# Practice M7: Bash Scripting and Automation

\* NOTE: Some of the exercises in this practice guide are unsuitable for execution in WSL or Docker environments. A virtual infrastructure should be used instead.

For the purpose of this practice, we will need at least one virtual machine with **AlmaLinux OS 9.x** (or **CentOS Stream 9**, or **Oracle Linux 9.x**, or **Rocky Linux 9.x**), **openSUSE Leap 15.x,** or **Debian 12.x** (or **Ubuntu Server 22.04/24.04**) installed. Of course, another version of the listed or another distribution can be used, but there can be some differences.

All commands that we are going to use in this practice will be accompanied by the appropriate prompt. This way, it will be easier for us to understand which user, in which folder, and on which machine is executing the command.

The next steps will be executed on an **AlmaLinux** 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.

Please note that we start each practice with a fresh machine or set of machines.

Please note that the output from the commands shown below may differ compared to the one you get.

## Part 1

### Scheduling with at

The **at** package may not be installed by default. This should not be an issue for us, as we should know by now how to add a missing package

Under **Red Hat**-based distributions this is done with:

[lsauser@almalinux ~]$ **sudo dnf install at**

Next, we must check if the **atd** service is up and running and if not, to change its status

[lsauser@almalinux ~]$ **systemctl status atd**

**...**

[lsauser@almalinux ~]$ **sudo systemctl enable --now atd**

Then we can check the current time

[lsauser@almalinux ~]$ **date**

And then schedule a job for **five** minutes after the current time (for example 18:48)

[lsauser@almalinux ~]$ **at 18:48**

Then enter the job steps (just one in our case):

at> **echo $(date) >> /tmp/at1.log**

We can have more than one command in a job, so let’s add one more

at> **echo "Done." >> /tmp/at1.log**

And then press **Ctrl+D** to send **EOF** and submit the job

Now, we can check the queue with

[lsauser@almalinux ~]$ **atq**

What if we want to use relative time instead of exact moment in the future? How can we do it?

Let’s create another job that will start in **two minutes** from now

[lsauser@almalinux ~]$ **at now + 2 minutes**

Then enter the job steps (just one in our case):

at> **echo $(date) >> /tmp/at2.log**

at> **echo "Done as well." >> /tmp/at2.log**

And then press **Ctrl+D** to send **EOT** and submit the job

Now, we check the queue

[lsauser@almalinux ~]$ **atq**

If we are quick enough, we will see the two jobs

Let’s create one more. It will archive and compress the **/etc** folder

[lsauser@almalinux ~]$ **at now + 30 minutes**

Then enter the job steps (just one in our case):

at> **tar czvf /tmp/etc.tar.gz /etc 2> /tmp/at3.log**

And then press **Ctrl+D** to send **EOT** and submit the job

Now, we check the queue

[lsauser@almalinux ~]$ **atq**

Perhaps, we will see just the last job

No matter if see all three or just the last one, let’s see how we can remove a job

To do so, we must execute a command like this

[lsauser@almalinux ~]$ **atrm 3**

Where **3** is the number or ID of the job

Then check again the queue

[lsauser@almalinux ~]$ **atq**

By now the first two jobs should have executed

We can check their output with

[lsauser@almalinux ~]$ **cat /tmp/at1.log**

**...**

[lsauser@almalinux ~]$ **cat /tmp/at2.log**

### Scheduling with cron

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

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

...

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

...

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

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

#### Simple Schedule

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

Open our crontab for editing

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

And add the following line at the end

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

Now, we must save and close it with **:wq**

After a minute a message will appear

We can wait a few more minutes in order to see some more messages

To check the list of installed cron jobs, we can execute

[lsauser@almalinux ~]$ **crontab -l**

Then stop the scheduled task by commenting the job

Open our crontab for editing

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

And make sure that the job is commented

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

Save and close it with **:wq**

#### Command Schedule #1

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

Open our crontab for editing

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

And add the following line at the end

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

Save and close it with **:wq**

Wait a while and check the results

Then stop the scheduled task as we did with the previous one

But wait, where are our cron jobs?

