C2110 UNIX and programming

6th Lesson

Petr Kulhánek, Jakub Štěpán

kulhanek@chemi.muni.cz

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

european social fund in the czech republic

EUROPEAN UNION

MINISTRY OF EDUCATION, YOUTH AND SPORTS

OP Education for Competitivness

INVESTMENTS IN EDUCATION DEVELOPMENT

CZ.1.07/2.2.00/15.0233

Contents

Scripts

- Scripts vs. programs
- Program compilation
- Running sample script and program

Variables

- Setting and removing variables
- Variables and processes
- String types

Scripts

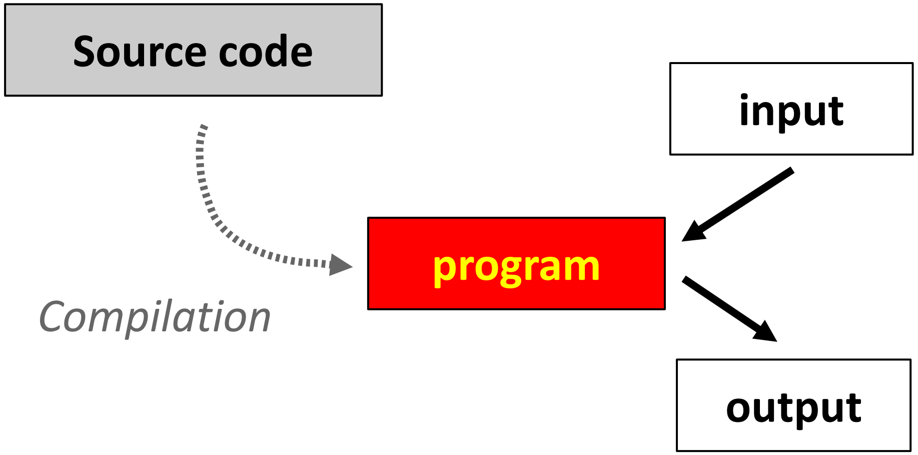
- Scripts vs. programs
- Program compilation
- Running sample script and program

Programs *vs.* Scripts

Program is machine instruction file processed directly by processor. It is created by procedure called **compilation** from source code.

Compiled languages:

C/C++
Fortran



Source code

Compilation

program

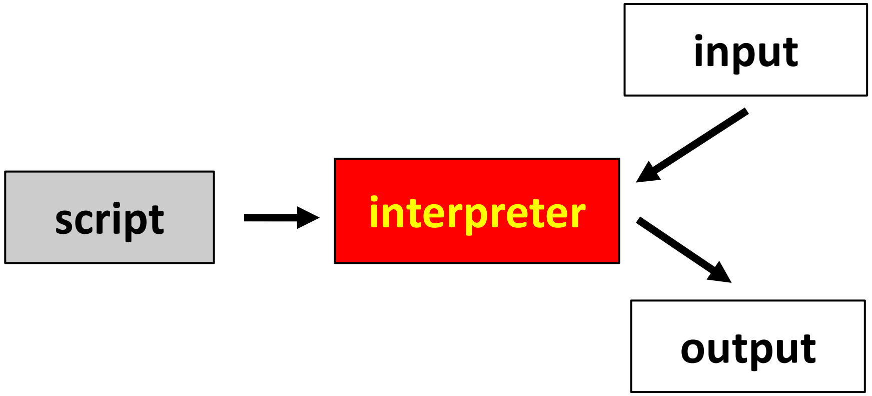
input

output

Script is text file containing commands and special constructions, these are processed by interpreter of scripting language.

Skripting languages:

