Lab – Start here! Introduction to BASH Scripting

Introduction

One of the great features of Linux is to writing scripts. Compared to writing Windows batch files, BASH scripting is much more flexible and comes with advanced features you won't find in a batch script. To understand how the BASH shell works, you must understand the logic of how Linux is built.

Simply put, BASH (Bourne-Again Shell) is the default shell we are provided within Linux distributions. It is the command line interpreter (CLI) for GNU (**G**NU's **N**ot **U**nix) operating system. When we open a terminal session in Kali or Ubuntu, we are using the BASH shell. Though GNU operating system provides different shells, BASH is the default out of the box shell for Linux.

A shell is a program that provides the command line (i.e., the all-text display user interface) on Linux and other Unix-like operating systems. It also executes (i.e., runs) commands typed into the terminal and displays the results. BASH is the default shell on Linux.

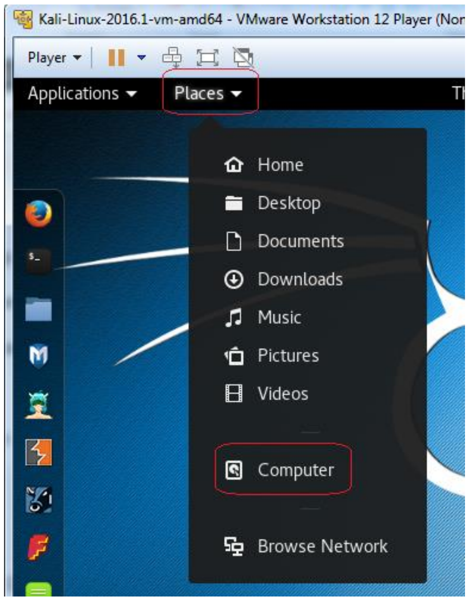
The GNU project is a mass collaborative initiative for the development of free software. Richard Stallman founded the project in 1978 at MIT.

When we type any syntax into a terminal window, it is the job of the BASH shell to interpret and execute the command. The BASH shell is located in the **bin** folder which is located at the root of the operating system. To access the root of any Linux system, we use the Unix command of **./**

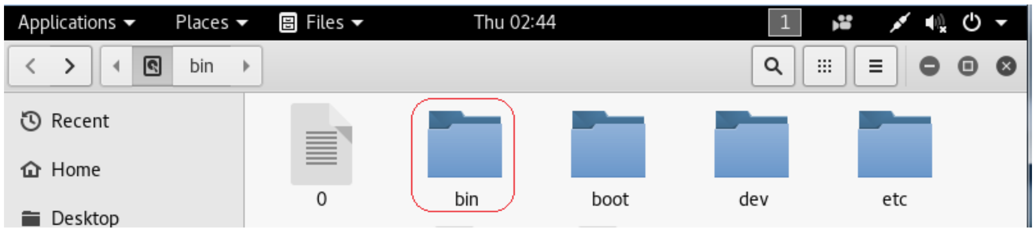
Let's create a visual of accessing the root in Kali.

Launch your Kali virtual machine.

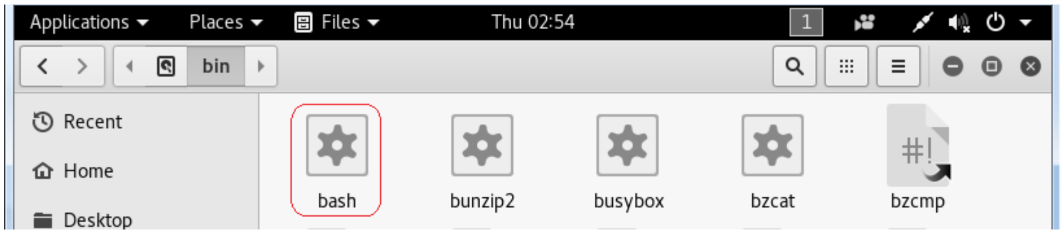
From the desktop, click **Places** and then open the link for **Computer**. You are now at the root for Kali or Ubuntu or any other GNU operating system.



The first directory we see at the root of our system is the bin folder.



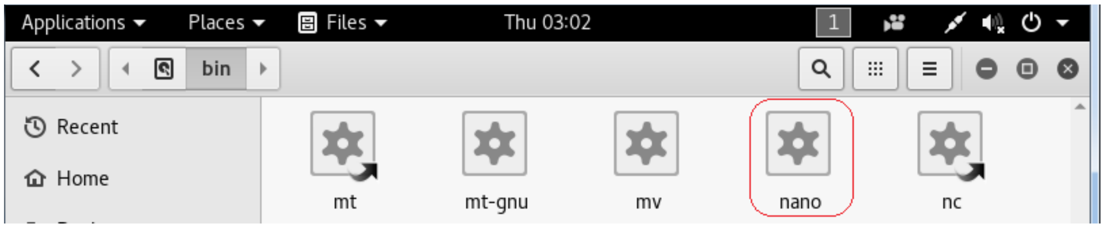
If we open the bin folder, we see the BASH interpreter.



So if you want to access the BASH interpreter, from the CLI, you would have to type, **./bin/bash**

Since the BASH interpreter is in the same folder as nearly all the default utilities and tools built into the GNU operating system, it makes it very easy for the completion of commands and the running of scripts using the CLI.

For instance, to create and edit scripts, we need a script editor. If you scroll through the **bin** folder, you'll see a script editor called nano.



Since the editor is in the same folder as BASH, it makes it very easy to create a script and save it without having to move in and out the bin folder. BASH has direct access to everything we need and use from the command line. Try to see the logic in how the operating is setup to use the BASH shell and BASH scripting.

Begin the lab…

As it is with life so it with learning technology, we have to crawl before we can walk. In this lab we'll be building a simple BASH script, going back and forth using the Nano editor to add text, make changes and using the CLI to run the script. Recall how the BASH shell knows how to find and open the Nano editor.

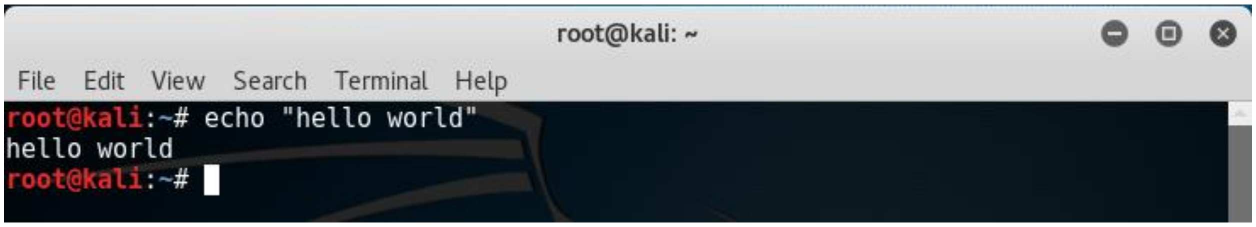
Open a new terminal window inside of Kali

Commands used in this script

A *command* is an instruction telling a computer to do something. An *argument* is input data for a command. *Standard output* is the display screen by default, but it can be redirected to a file, printer, etc.

echo - a built-in *command* in the bash shell that writes its *arguments* to *standard output*.

At the prompt, type echo "hello World" hit enter.



