

BASH Mastery

How to Build A Bash Script?

What is a Script?

- A shell script is a file containing COMMANDS FOR THE SHELL.
- Allows automation.
- This basically saves time 😊
- Stores sets of commands.
- Basically just commands : Plus we can practice the command line as well.

Bash Script Structure - Part 1- Core Components

- There are 3 core components of the Bash Script.
- How to RUN SCRIPT on your PC.
- Explain core components.

Scripts always begin with the SHABANG LINE.

Now why do we use this because it tells the interpreter what we need to do to interpret the file.

→ We basically tell the script that hey, this is the what language the script will be written in. 😊

```
#!/bin/bash
```

Lets just break that down.

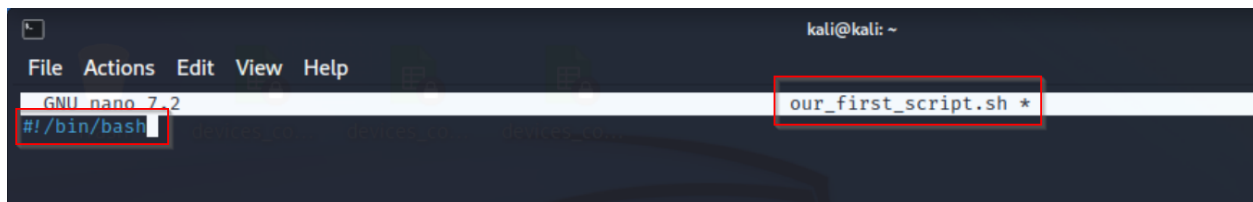
<code>#!</code>	We are telling the file that hey, this is where the file begins.
<code>/bin/bash</code>	This is the path to the interpreter.

Now am going to be creating our 1st script but here we will start with Shabang.

Type in the following:

Notice how at the beginning of the line there is NO SPACE.

The SHABANG starts at the beginning.

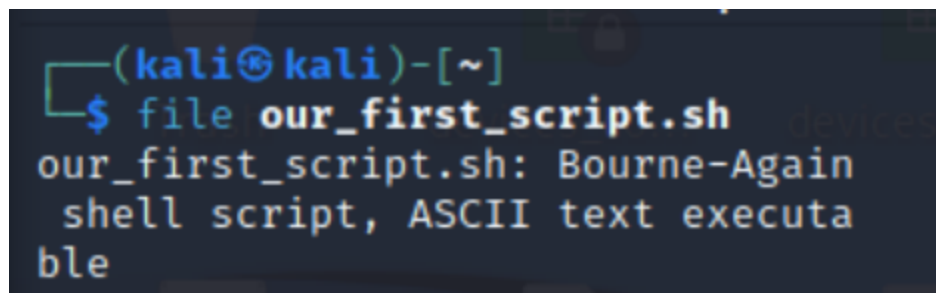


```

kali@kali: ~
File Actions Edit View Help
GNU nano 7.2 our_first_script.sh *
#!/bin/bash

```

Lets use the “file” command to view what type of file do we have.



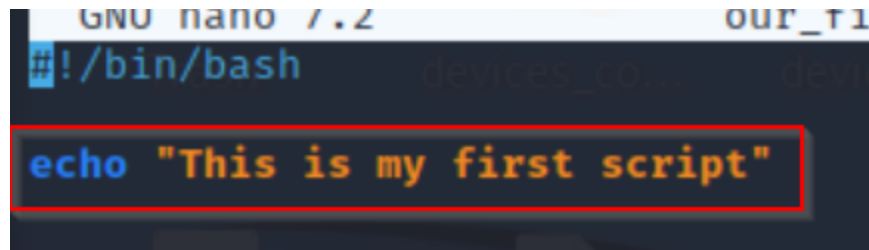
```

(kali㉿kali)-[~]
$ file our_first_script.sh
our_first_script.sh: Bourne-Again
shell script, ASCII text executable

```

→ We can see that BASH now identifies this as bash command.

