C2110 *UNIX and programming*

Lesson 7 / Module 1

PS / 2020 Distance form of teaching: Rev4

Petr Kulhanek

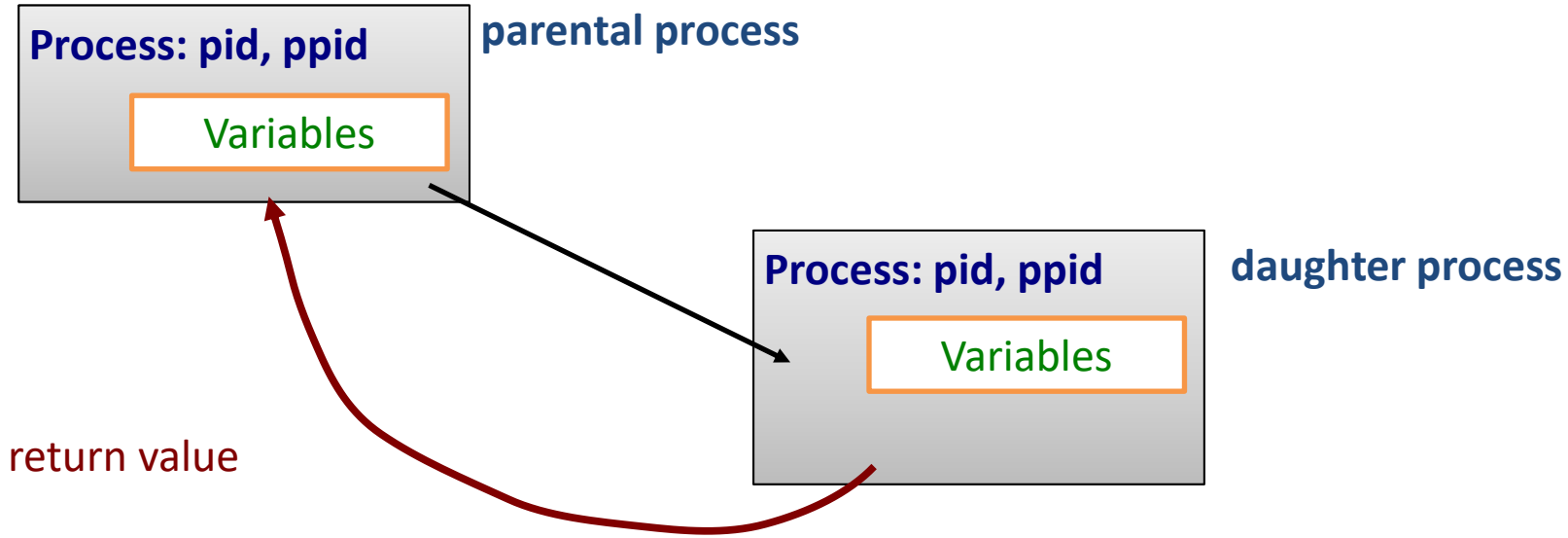
kulhanek@chemi.muni.cz

National Center for Biomolecular Research, Faculty of Science
Masaryk University, Kamenice 5, CZ-62500 Brno

Decision Making

I. Return value of the process

The ending process can communicate information about its run to the parent process with the use of **return values**. The return value is an integer that takes values from 0-255.



Return value:

0 = everything was successful (true)

> 0 = an error has occurred, return value then usually identifies the error **(false)**

Return value of the last executed command can be found using a variable **?**.

Return Value, Examples

```
$ mkdir test
$ echo $?
0
```

```
$ mkdir test
mkdir: cannot create directory `test ': File exists
$ echo $?
1
```

```
$ expr 4 + 1
5
$ echo $?
0
```

```
$ expr a + 1
expr: non-integer argument
$ echo $?
1
```

Command exit

Command **exit** is used to end a script run or interactive session. The optional argument of the command is the **return value**.

```
#!/bin/bash
if test "$1" -lt 0; then
      echo "Number is smaller than zero!"
      exit 1
fi
echo "Number is larger or equal zero."
exit 0
```

```
$ ./my_script 5
Number is larger or equal to zero.
$ echo $?
0
```

```
$ ./my_script -10
Number is smaller than zero!
$ echo $?
1
```

II. Command test

Command **test** is used to compare values and test file and directory types (man bash, man test). If the test is passed, the return value of the command is set to 0 (true).

