls	Lista arquivos e diretórios <b>ls -a</b> #Arquivos ocultos <b>ls -l</b> #Mais informações
rm	Remove arquivos ou diretórios <b>rm -f leiamet.txt</b> <b>rm -rf pasta</b>
mkdir	Cria um diretório <b>mkdir diretorio</b>
cp	Copia arquivos <b>cp manual.txt /home/manual</b>
mv	Move e/ou renomeia arquivos. <b>mv manual.txt ..</b> <b>mv manual.txt manual2.txt</b>
cat	Mostra o conteúdo do arquivo <b>cat manual.txt</b>
grep	Faz buscas em arquivos procurando linhas que atendam a expressão regular passada por parâmetro <b>grep apple fruitlist.txt</b> <b>ls   grep aula</b>

Mais comandos: [http://www.guiabuntupt.org/wiki/index.php?title=Comandos\\_basicos](http://www.guiabuntupt.org/wiki/index.php?title=Comandos_basicos)



## .Passagem por Parâmetro

É possível passar parâmetros para um shell script via linha de comando

```
1 #!/bin/bash
2 echo "Número de parâmetros passados: $#"
3 echo "Nome do Script Shell: $0"
4 echo "Primeiro Parâmetro: $1"
5 echo "Segundo Parâmetro: $2"
6
```

The screenshot shows a terminal window with a blue header bar. The header displays the user's name 'guga' and the host 'davinci', followed by the directory '/AulaShell'. The window has standard window controls (minimize, maximize, close) at the top right. The main area of the terminal shows the execution of a shell script named 'parametros.sh'. The user runs the command `chmod +x parametros.sh`, then `./parametros.sh 1234 "segundo parâmetro"`. The script outputs the following text:  
Número de parâmetros passados: 2  
Nome do Script Shell: ./parametros.sh  
Primeiro Parâmetro: 1234  
Segundo Parâmetro: segundo parâmetro

Comando	Descrição
<code>\$#</code>	Número de Parâmetros passados
<code>\$1, \$2, ...</code>	Os parâmetros passados para o script
<code>\$0</code>	O nome do script shell



## .Operações Aritméticas

Formato de uma expressão aritmética: \${(expressão)}

Variáveis não precisam ser precedidas de \$

Variáveis não definidas são inicializadas automaticamente com zero

Aritmética é somente de inteiros

```
1 #!/bin/bash
2 echo "Soma: $(( $1+$2 ))"
3
```

The screenshot shows a terminal window titled 'guga@davinci:~/AulaShell'. The user has run the command `./soma.sh 34 34`. The output is 'Soma: 68', demonstrating that variables not preceded by \$ are treated as integers. The terminal window has a blue header bar and standard window controls (minimize, maximize, close) in the top right corner.



## .Entrada do Utilizador

Pode ser pedida a entrada de um valor para o utilizador através do comando **read**

O comando **read** bloqueia a execução do script enquanto espera a entrada do valor pelo utilizador

Quando o utilizador clica <enter> a interpretação do script continua

```
1 #!/bin/bash
2
3 echo "Entre com o valor para a variável:"
4 read variavel
5 echo "O valor digitado foi: $variavel"
6
```

```
[guga@davinci AulaShell]$ chmod +x read.sh
[guga@davinci AulaShell]$ ./read.sh
Entre com o valor para a variável:
123
O valor digitado foi: 123
[guga@davinci AulaShell]$
```



## .Controle de Fluxo (IF)

Controle de fluxo são comandos que alteram o fluxo de execução do programa de acordo com o teste de condições

```
1 #!/bin/bash
2 if [ -e $linux ]
3 then
4     echo 'A variável $linux existe.'
5 else
6     echo 'A variável $linux não existe.'
7 fi
8
9 file=leiamme.txt
10 if test -r "$file"
11 then
12     echo "O Arquivo pode ser lido!"
13 else
14     echo "O Arquivo não pode ser lido!"
15 fi
```

The screenshot shows a terminal window titled 'guga@davinci:~/AulaShell'. The user runs the script 'fluxo.sh' and receives the expected output: 'A variável \$linux existe.' followed by 'O Arquivo pode ser lido!'.

```
[guga@davinci AulaShell]$ ./fluxo.sh
A variável $linux existe.
O Arquivo pode ser lido!
[guga@davinci AulaShell]$
```



## .Operadores

### Operadores de Texto

<b>strin1=string2</b>	Testa se as strings são iguais
<b>Strin1!=string2</b>	Testa se as strings são diferentes
<b>-n string</b>	Testa se a string é não nula
<b>-z string</b>	Testa se a string é nula