Lets edit our script. For formatting reasons normally leave a blank line and this is just for formatting reasons.

A screenshot of a terminal window. At the top, it shows 'GNU nano 2.9.2' and 'OUR_T1'. Below that, the prompt is '#!/bin/bash'. The command 'echo "This is my first script"' is entered and highlighted with a red rectangular box. The text is in a monospaced font with orange and blue colors.

Final core is the exit statement.

- Lets us know if the script has run successfully or not.
- Tells us that the shell has been exited.
- Exit codes can range from 0 - 255.
- Exit code 0 means success and other than that other codes means an error.
- Some exit codes have special means.
- Here is link to show what the different exit codes mean
- <https://tldp.org/LDP/abs/html/exitcodes.html>

Now when you create a script and write an exit code.

We can put an exit to indicate the following "exit 0"

Now we need to focus on do and the following.... we need to give our script executable bit.

Because as this point,we cannot run the script.

But this is weird, because well in Kali Linux it said that it was as shell script but in my MAC its telling me this is ACCII Text, but this is OKAY.

→ Because we are going to set an executable bit to our script.

```

Notion-2.1.12.dmg      this_is_our_1st_Shell_Script.sh
[saintrose@Josh-MBP Desktop % file this_is_our_1st_Shell_Script.sh
this_is_our_1st_Shell_Script.sh: Bourne-Again shell script text executable, ASCII text
saintrose@Josh-MBP Desktop % $

```

Lets hang tight and take a look : :

Now its important to take note of the permission that we give our files, and why do this matter because this will determine the security of our system.

But for now we will run this command “chmod +x <file_name>”

Lets break down the command

chmod	Command used to create
+x	setting an executable bit on the file

```

[saintrose@Josh-MBP Desktop % file this_is_our_1st_Shell_Script.sh
this_is_our_1st_Shell_Script.sh: Bourne-Again shell script text executable, ASCII text
[saintrose@Josh-MBP Desktop % $chmod +x this_is_our_1st_Shell_Script.sh

```

Now lets take a look at this :::::

1) Now we have created the permissions.

```

saintrose@Josh-MBP Desktop % chmod +x this_is_our_1st_Shell_Script.sh
saintrose@Josh-MBP Desktop % file this_is_our_1st_Shell_Script.sh
this_is_our_1st_Shell_Script.sh: Bourne-Again shell script text executable, ASCII text

```

We can see that typing ls —color, acts as a means the shell now recognises this as a script.

```
saintrose@Josh-MBP Desktop % ls --color
Notion-2.1.12.dmg          this_is_our_1st_Shell_Script.sh
saintrose@Josh-MBP Desktop %
```

So when we run a script we normally type in the following ::

1) “./” :: This will normally indicate or show that we are running our script.

Now the ./ means that hey , please can your run this in our directory.

and there we go.

whoooooooooooo. Now we are all good.

```
saintrose@Josh-MBP Desktop % ./this_is_our_1st_Shell_Script.sh
This is out 1st shell script
saintrose@Josh-MBP Desktop %
```

Structure of Bash Script - Part 2 : Professional Components.

- Learn about adding comments to our code.
- How to add extra information to make our code look more professional.
- Create inline comments to make your code look more professional.
- Add information to ensure that we can take ownership of our code.
- Add date information to let people know when our script was 1st created.
- Add DESCRIPTION TO YOUR SCRIPTS.
- Adding usage information to other people to know how to use your script.

What is a comment?

This allows us to add “Human readable comments” in our code 😊

→ We basically do not want these to be read by the shell.

Now how to indicate a comment.

We use the “#” symbol.

Now when you start writing your comment, put a space in between the `#!/bin/bash`.

→ Take notice of the space after the `/bin/bash`

→ And after the comment.