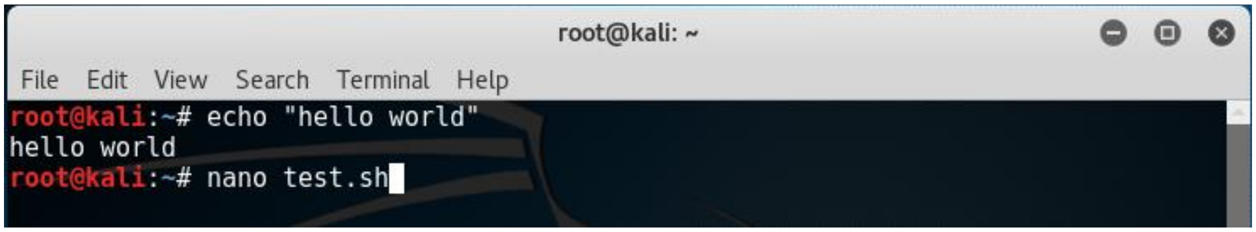
```
root@kali:~# echo "hello world"
hello world
root@kali:~# 
```

Let's break it down…. echo is the *command*, "hello world" is the *argument* and *standard input* is the hello world returned when you hit enter.

sleep - pauses the command for a specific amount of time. If I wanted to pause the output for 5 seconds, we use the sleep command followed by the number 5. 5 represents 5 seconds, 5m represents 5 minutes, and 5h represents 5 hours.

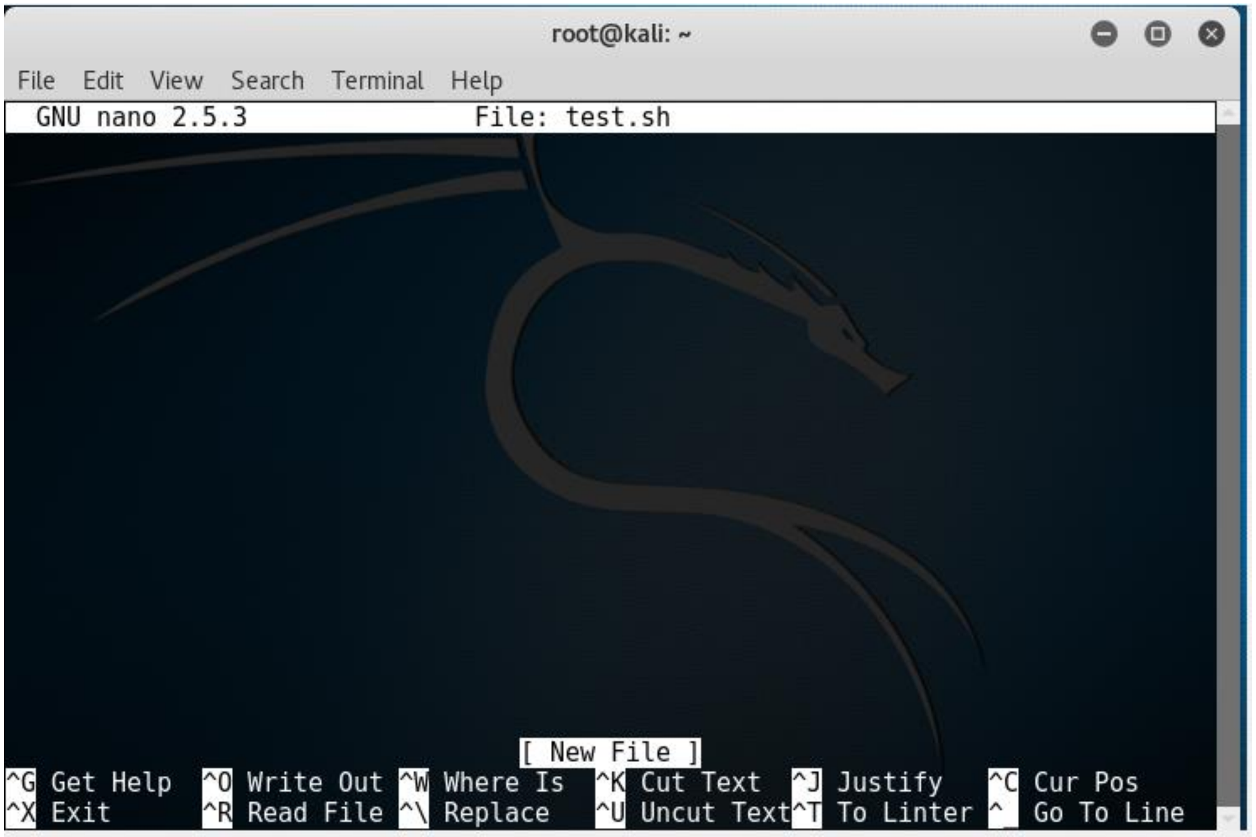
Let's put what we have so far into a script using nano. Nano is a text editor for Unix-like computing systems or operating environments using a command line interface.

At the terminal prompt type, nano followed be the name of the script. For this example, I've named the file we are about to create, test.sh. We are creating a file, not opening an existing file.



```
root@kali:~# echo "hello world"
hello world
root@kali:~# nano test.sh
```

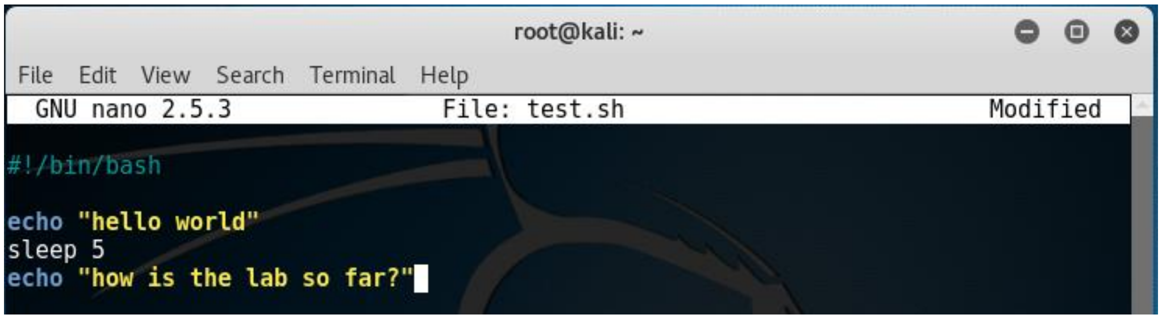
Hit enter and the nano editor opens up in the BASH shell.



```
  GNU nano 2.5.3              File: test.sh

                          [ New File ]
^G Get Help  ^O Write Out ^W Where Is  ^K Cut Text  ^J Justify   ^C Cur Pos
^X Exit      ^R Read File ^\ Replace   ^U Uncut Text^T To Linter ^_ Go To Line
```

Every script begins with The #! syntax used in scripts to indicate an interpreter for execution under UNIX / Linux operating systems. Remember at the beginning of the lab, we located the BASH shell at the root of the system, in the bin folder. We are telling the script, we are using the BASH interpreter to run the script and this is where it is located.

At the nano prompt, type the following…..

