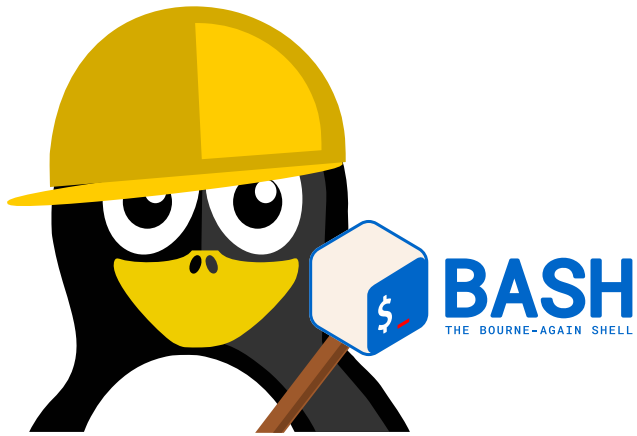
Leiden University
Medical Center

Scripting for Life Science Researchers

Basic Scripting



Mihai Lefter

Department of Human Genetics

Outline

- Introduction.
- First script.
- Variables.
- Special variables.
- Environment variables.
- Command substitution.
- Command line arguments.
- Introduction to processes.
- Interact with user input.

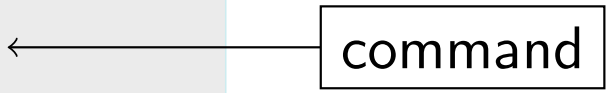
Scripts

- Plain text files containing a list of successive commands.
  - Anything you can normally run on the command line.
  - Executed sequentially.
  - Redirection possible.
  - Variables.
  - Flow control.
  - ...

First script

first_script.sh

```
1 echo 'Hello World!'
```

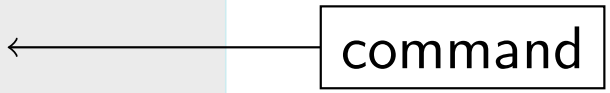


First script

Execution

first_script.sh

```
1 echo 'Hello World!'
```



terminal

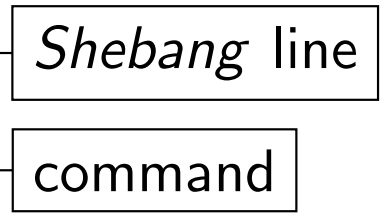
```
$ bash first_script.sh
Hello World!
```

First script

The shebang

first_script.sh

```
1  #!/bin/bash
2
3  echo 'Hello World!'
```



- The file interpreter follows the `#!` character sequence:
  - In this case is bash (the path to the actual interpreter).
- The `#!` must be the first characters in the script file!

The shebang - execution

first_script.sh

```
1  #!/bin/bash
2
3  echo 'Hello World!'
```

Shebang line

command

terminal

```
$ ./first_script.sh
bash: ./first_script.sh: Permission denied
```

The shebang - execution

first_script.sh

```
1  #!/bin/bash
2
3  echo 'Hello World!'
```

Shebang line

command

terminal

```
$ ./first_script.sh
bash: ./first_script.sh: Permission denied
$ ls first_script.sh
-rw-r--r-- 1 mlefter Users 34 aug 21 14:39 first_script.sh
```

The shebang - execution

first_script.sh

```
1  #!/bin/bash
2
3  echo 'Hello World!'
```

Shebang line

command

terminal

```
$ ./first_script.sh
bash: ./first_script.sh: Permission denied
$ ls first_script.sh
-rw-r--r-- 1 mlefter Users 34 aug 21 14:39 first_script.sh
$ chmod u+x scriptname # changing permission
```

The shebang - execution

first_script.sh

```
1  #!/bin/bash
2
3  echo 'Hello World!'
```

Shebang line

command

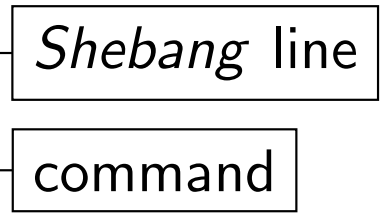
terminal

```
$ ./first_script.sh
bash: ./first_script.sh: Permission denied
$ ls first_script.sh
-rw-r--r-- 1 mlefter Users 34 aug 21 14:39 first_script.sh
$ chmod u+x scriptname # changing permission
$ ls first_script.sh
-rwxr--r-- 1 mlefter Users 34 aug 21 14:39 first_script.sh
```

The shebang - execution

first_script.sh

```
1  #!/bin/bash
2
3  echo 'Hello World!'
```



terminal

```
$ ./first_script.sh
bash: ./first_script.sh: Permission denied
$ ls first_script.sh
-rw-r--r-- 1 mlefter Users 34 aug 21 14:39 first_script.sh
$ chmod u+x scriptname # changing permission
$ ls first_script.sh
-rwxr--r-- 1 mlefter Users 34 aug 21 14:39 first_script.sh
$ ./first_script.sh
Hello World!
```

Naming convention

- Commonly the suffix ".sh" is appended:
  - To make it clear the file is a shell script.
  - Not required in order to work - Linux is extensionless.

