

# C2110 *UNIX and programming*

## Lesson 5 / Module 3

**PS / 2020 Distance form of teaching: Rev2**

Petr Kulhanek

[kulhanek@chemi.muni.cz](mailto:kulhanek@chemi.muni.cz)

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

# Bash

---

<https://www.gnu.org/software/bash/>

# Bash - Overview

**Unix shell** (also command processor, literally "unix shell") is the name of **text user interface**, which is the predecessor of the graphical user interface.

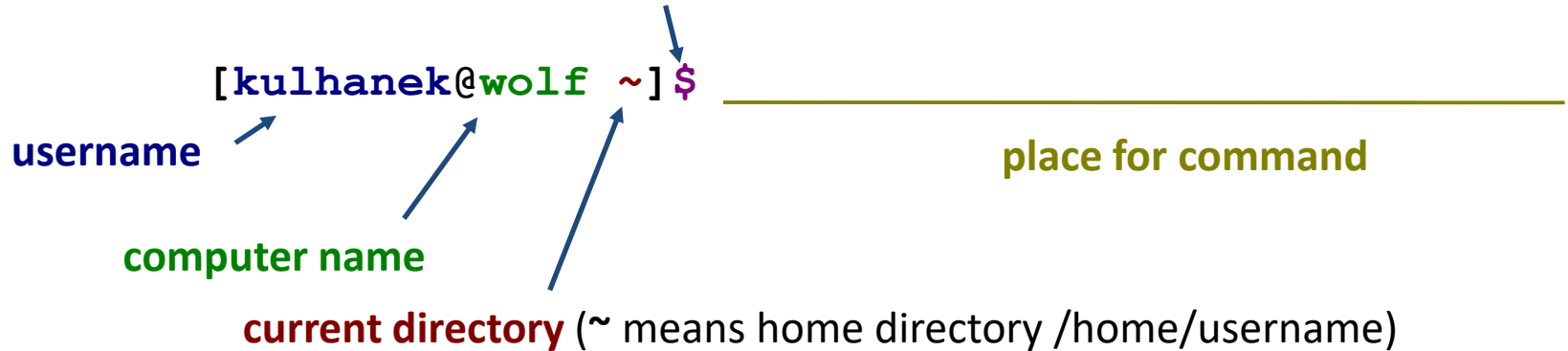
One of the Unix shells is **Bash**.

**Bash** is POSIX shell with several extensions. It is designed for operating systems based on the GNU project and can be run on most Unix operating systems. It is used as the default command interpreter in systems built on Linux kernel, as well as on Mac OS X or Darwin system. It can also be used on Microsoft Windows using the subsystem for Unix application (SUA), or POSIX emulation using software **Cygwin** and MSYS.

<https://cs.wikipedia.org>

# Interactive Mode

Prompt - type of user / prompt (\$ regular user, # super user, other possible %, >)



The command is executed by pressing **Enter** key.

**History:** Use the up and down arrow keys to scroll through the list of commands you have already entered. The history command can be reused or modified and the modified one can be used. The history is also accessible by command **history**.

**Autocomplete:** by pressing Tab key the command line interpreter tries to complete the spelled word. Command names, paths and file names are added (if one press does not cause anything, there are more options to add and repeated press will display them).

Shell interprets (expands) **wild characters and other special characters**, before the actual execution of the command. In interactive mode it is possible to **run control structures** of bash language.

Interactive mode is terminated by command **exit**.


# Bash Script

```
#!/bin/bash
# this is a comment
echo `This is a script in Bash interpreter!`
echo `Content of directory `pwd` is:`
ls    # prints content of directory
A=6   # sets value of variable A
echo `Value of variable A is $A`
echo `first command`; echo `second command`
./mycommand first_argument second_argument \  
    third_argument
```

order of command execution



immediately follows new line



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

# Where to write scripts (/programs)

Because **scripts** and program source codes are **text files**, you can use any text editor that allows you to save text in plain form (without formatting metadata).