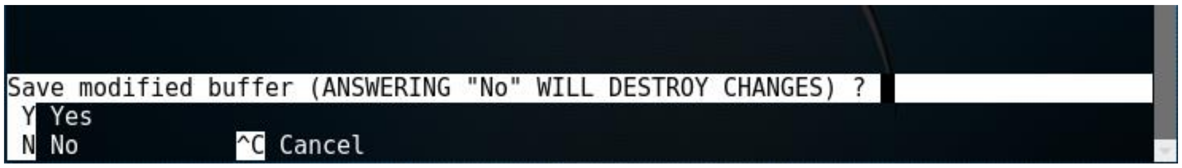


```
  GNU nano 2.5.3                File: test.sh                         Modified

#!/bin/bash

echo "hello world"
sleep 5
echo "how is the lab so far?"
```

Let’s save the script.

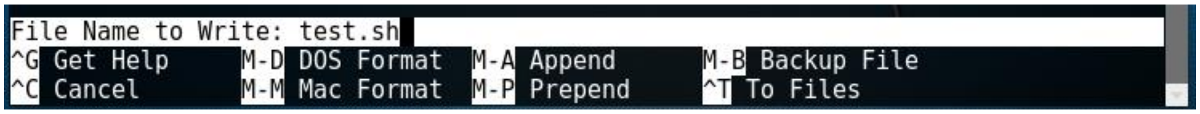
To save the script, press the CTRL+X keys on your keyboard at the same time.

Type in Y to save the changes.



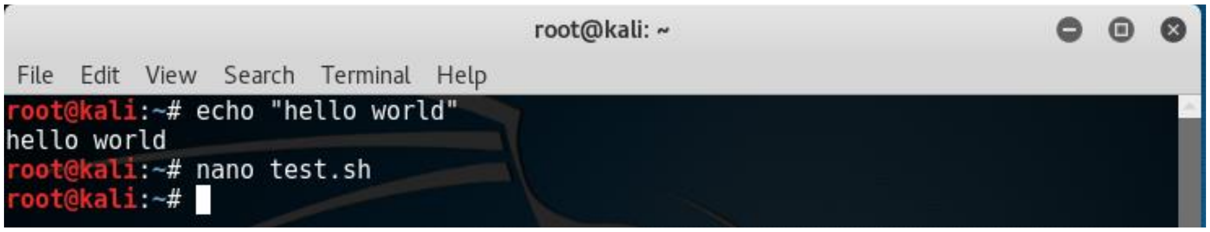
```
Save modified buffer (ANSWERING "No" WILL DESTROY CHANGES) ?
 Y Yes
 N No            ^C Cancel
```

Accept the default file named previously created by hitting the enter key.



```
File Name to Write: test.sh
^G Get Help       M-D DOS Format    M-A Append        M-B Backup File
^C Cancel         M-M Mac Format    M-P Prepend       ^T To Files
```

This bring you back to the terminal prompt.

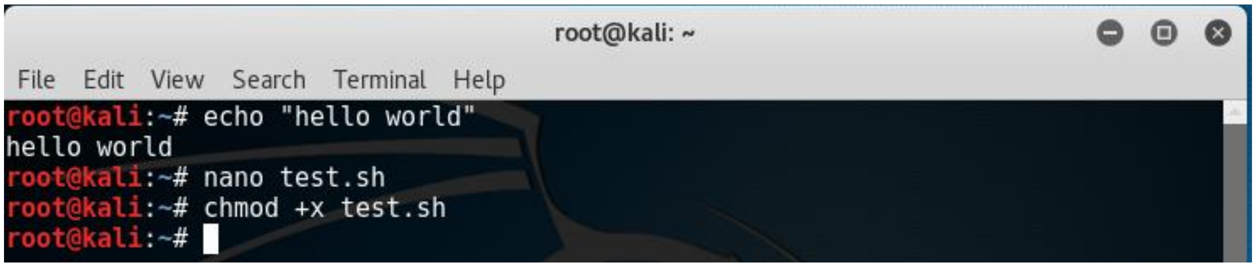


```
root@kali:~# echo "hello world"
hello world
root@kali:~# nano test.sh
root@kali:~# 
```

We now have script file saved as test.sh.

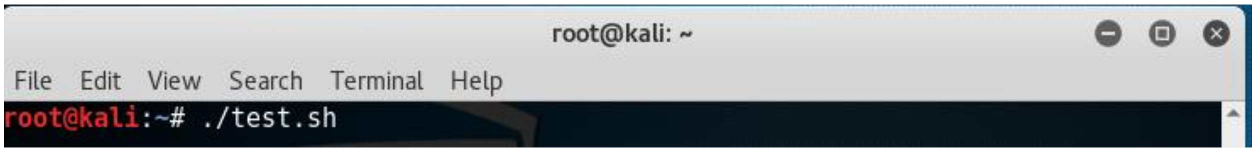
We now need to make the test.sh file executable.

At the prompt type **chmod +x test.sh** (change mode, this utility is also located in the bin file along with the BASH shell, the nano editor and now our script file.) The +x means run the chmod command ‘plus” make the following file executable.



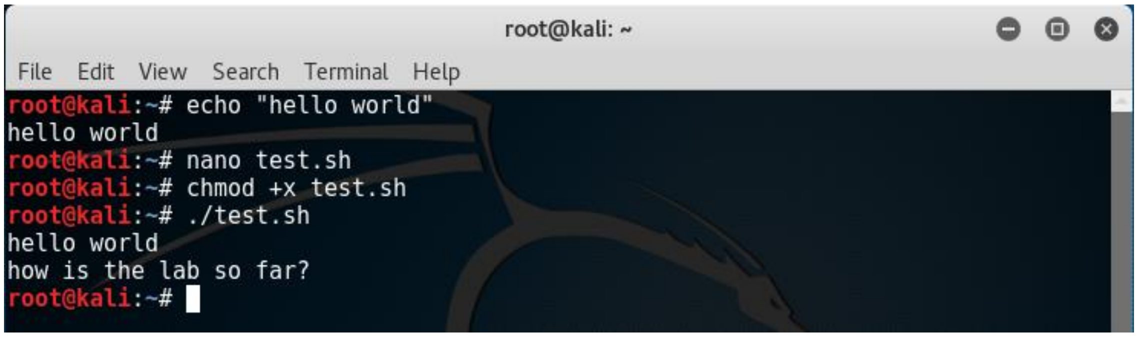
```
root@kali:~# echo "hello world"
hello world
root@kali:~# nano test.sh
root@kali:~# chmod +x test.sh
root@kali:~#
```

We have changed the mode of the file using the chmod command. All that is needed is to go type in the location of file along with the file name and hit enter.



```
root@kali:~# ./test.sh
```

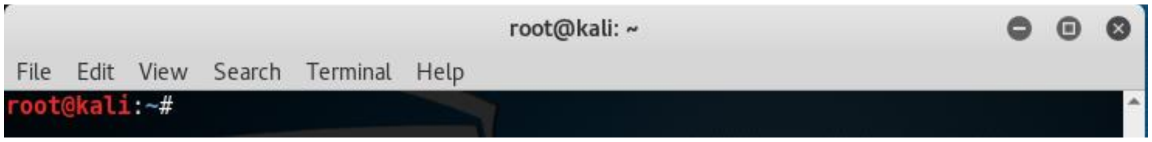
When you launch the script, it echoes "hello world" and pauses 5 seconds before it echoes. "how is the lab so far?"



```
root@kali:~# echo "hello world"
hello world
root@kali:~# nano test.sh
root@kali:~# chmod +x test.sh
root@kali:~# ./test.sh
hello world
how is the lab so far?
root@kali:~#
```

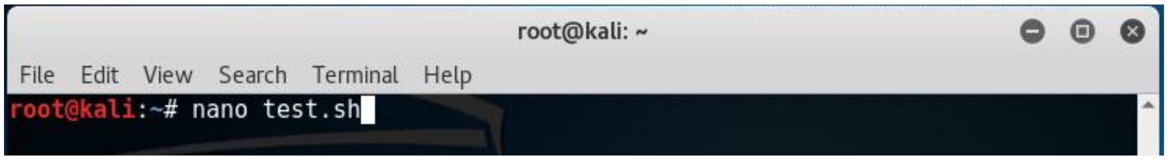
Another command we can use is the **clear** command. This clears the terminal of any text.

