C2110 UNIX and programming

8th Lesson

Petr Kulhánek, Jakub Štěpán

kulhanek@chemi.muni.cz

National Centre for Biomolecular Research, Faculty of Science
Masaryk University, Kotlářská 2, CZ-61137 Brno

INVESTMENTS IN EDUCATION DEVELOPMENT

CZ.1.07/2.2.00/15.0233

Contents

- Special variables
  - Script arguments
- Input / Formatted output
  - Commands printf, read
- Arithmetic operations
  - Command expr
- Command return value
- Conditions
  - Commands test, exit

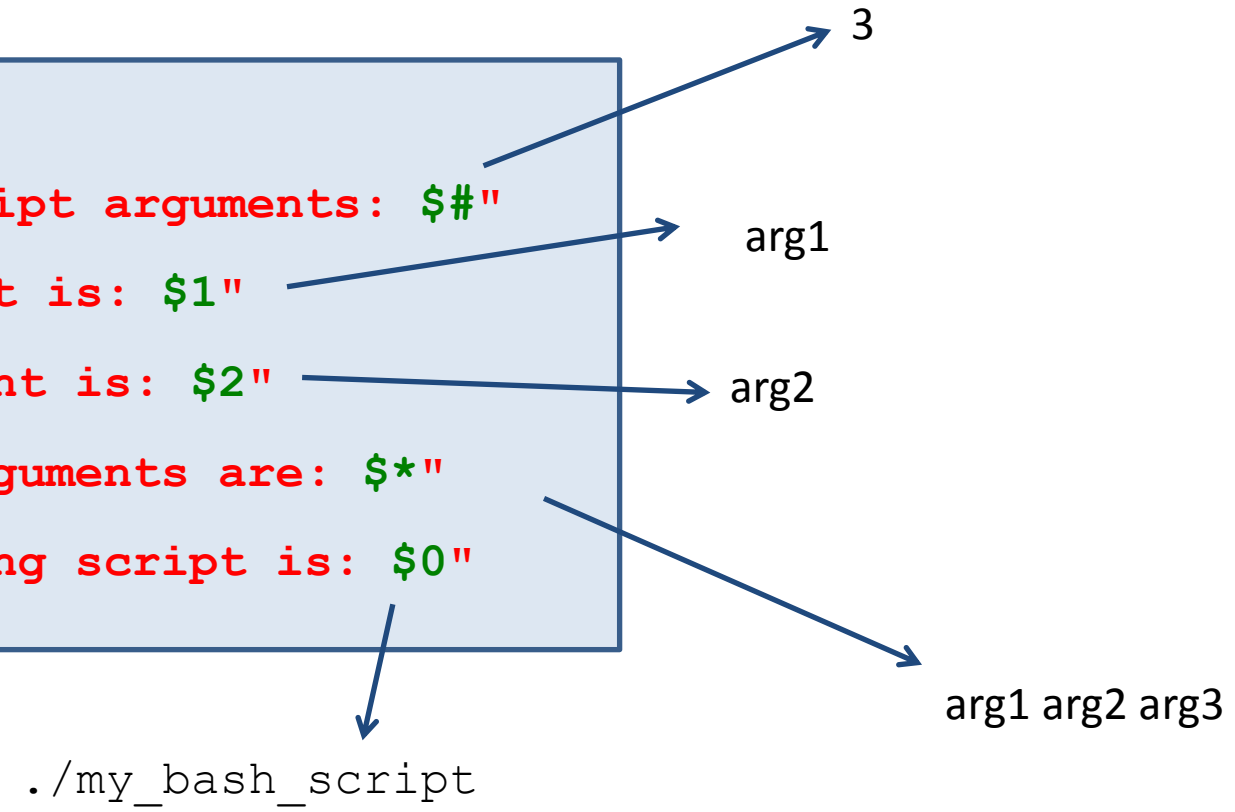
Special variables

- **Script arguments**

Script arguments

```
$ bash my_bash_script arg1 arg2 arg3
$./my_bash_script arg1 arg2 arg3
```

```
#!/bin/bash
echo "Number of script arguments: $#"
echo "First argument is: $1"
echo "Second argument is: $2"
echo "All script arguments are: $*"
echo "Name of running script is: $0"
```



Usage and meaning of arguments designes script author.