UW PICO 5.09

File: thi

```
#!/bin/bash

#This prints text to the screen.
echo "This is out 1st shell script"

exit 0
```

Now let us run the script.

```
THIS IS OUR 1ST SHELL SCRIPT
saintrose@Josh-MBP Desktop % ./this_is_our_1st_Shell_Script.sh
This is out 1st shell script
saintrose@Josh-MBP Desktop %
```

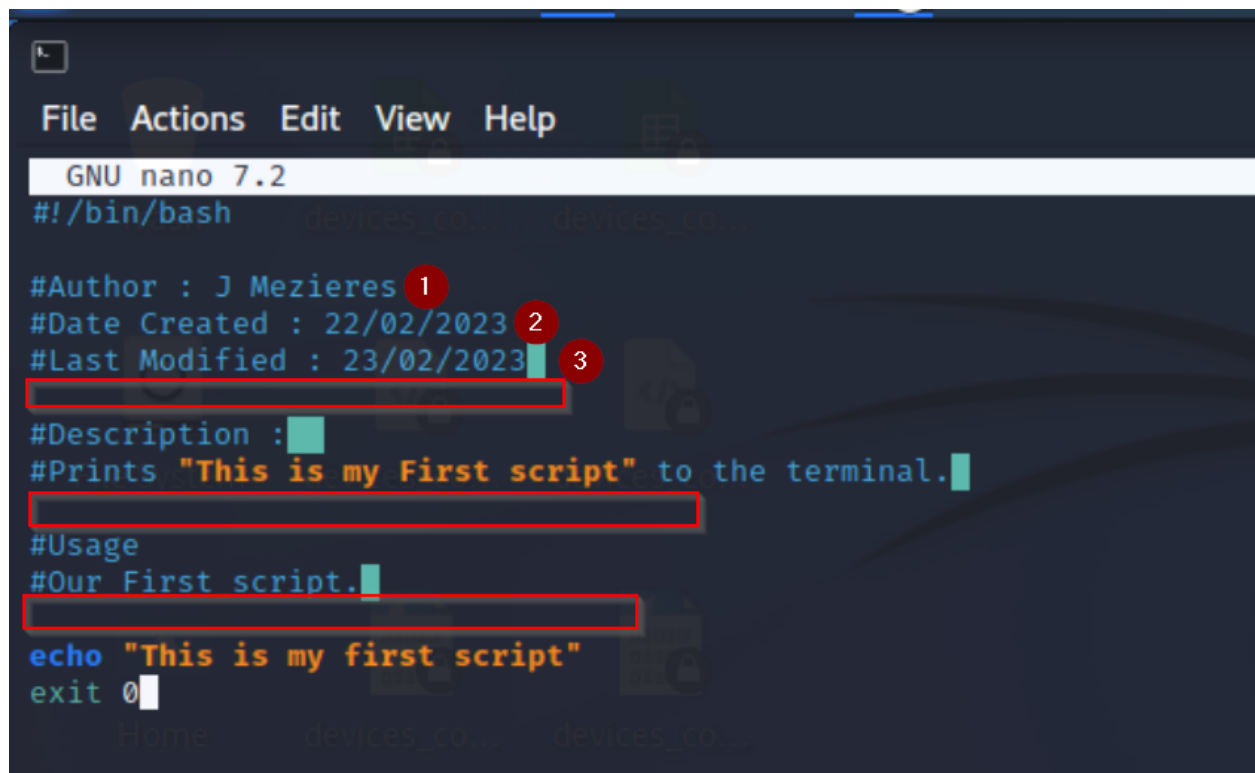
Notice how OUR COMMENTS are not showing.