Comments

- Start with the # character.
- Continue to the end of the line.

first_script.sh

```
1 #!/bin/bash
2 # This is a very simple script.
3
4 echo 'Hello World!' # command to be executed
```

- It's wise to comment the code:
  - Help with debugging or tests.

Variables

- A label assigned to a location or set of locations in computer memory holding an item of data.
- Used in:
  - Arithmetic operations.
  - Quantities manipulation.
  - String parsing.
- Variables are very useful in making scripts easier to manage.

Variables

Declaring and refering to

first_script.sh

```
1  #!/bin/bash
2
3  NAME=World!          # declaring and assigning a variable
4  echo Hello $NAME     # refering to a variable
```

terminal

```
$ ./first_script.sh
Hello World!
```

Variables

Declaring and refering to - good practice

first_script.sh

```
1  #!/bin/bash
2
3  NAME=World!          # declaring and assigning a variable
4  echo Hello ${NAME} # refering to a variable
```

terminal

```
$ ./first_script.sh
Hello World!
```

Variables

Bash is both space and case sensitive

first_script.sh

```
1  #!/bin/bash
2
3  NAME = World!       # declaring and assigning a variable
4  echo Hello ${NAME} # refering to a variable
```

terminal

```
$ ./first_script.sh
./first_script.sh: line 3: NAME: command not found
Hello
```

Variables

Bash is both space and case sensitive

first_script.sh

```
1  #!/bin/bash
2
3  NAME=World!          # declaring and assigning a variable
4  echo Hello ${name} # refering to a variable
```

terminal

```
$ ./first_script.sh
Hello
```

Quotes

terminal

```
$ message=Hello World!
World!: command not found
```

Quotes

terminal

```
$ message=Hello World!
World!: command not found
% $ message='Hello World!' # ' treat every character
 ↪   literally
```

Quotes

terminal

```
$ message=Hello World!
World!: command not found
$ message='Hello World!' # ' treat every character literally
$ echo ${message}
Hello World!
```

Quotes

terminal

```
$ message=Hello World!
World!: command not found
$ message='Hello World!' # ' treat every character literally
$ echo ${message}
Hello World!
$ name=World!
$ message='Hello ${name}' # name variable is not substituted
$ echo ${message}
Hello ${name}
```

Quotes

terminal

```
$ message=Hello World!
World!: command not found
$ message='Hello World!' # ' treat every character literally
$ echo ${message}
Hello World!
$ name=World!
$ message='Hello ${name}' # name variable is not substituted
$ echo ${message}
Hello ${name}
$ message="Hello ${name}" # " allow for value substitution
$ echo ${message}
Hello World!
```

Quotes - it's good practice to use them

first_script.sh

```
1  #!/bin/bash
2
3  NAME='World!'          # declaring and assigning a variable
4  echo "Hello ${NAME}" # refering to a variable
```

terminal

```
$ ./first_script.sh
Hello World!
```

Quiz 1

What's the output of the following script?

quiz1.sh

```
1  #! /bin/bash
2
3  path='/home/student01/'
4  echo "\${path}=${path}"
```

Quiz 1

What's the output of the following script?

quiz1.sh

```
1  #! /bin/bash
2
3  path='/home/student01/'
4  echo "\${path}=${path}"
```

