

Tom's Hardware is supported by its audience. When you purchase through links on our site, we may earn an affiliate commission. [Here's why you can trust us.](#)

[Home](#) > [How-to](#)

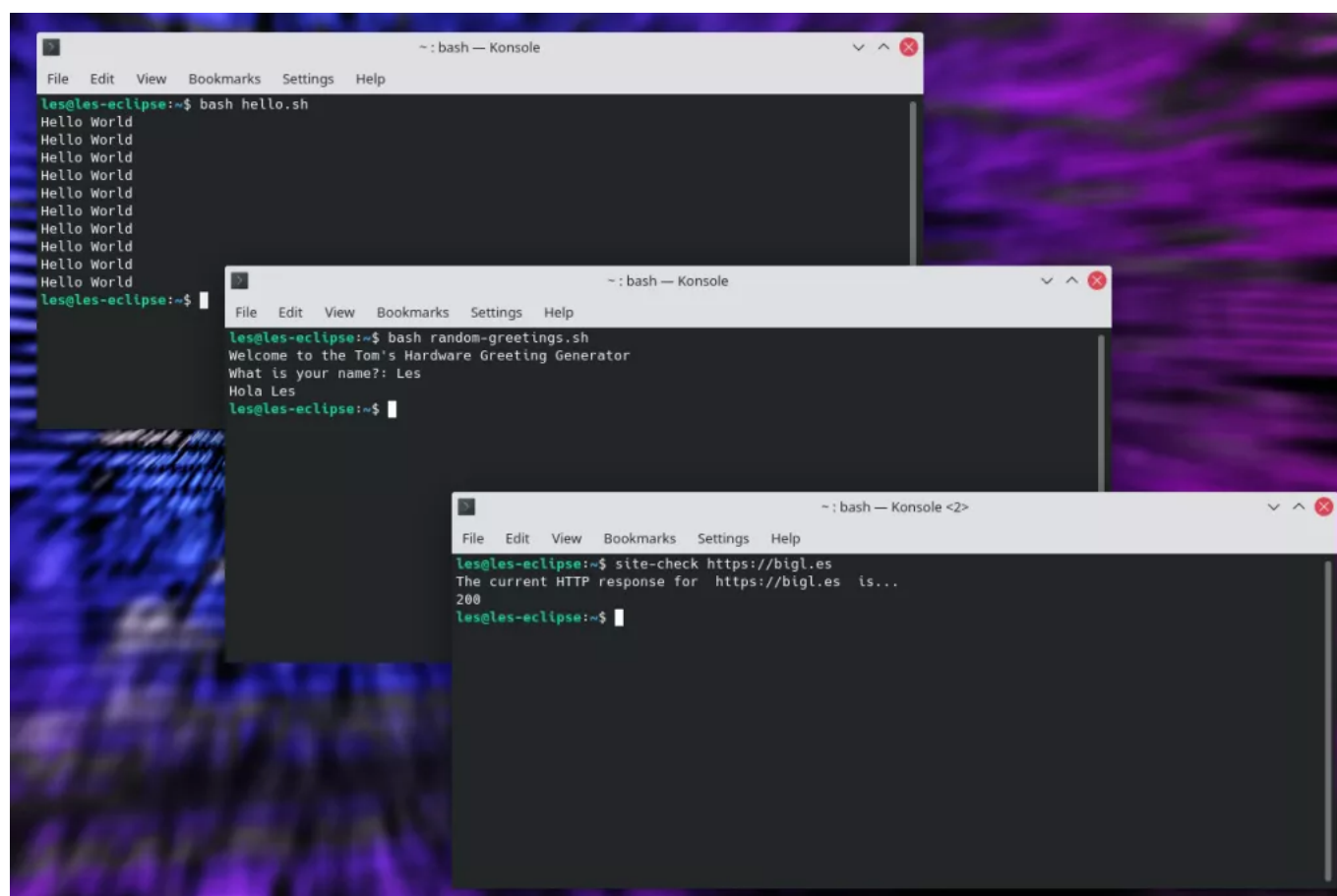
How To Write Bash Scripts in Linux

By [Les Pounder](#) published March 19, 2022

Automate your life with just a few lines of code



Comments (0)



(Image credit: Tom's Hardware)

Operating systems generally come with some form of scripting language which admins and power users can use to create custom tools. Unix and Linux are no different and come with many different options. The glue which holds many Linux projects together is made from Bash scripts. These simple to create, yet fiendishly powerful scripts can do everything from remotely backing up entire file systems to making an LED flash.