## Text editors:

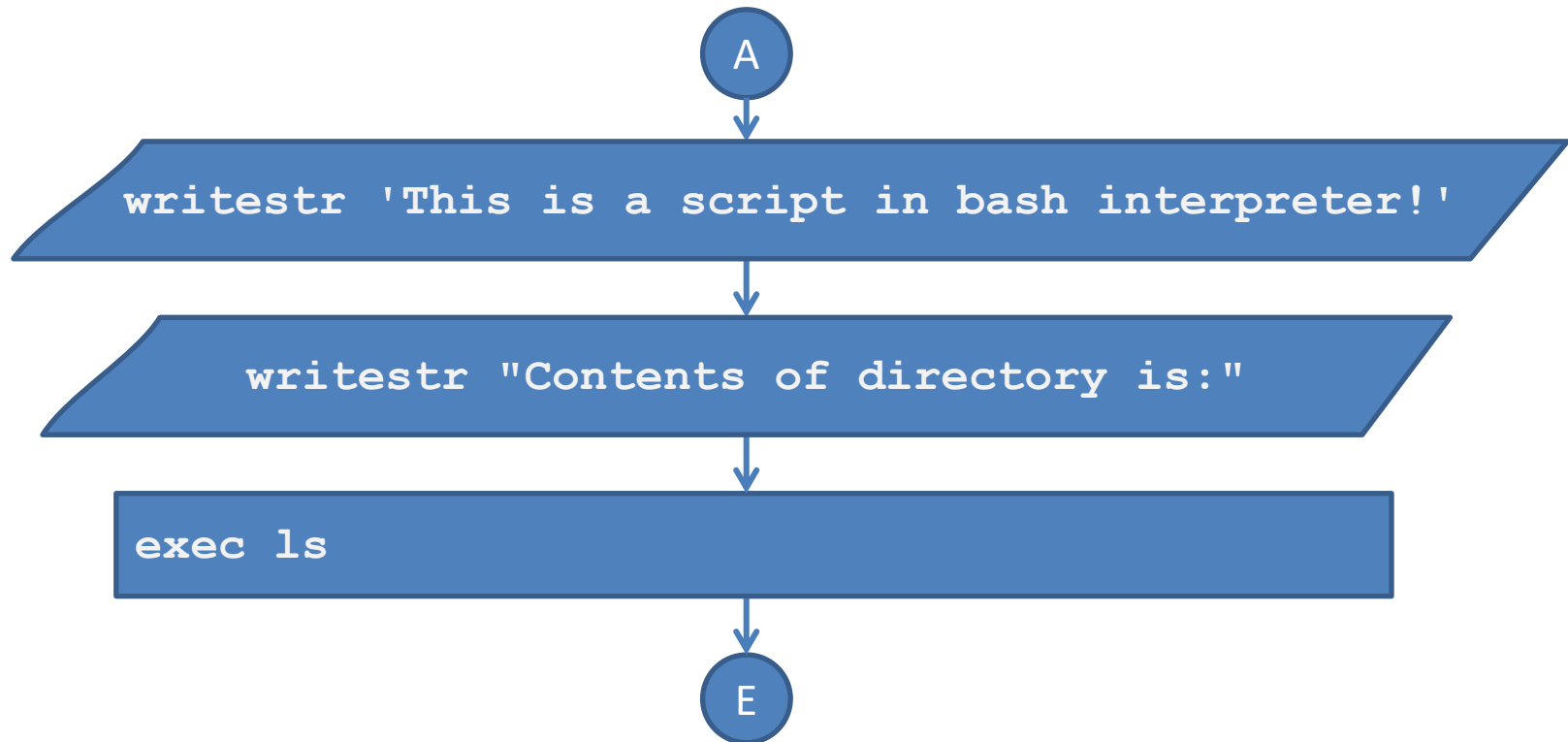
- vi
- kwrite
- kate
- **gedit**

A specialized development environment can also be used to write scripts and program source codes - **IDE (Integrated Development Environment)**. In addition to the editor, the IDE also contains a project manager, debugging tools (debugger) and more. Mostly available for more complex languages: *C, C++, Fortran, JavaScript, Python, PHP*, etc.