At the terminal prompt, type the command clear and hit enter. The screen is cleared.



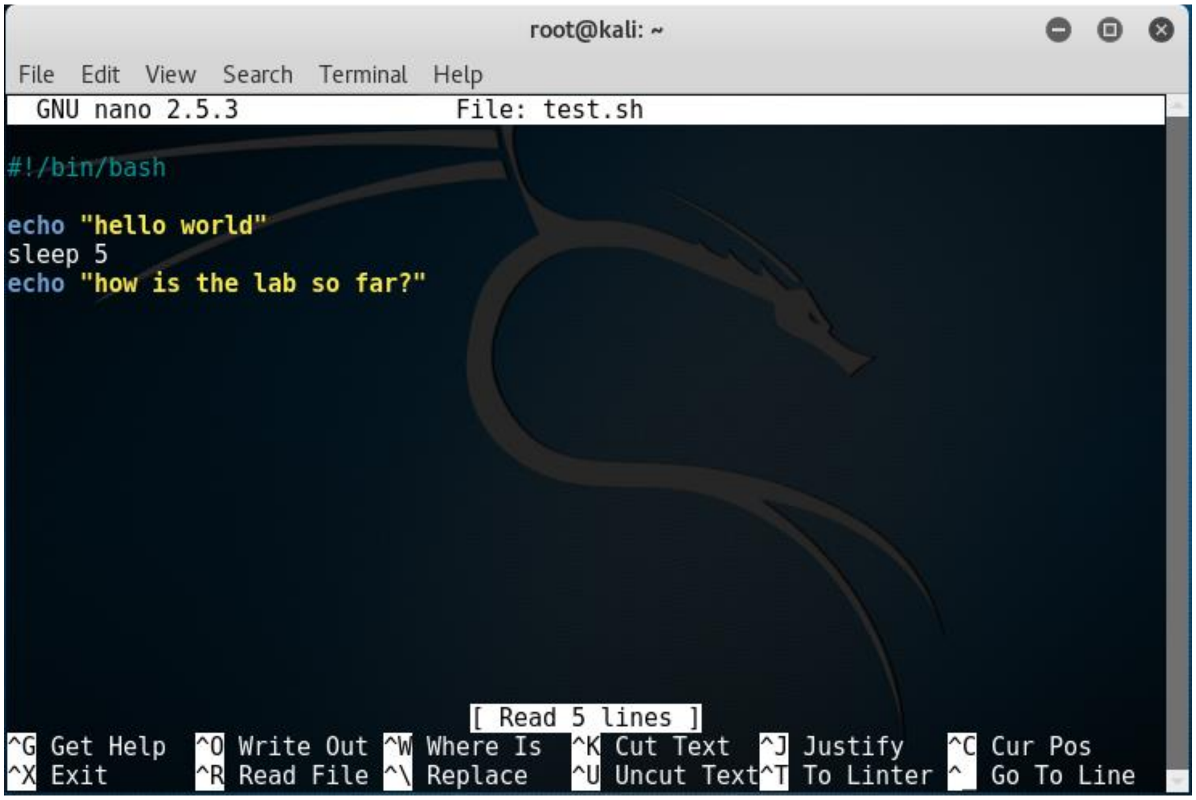
```
root@kali:~#
```

Let's go back into the nano file we created.



```
root@kali:~# nano test.sh
```

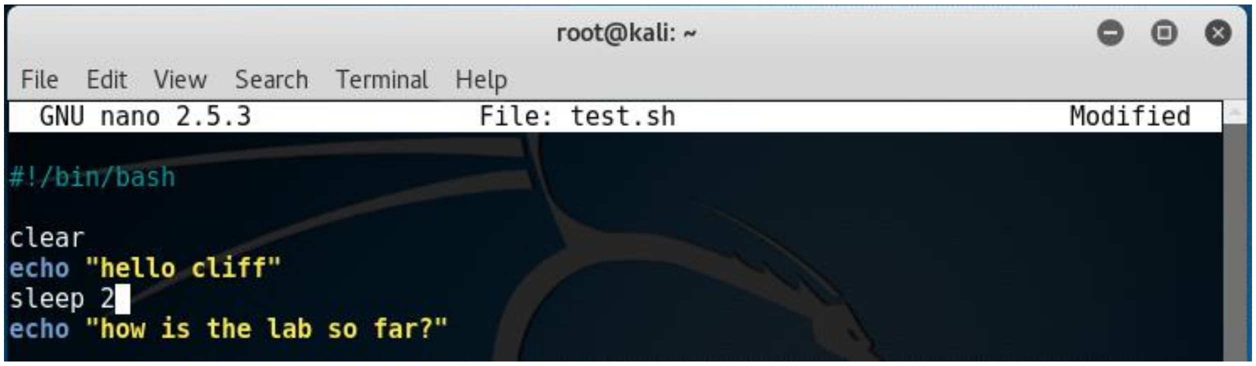
Hit enter….



```
  GNU nano 2.5.3                 File: test.sh

#!/bin/bash

echo "hello world"
sleep 5
echo "how is the lab so far?"




                              [ Read 5 lines ]
^G Get Help   ^O Write Out ^W Where Is   ^K Cut Text   ^J Justify    ^C Cur Pos
^X Exit       ^R Read File ^\ Replace    ^U Uncut Text ^T To Linter  ^_ Go To Line
```

Let’s start off the script with the clear command. Place your mouse at the end of the last line of the script and hit the up arrow until you are back to the start of the first echo command. Type the command clear, hit enter; this moves the echo command down one line.

Change “hello World” to say “hello <your name>’, change the sleep time to 2 seconds.

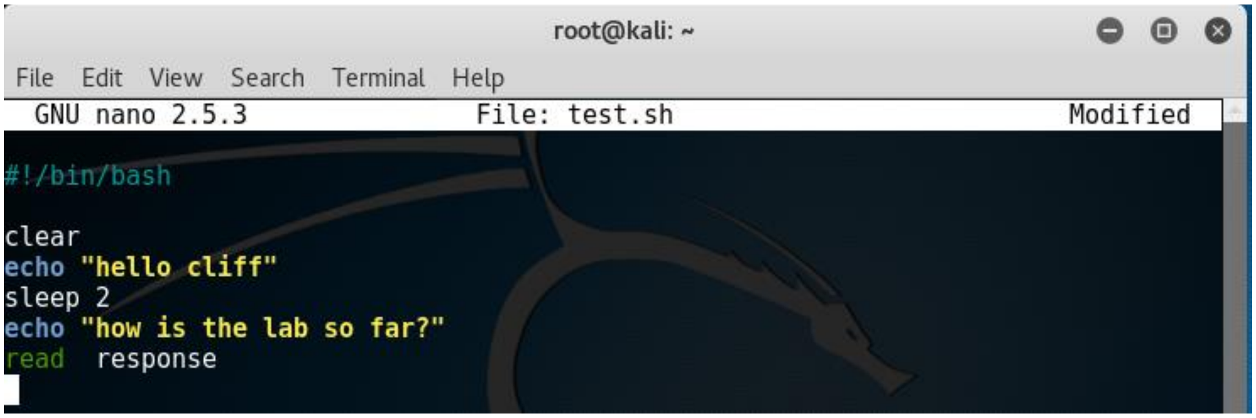


```
  GNU nano 2.5.3                 File: test.sh                        Modified

#!/bin/bash

clear
echo "hello cliff"
sleep 2
echo "how is the lab so far?"
```

The **read** command - Accepts user’s responses. Used for user-created variables ii storing information a user enters in response to a prompt.