Overview

Script arguments:

#	number of arguments, that were given on script start
0	script name
1 … 9	values of arguments 1 to 9, that were given on script start
*****	all arguments, that were given on script start

Processes:

?	return value of last processed command (process)
$	process identifier (PID)

Advanced arguments management:

If script needs more than 9 arguments, it is neccessary to use command **shift**. This command removes first argument from list of arguments.

```
for((I=1;I <= $#;I++)); do
      echo $1
      shift
done
```

Prints all script arguments sequentially.

Exercise

1. Write script, that print number of arguments that were given on script start.
2. Write script, that print symbols **A** one next to other. Number of symbols will be given as first script argument.
3. Write script, that print its process identifier **PID**. Then pause script execution for 5 minutes by command **sleep**. Use command **kill** in another terminal to terminate your paused script.

Input / output

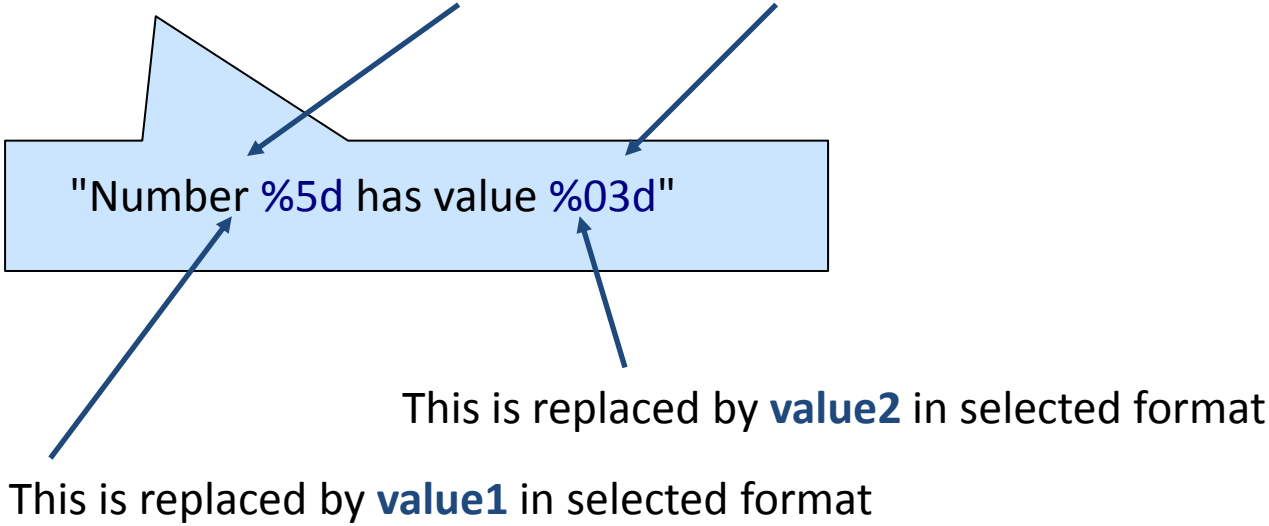
- Formated output - printf
- Input - read

Command printf

Command **printf** prints **formatted** text and numbers.

Syntax:

```
printf [format] [value1] [value2] ...
```



Further information: man bash, man printf

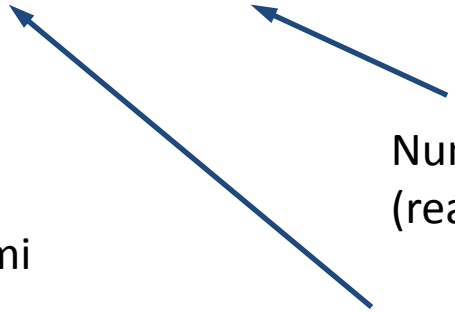
Command printf, příklady

```
$ I=10
$ B=12.345

$ printf "Variable I value is %d\n" $I
Variable I value is 10

$ printf "Given number B is %10.4f\n" $B
Given number B is    12.3450

$ printf "Given number B is %010.4f\n" $B
Given number B is 00012.3450

$ printf "Given number B is %+010.4f\n" $B
Given number B is +0012.3450

$ printf "Number I is %-5d and number B is %.1f\n" $I $B
Number I is 10    and number B is 12.3
```

Command printf, format

[] – optional part

%[flag][length][.precision]type



Number of decimal places (real numbers)

Total field length

Flag:

-	zarovnat doleva
0	prázdné místo zaplnit nulami
+	vždy uvést znaménko

Type:

d	integer
s	string (text)
f	real number

Special symbols:

\n	line break
\r	return to line begin
%%	symbol %

Further information: man bash, man printf

Command read

Command **read** read text from standard input and save it to variable. Command read while line and save first word to first variable, ..., rest of line is saved to last variable.