bash
gnuplot
awk
JavaScript
PHP



script

interpreter

input

output

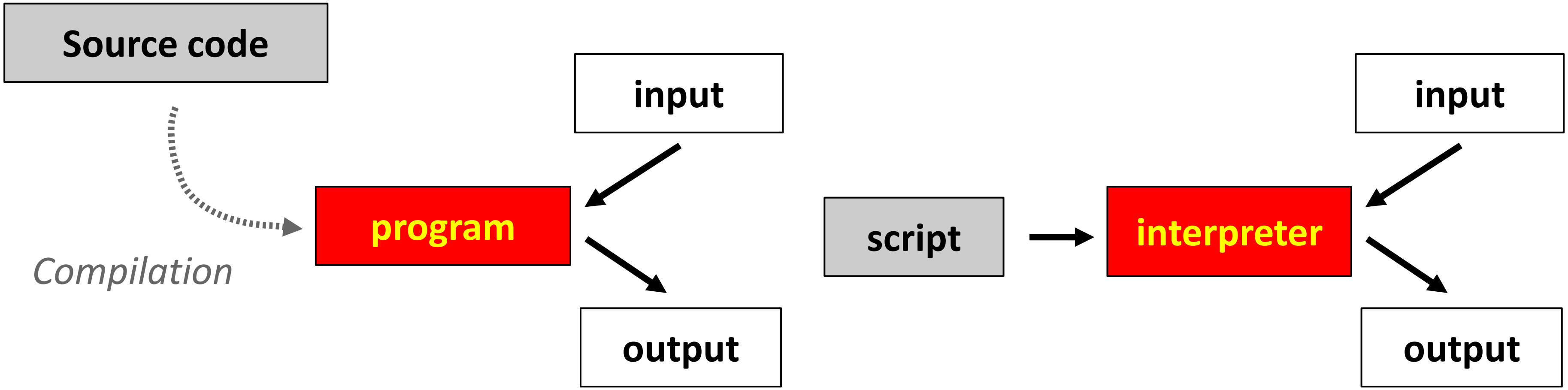
Programs *vs* Scripts, ...

- Easy optimization
- Fast processing

- Recompilation needed
- Self run code not available

- No recompilation
- Program can generate and run self running code

- Poor optimization
- Slower processing



Source code

Compilation

program

input

output

script

interpreter

input

output

How to write programs and scripts

Scripts are text files – thus any text editor can be used, that enables saving pure text (without any format metadata).

Text editors:

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

For complex programs and scripts development environments can be used – **IDE** (Integrated Development Enviroment). IDE contains next to editor extra tools as: project manager, debugger and more. Usually for more advanced and complex languages: *JavaScript*, *Python*, *PHP*, etc.

IDE:

- Kdevelop
- qtcreator
- NetBeans
- Eclipse

Program in C

Source code

```
#include <stdio.h>

int main(int argc,char* argv[])
{
   printf("This is C program! \n");
   return(0);
}
```

Compilation

```
$ gcc program.c -o program
```

C language compiler

Program name

Running program

```
$ ./program
```

file **program** needs permission to **execute**

Program in Fortran

Source code

```
program Hello

  write(*,*) 'This is Fortran program!'

end program
```

Compilation

```
$ gfortran program.f90 -o program
```

Fortran language compiler

Program name

Running program

```
$ ./program
```

file **program** needs permission to **execute**

Script in Bash

Script

```
#!/bin/bash

echo 'This is Bash script!'
```

Running script

```
$ bash script.bash
```

interpret Bash

file **script.bash** **does not need** permissions to **execute**

Script in gnuplot

Script

```
#!/usr/bin/gnuplot

set title "This is gnuplot script!"
plot sin(x)

pause -1
```

Running script

```
$ gnuplot skript.gnuplot
```

interpret gnuplot

file **script.bash** **does not need** permissions to **execute**

Exercise

1. Create four directories with names **task01, task02, task03, task04**
2. From directory **/home/kulhanek/Data/programs** copy **program.c , program.f90, skript.bash, a skript.gnuplot** to particular directories you created in 1.
3. Compile source codes of language C and Fortran. Run compiled programs.
4. What is size of compiled program in C language? Open program file in text editor, what is inside?
5. Run scripts **skript.bash a skript.gnuplot**.

Running scripts

1) Un-direct running

We run interpreter and as its argument we put script name.

```
$ bash my_bash_script_name

$ gnuplot my_gnuplot_script_name
```

Scripts **does not need** permission x (executable).

2) Direct running

We run directly script (shell runs interpreter automatically).

```
$ ./my_bash_script

$ ./muj_gnuplotu_script
```

Scripts **must have** x (**executable**) set and interpreter (first script line).

Interpreter specification

Interpreter specification (first script line):

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

Script in bash

```
#!/bin/bash

echo “This is bash script!"
```

Skript in gnuplot

```
#!/usr/bin/gnuplot

set xrange[0:6]

plot sin(x)

pause -1
```

- If no interpreter is specified, then system shell interpreter is used.
- Interpreter is ignored in case of un-direct running.

Interpreter specification

If absolute path may be changed over time (for example by using software modules), it may be specified dynamically:

```
#!/usr/bin/env interpreter
```

Interpreterhas to be in system path of variable PATH.