To make your CODE MADE LOOK PROFESSIONAL :

- 1) Author
- 2) Date of Creation.
- 3) Date the script was last modified.
- 4) Script description.
- 5) Script usage. - Command line arguments

Lets MODIFY our script to a setting that mimics that one on the screen

Now take note, its normally a good idea to leave a BLANK LINE below and above the description section.



```
GNU nano 7.2
#!/bin/bash

#Author : J Mezieres 1
#Date Created : 22/02/2023 2
#Last Modified : 23/02/2023 3
#Description :
#Prints "This is my First script" to the terminal.
#Usage
#Our First script.
echo "This is my first script"
exit 0
```

Now here we need to take note of how we have structured the beginning the of the script.

6) Setting up Secure Script Permissions.

- Basics of the file permissions.
- Setting permission's on our script correctly.
- Identify and understand script permissions. (Need to know what they mean)
- Keep your scripts secure with the right FILE PERMISSIONS.

This an IMPORTANT TOPIC, because it deal with "SECURITY"

- Now in the previous video we made a bit of security issues.
- When using the `chmod +x <script name>` : Notice how EVERYONE IN THE FILE SYSTEM now has access to the file.
- Doing a quick `ls -l` to view who has permissions and guess what : Everyone has those permissions.

```
total 36
drwxr-xr-x 2 kali kali 4096 Feb 19 05:00 Desktop
drwxr-xr-x 2 kali kali 4096 Feb 16 11:08 Documents
drwxr-xr-x 2 kali kali 4096 Feb 16 11:08 Downloads
drwxr-xr-x 2 kali kali 4096 Feb 16 11:08 Music
-rwxr-xr-x 1 kali kali 228 Feb 22 05:38 our_first_script.sh
drwxr-xr-x 2 kali kali 4096 Feb 16 11:08 Pictures
drwxr-xr-x 2 kali kali 4096 Feb 16 11:08 Public
drwxr-xr-x 2 kali kali 4096 Feb 16 11:08 Templates
drwxr-xr-x 2 kali kali 4096 Feb 16 11:08 Videos
```

What are File permissions?

Basically determine who CAN PERFORM CERTAIN ACTIONS ON a GIVEN FILE

Now how can we see who has what access to the file.

Here we can use the “ls -l” option.

```
total 36
drwxr-xr-x 2 kali kali 4096 Feb 19 05:00 Desktop
drwxr-xr-x 2 kali kali 4096 Feb 16 11:08 Documents
drwxr-xr-x 2 kali kali 4096 Feb 16 11:08 Downloads
drwxr-xr-x 2 kali kali 4096 Feb 16 11:08 Music
-rwxr-xr-x 1 kali kali 228 Feb 22 05:38 our_first_script.sh
drwxr-xr-x 2 kali kali 4096 Feb 16 11:08 Pictures
drwxr-xr-x 2 kali kali 4096 Feb 16 11:08 Public
drwxr-xr-x 2 kali kali 4096 Feb 16 11:08 Templates
drwxr-xr-x 2 kali kali 4096 Feb 16 11:08 Videos
```

Now in Linux : FILE PERMISSIONS represented by a SET of 10 Charecters that are STRUNG together, as we can see above.

```
drwxr-xr-x
drwxr-xr-x
```

The 1st character is either a “-” hyphen or a “d”. I take d at this point means “Directory”

```
drwxr-xr-x 2 kali kali 4096 Feb 19 05:00 Desktop
drwxr-xr-x 2 kali kali 4096 Feb 16 11:08 Documents
drwxr-xr-x 2 kali kali 4096 Feb 16 11:08 Downloads
drwxr-xr-x 2 kali kali 4096 Feb 16 11:08 Music
-rwxr-xr-x 1 kali kali 228 Feb 22 05:38 our_first_script.sh
drwxr-xr-x 2 kali kali 4096 Feb 16 11:08 Pictures
drwxr-xr-x 2 kali kali 4096 Feb 16 11:08 Public
drwxr-xr-x 2 kali kali 4096 Feb 16 11:08 Templates
drwxr-xr-x 2 kali kali 4096 Feb 16 11:08 Videos
```

now unless if there a specific reason not to : Only the file owner should have READ + WRITE + Execute commands on the page.