We can check here

[lsauser@almalinux ~]$ **sudo ls -al /var/spool/cron/**

And then get the contents of the file for our user (for example, lsauser)

[lsauser@almalinux ~]$ **sudo cat /var/spool/cron/lsauser**

It is not a good practice to manipulate this file directly

#### Command Schedule #2

One more scheduled task – this time to archive the **/etc** folder every day at specific time (for example, at 19:15)

Open our crontab for editing

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

And add the following line at the end

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

Save and close it with **:wq**

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

### Scheduling with timers

Let’s first see what timers are available in our system with

[lsauser@almalinux ~]$ **systemctl list-timers --all**

Let’s pick one of them. For example, the **dnf-makecache.timer** one on **Red Hat**-basedsystems

Ask for its status just as we do with service units

[lsauser@almalinux ~]$ **systemctl status dnf-makecache.timer**

We can ask for even more detailed information with

[lsauser@almalinux ~]$ **systemctl cat dnf-makecache.timer**

Okay, let’s create our own

Transient timers are a good entry point to start with the systemd timers

#### Transient #1

We can create a timer that will execute 1 minute after its creation and will create a simple text file

To do so, we must execute the following

[lsauser@almalinux ~]$ **sudo systemd-run --on-active=60 echo "Executed at $(date)" > /tmp/timer1.log**

We can see that two new units got created – a service and timer

Note the name of the service unit

It should appear in the list of timers

[lsauser@almalinux ~]$ **systemctl list-timers --all**

After a while it will execute

If we check the resulting file

[lsauser@almalinux ~]$ **cat /tmp/timer1.log**

We won’t see anything

It appears that we cannot use redirection here the way we used to do it

If we repeat the timer creation and check the content of the service unit, we will see why 😉

We could check the journal to see if the service got executed

[lsauser@almalinux ~]$ **sudo journalctl -u <name-of-the-service>.service**

We can repeat the above command but change **.service** to **.timer**

Indeed, our task was executed

#### Transient #2

Now, let’s create a simple service unit

[lsauser@almalinux ~]$ **sudo vi /etc/systemd/system/free-mem.service**

With the following content

**[Unit]**

**Description=Logs system free memory**

**Wants=free-mem.timer**

**[Service]**

**Type=oneshot**

**ExecStart=/usr/bin/free -h**

**[Install]**

**WantedBy=multi-user.target**

Save and close the file

Now, start it with a transient timer

[lsauser@almalinux ~]$ **sudo systemd-run --on-active=60 --unit free-mem.service**

And check if it appears in the list of timers

[lsauser@almalinux ~]$ **systemctl list-timers --all**

After 60 seconds, check the journal for the output of the service execution

[lsauser@almalinux ~]$ **sudo journalctl -u free-mem.service**

Yay, not only did our timer execute successfully but we also created a service that worked 😊

We can check the status of our service the way we do with others

[lsauser@almalinux ~]$ **systemctl status free-mem.service**

And here, we can see the execution results as well

#### Calendar

We will reuse our service unit from the previous section

This time, we will create a timer that will be executed on schedule (every two minutes)

First, let’s explore a few expressions

Check the weekly pattern

[lsauser@almalinux ~]$ **systemd-analyze calendar weekly**

Next, check if we want something to execute every Wednesday at noon

[lsauser@almalinux ~]$ **systemd-analyze calendar "Wed \*-\*-\* 12:00:00"**

What about noon on Monday and Tuesday but only between 01 and 04 every month?

[lsauser@almalinux ~]$ **systemd-analyze calendar "Mon,Tue \*-\*-01..04 12:00:00"**

Our aim here is much simpler

We would want our timer to trigger every two minutes

[lsauser@almalinux ~]$ **systemd-analyze calendar "\*-\*-\* \*:0/2:00"**

We can shorten it to

[lsauser@almalinux ~]$ **systemd-analyze calendar "\*:0/2"**

We may even ask for the first 7 iterations

[lsauser@almalinux ~]$ **systemd-analyze calendar --iterations=7 "\*:0/2"**

*Please note that the* ***--iterations*** *option as some sub-commands (like* ***timespan****) may not work on your system*