Binary operators (requiring two arguments):

```
test argument1 operator argument2
```

there must be spaces

Preferred "alternative" notation:

```
[[ argument1 operator argument2 ]]
```

there must be spaces

Unary operators (requiring one argument):

```
test operator argument1
```

there must be spaces

Preferred "alternative" notation:

```
[[ operator argument1 ]]
```

there must be spaces

Command test, Integers

Comparing integers:

```
[[ number1 operator number2 ]]
```

!=	does not equal
==	equals
<	smaller
<=	less than or equal to
>	larger
>=	greater than or equal to

Operator:

- **-eq** equals
- **-ne** does not equal (not equal)
- **-lt** smaller than (less than)
- **-le** less than or equal to (less or equal)
- **-gt** greater than
- **-ge** greater than or equal to (greater or equal)

Additional information: man bash, man test

Examples:

```
[[ I -eq 5 ]]
```
is value of variable I equal to 5?

```
[[ J -le K ]]
```
is value of the variable K less than or equal than value of variable K?

when using [[…]] and operators for comparing integers, it is not necessary to use the $ operator to get the value of a variable

Command test, Strings

String comparison

```
test string1 operator string2
[[ string1 operator string2 ]]
```

Operator:

`==`	the strings are identical (`=` can also be used)
`!=`	strings vary

Examples:

```
[[ $A == "Hi" ]]
```

does variable A contain the string “Hi"?

```
[[ $J != $K ]]
```

does the variable K contain a different string than the variable K?

The $ operator MUST be used to get the value of a variable.

Command test, Strings II

String testing

```
test operator string1
[[ operator string1 ]]
```

Operator:

- **-n** tests whether string **does not have** zero length
- **-z** tests whether string **has** zero length
- **-f** tests whether string is the name of an existing **file**
- **-d** tests whether string is the name of an existing one **directory**

Examples:

```
[[ -n $I ]]
```

does the variable I contain a value?

```
[[ -f $K ]]
```

does the K variable contain name of an existing file?

The $ operator MUST be used to get the value of a variable.

Manipulating File/Directory Names

Commands:

basename - prints the file name, potentially removes the extension from the name
dirname - prints the directory name

Commands work with plain text, the names may not refer to existing files.

Examples:

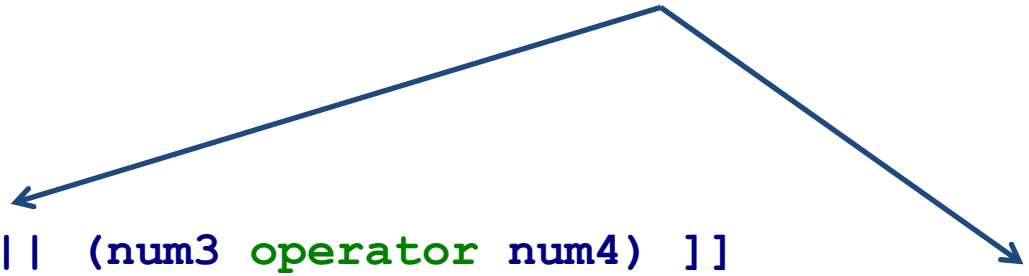
`basename test.txt .txt`	prints "test"
`basename directory/test.txt`	prints "test.txt"
`NAME=`basename "$FILE" .doc``	inserts the file name without the .doc extension from FILE variable into NAME variable
`dirname directory/test.txt`	prints "directory"
`DIR=`dirname "$FILE"``	inserts the directory name from FILE variable into DIR variable

Command test, Logical Operators

Logical operators:

```
||    logical or
&&    logical and
!     negation
```

same result, different way of interpretation!



Examples:

```
[[ (num1 operator num2) || (num3 operator num4) ]]
[[ (num1 operator num2) ]] || [[ (num3 operator num4)]]
```

I do not recommend using

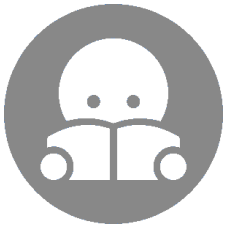
- More complex conditions can be created using logical operators.
- If we do not know priority of operators or we are not sure, then we use parentheses.
- Bash uses **lazy evaluation** of conditions, which manifests in evaluating only that part of the logical condition that must be evaluated to determine the resulting logical value.

Lazy evaluation

```
[[ expr1 || expr2 ]] <-> [[ expr1 ]] || [[ expr2 ]]
```

```
F || F = F
F || T = T
T || F = T
T || T = T
```

If the first expression is true (**T**), so the result is always true. Therefore, expr2 is evaluated only if the first expression is not true.



Trick:

```
mkdir directory || exit 1
```

if command mkdir fails (**F**), the exit command is called and the script is terminated

```
[[ expr1 && expr2 ]] <-> [[ expr1 ]] && [[ expr2 ]]
```

```
F && F = F
F && T = F
T && F = F
T && T = T
```

If the first expression is false (**F**), the result is always false. Therefore, expr2 is evaluated only if the first expression is true.

Command test, Examples

```
[[ (I -ge 5) && (I -le 10) ]]
```
Is the value of variable I in the interval <5;10>?

```
[[ (I -lt 5)||(I -gt 10) ]] OR [[ ! ((I -ge 5)&&(I -le 10)) ]]
```
Is the value of variable I outside the interval <5;10>?

```
[[ I -ne 0]]
```
Is the value of the variable I different from zero?

```
[[ $A == "test" ]]
```
Does variable A contain the string "test"?

```
[[ $A ! = "test" ]]
```
Does variable A contain a string other than "test"?

```
[[ -z $A ]]
```
Does variable A contain an empty string?

```
[[ -f $NAME ]]
```
Is there a file whose name is in the NAME variable?

```
[[ ! (-d $NAME) ]]
```
Isn't there a directory whose name is in the NAME variable?