By adding a variable to the read command, we can store and read back a user’s response.

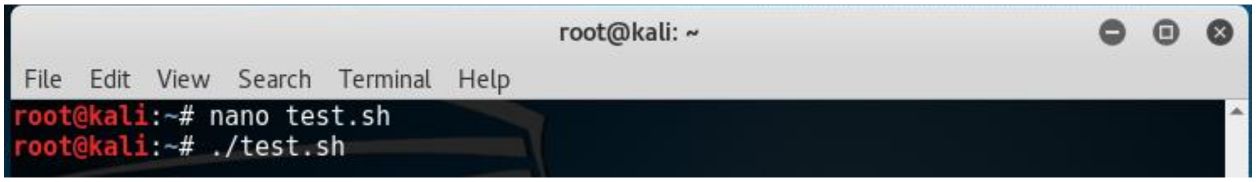
We first have to create the variable. Let’s use “response” as our variable.



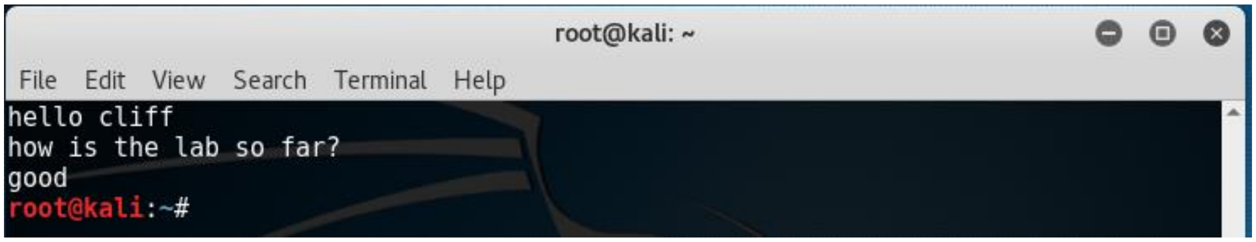
```
  GNU nano 2.5.3              File: test.sh                    Modified

#!/bin/bash

clear
echo "hello cliff"
sleep 2
echo "how is the lab so far?"
read  response
```

Save the file by pressing the CTRL+X, keys at the same time. Press Y for yes and hit enter to accept the current name of the file.

Run the script again.



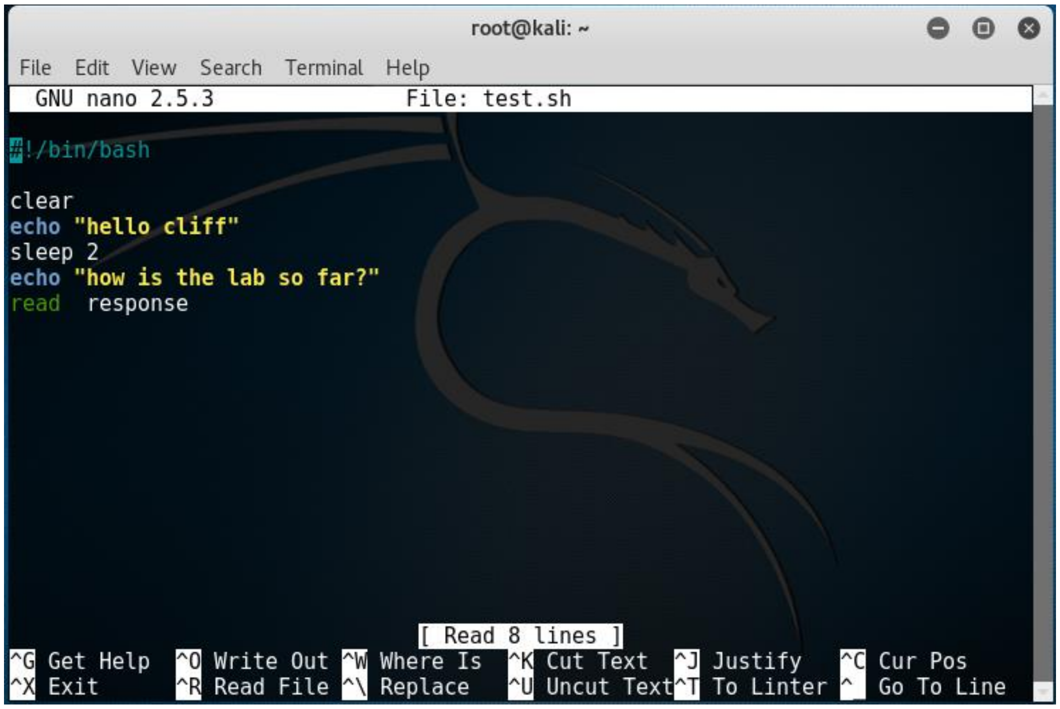
```
root@kali:~# nano test.sh
root@kali:~# ./test.sh
```



```
hello cliff
how is the lab so far?
good
root@kali:~#
```

The script clears the screen, says hello <your name> and ask the question, "how the lab is so far?" and now it waits for the user's response. I typed in good, hit enter, and it brought me back to the terminal prompt.

