

C2110 *UNIX and programming*

Lesson 6 / Module 1

PS / 2020 Distance form of teaching: Rev1

Petr Kulhanek

kulhanek@chemi.muni.cz

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

Variables

Variables

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

Variable settings:

```
$ VARIABLE_NAME=value  
$ VARIABLE_NAME="value with spaces"
```

can not be a gap between **variable name** and =



Access to the value of a variable:


```
$ echo $VARIABLE_NAME
```

"TEXT \$ {VARIABLE}TEXT"

To delete a variable:

```
$ unset NAME_PROMENNE
```

if the value is to be part of a text, the variable name is enclosed in braces



Overview of all set variables:

```
$ set
```

Variable Setting



```
$ VARIABLE_NAME="value with spaces"
```

```
$ VARIABLE_NAME="value with spaces"
```

interpreted as name of program gap

interpreted as argument of program

```
$ VARIABLE_NAME= "value with spaces"
```

VARIABLE_NAME is set to an empty string,
value of the variable is available only to the
running program

interpreted as the name of the program



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

you can specify several variables and their
values (pairs NAME=VALUE are separated by
a space), which are only available for the
running program

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

Strings

In the Bash language, four types of strings can be used:

- **without quotes**

A=try

B=* ←

C=\$A ←

there is no expansion (it is not an argument, but the value of a variable)

is replaced by the value of the variable A

- **with quotation marks**

A="foo bar" ←

B="* \$A" ←

value of variable contains two words separated by a space

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

- **with single quotes (apostrophe)**

A='foo bar' ←

B='* \$ A' ←

text is given exactly, without any expansion or transformation

- **with inverted single quotes (inverted apostrophe)**

A="`ls -d`"

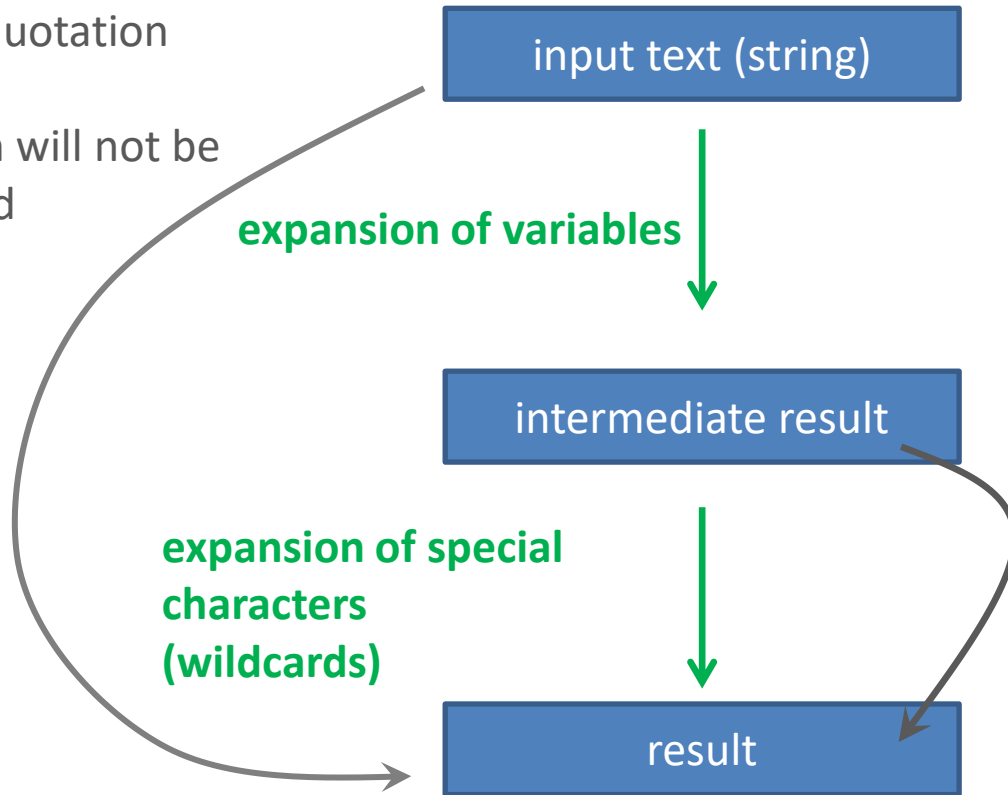
B="count : `ls | wc -l`"

standard output of **command** indicated in inverted quotation marks is placed in between them

String/Command Line Expansion

String/command line expansion order:

if the text is enclosed in single quotation marks ('), expansion will not be performed



if the text is enclosed in standard quotation marks (") or is not contained in a word that could be expanded, the expansion will not be performed

More details: man bash

Exercise I

Work in interactive mode of shell.