Let’s create a timer unit the same way that we created the service unit

Open a file for editing

[lsauser@almalinux ~]$ **sudo vi /etc/systemd/system/free-mem.timer**

Enter the following content

**[Unit]**

**Description=Runs a service every two minutes**

**[Timer]**

**OnCalendar=\*:0/2**

**Persistent=true**

**[Install]**

**WantedBy=timers.target**

Save and close the file

Check if the timer appears amongst the others

[lsauser@almalinux ~]$ **systemctl list-timers --all**

If it doesn’t appear, we must enable and start it

[lsauser@almalinux ~]$ **sudo systemctl enable --now free-mem.timer**

Now, check again

[lsauser@almalinux ~]$ **systemctl list-timers --all**

It should be there. We can see when it is expected to be executed

Wait a while, and then check the journal

[lsauser@almalinux ~]$ **sudo journalctl -u free-mem.service**

Okay, it is working 😊

Should we want, we can stop it and disable it with

[lsauser@almalinux ~]$ **sudo systemctl disable --now free-mem.timer**

## Part 2

Let’s start with the building blocks

### Simple Scripts #1

We may think of scripts as just text files with some sequence of commands there

#### Script #1

Create a simple text file **~/s1.sh** with the following content

**#!/bin/bash**

**# Prints Hello World and exits**

**echo "Hello World!"**

Save it and close it

Make it executable with

[lsauser@almalinux ~]$ **chmod +x s1.sh**

Execute it

[lsauser@almalinux ~]$ **./s1.sh**

We **cannot** execute it with just

[lsauser@almalinux ~]$ **s1.sh**

As it (the file) is not part of our **PATH** environment variable

One of the most important parts of our script is the signature (**#!/bin/bash**)

It is used by the shell to determine how to handle or execute the file (in our case to use the **bash** command)

Of course, we can omit it (not a good idea) but then we should execute the script with

[lsauser@almalinux ~]$ **bash s1.sh**

The above can be used even if there is a signature, but we usually use the shorter way of execution

#### Script #2

Now, let’s create another one **~/s2.sh** with the following content

**#!/bin/bash**

**# Stores current date and time in a file**

**filename=/tmp/date-time.txt**

**echo "Now is $(date)" > $filename**

Save it and close it

Make it executable with

[lsauser@almalinux ~]$ **chmod +x s2.sh**

Execute it

[lsauser@almalinux ~]$ **./s2.sh**

Now, check the result

[lsauser@almalinux ~]$ **cat /tmp/date-time.txt**

Still not impressed? Let’s try something else 😊

#### Script #3

When creating users in the classical way, we execute at least two commands – **useradd** and **passwd**

Imagine, that there are also others. For example, writing a custom readme.txt file in its home folder (yes, we can use the **/etc/skel** folder but we do not want to 😉 )

So, we can create an oversimplified script (**~/s3.sh**) and use it every time, we want to create a new user

**#!/bin/bash**

**# Creates a user**

**username=demo**

**echo "\* Create user $username"**

**useradd -m -s /bin/bash $username**

**echo "\* Set password for the $username user"**

**passwd $username**

**echo "\* Create a readme.txt file"**

**echo "Welcome to the club, $username" > /home/$username/readme.txt**

**echo "\* Done."**

Save it and close it

Make it executable with

[lsauser@almalinux ~]$ **chmod +x s3.sh**

Execute it

[lsauser@almalinux ~]$ **./s3.sh**

Hm, it doesn’t work. Of course, it won’t. We must execute it with **sudo**

[lsauser@almalinux ~]$ **sudo ./s3.sh**

Now, it works 😊

We are getting better and better. Let’s check the result

[lsauser@almalinux ~]$ **tail /etc/passwd**

**...**

**demo:x:1001:1001::/home/demo:/bin/bash**

[lsauser@almalinux ~]$ **sudo cat /home/demo/readme.txt**

This is another story 😊

Of course, having to edit the script every time when we need to create another user is not a good idea

Later, we will see a few ways to parametrize it

### Sourcing vs Execution

There are two ways which we can use to execute what is inside a script

Sourcing executes the instructions of a script as part of the current session, while execution starts a child session in which the script is executed

The main difference is that with sourcing all changes (new variables, changed variables, etc.) to the environment persist

Let’s see it in action with the following (**~/s4.sh**) script

**#!/bin/bash**

**# Sourcing vs Execution**

**MYVAR=Hello**

**echo "MYVAR=$MYVAR"**

**echo "Executed on $(date)" | tee /tmp/s4.txt**

Save it and close it

Make it executable with

[lsauser@almalinux ~]$ **chmod +x s4.sh**

Before executing it, let’s check if there is the MYVAR variable in our session

[lsauser@almalinux ~]$ **echo $MYVAR**

No, there isn’t. Now, execute it

[lsauser@almalinux ~]$ **./s4.sh**

And check the result (besides what we saw on the screen)

[lsauser@almalinux ~]$ **echo $MYVAR**

[lsauser@almalinux ~]$ **cat /tmp/s4.txt**

Hm, the variable is empty, but the file has content

Now, let’s source it instead of executing it

This can be done either

[lsauser@almalinux ~]$ **source s4.sh**

Or

[lsauser@almalinux ~]$ **. ./s4.sh**

*Execute just one of the above*

And check the result (besides what we saw on the screen)

