C2110 *UNIX and programming*

7th lesson

Bash - scripting language

Petr Kulhánek

kulhanek@chemi.muni.cz

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

Contents

- **Scripting in bash**
  - **basic syntax, comparison with flowcharts, script execution**
- **Variables**
  - **setting and accessing the values, variables and processes, string enterpretation**
- **Input and Output**
  - **read, echo, printf**
- **Arithmetic operations**
  - **operations with integers**

Scripting in Bash

Script in Bash

```
#!/bin/bash
# this is a comment
echo 'This is a script in Bash interpreter!'
echo "The content of`pwd` is:"
ls     # displey content of directory
A=6    # sets the value of variables A
echo "The value of variable A is $A"
echo "command one"; echo "command two"
./mycommand first_argument second_argument \
            third_argument
```

Order of command execution

immediately followed by new line

- blank lines are ignored
- text prefixed by **#** is ignored (used to comment on the script functionality)
- one-line can include multiple commands, commands are separated by a semicolon **;**
- one command can be written to multiple lines using backslash \

Script/Program editors

Since the **scripts** and program source codes are **text files**, any text editor can be used to save code in plain text form (without formatting metadata).

Text editors:

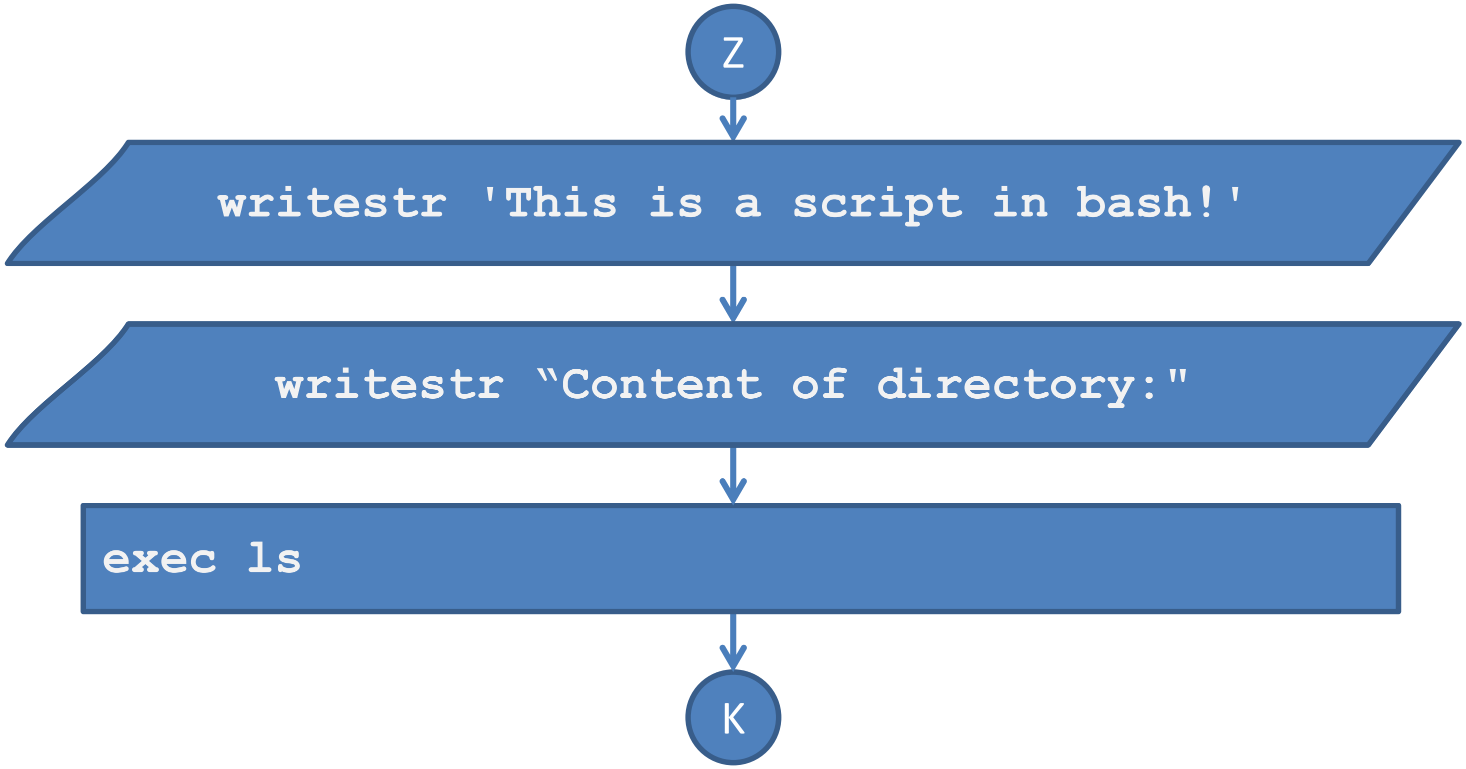
- vi
- kwrite
- kate
- **gedit**

You can also use the specialized development environment - **IDE (Integrated Development Environment)** to write script or source code. In addition to the plain editors, the IDE includes project manager, debugging tools (debugger) and more. It is mostly available for more complex languages: *C, C++, Fortran, JavaScript*, *Python*, *PHP*, etc.

Development environment:

- Kdevelop
- qtcreator
- NetBeans
- Eclipse

Flowchart and scripts in bash


Z
writestr 'This is a script in bash!'
writestr "Content of directory:"
exec ls
K

Commands execution order

```
#!/bin/bash
echo 'This is a script in bash!'
echo "Content of directory:"