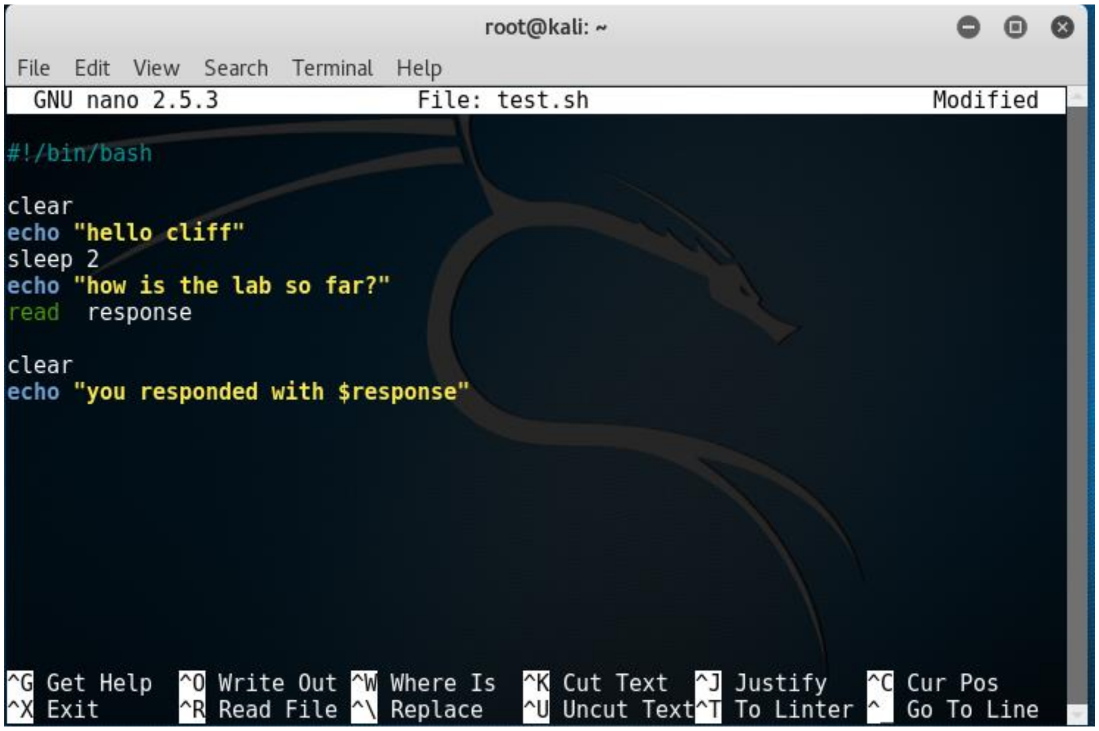
Let's go back into the script and do something with the information that was inputted. Open the script.



```
  GNU nano 2.5.3                File: test.sh

#!/bin/bash

clear
echo "hello cliff"
sleep 2
echo "how is the lab so far?"
read  response




                                [ Read 8 lines ]
^G Get Help  ^O Write Out ^W Where Is  ^K Cut Text  ^J Justify   ^C Cur Pos
^X Exit      ^R Read File ^\ Replace   ^U Uncut Text^T To Linter ^_ Go To Line
```

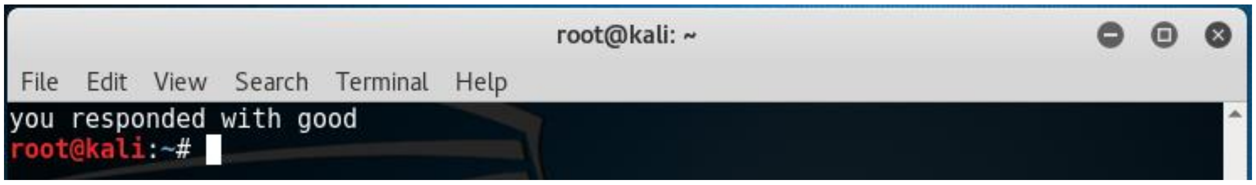
On the next line, add the clear command, hit enter and add another echo command followed by "you responded with $response"

The $ sign lets the script know we are using the variable response used with the read command. Whatever the response to the questions is, the script knows to use it.



```
  GNU nano 2.5.3              File: test.sh                        Modified

#!/bin/bash

clear
echo "hello cliff"
sleep 2
echo "how is the lab so far?"
read  response

clear
echo "you responded with $response"

^G Get Help  ^O Write Out ^W Where Is  ^K Cut Text  ^J Justify   ^C Cur Pos
^X Exit      ^R Read File ^\ Replace   ^U Uncut Text^T To Linter ^_ Go To Line
```

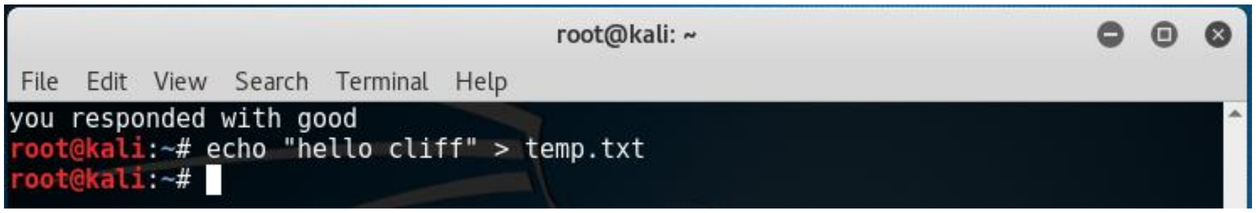
Save the file, run the script and insert your response.



```
you responded with good
root@kali:~# 
```

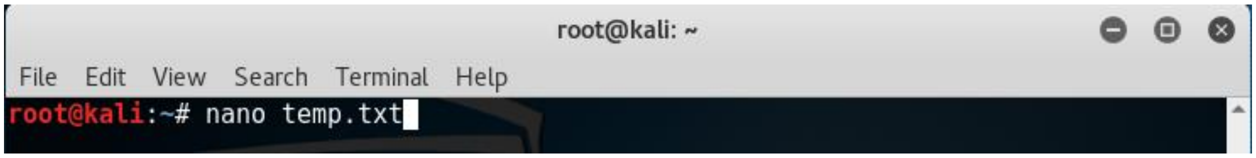
We can also send the output to a text file.

If we follow up the echo “hello Cliff” with the > and we name the file, we will have the file saved with the output of the echo command.

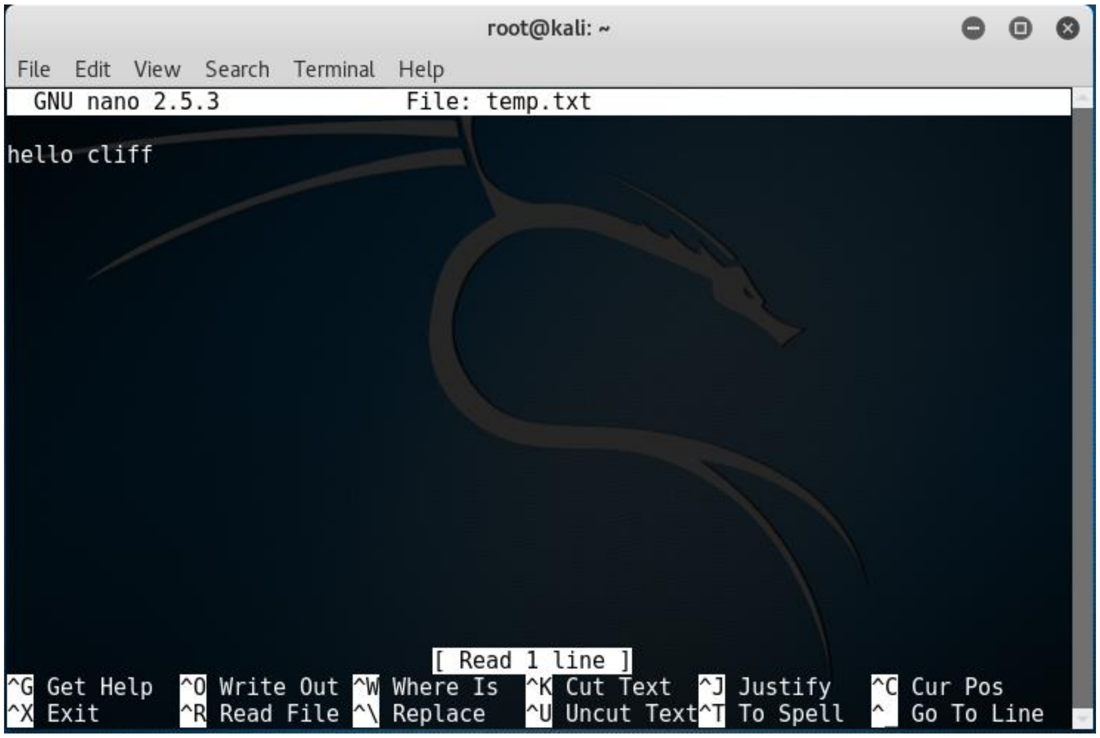


```
you responded with good
root@kali:~# echo "hello cliff" > temp.txt
root@kali:~# 
```

To open the text file, we open the temp.txt file using nano.