terminal

```
$ ./quiz1.sh
$path=/home/student01/
```

Variables

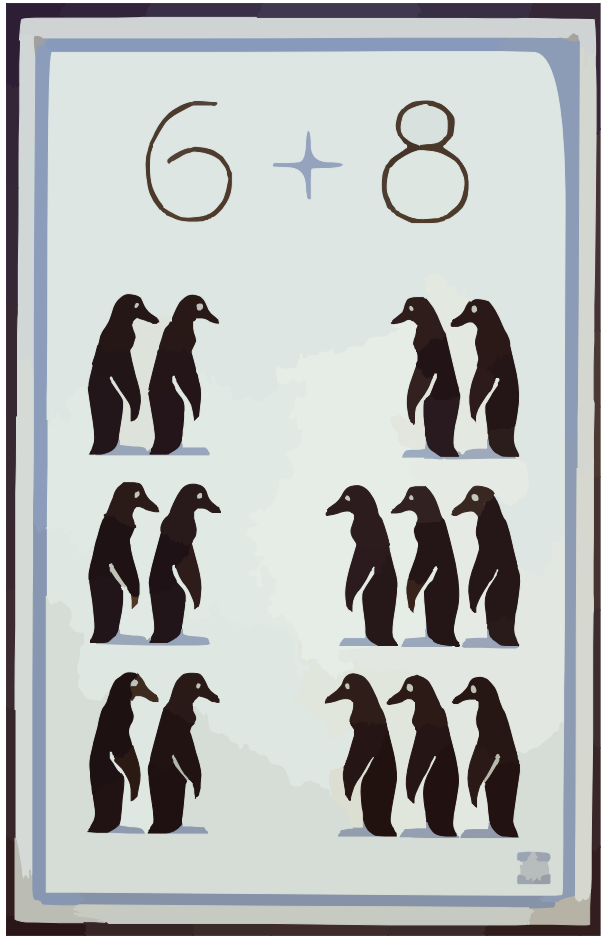
Types

- No variable types:
  - A blessing and a curse.
- By default everything is a string.
- Arithmetic operations and comparisons on variables is permitted:
  - The variable should contain only digits.

Arithmetic

Only integer operations are possible.

- let
- expr
- **((expression)) - detailed next**



((expression)) - arithmetic expansion

terminal

```
$ a=5
$ b=$(( ${a} + 3 ))    # Spaces allowed but not required
$ echo ${b}
8
```

((expression)) - arithmetic expansion

terminal

```
$ a=5
$ b=$(( ${a} + 3 ))    # Spaces allowed but not required
$ echo ${b}
8
$ echo $(( a + 10 ))   # You can use variables directly
15
```

((expression)) - arithmetic expansion

terminal

```
$ a=5
$ b=$(( ${a} + 3 ))    # Spaces allowed but not required
$ echo ${b}
8
$ echo $(( a + 10 ))   # You can use variables directly
15
$ (( a *= 5 ))         # Operation performed
$ echo ${a}
25
```

((expression)) - arithmetic expansion

terminal

```
$ a=5
$ b=$(( ${a} + 3 ))    # Spaces allowed but not required
$ echo ${b}
8
$ echo $(( a + 10 ))   # You can use variables directly
15
$ (( a *= 5 ))         # Operation performed
$ echo ${a}
25
$ echo $(( a / 25 ))   # Division
1
```

((expression)) - arithmetic expansion

terminal

```
$ a=5
$ b=$(( ${a} + 3 ))    # Spaces allowed but not required
$ echo ${b}
8
$ echo $(( a + 10 ))   # You can use variables directly
15
$ (( a *= 5 ))         # Operation performed
$ echo ${a}
25
$ echo $(( a / 25 ))   # Division
1
$ (( ++a ))            # Pre-increment a
$ echo ${a}
2
```

Quiz 2

What's the execution output of the following script?

quiz2.sh

```
1  #!/bin/bash
2
3  a=0
4  b='1+a'
5
6  echo $b
7  echo $(( 10 - 5 / (b + 1) ))
```

Quiz 2

What's the execution output of the following script?

quiz2.sh

```
1  #!/bin/bash
2
3  a=0
4  b='1+a'
5
6  echo $b
7  echo $(( 10 - 5 / (b + 1) ))
```

terminal

```
$ ./quiz2.sh
1+a
8
```

Command Substitution (Expansion)

$(COMMANDS)

- Replace the output of a command with the command itself.
  - Expanded commands are executed in a subshell.
  - Their *stdout* data is what the substitution syntax expands to.

terminal

```
$ date
ma 28 aug 2017 8:31:33 CEST
$ echo Current date is $(date)
Current date is ma 28 aug 2017 8:31:36 CEST
```

Command Substitution (Expansion)

$(COMMANDS)

terminal

```
$ pwd
/home/student01
$ cwd=$(pwd)
$ cd project_genetics
$ pwd
/home/student01/project_genetics
$ echo "moved from ${cwd} to $(pwd)"
moved from /home/student01 to /home/student01/project_genetics
$ cd ${cwd}
$ pwd
/home/student01
```

Quiz 3

What is the difference between lines 2 and 3?

quiz3.sh

```
1 #!/bin/bash
2 echo ${PWD}
3 echo $(pwd)
```

Quiz 3

What is the difference between lines 2 and 3?

quiz3.sh

```
1 #!/bin/bash
2 echo ${PWD}  # access variable
3 echo $(pwd)  # command substitution
```

Quiz 3

What is the difference between lines 2 and 3?

quiz3.sh

```
1 #!/bin/bash
2 echo ${PWD}  # access variable
3 echo $(pwd)  # command substitution
```

terminal

```
$ ./quiz3.sh
/home/student01
/home/student01
```

Command Substitution (Expansion)

Quiz 4

Set variable *entries* with the current directory total files number.