ls
```

Execution of scripts

1) Indirect execution

The script is given as argument of language interpreter.

```
$ bash my_bash_script
```

Scripts **may not have** flag x (executable).

2) Direct execution

Script is executed directly (shell automatically starts the interpreter).

```
$ chmod u+x my_bash_script
$./my_bash_script
```

Scripts must have **x** flag (executable) and set **interpreter** (part of the script).

```
#!/bin/bash

echo 'This is a script in Bash!'
```

Interpreter specification

Specifications of interpreter (the first line of the script):

```
#!/absolute/path/to/script/interpreter
```

Script in Bash

```
#!/bin/bash

echo " This is bash script!"
```

Script in gnuplot

```
#!/usr/bin/gnuplot

set xrange[0:6]

plot sin(x)

pause -1
```

- If the interpreter of script is not specified during the direct execution, the system shell interpreter (bash) is used.
- Interpreter specified in the script is ignored during the indirect script execution.
- Interpreter specification is always appropriate in the script because it is used by text editors to highlight syntax. (file name is specified without suffix or with suffix **.sh**)

Exercise I

1. Write a bash script that prints the text "current directory is:" followed by an absolute path to the current directory. Execute the script indirectly by bash.
2. Extend the functionality of the previous script that in addition to the above, it prints "Contents of directory is: " followed by a list of files and directories in long format. Execute the script directly.

Variables

Variables

In Bash, variable is **named memory location** that contains a value. The value of the Bash variable is always **string (text)**.

There must be **no spaces between** the variable **name** and a **=**

Setting variables:

```
$ VARIABLE_NAME=value
$ VARIABLE_NAME =“value with space"
```

Accessing the variable value:

```
$ echo $VARIABLE_NAME
```

"TEXT ${VARIABLE}TEXT"

if the value of the variable is a part of the larger text, the variable name is put into double quotation marks.