[lsauser@almalinux ~]$ **echo $MYVAR**

[lsauser@almalinux ~]$ **cat /tmp/s4.txt**

Now the variable persists, just as promised 😉

### Loops #1

Let’s try the loop commands in bash

#### seq

But first, let’s check the **seq** command

Should we want to create a range of numbers, for example between 1 and 10, we can execute

[lsauser@almalinux ~]$ **seq 1 10**

We may use a custom step, for example 2

[lsauser@almalinux ~]$ **seq 1 2 10**

We can go the opposite – from a big number down to a smaller one

[lsauser@almalinux ~]$ **seq 10 -2 1**

A leading zero can be added with

[lsauser@almalinux ~]$ **seq -w 1 2 10**

We may use a custom format especially for floating point numbers, for example

[lsauser@almalinux ~]$ **seq -f '%.2f' 1 0.35 5**

Or create a list of numbers separated by comma

[lsauser@almalinux ~]$ **seq -s ',' 1 5**

Now, let’s use the above together with **for**

#### for #1 (items iteration)

The standard usage of for is to iterate over a list of items

It can execute one or more commands for each member of the list

We can start with this example

[lsauser@almalinux ~]$ **for i in a b c; do echo $i; done**

We can change the echo to something else. Or, we can add another command there

[lsauser@almalinux ~]$ **for i in a b c; do echo $i; touch file-$i.txt; done**

The above will create a file for each item

Check the result

[lsauser@almalinux ~]$ **ls -l file\***

The list of items can come from somewhere else

It may come from file or from the output of another command

Let’s try the second option first

[lsauser@almalinux ~]$ **for i in $(seq 1 5); do echo "item: $i"; done**

Of course, we can use another technique that we already know, to achieve the same

What about this?

[lsauser@almalinux ~]$ **for i in {1..5}; do echo "item: $i"; done**

Now, let’s use the output of a “regular” command like **ls**

[lsauser@almalinux ~]$ **for i in $(ls -l); do echo "item: $i"; done**

Wow, what a mess. Let’s change it to (change the small letter L to the number 1)

[lsauser@almalinux ~]$ **for i in $(ls -1); do echo "item: $i"; done**

It works 😊

And what about the content of a file? Let’s test this as well

Create **~/for.txt** file with the following content

**sofia**

**varna**

**plovdiv**

**ruse**

Save it and close it

Now, use it with for (in fact, we already know all we need)

[lsauser@almalinux ~]$ **for i in $(cat ~/for.txt); do echo "item: $i"; done**

We can extend it to something like this

[lsauser@almalinux ~]$ **for i in $(cat ~/for.txt | sort); do echo "item: $i"; done**

What if we want the list sorted in reverse? 😉

#### for #2 (c-style)

Sometimes, we may see and/or use another form of the **for** command

It is more like what we used to use with different programing languages

Try this

[lsauser@almalinux ~]$ **for ((i=0;i<=5;i++)); do echo "item: $i"; done**

It looks familiar

#### Nested for

We may even nest two (or more) **for** blocks

Let’s test the following

[lsauser@almalinux ~]$ **for i in {1..5}; do for j in {1..5}; do echo "i=$i,j=$j"; done; done**

It will work with the c-style version as well

[lsauser@almalinux ~]$ **for ((i=1;i<=5;i++)); do for ((j=1;j<=5;j++)); do echo "i=$i,j=$j"; done; done**

## Part 3

### Loops #2

There are two more ways to create a loop – **while** and **until**

#### while

While executes command or set of commands as long as a condition is true

We can test it by executing the following

[lsauser@almalinux ~]$ **count=1**

[lsauser@almalinux ~]$ **while [ $count -le 5 ]; do echo "item: $count"; sleep 5; done**

Wait, this doesn’t work the way we wanted …

Why? How can we stop it?

