# Practice M8: Bash Scripting and Automation

For the purpose of the current practice we will need one virtual machine. It could be with either **CentOS 8.3**, **openSUSE Leap 15.2**, or **Ubuntu 20.04**. Of course, another version of the listed or another distribution can be used, but there can be some differences.

All the commands, which we will used until the end of this practice, will be accompanied by the appropriate prompt. This way it will be easier for us to know which user, in which folder, and on which machine is executing the command.

Next steps will be executed on **CentOS** machine. If there is a big difference in the way we issue the command, or in its result between distributions, it will be stated clearly.

## Part 1

#### Scheduling

We can check if the **crond** service is operational and if not, start it with:

[lsauser@centos ~]$ **systemctl status crond**

...

[lsauser@centos ~]$ **sudo systemctl enable crond**

...

[lsauser@centos ~]$ **sudo systemctl start crond**

Please note that the service, could have different name amongst distributions. For example, just **cron**.

Let's create a schedule to run the **wall** command every minute:

[lsauser@centos ~]$ **crontab -e**

...

\* \* \* \* \* wall 'Hello world! I am running on schedule :)'

Now, we must save it with **:wq**. After a minute a message will appear. We can wait few more minutes in order to see some more messages. Then stop the scheduled task with:

[lsauser@centos ~]$ **crontab -e**

...

#\* \* \* \* \* wall 'Hello world! I am running on schedule :)'

Save it with **:wq**.

**Command Schedule #1**

It is time to create a schedule with more complex command. It will count the running processes every three minutes and store the result in a file:

[lsauser@centos ~]$ **crontab -e**

...

\*/3 \* \* \* \* ps ax | wc -l >> /tmp/running\_processes.log

When we are convinced that it is working as expected, we can stop the scheduled task as we did with the previous one.

**Command Schedule #2**

One more scheduled task – this time to archive the **/etc** folder every day at specific time:

[lsauser@centos ~]$ **crontab -e**

...

15 19 \* \* \* tar -czf /backup/etc-$(date +\%Y-\%m-%\d).tar.gz /etc > /tmp/tar.log 2>&1

When we are convinced that it is working as expected, we can stop the scheduled task as we did with the previous one.

## Part 2

**Simple Interactive Script**

Let's create a simple interactive script:

[lsauser@centos ~]$ **vi p2-1-user-input.sh**

#!/bin/bash

#

# Ask for user input

# p2\_1\_user\_input.sh

#

read -p 'Enter your name: ' USR\_NAME

read -p "Okay $USR\_NAME, what is your favourite color? " USR\_COLOR

echo 'So, '$USR\_NAME' You like '$USR\_COLOR

echo 'I like it too, but mine favorite color is Blue.'

Save the file and close the editor

Now, make the file executable

[lsauser@centos ~]$ **chmod +x p2-1-user-input.sh**

And start it

[lsauser@centos ~]$ **./p2-1-user-input.sh**

**Script with Parameters and a Loop**

In this exercise we will create a script that will accept command line parameters and will execute tasks in a loop:

[lsauser@centos ~]$ **vi p2-2-params.sh**

#!/bin/bash

#

# Create number of files with a spcific prefix

# p2\_2\_params.sh

#

if [ $# -ne 2 ]; then

echo 'Wrong execution!';

echo "Usage: $0 file\_prefix file\_num"

exit 1;

fi

for i in $( seq -w 1 $2 ); do

echo "File num: $i" >> $1$i.txt

done

Save the file and close the editor

Then make the file executable

[lsauser@centos ~]$ **chmod +x p2-2-params.sh**

Now execute the file and test it

[lsauser@centos ~]$ **./ p2-2-params.sh**

**Simple Script Game (Guess the Number)**

In this exercise we will create a simple game that is also known as Guess the number. Even though it is not directly related the regular duties of an administrator, it demonstrates common techniques, that can be seen in many scripts:

[lsauser@centos ~]$ **vi guess-params.sh**

#!/bin/bash

#

# guess.sh - number guessing game

#

# used system variables and routines:

# - RANDOM - generates random integer number

# between 0 and 32767

# - $0 - script name

# - $1 - argument no.1

# - $# - total number of arguments

#

# random number limit

maxno=100

# check how many agruments we have and if

# the requirements are not met a message is

# displayed

if [ $# -ne 1 ]; then

echo "Usage: $0 no\_tries"

exit 1

fi

# calculate the number to be guessed

# it will be something between 0 and $maxno-1

guessnum=$((RANDOM%$maxno))