Deleting variables:

```
$ unset VARIABLE_NAME
```

List of all defined variables:

```
$ set
```

Set variables

```
$ VARIABLE_NAME=“value with space"
```

~~$~~ VARIABLE_NAME =“value with space"

interpreted as
a program name

space

interpreted as
an program argument

~~$~~ VARIABLE_NAME= “value with space"

VARIABLE_NAME is set
to empty string,the variable value is
available only to executed program.

space

interpreted as
a program name

```
$ VARIABLE_NAME=“value with space" program [arg1...]
```

Several variables with theirs values can be specified (entries VAR=VALUE are separated by space), that are only available for the running program.

if program name contains an equal sign, the name must be given in quotation marks

Strings

Four types of the string can be used in Bash

- **without quotation marks**

```
A=magic
B=*
C=$A
```

Wild card expansion is context dependent.

replaced by the value of the variable A

- **with quotation marks**

```
A="magic trick"
B="* $A"
```

the value of the variable contains two words separated by a space

is replaced by the value of the variable A, the asterisk is not expanded (in quotation marks)

- **with apostrophes**

```
A='magic trick'
B='* $A'
```

Exact text without any expansion or transfomation

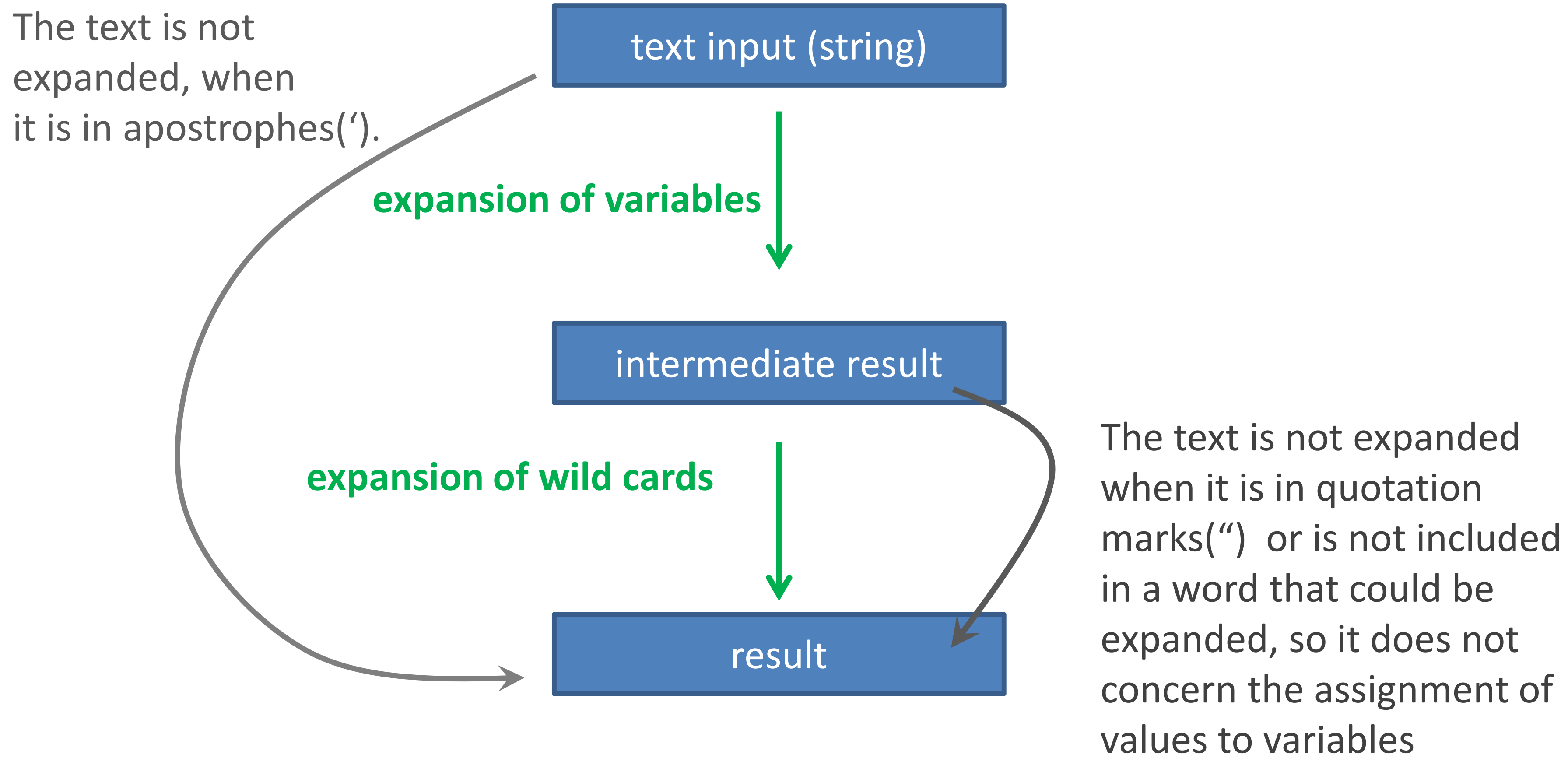
- **With grave accents**

```
A="`ls -d`"
B="number : `ls | wc -l`"
```

