Umeå University
Department of Applied Physics and Electronics

Linux as Development Environment 7.5 ECTS
5EL142 HT-16

Assignment 10 - Bash Scripting

Submitted	2017-06-05
Author:	Lorenz Gerber, 20161202-2033 (`lorenzottogerber@gmail.com`)
Instructor:	Sven Rönnbäck, John Berge, Björne A Lindberg

1 Introduction

Short report with a reflection about the work. What went well what was difficult. Name some limitations of the chosen solution and/or propose some improvements to the provided scripts.

2 Timetrack

Timetracker was implemented using the `case` statement. The tracker information is stored in a file called time.log. The script handles the cases when there is either a tracker running or not for all three arguments. The commmand `date +%s` is used to obtain the seconds since 1970-01-01 00:00:00 UTC. Then `typeset -l` is used to convert and store the value as an integer variable for later arithemtics.

3 Checkuser

Checkuser is a script that checks if a specific user is logged into the system. The user name is given either as command line argument or is queried interactively. `read` is used to obtain the user input. `awk` is used to extract usernames of all logged in users from the output of command `who`. The whole process of who, awk and test is lined up with pipe commands.

4 Check5D

Check5D is a script that checks the `make` build script of exercise 5D. The script tests five different operations:

- `make all` executes without errors
- executables are created
- executable execute and return correct result
- `make install` works
- `make uninstall` works
- `make clean` works

5 Test Suite for electrotest Application

A test suite for the compiled 'electrotest' application was written as a bash script according to the specifications given. In brief, the compiled program is started as a background process and the input is fed with a 'here' document. The application output is caught in a tempfile. In test cases needed, the output of the program is verified by using grep on the tempfile. For the case that the prorgram does not terminate on the given input, a short wait time is built into each test case after which the PID of the background process is queried for existence. If it still exists, the test case is failed and the background process stopped. The normal evaluation of testcases is based either on the output of the program or on the exit code of the background process which is queried after the short wait time. Below follows a short pseudo code description of the test script:

```
checkUsage()
create tempFile
run electrotest as background process and divert output into tempfile
get PID from background process
sleep for a short time
if backprocess still runs
    outOfTime or doesNotCatchException
    kill background process
else
    wait PID
    store exit status
    TestCase() #e.g. grep
    check exit status
    remove tempFile
fi
```

5.1 Testing electrotest

My implementation of electrotest from exercise 6 does not catch negative resistance values. Neither does it throw an exception for out of range values. These cases could be corrected by simply verifying user input and exiting with an error code. One design strategy in my electrotest was found that makes testing more complicated: The user input for ‘sequential’ or ‘parallel’ user input verifies the input and infinitely jumps back to query the user until the given input is valid. I guess that this is a rather unwanted behaviour for headless/console applications. It would be better to just exit with a explanation to stderr to be more transparent.

6 Reflection

Since many years, I only use -nix operating systems and have therefore spent already quite some time on the command line and with writing small scripts. However, for some reason I never really got warm with Bash scripting and I was thinking already quite some times why this might be. In this exercise, I deliberately chose bash scripting for the second part instead of Python, mostly to figure out why bash seems so unintuitive to me. I think one reason could be that in most everyday situatinos, the only thing needed are oneliners which one can easily find online. Usually in some super ‘clever’ short form that require half an hour thinking to really decipher what they do... Now in this exercise, the tasks become more and more like ‘normal’ programming with common control flows and so on. I think this could have helped a bit for my bash scripting future. I seemd to get used a bit more to many concepts in scripting now. Maybe the problem is that if one does not do it often enough, it is hard to remember the rather awkward syntax and many quirky but powerful details in bash. This was for me one of the most giving exercises so far in this course.