Quiz 4

Set variable *entries* with the current directory total files number.

quiz4.sh

```
1 #!/bin/bash
2 entries=$(ls | wc -l)
3 echo ${entries}
```

Quiz 4

Set variable *entries* with the current directory total files number.

quiz4.sh

```
1 #!/bin/bash
2 entries=$(ls | wc -l)
3 echo ${entries}
```

terminal

```
$ ls
first_script.sh         special_variables1.sh
special_variables2.sh   user_input_example.sh
$ ./quiz4.sh
4
```

Special Variables

Command line arguments

- Scripts can accept command line arguments just like any other Linux commands.
- Refer to them within the script as ${1}, ${2}, ...

first_script.sh

```
1 #!/bin/bash
2
3 echo "Hello ${1}!"
```

terminal

```
$ ./first_script.sh students
Hello students!
```

Special Variables

Other I

special_variables1.sh

```
1 #!/bin/bash
2 echo ${0} # The Bash script name
3 echo ${#} # Script arguments number
4 echo ${@} # All the arguments supplied to the Bash script
5 echo ${?} # The exit status of the most recently run process
6 echo ${$} # The process ID of the current script
```

terminal

```
$ ./special_variables1.sh arg1 arg2
./special_variables1.sh
2
arg1 arg2
0
19803
```

Other II

special_variables2.sh

```
1  #!/bin/bash
2  echo ${USER}      # The user running the script
3  echo ${HOSTNAME}  # The machi ne hostname
4  echo ${SECONDS}   # Seconds number since script started
5  echo ${RANDOM}    # Returns a random number
6  echo ${LINENO}    # Returns the current script line number
```

terminal

```
$ ./special_variables2.sh
student01
research-pc
0
16878
6
```

Processes

A computer program instance that is being executed

- A computer program:
  - A passive collection of instructions.
  - A process is the actual execution of those instructions.
  - Several processes may be associated with the same program.
- A child process:
  - A subprocess launched by another process, its parent.

Subshell

- A separate instance of the running shell process.
- Subshell variables are not visible outside the subshell.
- They are not accessible to the parent shell.

Subshell

- A separate instance of the running shell process.

subshell_example.sh

```
1  #!/bin/bash
2  echo "${PWD}"
3  (
4    cd /usr
5    echo "${PWD}"
6  )
7  echo "${PWD}" # Still in the original directory.
```

terminal

```
$ ./subshell_example.sh
/home/student01
/usr
/home/student01
```

Environment (Global) Variables

- Are created when starting a shell process.
- Defined new variables are inherited by child processes.
  - More on processes tomorrow.

terminal

```
$ printenv  # list environmental variables with values
XDG_VTNR=7
LC_PAPER=nl_NL.UTF-8
LC_ADDRESS=nl_NL.UTF-8
XDG_SESSION_ID=c2
...
HOME=/home/student01
...
$ echo ${HOME}
/home/student01
```

Exporting Variables

- Make them available in child processes.

print_city.sh

```
1 #!/bin/bash
2 echo "${city}"
```

terminal

```
$ city='Leiden'
$ ./print_city.sh

$ export city
$ ./print_city.sh
Leiden
```

Interact With User Input

By using the `read` command

user_input_example.sh

```
1  #!/bin/bash
2  greet='Hello'
3  # Ask user's name
4  echo 'What is your name?'
5  read name # variable assignment
6  # Say hello
7  echo "${greet} ${name}"
```

Interact With User Input

By using the `read` command

user_input_example.sh

```
1  #!/bin/bash
2  greet='Hello'
3  # Ask user's name
4  echo 'What is your name?'
5  read name # variable assignment
6  # Say hello
7  echo "${greet} ${name}"
```

terminal

```
$ ./user_input_example.sh
What is your name?
-
```

Interact With User Input

By using the `read` command

user_input_example.sh

```
1  #!/bin/bash
2  greet='Hello'
3  # Ask user's name
4  echo 'What is your name?'
5  read name # variable assignment
6  # Say hello
7  echo "${greet} ${name}"
```

terminal

```
$ ./user_input_example.sh
What is your name?
Gulliver_
```

Interact With User Input

By using the `read` command

user_input_example.sh

```
1  #!/bin/bash
2  greet='Hello'
3  # Ask user's name
4  echo 'What is your name?'
5  read name # variable assignment
6  # Say hello
7  echo "${greet} ${name}"
```

terminal

```
$ ./user_input_example.sh
What is your name?
Gulliver
Hello Gulliver!
```

Leiden University
Medical Center

Acknowledgements

Jeroen Laros
Magnus Palmblad
Jonathan Vis
Mihai Lefter
Yassene Mohammed