the output of the quoted commas is put in the inverted quotes

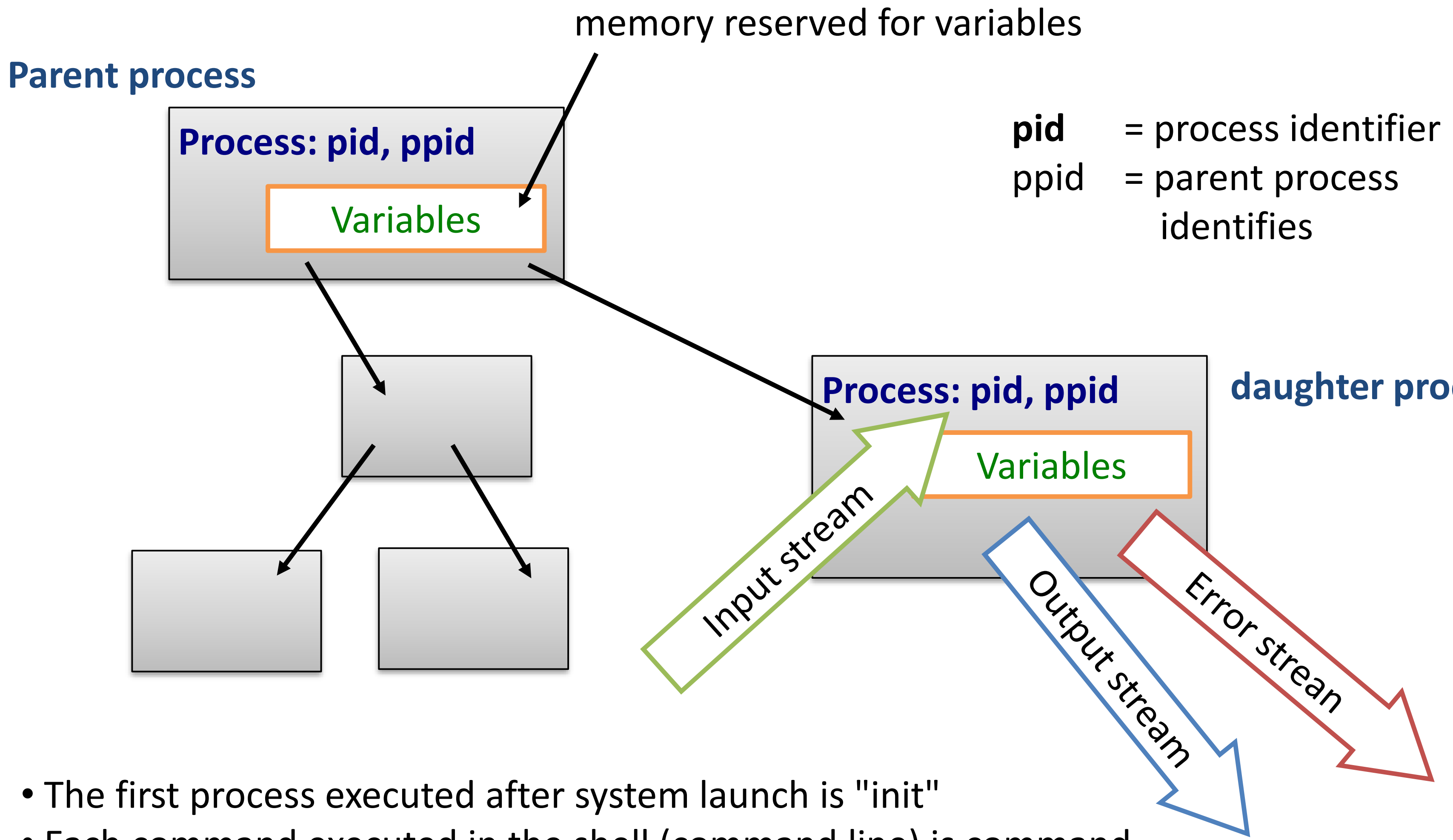
Expansion of string/command line

Order of string expansion/command line:


The text is not expanded, when it is in apostrophes(‘).
text input (string)
expansion of variables
intermediate result
expansion of wild cards
result
The text is not expanded when it is in quotation marks(“) or is not included in a word that could be expanded, so it does not concern the assignment of values to variables

Processes

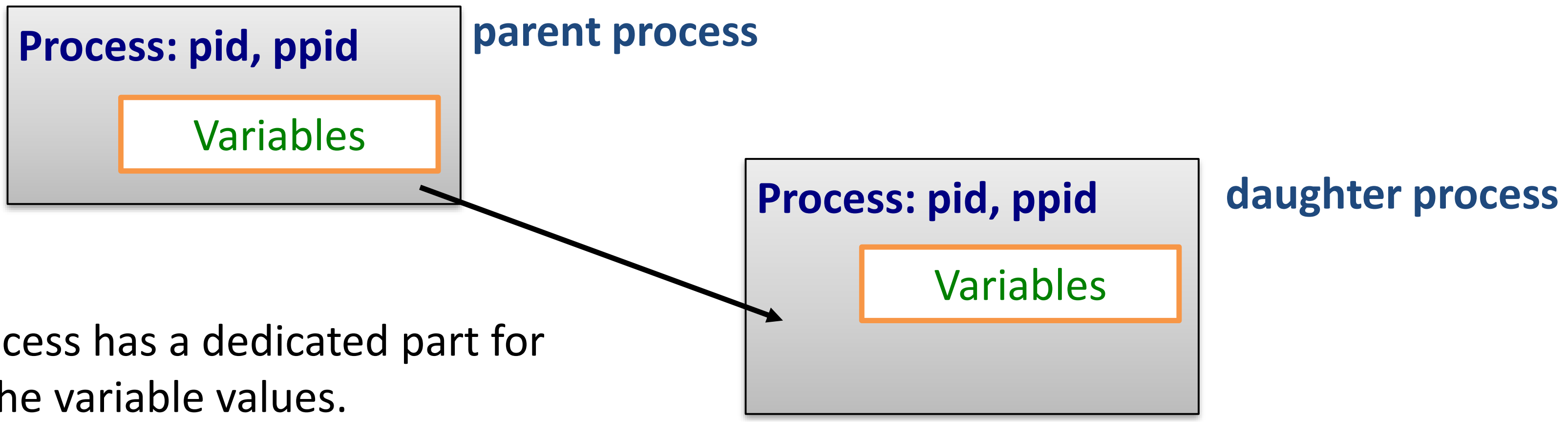
Process is instance of a running **program**



pid	= process identifier
ppid	= parent process identifies

- The first process executed after system launch is "init"
- Each command executed in the shell (command line) is command

Proměnné a procesy



Each process has a dedicated part for storing the variable values.

Daughter process at the moment of execution **gets the copy** of variables (exported) and theirs values from parent process. These variables can be changed or removed. The new variables can be also set or remove new variables**. However all these changes after end of the daughter process are removed.** Changes will **not affect the values of the original variables** or parent process.