Syntax:

```
read A      # whole line is saved to variable A
read A B    # first word is saved to variable A
            # rest of line is saved to variable B
```

Example:

```
echo -n "Write number: "
read A
echo "You wrote number: $A"
```

Attention: do not use command read together with pipes

```
echo "text" | read A
echo $A
```

A does not hold value "text"

Further information: man bash

Arithmetic operations

Arithmetic operations

Arithmetic operations with **whole numbers** may be done in block **((...))**.

Possible input:

```
(( I = I + 1 ))
(( I++ ))
I=$(( I + 1 ))
echo "Value I incremented by 1 : $(( I + 1 ))"
```

Result value is printed to standard output

Operators:

`=`	assignment
`+`	addition
`-`	sustraction
`*`	multiply
`/`	division
`%`	division remainder
`++`	increment by 1
`--`	decrement by 1

Further information: man bash

Command expr

Command **expr** solve mathematical expressions, results are printed to standard output.

Examples:

```
$ expr 1 + 2
3

$ expr 2 \* 3
6

I=`expr $I + 1`
```

**** prevent special symbol expansion * to file and directory names in current directory

Variable value is given

Result is assigned to variable I

Further information: man expr

Exercise

1. Wtite script, that print first argument in format **%4d**.
2. Write script, that reads number from standard input and prints it iín following format: including sign, length will be 5 symbols, empty space will be filled by zeros:

   Given number is : +0003

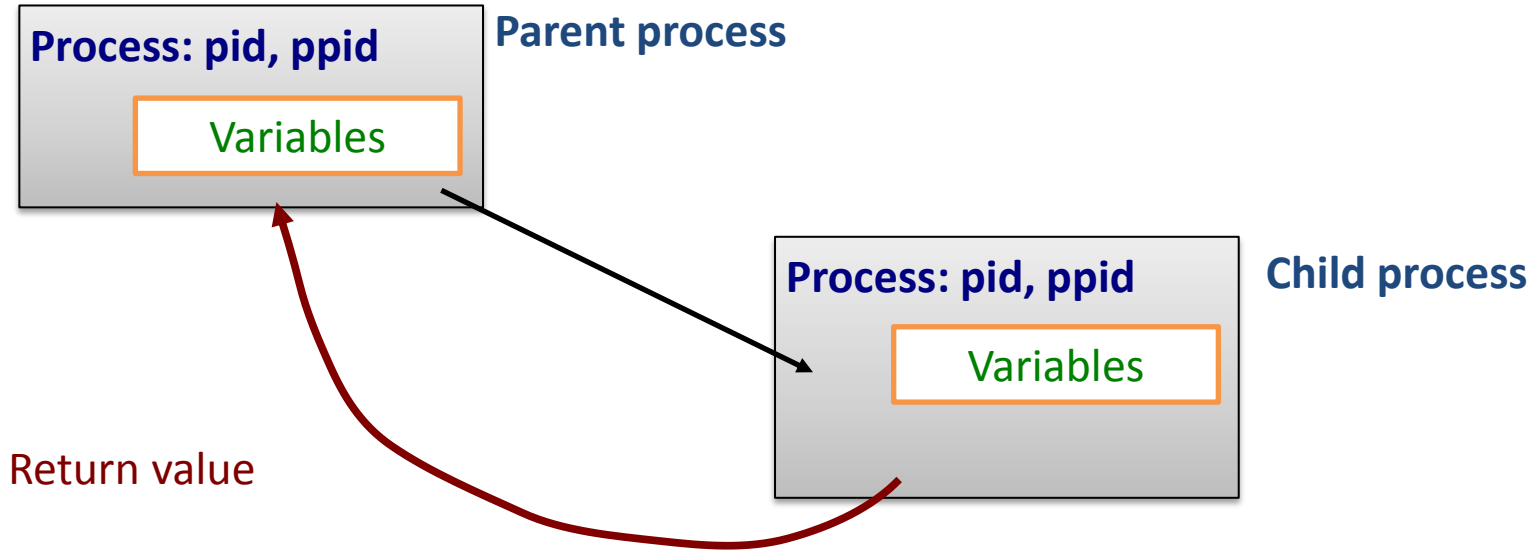
3. What happens if script from exercise 2 is given number: 123456?
4. Write script, that will accept **2 numbers as arguments**. These numbers will then be printed together with their sum.