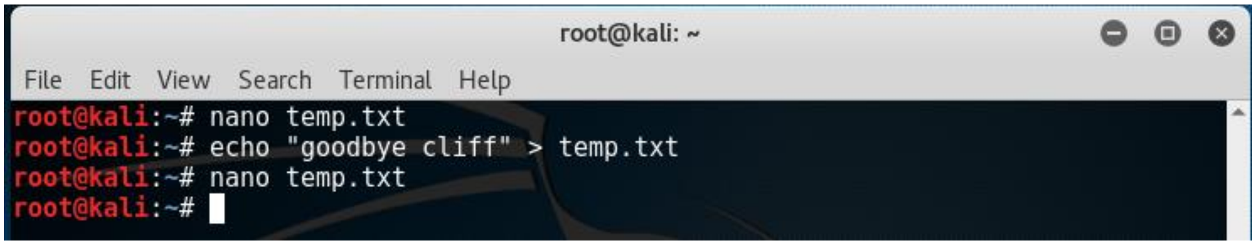


```
root@kali:~# nano temp.txt
```

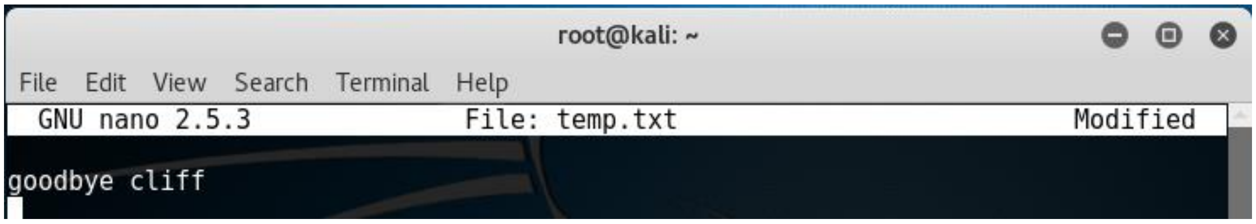


```
  GNU nano 2.5.3              File: temp.txt

hello cliff








                              [ Read 1 line ]
^G Get Help  ^O Write Out ^W Where Is  ^K Cut Text  ^J Justify   ^C Cur Pos
^X Exit      ^R Read File ^\ Replace   ^U Uncut Text^T To Spell  ^_ Go To Line
```

If we repeat the output to the text file but change the "hello cliff" to "goodbye Cliff" the first entry is overwritten.

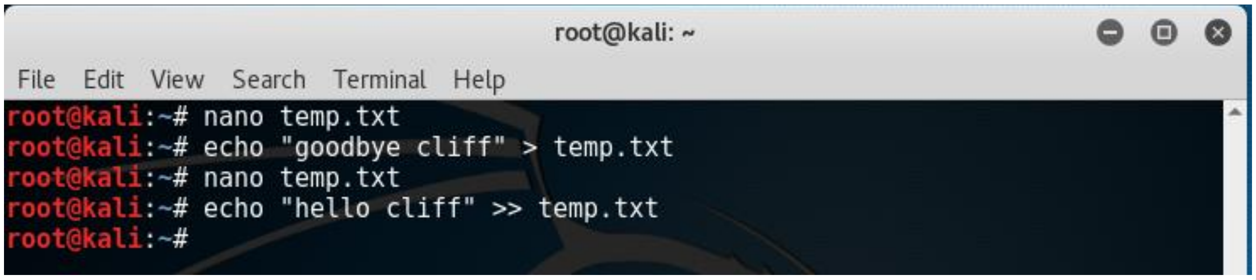


```
root@kali:~# nano temp.txt
root@kali:~# echo "goodbye cliff" > temp.txt
root@kali:~# nano temp.txt
root@kali:~# 
```

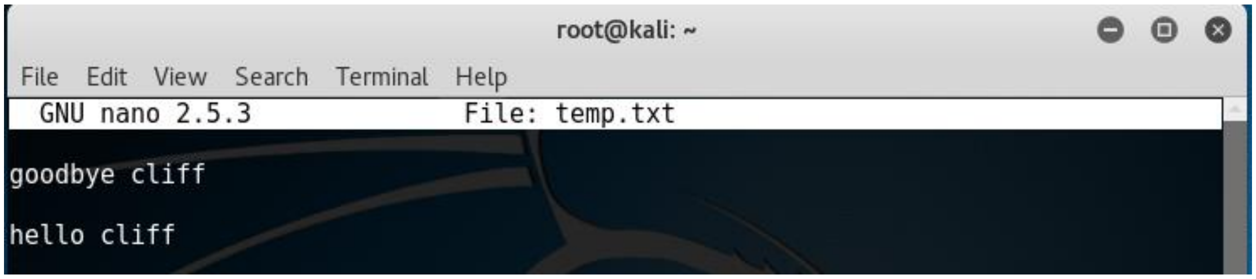


```
  GNU nano 2.5.3              File: temp.txt                        Modified

goodbye cliff
```

To not overwrite our first entry and save the next output to the file on a new line, add one additional > to the echo command.

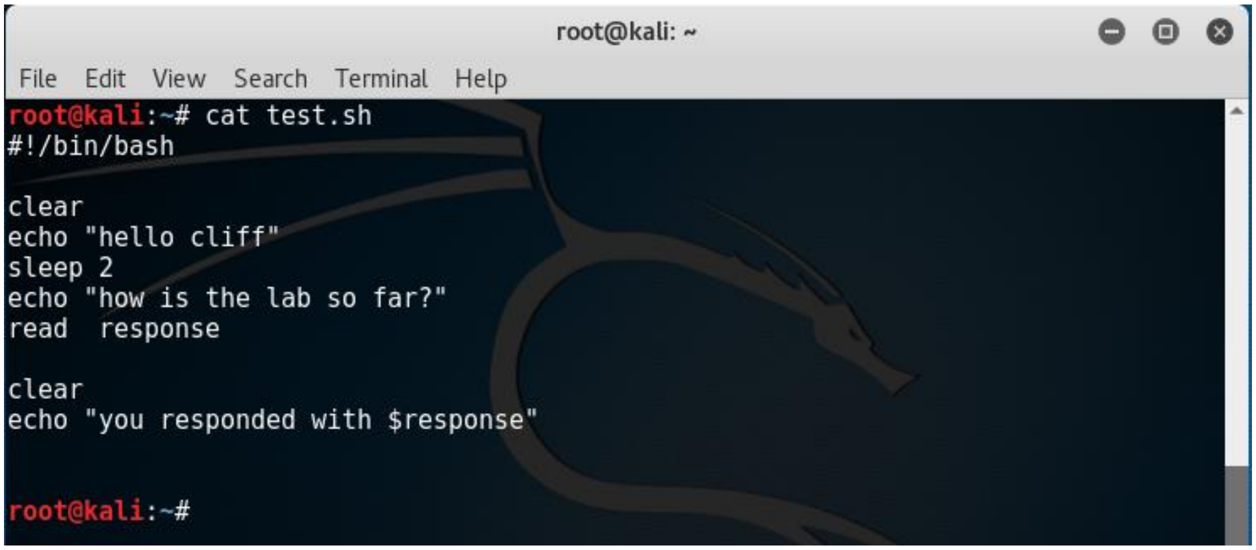


```
root@kali:~# nano temp.txt
root@kali:~# echo "goodbye cliff" > temp.txt
root@kali:~# nano temp.txt
root@kali:~# echo "hello cliff" >> temp.txt
root@kali:~#
```



```
 GNU nano 2.5.3                 File: temp.txt

goodbye cliff

hello cliff
```

If I want to see what is inside a file without using an editor, I can use the **cat** command. Here I use the **cat** command to see what is inside or test,sh script file.