Script in bash

```
#!/usr/bin/env bash

echo "This is bash script!"
```

Script in gnuplot

```
#!/usr/bin/env gnuplot

set xrange[0:6]

plot sin(x)

pause -1
```

Exercise

1. Change access permissions to files **skript.bash a skript.gnuplot** (command **chmod**).
2. Make sure that scripts can be run directly.
3. What happens when we use interpreter bash for **script skript.gnuplot**?

Variables

- Variable setting and unsetting
- Variables and processes
- String types

Variables

In Bash language variable is **named memory place**, that contain value. Variable value is **always** of type **string (test)**.

No space between variable name and **=**

Variable set:

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

Access to variable value:

```
$ echo $VARIABLE_NAME
```

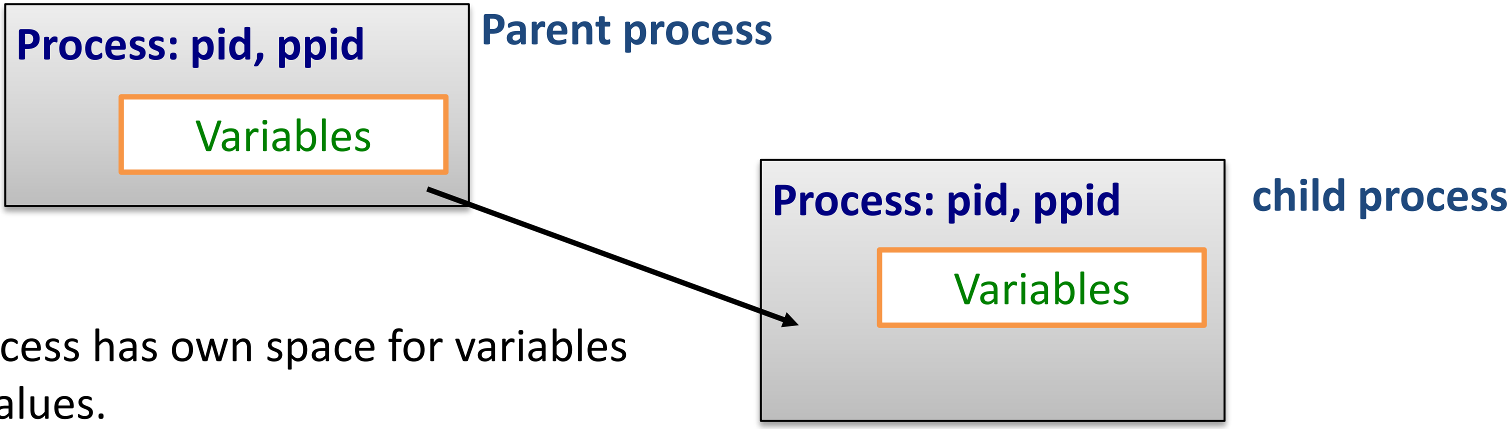
Unsetting of variable:

```
$ unset VARIABLE_NAME
```

Overview of all variables:

```
$ set
```

Variables and processes



Each process has own space for variables and its values.

Child process when started gets **copy** of **exported** variables and its values from parent process. These variables can be changed by any way or remove them and new variables can be defined too. **All these changes are not visible to original variables in parent process and are deleted when child process finishes.**

Export proměnné:

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

export

export with assignment

Strings

In Bash llanguage there are four string types:

- **no quotes**

```
A=pokus
B=*
C=$A
```

Expands to list of files and directories in current working directory (advanced constructions can be used)

Value of variable A is inserted

- **with quotes**

```
A="pokus hokus"
B="* $A"
```

Variable contains value with 2 words separated by space

Value of A is inserted but no expansion is done (star is in quotes)

- **single quote (apostrophe)**

```
A='pokus hokus'
B='* $A'
```

Text is saved in exact way, no variable insertion, no expansion is done.