## Development environment:

- Kdevelop
- qtcreator
- NetBeans
- Eclipse

# Flowchart and Bash Script



order of command  
execution

```
#!/bin/bash
```

```
echo 'This is a script in bash interpreter!'
```

```
echo "Contents of directory is:"
```

```
ls
```



# Non-interactive Mode - Scripts

## 1) Indirect start

We run the language interpreter and give the name of the script as an argument.

```
$ bash my_bash_script
```

Scripts **do not have to** have the x flag set (executable).

## 2) Direct start

We run the script directly (shell automatically starts the interpreter).

```
$ chmod u+x my_bash_script
```

```
$ ./my_bash_script
```

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

```
#!/bin/bash ←
```

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



# Change of Access Rights

Access rights determine what operations a user can perform on files or directories in the file system.

## Access rights:

<b>r</b>	ability to read file	to list contents of directory
<b>w</b>	ability to change file	to change contents of directory (create or delete file/directory)
<b>x</b>	ability to run the file	to enter directory

Each file or directory has a designated owner and user group. Access rights are listed separately for **file owner (u)**, **user group (g)** and **other users (o)**.

```
$ ls -l  
  u  g  o ← access rights  
-rw-rw-r--  1 kulhanek lcc  858 2008-10-17 11:58 script.sh
```

```
$ chmod u+x script.sh    add (+) / remove (-) of a right
```

```
$ ls -l  
-rwxrw-r--  1 kulhanek lcc  858 2008-10-17 11:58 script.sh
```

# Specification of Interpreter

Interpreter specification (first line of script):

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

Bash script

```
#!/bin/bash  
  
echo "This is a bash script!"
```

Gnuplot script

```
#!/usr/bin/gnuplot  
  
set xrange[0:6]  
  
plot sin(x)  
  
pause -1
```

- If script interpreter is not specified when it is run directly, the system interpreter is used shell (bash).
- The interpreter specified in the script is ignored during indirect execution.
- It is appropriate to always include the interpreter in the script, because it is used by text editors to highlight syntax.

# Commands for Exercises

**echo** prints the entered text (string)

**grep** prints lines that contain the specified PATTERN

## Examples of use:

```
$ echo "The result is:"  
prints the string "The result is:"
```

```
$ grep TEMP rst.out  
prints each line from the file rst.out which contains the string "TEMP"
```

The grep command is very useful, so I recommend that you go through its detailed documentation (`man grep`)

# Exercise 1

## Interactive mode:

1. Run the command `bash` in the terminal. What happened? End the second session with the `exit` command.
2. Using commands `ps`, find out the number of the process that interprets the command line.
3. End the process with the command `kill -9 pid`, where `pid` the process number). What will happen?

# Exercises 2

## Scripting:

1. Create a directory named **script**.
2. To the directory, copy the file **script.bash** from the directory `book /home/kulhanek/Documents/C2110/Lesson05/programs`.
3. Run directly script **script.bash**.
4. Create a directory named **measurement**.
5. To the directory, copy the file **mereni1.log** from the directory `/home/kulhanek/Documents/C2110/Lesson05/data`.
6. Using suitable commands, extract the line containing the temperature of measurement from file **mereni1.log** (a description of the data is given in the analyzed file).
7. In a text editor, write a script that will print the temperature and pressure of the measurement from the file **mereni1.log**

# Additional Information

---



# Specification of Interpreter II

In case the absolute path to the interpreter changes (e.g., when using software modules), it is advisable to use the following construction:

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

The interpreter must be in one of the directories specified by the PATH system variable.

## Bash script

```
#!/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
```