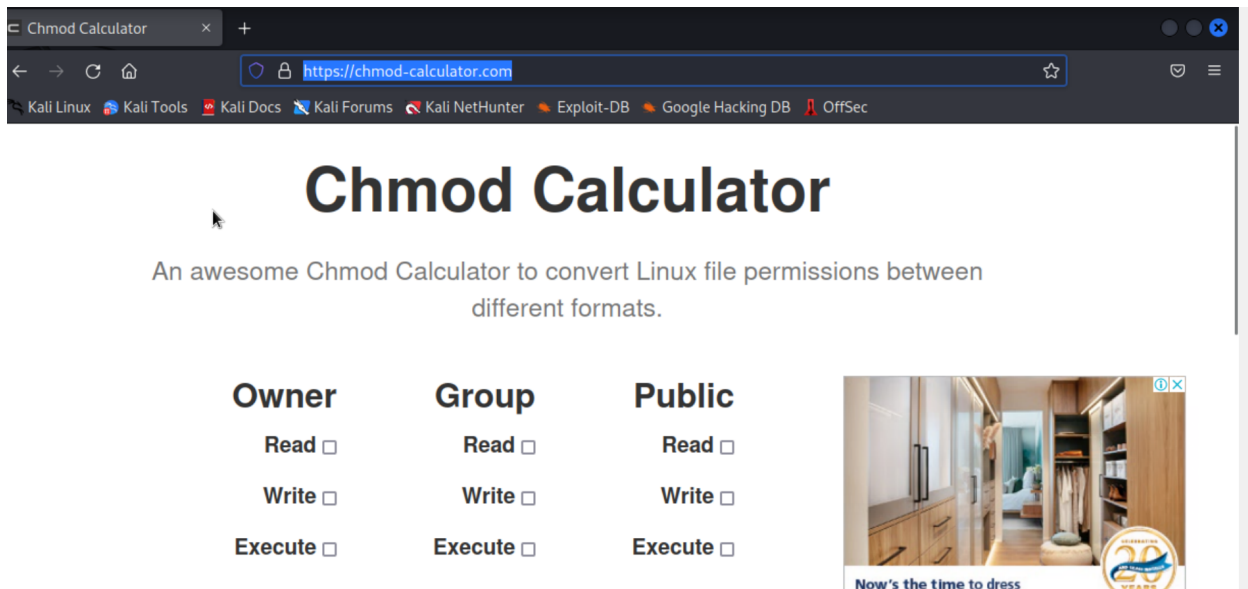
Everyone else should only be allowed to read it. 😊

→ You need to know how to make you script more secure.

You can use the “chmod” command.

Sometime we can have information overload and if you would like to, you can go to : chmod-calculator.com to calculate which permissions.

→ We tried an exercise that would not let anyone run the bash script.



The command that we wrote is :

```
(kali㉿kali)-[~]
└─$ chmod 000 our_first_script.sh

(kali㉿kali)-[~]
└─$ ./our_first_script.sh
ash: permission denied: ./our_first_script.sh

(kali㉿kali)-[~]
└─$ sudo su
[sudo] password for kali:
(root㉿kali)-[/home/kali]
└─$ ./home/kali/
bash: ./home/kali/: No such file or directory

(kali㉿kali)-[/home/kali]
└─$ ls
Desktop  Documents  Downloads  Music  our_first_script.sh  Pictures  Public  Templates  Videos

(kali㉿kali)-[/home/kali]
└─$ ls -l
total 36
drwxr-xr-x 2 kali kali 4096 Feb 24 00:31 Desktop
drwxr-xr-x 2 kali kali 4096 Feb 16 11:08 Documents
drwxr-xr-x 2 kali kali 4096 Feb 16 11:08 Downloads
drwxr-xr-x 2 kali kali 4096 Feb 16 11:08 Music
----- 1 kali kali 228 Feb 22 05:38 our_first_script.sh
drwxr-xr-x 2 kali kali 4096 Feb 16 11:08 Pictures
```

But not you need to remember the following:

- 1) Set the code to 755.
- 2) if you want to give someone write permissions that also need to have read permissions as well 😊

Completing the 1st exercise

Backup Script Project Brief

Scenario

Your boss has noticed that you do some very valuable work for the company, and, to ensure it doesn't get lost, they have suggested that you create a script that you can use to easily backup all the files in your home directory.