[[...]], test, [...]

```
[[ (I -ge 5) && (I -le 10) ]]
```

preferred notation

```
test $I -ge 5 && test $I -le 10
[ ($I -ge 5) && ($I -le 10) ]
```

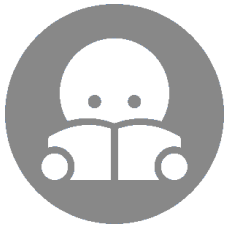
requires more complicated notation, the use of the $ operator, and possibly quotation marks

```
[[ -f $I ]]
```

preferred notation

```
test -F "$I"
[ -f "$I"]
```

requires more complicated notation, the use of the $ operator, and possibly quotation marks

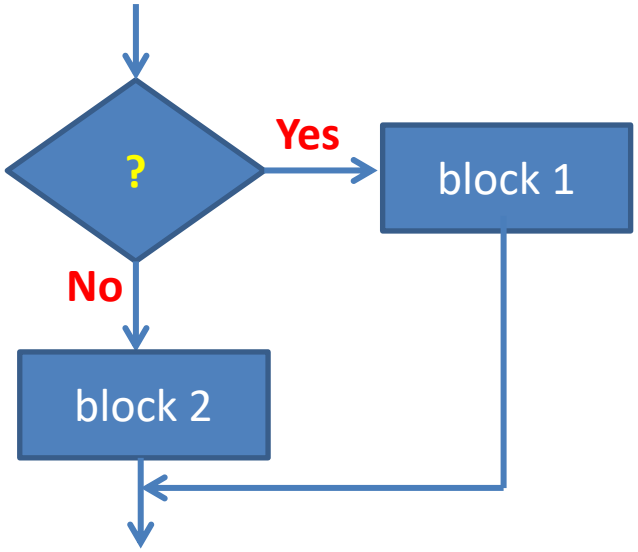


Details:

- man test
- man bash (CONDITIONAL EXPRESSIONS)

Conditions

Conditional block execution



Conditions

```
if command1
      then
            command2
            ...
      fi
```

```
if command1
      then
            command2
            ...
      else
            command3
            ...
      fi
```

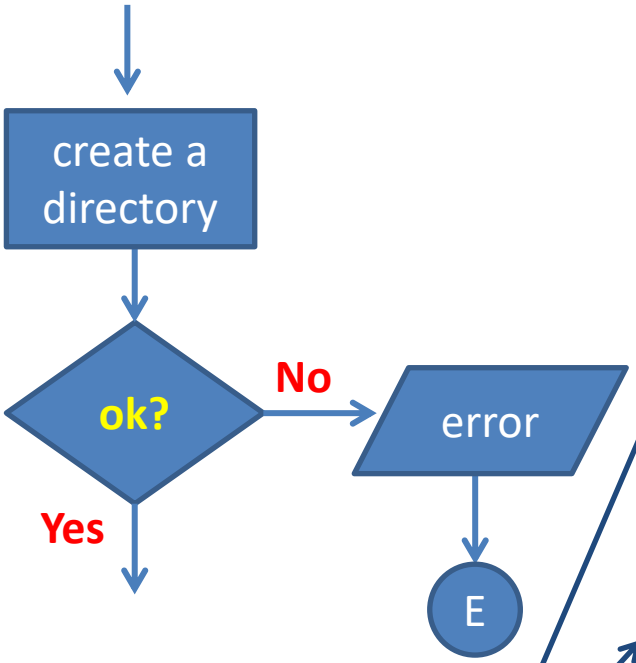
If **command1** ends with a return value **0, command2** will be performed. Otherwise, **command3** will be executed.

Compact notations:

```
if command1; then
      command2
      ...
fi
```

```
if command1; then
      command2
      ...
else
      command3
      ...
fi
```

Practical Example - Condition



functionally identical notations

```
mkdir directory 2> /dev/null
if [[ $? -ne 0 ]]; then
      echo "I can't create directory!"
      exit 1
fi
```

```
mkdir directory 2> /dev/null
if test $? -ne 0; then
      echo "I can't create directory!"
      exit 1
fi
```

```
if ! mkdir directory 2> /dev/null; then
      echo "I can't create directory!"
      exit 1
fi
```

Exercise I

1. Try the examples from the previous page. Use the command ls to monitor the existence of the directory and change commands mkdir and rmdir.
2. Write a script that prints the result of the ratio of two numbers. The user enters the values interactively after running the script. The script handles possible division by zero.
3. Write a script that asks for the file name. The script prints an error message if the file does not exist. Otherwise, it writes it to standard output.