1. Set value of variable A to 55.
2. Print the value of the variable A (command echo)
3. List all variables set in the given terminal. Is there a variable A between them? Use the command less or more to clarify the statement.
4. Use the grep command to list only the line containing the record for variable A. Choose a search pattern that is independent of the value of the variable.
5. List all set 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. Delete the variable A.
9. Verify that you deleted the variable (following the procedure in step 4).
10. Set variables A, B and C one by one according to the examples on slide 5. Check their value step by step by set and echo commands. Analyze any discrepancies.

Arithmetic Operations

Arithmetic operations

- Arithmetic operations with integers can be performed in `((...))` block.
- Characters in the block are interpreted as variable names. Therefore, it is not necessary to use `$` operator to obtain their value.
- The values of the variables are interpreted as integers. If the conversion fails, a value of zero is used.


Possible entries:

```
(( I = I + 1 ))  
(( I++ ))
```

Arithmetic operation with obtaining the result:

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

value of the result is place to
the position of the sign



More details: `man bash`

Arithmetic Operations

- Arithmetic operations with integers can be performed in `((...))` block.
- Characters in the block are interpreted as variable names. Therefore, it is not necessary to use `$` operator to obtain their value.
- The values of the variables are interpreted as integers. If the conversion fails, a value of zero is used.

Possible entries:

```
(( I = I + 1 ))  
(( I++ ))
```

more appropriate notation

Arithmetic operation with obtaining the result:

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

value of the result is place to
the position of the sign

Next information: man bash

Operators

- = assignment
- + addition
- subtraction
- * multiplication
- / **integer** division
- % rest after **integer** division (modulo)
- ++ increment (increase value by 1)
- decrementation (decrease in value by 1)

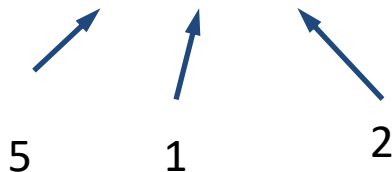
Example:

A=5

((B = A / 3))

((C = A % 3))

echo \$A \$B \$C



Command `expr`



Command `expr` evaluates mathematical expressions, results are printed to standard output.

Examples:

```
$ expr 1 + 2  
3
```

`\` prevents the special character `*` from expanding to the names of files and directories located in the current directory

```
$ expr 2 \* 3  
6
```

we pass the value of the variable, necessary to use the operator `$`

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

More details: `man expr`

we insert the result into the variable `A`

Another option is to use the command `bc`, which can work with real numbers.

Exercises II

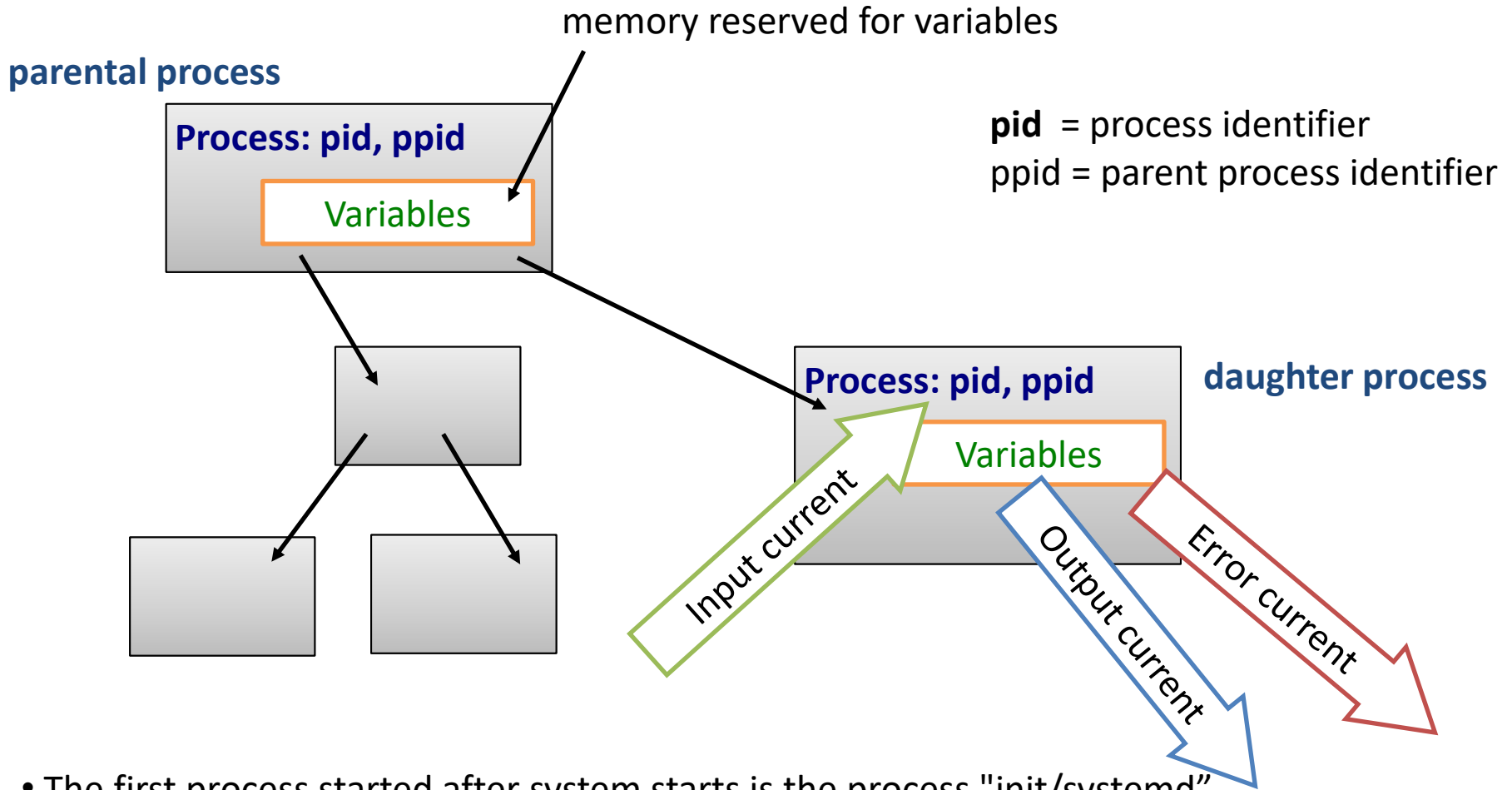
1. Write a script in which you set the variables A and B to the values 5 and 6. Next, list the value of their sum, difference, proportion and multiplication using the echo command.