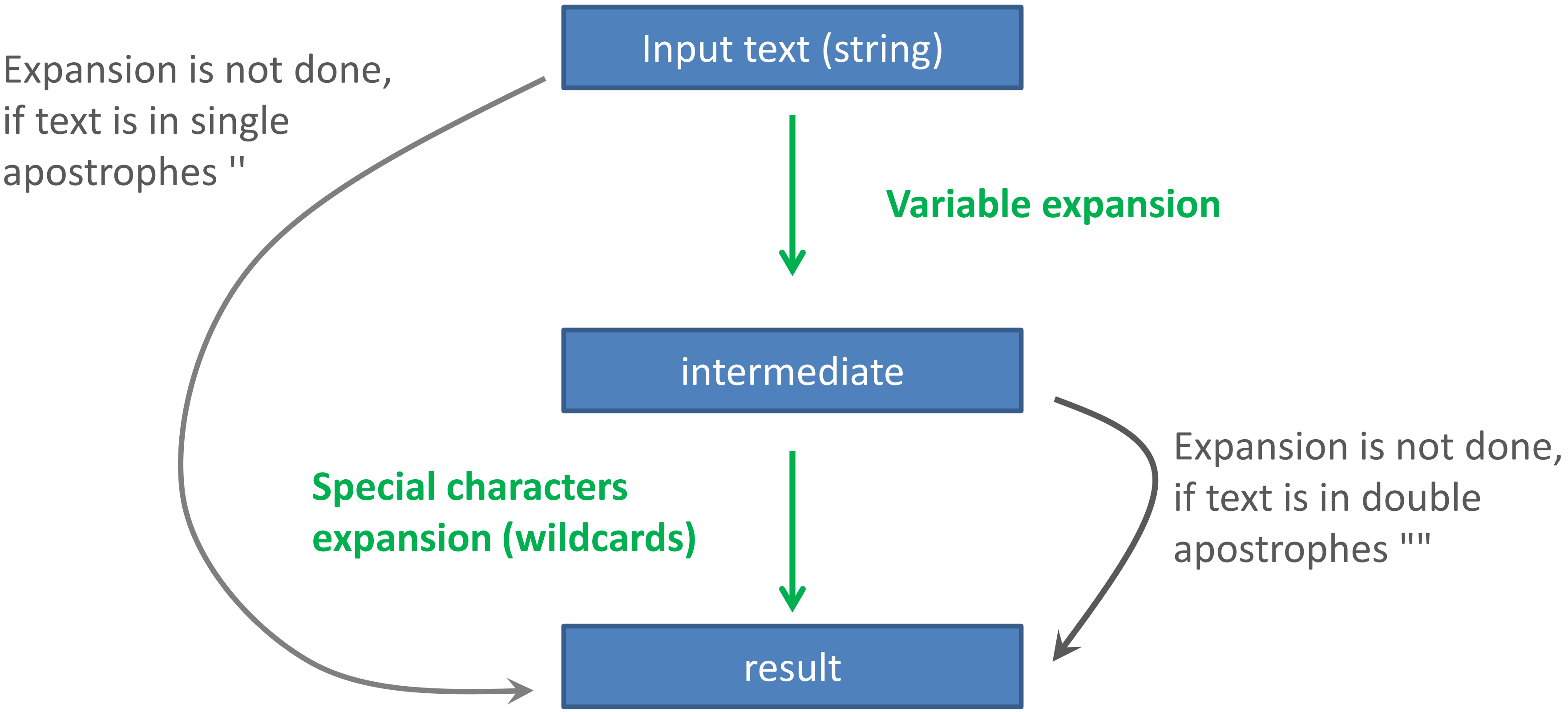
- **backward single quote (backward apostrophe)**

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

To **place** where are backward apostrophes, **command output** is inserted

Variables and special symbols

Text expansion order:



Input text (string)

Variable expansion

intermediate

Special characters expansion (wildcards)

result

Expansion is not done, if text is in single apostrophes ''

Expansion is not done, if text is in double apostrophes ""

Commands for exercise

more prints text from file or standard input by pages (appropriate to view long texts)

less similar to **more** with extended functionality (for example movement to both directions in text)

xargs runs command with arguments that are from standard input. Appropriate to create long argument list.

grep prints lines from files or standard input that match given search PATTERN

Examples:

```
$ set | more
    lists existing variables and functions by pages
$ cat *.txt | less
    prints contents of all files with extension .txt by pages
$ cat directory_list.txt | xargs mkdir
    creates directories with names according to contents of file
    directory_list.txt
$ grep AHOJ file.txt
    prints particular lines from soubor.txt, that contain text AHOJ
```

Exercise

1. Set variable **A** to value 55.
2. Print value of variable **A** (command **echo**)
3. List all variables. Is there variable A (try to use command **less** and **more**)?
4. Use command **grep** and print line containing variable **A** record. Select search pattern independent on variable value.
5. Print all variables with name beginning with **A** (grep ^TEXT).
6. Change variable **A** value to **"this is long string"**.
7. Print value of variable **A**.
8. Unset variable **A**.
9. Make sure it is unset (use procedure as in 4).
10. Set variables **A**, **B** and **C** as on previous page 19. Check their values by **set** or **echo**.
11. Create file **directories.txt**, with words **pokus1**, **pokus2**, **pokus3** on separate lines. Use command **xargs** to create directories of same names.