Export proměnné:

```
$ export VARIABLE_NAME
$ export VARIABLE_NAME="value"
```

export

export with assigning

Exercise II

Work in shell interactive mode.

1. Set the variable A to value 55
2. Print the value of variable A (echo command)
3. Display all environment variables in the terminal. Is among them A variable? Use the command less or more to make the listing more transparent.
4. Use grep command and display only the record that contains the variable A. For search, use the variable value independent pattern.
5. Display all variables whose names begin with the letter A (grep ^TEXT) .
6. Change the value of the variable to "this is a long string"
7. Print the value of the variable A.
8. Remove the variable A
9. Verify that the variable was removed (procedure in 4. task).
10. Gradually assign the variables A, B and C according to the examples listed on page 13. Gradually validate their values by set and echo commands. Analyze any contradictions.

Work in interactive shell mode shell.

Cvičení III

Work in new terminal

1. Clear the PATH variable. How does the change affect command line functionality? Try to run ls and pwd. Explain the behavior.
2. When is expanded the wild card * in the following example:

```
$ B="Content of directory is *"
$ echo $B
```

3. Write a script called print_C that prints the value of the variable C. Explain behavior of script in the following examples:

```
$ ./print_C
$ C="value 1" ./print_C
$ echo $C
$ C="value 2"
$ echo $C
$ ./print_C
$ export C
$ ./print_C
```

Input/Output

Command read

Command **read** is used to **read text** from standard input (**ie. to read interactive input**) and to assign it to variables. The command always reads the entire line, first variable stores first word, …, last variable stores the rest of the line

Syntax:

```
read A       # entire line is stored in the variable A
read A B     # first word is stored to variable A
             # rest of the line is stored in the variable B
```

Příklad:

```
echo -n "Enter value:  "
read A
echo "Entered value is: $A"
```

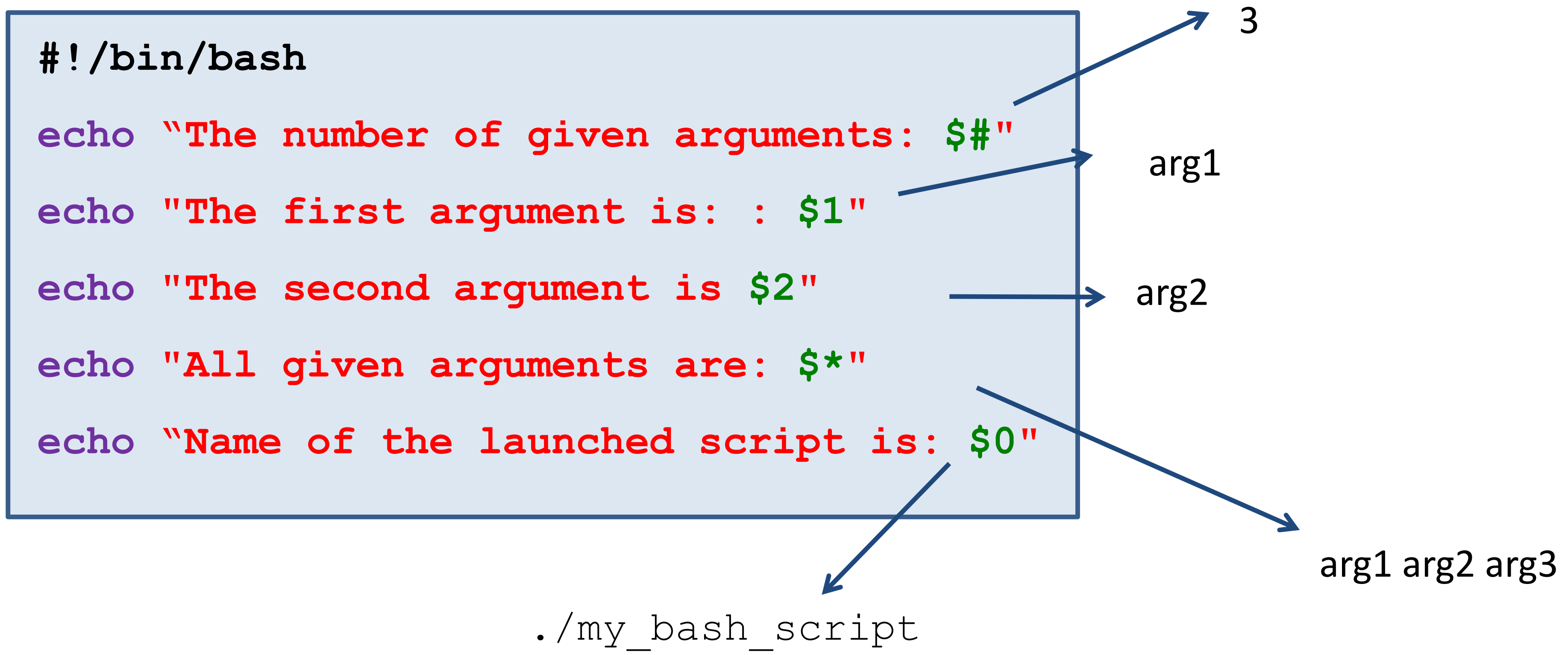
Caution: Do not use read and pipe in one command