Your boss also thinks that this script will be very useful for others in the office, and wants you to make sure the script is professionally formatted before sharing it with your colleagues.

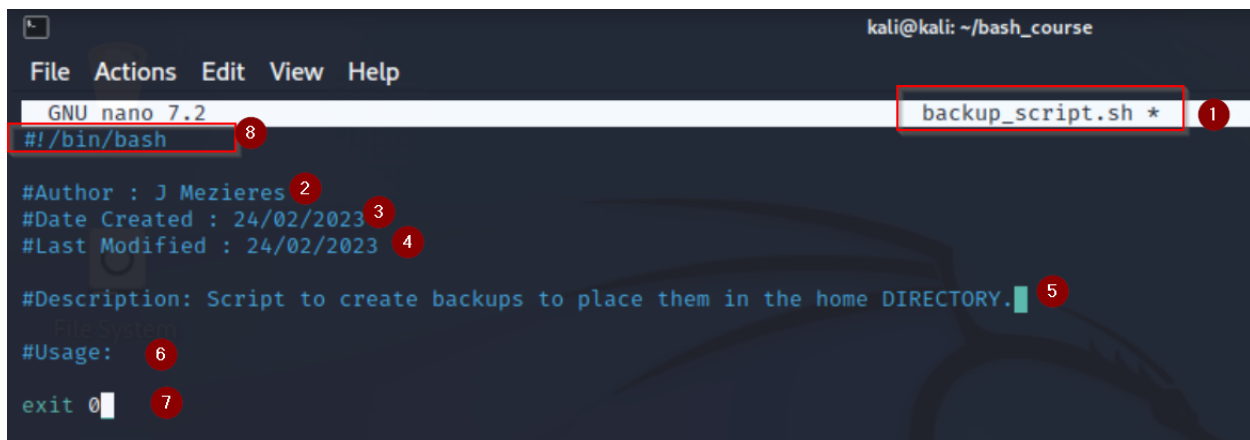
Your Task

Step 1: Create a bash script called `backup_script` in your `~/bash_course/` directory.

This script should backup all the files in your home directory and save them all in a `.tar` archive. **Guidance is provided below on how to do this.**

When writing your script, remember to add all the core components of your script:

1. The “Shebang” line
2. Commands (**see the "extra guidance" section below**)
3. An exit statement with an appropriate exit code.



The screenshot shows a terminal window with the nano editor open at `~/bash_course`. The file `backup_script.sh` is being edited. The content of the script is as follows:

```
GNU nano 7.2 backup_script.sh *
#!/bin/bash

#Author : J Mezieres
#Date Created : 24/02/2023
#Last Modified : 24/02/2023

#Description: Script to create backups to place them in the home DIRECTORY.
#Usage:
exit 0
```

Numbered annotations (1-8) are placed on the image to highlight specific parts of the script:

- 1: The filename `backup_script.sh` in the title bar.
- 2: The author comment `#Author : J Mezieres`.
- 3: The date created comment `#Date Created : 24/02/2023`.
- 4: The last modified comment `#Last Modified : 24/02/2023`.
- 5: The description comment `#Description: Script to create backups to place them in the home DIRECTORY.`.
- 6: The usage comment `#Usage:`.
- 7: The exit statement `exit 0`.
- 8: The shebang line `#!/bin/bash`.

Step 2: Give the script the correct permissions.