### Operadores Aritméticos

<b>expr1 -eq expr2</b>	Testa se as expressões são iguais
<b>expr1 -ne expr2</b>	Testa se as expressões são diferentes
<b>expr1 -gt expr2</b>	Testa se a expr1 é maior que a expr2
<b>expr1 -ge expr2</b>	Testa se a expr1 é maior ou igual a expr2
<b>expr1 -lt expr2</b>	Testa se a expr1 é menor que a expr2
<b>expr1 -le expr2</b>	Testa se a expr1 é menor ou igual a expr2
<b>!expr1</b>	Testa se expr1 é falsa

```

1 #!/bin/bash
2 echo "Você deseja executar esta
3 operação? (s/n)"
4 read resposta
5 if [ "$resposta" = "s" ] ; then
6     echo "Sim o usuário deseja!"
7 else
8     echo "Não o usuário não deseja!"
9 fi
10 num1=10
11 num2=20
12 if [ $num1 -eq $num2 ] ; then
13     echo "O número $num1 é igual que $num2"
14 elif [ $num1 -ge $num2 ] ; then
15     echo "O número $num1 é maior que
16 $num2"
17 else
18     echo "O número $num1 é menor que
19 $num2"
20 fi

```



## .Operadores

### Operadores de Ficheiros (arquivos)

<b>-d arq</b>	Testa se arq é um diretório
<b>-e arq</b>	Testa se o arquivo existe
<b>-f arq</b>	Testa se arq é um arquivo regular
<b>-r arq</b>	Testa se o arquivo pode ser lido
<b>-u arq</b>	Testa se o arquivo tem tamanho diferente de zero
<b>-w arq</b>	Testa se o arquivo pode ser escrito
<b>-x arq</b>	Testa se o arquivo pode ser executado



## .Repetição FOR

Em Shell Script o **for** realiza um loop em uma determinada lista de valores.  
A lista pode ser um conjunto de strings, arquivos, etc..

```
1 #!/bin/bash
2
3 echo "Laço em lista de strings:"
4 echo ""
5 for var in eu gosto de programar
6 do
7     echo $var
8 done
9
10 echo "Laço em lista de arquivos:"
11 echo ""
12 for file in $(ls f*.sh)
13 do
14     echo $file
15 done
```

The terminal window shows the execution of the script 'for.sh' and its output. The output consists of two loops: one printing the words 'eu', 'gosto', 'de', and 'programar' on separate lines, and another printing the files 'fluxo2.sh', 'fluxo.sh', and 'for.sh' on separate lines.

```
[guga@davinci AulaShell]$ ./for.sh
Laço em lista de strings:
eu
gosto
de
programar
Laço em lista de arquivos:
fluxo2.sh
fluxo.sh
for.sh
[guga@davinci AulaShell]$
```

# Sintaxe Básica Shell

## Exemplo de Aplicação

execução de verificação

# Exemplo de Aplicação

## Backup Automático

```
1 #!/bin/bash
2
3 dir_orig="/home/guga/AulaShell/"
4 dir_dest="/home/guga/AulaShell/backup"
5
6 data_dia=`date +%d`
7 data_mes=`date +%m`
8 data_ano=`date +%Y`
9
10 data="$data_dia.$data_mes.$data_ano"
11
12 if [ -e $dir_orig ]; then
13     echo "Diretório Origem já existente!"
14 else
15     echo "Diretório não existe"
16     exit 1
17 fi
18
19 dir_final="$dir_dest/bkp_$data/"
20
21 if [ -e $dir_final ]; then
22     echo "Diretório Destino já existente!"
23 else
24     echo "Diretório não existe. Será criado!"
25     mkdir -p $dir_final
26 fi
27
28
29 for file in $(ls *.sh)
30 do
31     cp $file $dir_final
32 done
33
34 # mensagem de resultado
35 echo "Seu backup foi realizado com sucesso."
36 echo "Diretório: ${dir_orig}"
37 echo "Destino: ${dir_final}";
38 exit 0
```

# Exemplo Aplicação



## .Agendando o Backup

Para agendar o backup deveremos usar o seguinte comando:  
**crontab -e**

```
guga@davinci:~/AulaShell/backup/bkp_25.02.10 [guga@davinci bkp_25.02.10]$ crontab -e
20 22 * * * /home/guga/AulaShell/backup_diario.sh
[guga@davinci bkp_25.02.10]$
```

Minutos (0-59)  
Hora (0-23)  
Dia do mês (1-31)  
Mês (1-12)  
Dia da semana (0-6)

Campo	Função
Minuto	0-59
Hora	0-23
Dia do Mês	1-31
Mês	1-12
Dia da Semana	0-6 (O “0” é Domingo, “1” segunda, etc...)



# Na internet

## Comandos Linux

- [http://www.guiaubuntupt.org/wiki/index.php?title=Comandos\\_basicos](http://www.guiaubuntupt.org/wiki/index.php?title=Comandos_basicos) |
- Moodle, Diapositivos das aulas e Internet

