Unix Fundamentals – Bash Scripting

Marek Kozłowski
Faculty of Mathematics and Information Sciences
Warsaw University of Technology

Never use copy&paste for the following exercises. Retype all commands manually!
Don't just read those exercises and examples. Do them, please!

1. **Part I**

   (a) Download the file '`bash1.sh`' and make it executable:
   `$ chmod +x bash1.sh`

   (b) Open it with '`vim`' (preferred) or any editor of your choice. Read it. Its contents should be self-explanatory.

   (c) Run it with and without parameters:
   `$ ./bash1.sh`
   `$ ./bash1.sh one two three`
   As you remember if you are using the '`vim`' you may run the script as follows:
   `<Esc>:!./%`
   `<Esc>:!./% one two three`

   (d) Create your own script:
   `$ echo '#!/bin/bash' > myscript.sh`
   make it executable:
   `$ chmod +x myscript.sh`
   and open (with '`vim`').

   (e) Based on '`bash1.sh`' add some multi-line output (help message) with '`cat`'. Check if it works.
   Remember: no blanks may precede the closing tag!

   (f) Define the function '`help()`', which prints that help message and invoke it. Your script should look like:

```
#!/bin/bash

help() {
cat << TAG1
Something
    to
  be
      printed
TAG1
}

help
```

   Check if it works.

   (g) Check if you can define a function on the command line. Does `bash` help with auto-completion?
   `$ myfun() { echo "The 1st parameter is $1"; }`

`$ myf<Tab>`
`$ myfun someparam`
Assuming you didn't forget leading and trailing blanks and the semicolon (read the tutorial again!) the answer for both questions is 'yes'. Yes, `bash` built-ins are in fact functions.

2. **Part II**

   (a) Download the file '`bash2.sh`' and make it executable:
   `$ chmod +x bash2.sh`

   (b) Run it from the command line. Mind that at some point it will require closing all '`gvim`' instances:
   `$ ./bash2.sh`

   (c) Open it with '`vim`' (preferred) or any editor of your choice. Read it. Its contents should be self-explanatory. Some notes:

   - remember that – unlike the C '`break`' instruction – in `bash` double semicolons (`;;`) are obligatory after each '`case`' choice block!
   - take yet another look at the '`while`' loop: commands' exit status is commonly used as a logical condition; for silent mode ignore both: '`stdout`' and '`stderr`' – as you remember we do that as follows:
     `$ somecommand 1>/dev/null 2>&1`

   (d) '`while`' is quite often invoked together with '`sleep`'. A simple example:
   `$ while true; do date; sleep 1; done`
   Terminate this infinite loop with the '`SIGINT`' signal (`Ctrl-C`).

   (e) Extend your script (`myscript.sh`) so it accepts five options: '`-a`', '`-b`' (with a value), '`-c`' (with a value), '`-d`' and '`-h`' (help). An exemplary solution:

```
#!/bin/bash

help() {
cat << TAG1
Something
    to
  be
      printed
TAG1
}

while getopts "ab:c:dh" some_variable 2>/dev/null
do
  case ${some_variable} in
    a) echo "option \‘-a’ active" ;;
    b) echo "option \‘-b’ active; value: $OPTARG" ;;
    c) echo "option \‘-c’ active; value: $OPTARG" ;;
    d) echo "option \‘-d’ active" ;;
    h) help ;;
    ?) echo "unrecognized option"; exit 1 ;;
  esac
done
```

   Check if it works as expected. Experiment with grouping or specifying options in any order.

3. **Part III**

(a) Download the file '`bash3.sh`' and make it executable:
`$ chmod +x bash3.sh`

(b) Run it from the command line:
`$ ./bash3.sh`

(c) Open it with '`vim`' (preferred) or any editor of your choice. Read it **thoroughly**. It contains a lot of useful copy&paste examples you may need but focus on why and how the work.

(d) Read the man page for the '`test`' command. Then run '`man bash`' and seek for the '`CONDITIONAL EXPRESSIONS`' section.

(e) Do experiments on your own until you are quite sure you have some fluency in conditional command execution.

(f) Our script (`myscript.sh`) accepts options in any order (or even grouped) but it behaves a little bit different depending on the order of the options. Moreover we cannot define mutually exclusive options. Let's try to improve it a little bit:

```
#!/bin/bash

help() {
cat << TAG1
Something
    to
  be
      printed
TAG1
}

while getopts "ab:c:dh" some_variable 2>/dev/null
do
  case ${some_variable} in
    a) A=1 ;;
    b) B=1; B_VAL=$OPTARG ;;
    c) C=1; C_VAL=$OPTARG ;;
    d) D=1 ;;
    h) help; exit 0 ;;
    ?) echo "unrecognized option"; exit 1 ;;
  esac
done

[ ! -z $A ] && echo "option \‘-a’ active"
[ ! -z $B ] && echo "option \‘-b’ active", value: $B_VAL"
[ ! -z $C ] && echo "option \‘-c’ active", value: $C_VAL"
[ ! -z $D ] && echo "option \‘-d’ active"
```

The above script seems to be a perfect template for any bash scripts you'll write in the future!

4. **Part IV**

(a) Download the file '`bash4.sh`' and make it executable:
`$ chmod +x bash4.sh`

(b) Run it from the command line:
`$ ./bash4.sh`

(c) Open it with '`vim`' (preferred) or any editor of your choice. Read it. Feel free to modify it and do experiments on your own.

(d) Mind that `bash` supports only integer arithmetic. There are no decimal / floating point values.

(e) For the string chopping explanation more in detail refer to the IBM tutorial *Bash by example* (https://www.ibm.com/developerworks/library/l-bash/index.html), sections:

- *Chopping strings overview*
- *Chopping strings like a pro*

(f) Unfortunately most `bash` documentation explains this language *by example*. The same refers to the most comprehensive (and undoubtedly the longest!) *Advanced Bash-Scripting Guide* (http://tldp.org/LDP/abs/html/). Fortunately our 4 part short tutorial covers at least 99% of all bash code you can ever see in real `bash` scripts.