Variables and Processes



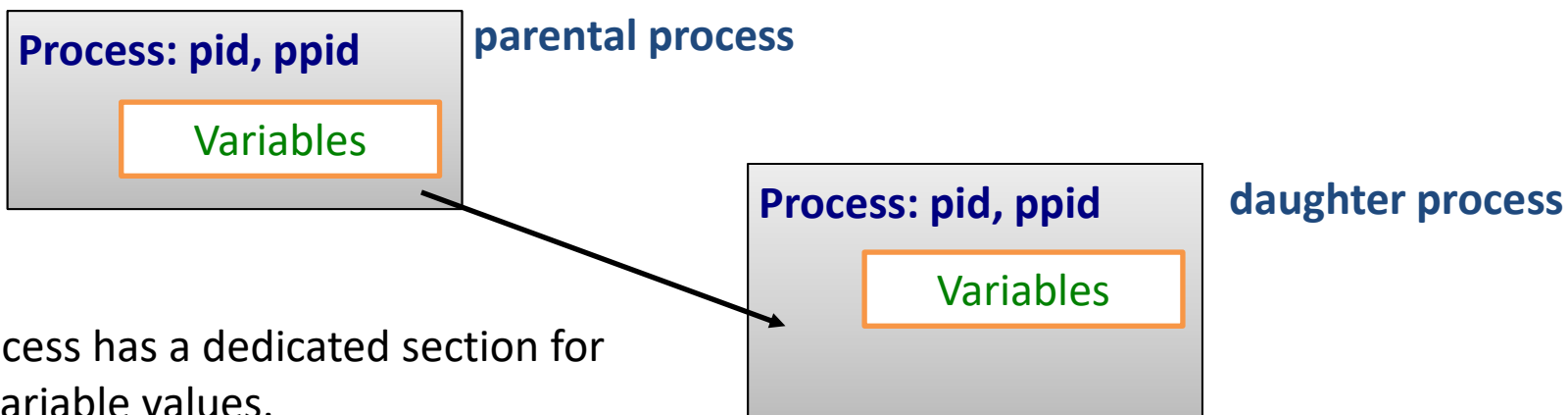
Processes

Process is an instance of a running **program**.



- The first process started after system starts is the process "init/systemd"
- Every command run in shell (command line) is a process

Variables and Processes



Each process has a dedicated section for storing variable values.

The child process at the time of its start **gets a copy of** variables (exported) and their values from the parent process. You can change or delete these variables as needed. It can also set or delete new variables. **However, all the changes will disappear after the end of the daughter process.** Changes taking place **do not manifest** on values of **original variables** in the parent process.

Variable export:

```
$ export VARIABLE_NAME ← export
```

```
$ export VARIABLE_NAME="value" ← export with assignment
```


Exercise III

Work in a new terminal.

1. Clear the PATH variable. How will the change affect command line functionality? Try running the command `ls` and `pwd`. Explain the behavior.
2. When is the expansion of wildcard `*` taking place in the following example:

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

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

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