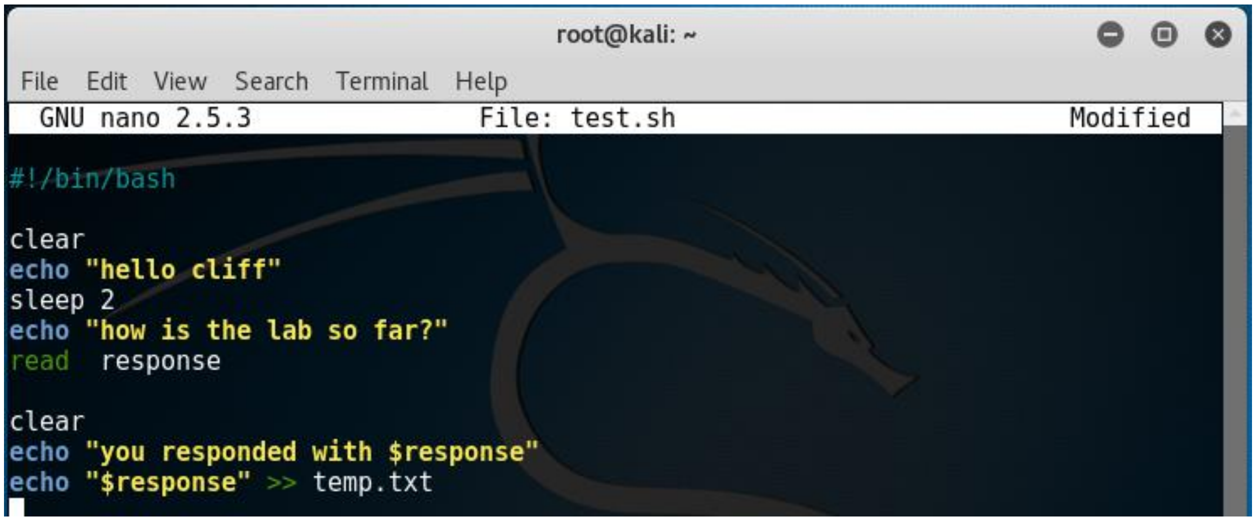


```
root@kali:~# cat test.sh
#!/bin/bash

clear
echo "hello cliff"
sleep 2
echo "how is the lab so far?"
read  response

clear
echo "you responded with $response"


root@kali:~#
```

Open the test.sh file using nano.

Tip! You can use the up arrow to see the command history of the terminal. Keep hitting the up arrow until you see the command you want.

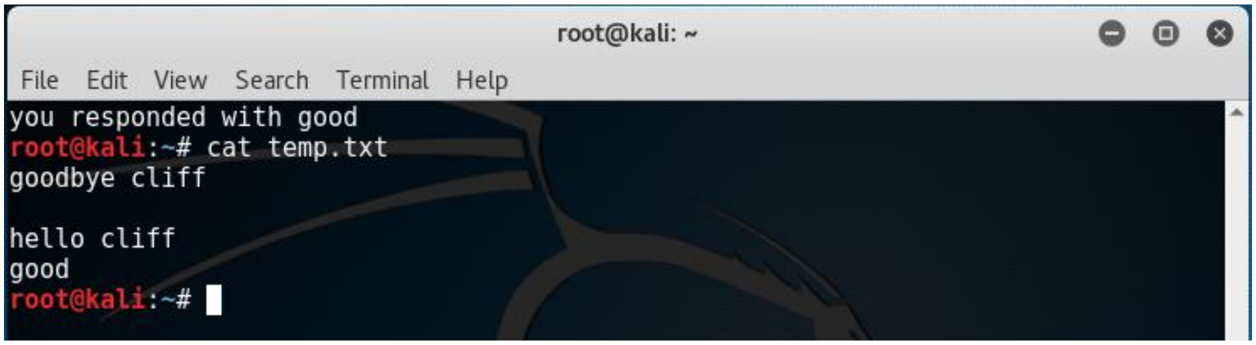
On a new line, add an echo command that saves the $response to a file called temp.txt. Make sure that the response is saved to a new line and does not overwrite any saved data.



```
  GNU nano 2.5.3              File: test.sh                        Modified

#!/bin/bash

clear
echo "hello cliff"
sleep 2
echo "how is the lab so far?"
read  response

clear
echo "you responded with $response"
echo "$response" >> temp.txt
```

Save the file and run the script.

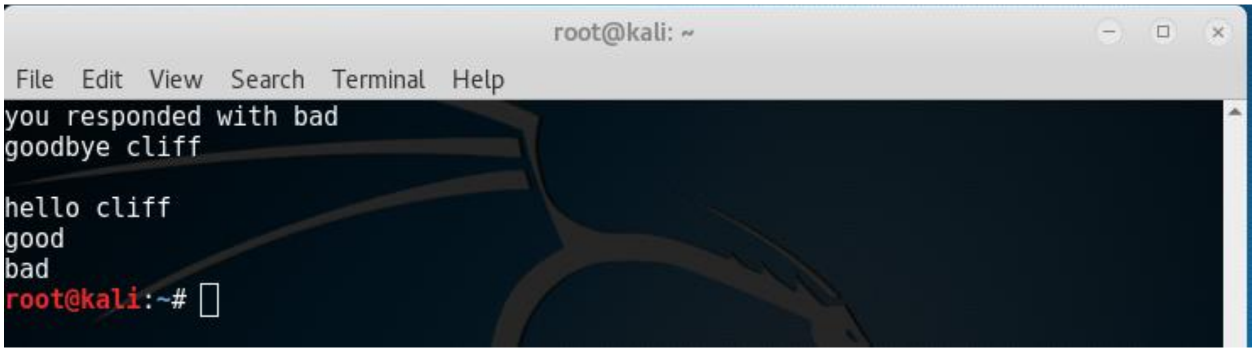
Check the contents of the temp.txt file using **cat** command.



```
you responded with good
root@kali:~# cat temp.txt
goodbye cliff

hello cliff
good
root@kali:~# 
```

You can also add a line to our test.sh file that says to output the content of the temp.txt using the **cat** command.

Add the line to the script file, save and run the script but this time use the word "bad" as your response to the question.



```
you responded with bad
goodbye cliff

hello cliff
good
bad
root@kali:~# 
```

This was a basic BASH shell scripting lab, but again, we must crawl before we can walk.

End of the lab!