Press **Ctrl+C** to stop it

The reason that it is not behaving like we expect is that there isn’t anything inside the loop that changes the value of the index (**$count**) variable. In fact, we created an infinite loop

Now, change the loop body to

[lsauser@almalinux ~]$ **while [ $count -le 5 ]; do echo "item: $count"; sleep 5; count=$((count+1)); done**

This is another story 😉

Of course, we can place many other commands inside the loop, but we intentionally keep the things simple

#### until

This one behaves exactly the opposite compared to the **while** loop

It executes its body until the condition is not met

Let’s test it with this

[lsauser@almalinux ~]$ **count=1**

[lsauser@almalinux ~]$ **until [ $count -gt 5 ]; do echo "item: $count"; sleep 5; count=$((count+1)); done**

We should be careful here as well. We **must** have something to change the value of the index variable

### Simple Scripts #2

#### **Interactive Script #1**

Let's create a simple interactive script:

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

#!/bin/bash

#

# Ask for user input

# user-input-1.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@almalinux ~]$ **chmod +x user-input-1.sh**

And start it

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

#### **Interactive Script #2**

Let's create another simple interactive script:

[lsauser@almalinux ~]$ **vi user-input-2.sh**

#!/bin/bash

#

# Ask for user input

# user-input-2.sh

#

mynum=$((RANDOM%100))

read -p 'Enter a number between 0 and 100: ' yournum

echo 'So, your number is '$yournum' and mine is '$mynum

if [ $mynum -gt $yournum ]; then

echo 'Mine is bigger.'

elif [ $mynum -lt $yournum ]; then

echo 'Mine is smaller.'

else

echo 'They are equal.'

fi

Save the file and close the editor

Now, make the file executable

[lsauser@almalinux ~]$ **chmod +x user-input-2.sh**

And start it

[lsauser@almalinux ~]$ **./user-input-2.sh**

#### Interactive Script with Loop

Let’s repeat one and the same action, until the user decides to stop us by answering a question in a certain way

Create the following script

[lsauser@almalinux ~]$ **vi user-input-3.sh**

#!/bin/bash

#

# Ask for user input and loops

# user-input-3.sh

#

read -p 'Do you want to see a random number? (y/n) ' answer

while [ $answer != 'n' ]; do

echo "Here is one random number – $((RANDOM%100))"

read -p 'Do you want to see a random number? (y/n) ' answer

done

Save the file and close the editor

Now, make the file executable

[lsauser@almalinux ~]$ **chmod +x user-input-3.sh**

And start it

[lsauser@almalinux ~]$ **./user-input-3.sh**

We can remove the redundant question with a solution like this (not the only one, nor the optimal one):

[lsauser@almalinux ~]$ **vi user-input-4.sh**

#!/bin/bash

#

# Ask for user input and loops

# user-input-4.sh

#

while true; do

read -p 'Do you want to see a random number? (y/n) ' answer

if [ $answer != 'n' ]; then

echo "Here is one random number – $((RANDOM%100))";

else

break;

fi

done

Here we used the **break** command to escape from a loop once a condition is met

Save the file and close the editor

Now, make the file executable

[lsauser@almalinux ~]$ **chmod +x user-input-4.sh**

And start it

[lsauser@almalinux ~]$ **./user-input-4.sh**

Let’s do one more iteration

[lsauser@almalinux ~]$ **vi user-input-5.sh**

#!/bin/bash

#

# Ask for user input and loops

# user-input-5.sh

#

answer='x'

while [ $answer != 'n' ]; do

read -p 'Do you want to see a random number? (y/n) ' answer

if [ $answer = 'y' ]; then

echo "Here is one random number – $((RANDOM%100))";

fi

done

Save the file and close the editor

Now, make the file executable

[lsauser@almalinux ~]$ **chmod +x user-input-5.sh**

And start it

[lsauser@almalinux ~]$ **./user-input-5.sh**

So many ways to solve such a simple task and we haven’t seen all possible solutions 😊

#### 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@almalinux ~]$ **vi params.sh**

#!/bin/bash

#

# Create number of files with a spcific prefix

# 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

exit 0

Save the file and close the editor

Then make the file executable

[lsauser@almalinux ~]$ **chmod +x params.sh**

Now execute the file and test it

[lsauser@almalinux ~]$ **./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 to the regular duties of an administrator, it demonstrates common techniques that can be seen in many scripts:

[lsauser@almalinux ~]$ **vi guess.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

Save the file and close the editor

Then make the file executable

[lsauser@almalinux ~]$ **chmod +x guess.sh**

Now execute the file and test it

[lsauser@almalinux ~]$ **./guess.sh 3**

## 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@almalinux ~]$ **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@almalinux ~]$ **bash bash-showcase.sh**

...

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

...

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

...

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

We should see the differences in the script’s 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 to get the details.

Let's execute the following set of commands:

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

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

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

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

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

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

Now, we can execute one full logon-logoff cycle. First, we must log off 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@almalinux ~]$ **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@almalinux ~]$ **export MYVAR=global**

[lsauser@almalinux ~]$ **./global-vs-local.sh**