Because the script will be shared with others in your organisation, your script should have the following permissions:

- The file owner (i.e you) should have read, write, and execute permissions.
- Everyone in the file’s group (i.e. your colleagues) should have read and execute permissions. For security purposes, they should not have write permissions.

- Everyone else in the organisation should only have read permissions.

Use permissions-calculator.org/ to figure out the correct octal code to use with the `chmod` command to achieve this.

Step 3: Run your script multiple times to check that it's working.

You should see it creating multiple backups for you in your `~/bash_course` folder.

```
(kali@kali)-[~/bash_course]
$ chmod 744 backup_script.sh
```

Doing an `ls -l` to show all the fields if the permissions I set are correct.

```
(kali@kali)-[~/bash_course]
$ ls -l
total 4
-rwxr--r-- 1 kali kali 177 Feb 24 04:55 backup_script.sh
```

```

(kali㉿kali)-[~/bash_course]
$ ls
backup_script.sh

(kali㉿kali)-[~/bash_course]
$ ./backup_script.sh
/home/kali/bash_course/
/home/kali/bash_course/backup_script.sh
/home/kali/Desktop/
/home/kali/Documents/
/home/kali/Downloads/
/home/kali/Music/
/home/kali/our_first_script.sh
/home/kali/Pictures/
/home/kali/Public/
/home/kali/Templates/
/home/kali/Videos/

```

This is what the script looked like inside.

```

kali@kali: ~/bash_course
File Actions Edit View Help
GNU nano 7.2 backup_script.sh
#!/bin/bash

#Author : J Mezieres
#Date Created : 24/02/2023
#Last Modified : 24/02/2023

#Description: Script to create backups to place them in the home DIRECTORY.
#Usage:
tar -cvf ~/bash_course/my_backup_"$(date +%d-%m-%Y_%H-%M-%S)".tar ~/* 2>/dev/null

```

Section 9: Adding Scripts to Your Path

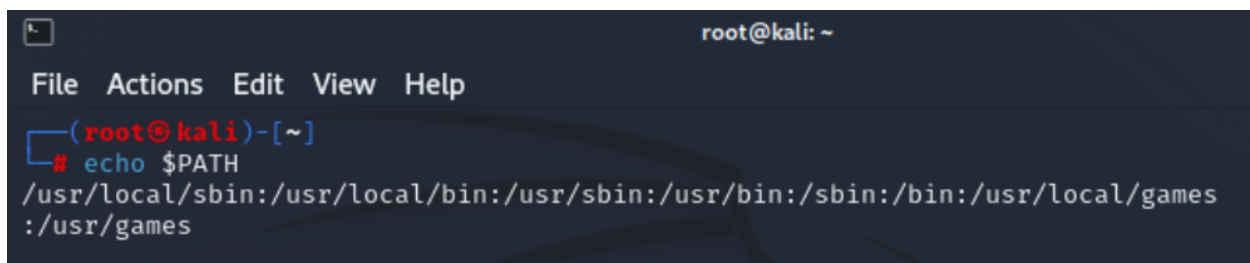
- What is the PATH Variable.
- How to add directories to your PATH VARIABLE.
- Explain the purpose of the Path variable.
- Add new directories to your path.
- Run scripts from any where in our system :)
- Allows us to run our scripts from anywhere we are in the network.
- This is awesome because this can save us time from always having to run the script from a specific location.

What is a System Path?

In Linux this will tell our system which path to search for Executable files in.

How can we see which directories are in our Path.

In Kali Linux type : `echo $PATH`

A terminal window titled 'root@kali: ~' with a menu bar (File, Actions, Edit, View, Help). The prompt is '(root@kali)-[~]'. The command '# echo \$PATH' has been entered, and the output is displayed on the next line: '/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/local/games:/usr/games'.

- Basically, when we run a command, its going to look in all of these directories, to look for the command and this will tell us and inform us all the commands in the system 😊
- Now what we could just do is jsut the script to anyone of the folder in our path.
- But for this example ,what we are going to do is create folder that we can create to our path.