More information: man bash

~~**echo "text" | read A**~~
~~**echo $A**~~ ← it would **not store** the value "text"

Script arguments

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



```
#!/bin/bash
echo “The number of given arguments: $#"
echo "The first argument is: : $1"
echo "The second argument is $2"
echo "All given arguments are: $*"
echo “Name of the launched script is: $0"
```

Use and meaning of arguments is determined by the author of the script.

Script arguments- variables

Script arguments(special variables):

#	the number of arguments with which the script was executed
0	name of the executed script
1 … 9	the values of arguments 1-9, with which the script was executed
*****	all the arguments with which the script was executed

Advanced work with arguments:

"$@" all the arguments with which the script was started embedded in the quotation marks (it handles the input where the arguments contain spaces)

Different behavior towards **$*** or **"$*"**!!

When more than nine arguments is needed, it is necessary to use **shift** command. The command removes first from the given list of arguments

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

Gradually prints the arguments of the script.

Exercise IV

1. Write a script that will ask user for his favorite color, which is then printed on the screen.
2. Write a script that prints the number of given arguments, and the value of the first argument

Command echo

Command **echo** is used to print of unformatted text to the standard output stream.

Syntax:

```
echo [options] [string1] [string2] ...
```

The command prints the string in a specified order separated by space.

Useful options:

-n	do not move to the next line

Příklady:

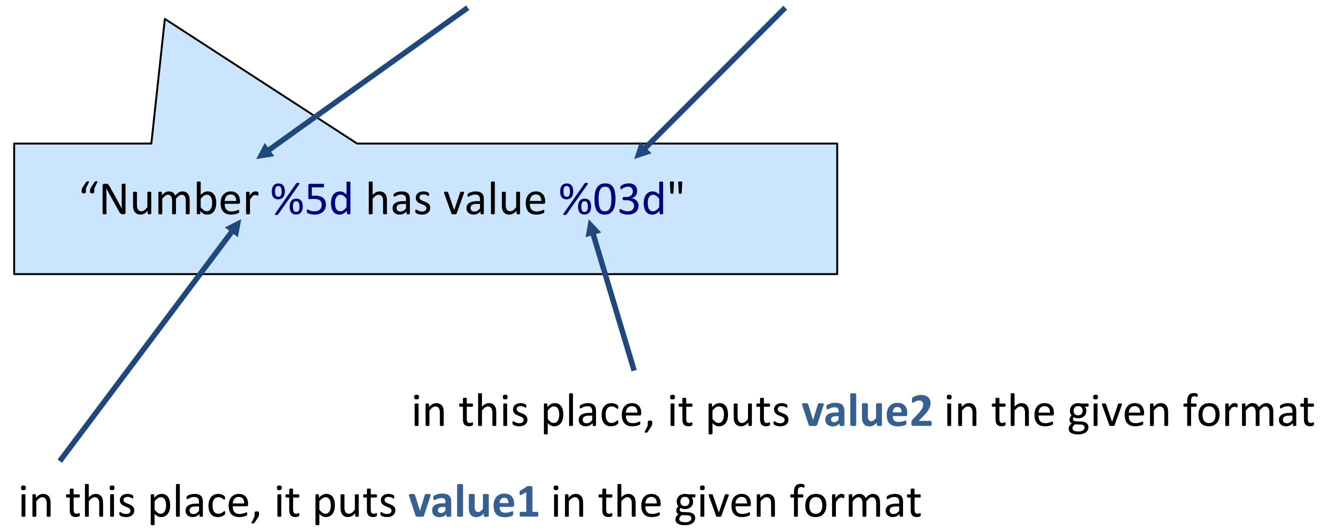
```
$ echo "Ema"        "grinds" "meat."
Ema grinds meat.
$ echo "Ema   grinds   meat."
Ema   grinds   meat.
$ echo -n "Enter color: "
$ A=5
$ echo "Value of variable A is $A."
Value of variable A is 5.
```

Command printf

Command **printf** is used to print formatted text and number.

Syntax:

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



“Number %5d has value %03d"

in this place, it puts **value2** in the given format

in this place, it puts **value1** in the given format

Příkaz printf, příklady

```
$ I=10
$ B=12.345

$ printf "The value of the variable I is %d\n" $I
The value of the variable I is 10

$ printf " The value of the variable B is %10.4f\n" $B
The value of the variable B is    12.3450

$ printf " The value of the variable B is %010.4f\n" $B
The value of the variable B is 00012.3450

$ printf " The value of the variable B is %+010.4f\n" $B
The value of the variable B is +0012.3450

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

Command printf, format

an optional part

% [flags][length][.precision]type

Number of decimal places after decimal point (real numbers)

Total length of array

Flag:

-	align to left
0	empty space filled with zeros
+	always print sign

Type:

d	integer
s	string (text)
f	real number

Wild cards:

\n	end of line
\r	go back to the beginning of line
%%	character %

More information: man bash, man printf

Cvičení V

1. Write a script that asks the users for their favorite color, which is then printed to the terminal. Question should be printed in the way that user would enter the color on the same line as question.
2. Practice printf command by executing the commands listed in examples.
3. Write a script that prints the first argument of the script in the format %4d.
4. Write a script that reads from standard input a number, that will be printed by following specifications: prints sign, total length of number will be five, empty space will be filled with zeros:

   The entered number is: +0003
5. What if to the script from the exercises 4 we submit number: 123456?

Home works

Arithmetic operations

Arithmetic operations

Arithmetic of integer operations is possible to do in the block **((...))**.

Possible entries:

prints the value of the result in the place of the entry

```
(( I = I + 1 ))
(( I++ ))
I=$(( I + 1 ))
echo "Value I is increased by one : $(( I + 1 ))"
```

Operators:

`=`	assignment
`+`	addition
`-`	subtraction
`*`	multiplication
`/`	division
`%`	division remainder
`++`	increment (increase value by 1)
`--`	decrement (decrease value by 1)

More information: man bash

Command expr

Command **expr** interpretes mathematical expressions, the results are printed to stadart output.

Example:

```
$ expr 1 + 2
3

$ expr 2 \* 3
6

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

\ prevents the expansion of the special character ***** to file and directory names located in current directory

Work with the values of the variables

Result is assigned into the variables I

More information: man expr

Another option is to use the command **bc**, that can work with real numbers.

Home work

1. Write a script, which will submit two numbers as arguments. The script prints the numbers and the sum of the numbers
2. Write a script that asks gradually for two numbers given by user. After entering the numbers, it prints product of the numbers.