# Princípios da Computação

Shell scripts (2nd part)

isep Instituto Superior de **Engenharia** do Porto

# Arithmetic operations

isep Instituto Superior de Engenharia do Porto

# Arithmetic operations

- **`expr`**
    - Evaluates an expression and writes the result on stdout.
    - Operators and operands must be passed as separate arguments (spaces between them).

```
a=5
b=3
result=$(expr $a + $b)
```

# Arithmetic operations

- It is possible to achieve the same effect using the **$((***expression***))** operator.

- The variable expansion symbol **$** is not necessary within the arithmetic environment.

```
a=5
b=3
result=$((a + b))
```

# Some operators

| | |
|---|---|
| + | Add |
| - | Subtract |
| * | Multiply |
| / | Integer division |
| % | Remainder of integer division |

See the **`expr`** command manual page for the complete list of operators.

# Control flow (continue)

# Loop: `while` statement

- Keywords: **`while`** - **`do`** - **`done`**

- The condition is always evaluated (must be true) before executing the loop code.

```
while condition
do
   # Some action here.
done
```

# Loop: `while` statement

- The condition can be evaluated by testing the values of variables.

```
value=0
while [ ! $value -gt 0 ]
do
   read -p "Insert a positive integer: " value
done


echo "Value: $value"
```

# Loop: `while` statement

- The condition can also be evaluated by the success of a command (or command pipeline).

```
site='www.google.com'
while ! ping -c 1 $site 2> /dev/null
do
   echo "Can not reach $site" >&2
   sleep 60
done
echo "$site is available"
```

# Loop: `until` statement

- Keywords: **`until`** - **`do`** - **`done`**

- The condition is always evaluated (must be false) before executing the loop code.

```
until condition
do
   # Some action here.
done
```

# Loop: `until` statement

```
site='www.google.com'
until ping -c 1 $site 2> /dev/null
do
   echo "Can not reach $site" >&2
   sleep 60
done
echo "$site is available"
```

This condition is the opposite of the previous "while" example.

# Skipping a loop

- **`break`**
    - The loop ends immediately.
    - No more loop code will be executed.
- **`continue`**
    - Immediately ends the current iteration of the loop.
    - Skips to the next iteration of the loop.

# Decision: `case` statement

- Keywords: **`case`** - **`in`** - **`esac`**

```
case test_value in
   value1)
      # Some action here (there may be more case values).
      ;;
   *)
      # Default action here.
      ;;
esac
```

The ;; breaks (finishes) the case.

# Loop: `case` statement

```
read -p 'Yes/No?' answer
case $(echo $answer | tr [A-Z] [a-z]) in
'y'|'yes')
   echo 'Work saved.'
   ;;
'n'|'no')
   echo 'Work discarded.'
   ;;
*)
   echo 'Option not supported!'
   ;;
esac
```

# Functions

isep Instituto Superior de Engenharia do Porto

# Functions

- A function serves to write a functionality that can be called at various points in the script.

- It is characterized:
    - by its name followed by a pair of parentheses, and
    - by the sequence of commands delimited by braces.

- Functions are defined at the beginning of the script, so that they are known when the main script calls them.

# Calling a function

- A function is called by its name, followed by the arguments, just like a command line.
    - The function has its own local variables with the arguments given to it: **`$#`**, **`$*`**, **`$0`**, **`$1`**, etc.
- These local variables are distinct from the variables in the main script, although they have the same names.

# Returning values

- A function does not explicitly return values.
- The variables in a script are global.
- A function can manipulate the global variables of the script, and the changes are visible in the main script.
- The return is made by the updated global variables.

```
#!/bin/bash

global_var=0

my_function()
{
   echo Function: received $# arguments.
   global_var=$#
}

# MAIN SCRIPT
echo Main script: $global_var
my_function one two three
echo Main script: $global_var
exit 0
```

```bash
#!/bin/bash

global_var=0

my_function()
{
   echo Function: received $# arguments.
   global_var=$#
}

# MAIN SCRIPT
echo Main script: $global_var
my_function one two three
echo Main script: $global_var
exit 0
```

**Output:**

```
Main script: 0
Function: received 3 arguments.
Main script: 3
```

# Exercises / examples

# Exercise 1

- Write a script that receives exactly two values from the command line and displays the sum of the two values.
- If the script does not receive exactly two values from the command line, it should terminate with exit status 1.

# Exercise 1 (solution)

```
#!/bin/bash


if [ $# -ne 2 ]; then
   echo "usage: $(basename $0) value1 value 2" >&2
   exit 1
fi


result=$(expr $1 + $2)
echo $1 + $2 = $result


exit 0
```

# Exercise 2

- Write a script that receives a series of file names from the command line.
- For each existing file, its type should be presented, using the file command.
- If the name does not match a file, an information message should be displayed.

# Exercise 2 (solution)

```
#!/bin/bash


for filename
do
   if [ -f $filename ]; then
      file $filename
   else
      echo "$filename: not a file."
   fi
done


exit 0
```

# Exercise 3

- Write a script that asks the user for the path to a directory.

- If the directory exists, the script must display the number of lines of text from each file present in that directory, whose name ends in **`.txt`**.

- At the end, the script should show the number of **`.txt`** files found.

# Exercise 2 (solution, part 1/2)

```
#!/bin/bash

read -p "Directory: " directory

if [ ! -d "$directory" ]; then
   echo "$directory: not a directory." >&2
   exit 1
fi

# continues next slide...
```

# Exercise 3 (solution, part 2/2)

```
# continues here...


counter=0
for filename in ${directory}/*.txt
do
      wc -l $filename
      counter=$((counter + 1))
done


echo "$counter .txt files found."
exit 0
```