Now in our Bash folder, lets create a new folder called Scripts.

```
File Actions Edit View Help
(kali@kali)-[~]
└─$ cd bash_course
(kali@kali)-[~/bash_course]
└─$ mkdir "Scripts"
(kali@kali)-[~/bash_course]
└─$
```

Now we are going to move this script into the Scripts folder.

```
(kali@kali)-[~/bash_course]
└─$ mkdir "Scripts"
(kali@kali)-[~/bash_course]
└─$ ls
backup_script.sh  my_backup_24-02-2023_05-00-53.tar  Scripts
(kali@kali)-[~/bash_course]
└─$ mv backup_script.sh Scripts
```

Lets go to our home directory.

1) This can be done by typing “~”

2) type “nano ~/.profile

(This is a hidden file.)


```
(kali㉿kali)-[~]  
$ cd ~; nano ~/.profile  
  
(kali㉿kali)-[~]  
$
```

but now we want to add our folder so that our scripts can be run.

Now, when at the end of the file, you will need to type out the following.

```
export PATH="$PATH:$HOME/bash_course/Scripts"
```

```
GNU nano 7.2  
# see /usr/share/doc/bash/examples/startup-files for examples.  
# the files are located in the bash-doc package.  
  
# the default umask is set in /etc/profile; for setting the umask  
# for ssh logins, install and configure the libpam-umask package.  
#umask 022  
  
# if running bash  
if [ -n "$BASH_VERSION" ]; then  
    # include .bashrc if it exists  
    if [ -f "$HOME/.bashrc" ]; then  
        . "$HOME/.bashrc"  
    fi  
fi  
  
# set PATH so it includes user's private bin if it exists  
if [ -d "$HOME/bin" ] ; then  
    PATH="$HOME/bin:$PATH"  
fi  
  
# set PATH so it includes user's private bin if it exists  
if [ -d "$HOME/.local/bin" ] ; then  
    PATH="$HOME/.local/bin:$PATH"  
fi  
  
export PATH="$PATH:$HOME/bash_course/Scripts"
```

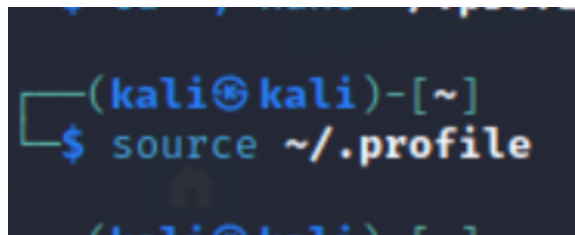
Now this will be added on as a path variable to the end of all of the paths that we have.

Note : the .profile script is normally loaded up when our computer boots up.

So if we want the shell to read our updated scripts we have 2 options.

- 1) Reboot our machine.
- 2) Use the “source” command to update the file.

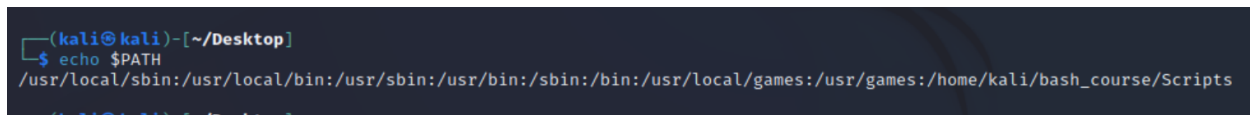
So run the following command which is : “source ~/.profile”



```
(kali@kali)-[~]  
$ source ~/.profile
```

- 3) Now type “echo \$PATH”

We should see the file that you just added.



```
(kali@kali)-[~/Desktop]  
$ echo $PATH  
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/local/games:/usr/games:/home/kali/bash_course/Scripts
```

And bam Now all scripts inside this folder should be able to be executed 😊