Conditions

- Command return value
- Conditions

Command return value

A process may at the end inform its parent process about its run by **return value**. Return value is whole number with values in range 0-255.



Return value:

0 = **successful run – no error**

> 0 = **error occured**, return value may be used to identify error

Return value of last process command may be obtained by special 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 test, whole numbers

Command **test** is used to compare values and testing types. If test is successful, then return value of test command is set to 0.

Whole numbers comparison:

```
test number1 operator number2
```

Operator:

-eq	equal to
-ne	not equal
-lt	less than
-le	less or equal
-gt	greater than
-ge	greater or equal

Further info: man bash, man test

Command test, strings

String comparison

```
test string1 operator string2
```

Operator:

==	strings are identical
!=	strings differ

String testing

```
test operator string1
```

Operator:

-n	test if length of string is **non-zero**
-z	test if length of string is **zero**
-f	test if string is **existing file**
-d	test if string is **existing directory**

Conditions

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

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

If **command1** has return value **0**, then command **command2** is processed. If return value is non-zero, **command3** is processed.

Compact form:

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

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

Command exit

Command **exit** end script or interactive session. Optional argument is return value.

```
#!/bin/bash
if test "$1" -le 0; then
      echo "Whole number is not lower then zero!"
      exit 1
fi
echo "Number is higher then zero."
exit 0
```

```
$ ./my_script 5
Number is higher than zero.
$ echo $?
0
```

```
$ ./my_script -10
Number is not lower then zero!"
$ echo $?
1
```

Nesting

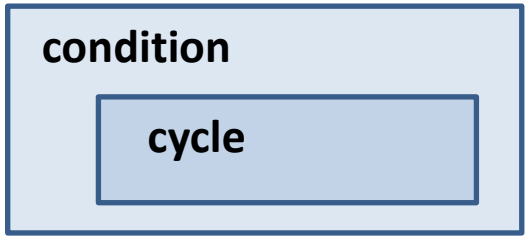
Control structures may be nested.

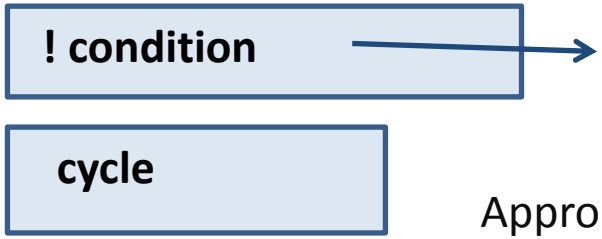
```
for((I=1;$I <= 10;I++)); do
        for((J=1;$J <= 10;J++)); do
                echo $(($I+$J))
        done
done
```

Outer cycle

Inner cycle

By script design it is advantageous to avoid unneccesary nesting (main reason is better orientation in script text).





Appropriate design to test user input.

Exercise

1. Write script, that read two numbers from standard input. These numbers will be printed with information which one is higher (script autor may design output form).
2. Write script, that will accept two numbers as arguments. Script prints these numbers together with their quotient. Use condition to prevent division by zero.
3. Save list of files and directories that are in your home directory to file **list.txt**
4. Write script, that accepts file name as argument. Script test if file exists and if yes, then prints its contents and number of lines in file. Test your script on file **list.txt**.

Home work

1. Print filled square from symbols "X" to terminal. Size of square gives user as script argument.
2. Rewrite previous script so that it prints only square outlines.
3. Write script, that prints two right-angled triangles in following orientations. Leg length will be read from user from standard input.

```
X

X X

X X X


X X X

X X

X
```

4. Print circle or ring from "X" symbols. Radius and circle / ring option will be read from user from standard input.