# uncomment the following line to see

# the generated number

# echo $guessnum

# guess counter - it tracks user attempts

guesscnt=1

# repeats the loop until the number is guessed or

# the stated number of attempts is reached - whichever

# happens first

while [ $guesscnt -le $1 ]; do

echo -n "Your guess attempt no.$guesscnt is: "

read guess

# check if the user guessed the number

if [ $guess -eq $guessnum ]; then

echo "Congratulations! You guessed it!"

exit 0

fi

# checks if the given number is smaller than

# the one to be guessed and prints a message

if [ $guess -lt $guessnum ]; then

echo "No. Try a bigger number."

fi

# checks if the given number is bigger than

# the one to be guessed and prints a message

if [ $guess -gt $guessnum ]; then

echo "No. Try a smaller number."

fi

# increase the guess attempts counter

guesscnt=$((guesscnt+1))

done

# if the maximum number of requested attemts was

# reached and the user did not guess the number then

# the game is over

if [ $guesscnt -gt $1 ]; then

echo "GAME OVER. My number was: $guessnum"

fi

exit 0

## Part 3

##### Exam Simulation

##### Test

Test exam simulation with 30 questions similar to those that will be on the exam.

Check the **M8-Exam-Simulation.pptx** file.

##### Practice

Execute set of tasks similar to those that will be on the exam.

Check the **M8-Exam-Simulation.docx** file.

***Part 4 (Additional) \****

**Bash Showcase**

In order to check when and how a sub-shell is created, we will write a small script named **bash-showcase.sh** that will help us see what is going on:

[lsauser@centos ~]$ **vi bash-showcase.sh**

#!/bin/bash

#

# bash\_showcase.sh

#

# Shows the invocation/execution mode and the process tree:

# - OWN - in its own shell

# - SOURCED - as part of the invoking shell

# The key wait is added for cases in which the script is executed with "exec ./bash\_showcase.sh"

#

[ "$0" = "$BASH\_SOURCE" ] && EXECWAY=OWN || EXECWAY=SOURCED;

echo 'Execution mode: '$EXECWAY

ps -o pid -o ppid -o cmd --forest

read -n 1 -s -p "Press any key to continue"

echo

Save and close the file. Then make it executable. Finally, test it with:

[lsauser@centos ~]$ **bash bash-showcase.sh**

...

[lsauser@centos ~]$ **./bash-showcase.sh**

...

[lsauser@centos ~]$ **source bash-showcase.sh**

...

[lsauser@centos ~]$ **exec bash-showcase.sh**

We should see the differences in the script behavior based on the way it was executed.

**Environment Configuration Files Order**

We all know that there is a special order when executing the environment configuration scripts. There is an easy way to force the system to show us what is going on. This way it will be easier for us get the details.

Let's execute the following set of commands:

[lsauser@centos ~]$ **echo 'echo "ENDED: /etc/bashrc"' | sudo tee -a /etc/bashrc**

[lsauser@centos ~]$ **echo 'echo "ENDED: /etc/profile"' | sudo tee -a /etc/profile**

[lsauser@centos ~]$ **echo 'echo "ENDED: .bashrc"' >> .bashrc**

[lsauser@centos ~]$ **echo 'echo "ENDED: .bash\_profile"' >> .bash\_profile**

[lsauser@centos ~]$ **echo 'echo "ENDED: .bash\_logout"' >> .bash\_logout**

[lsauser@centos ~]$ **echo 'read -p "Press any key ..."' >> .bash\_logout**

Now, we can execute one full logon-logoff cycle. First, we must logoff from our current session. As we can see there are some additional messages on the screen. We will see more during the cycle.

**Global vs Local Variables**

Let's check the scope differences of the global (for the environment) and the local (in a script) variables. Again, we will create a small script:

[lsauser@centos ~]$ **vi global-vs-local.sh**

#!/bin/bash

#

# global\_vs\_local.sh

#

# Global vs Local variables

#

# if there is no local definition, the global one will be used

echo '# this is the value of MYVAR before local initialization'

echo 'MYVAR =' $MYVAR

# now we have local variable with the same name as existing global variable

MYVAR=local

# this statement will print the local variable

echo '# this is the value of MYVAR after local inititalization'

echo 'MYVAR =' $MYVAR

# we unset the local variable

unset MYVAR

# but this will not make the script use the global one instead

echo '# this is the value of MYVAR after unsetting it'

echo 'MYVAR =' $MYVAR

Save and close the file. Then make it executable. Finally, execute it:

[lsauser@centos ~]$ **export MYVAR=global**

[lsauser@centos ~]$ **./global\_vs\_local.sh**