Bash scripts behave like those of other programming languages. The code runs line by line in a sequence. We can use conditional logic (selection) to alter the path that our code takes and the code can iterate and repeat the execution of the code until a condition is met or indefinitely.

Bash, the Bourne Again SHell is one of many different shells available to Unix and Linux users (Zsh, Ksh, Tcsh, Fish are others) and is commonly installed with many of the main distributions. We ran Kubuntu 21.10 when writing this tutorial, but the same code will also work on a Raspberry Pi (with Raspberry Pi OS) or almost any other Linux distribution.

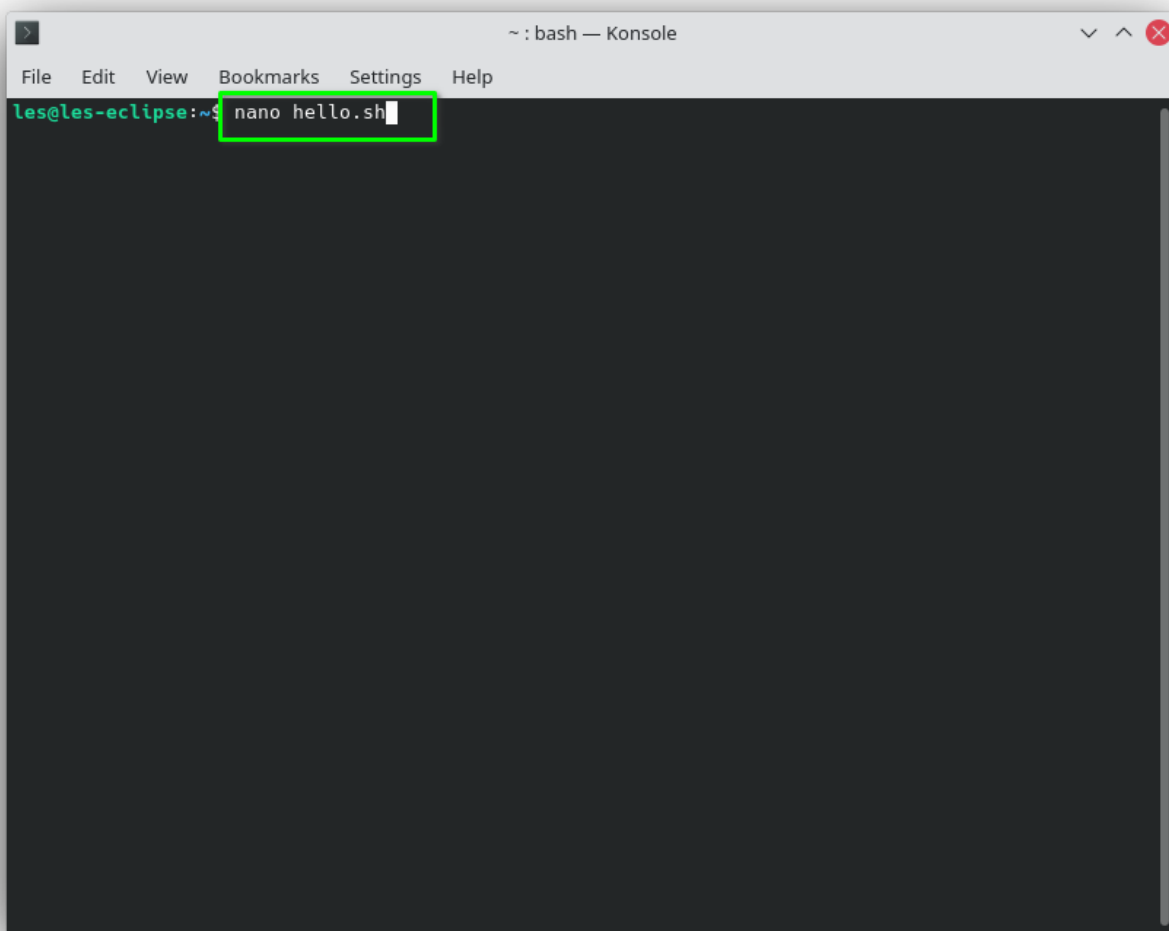
In this how-to we shall take our first steps into creating Bash scripts with three example projects. The first is the venerable “Hello World”, then we learn how to create interactive scripts, before finally creating a real script to check the status of a site / server. We then go the extra mile and learn how to make our scripts executable and available system-wide.

How to Write a “Hello World” Bash Script

We'll start simple with “Hello World”. This may seem trivial, but a hello world test is useful to understand the workflow of creating a Bash script, and to test the basic functionality. We'll be using the nano text editor, but you can also write your scripts in another terminal or GUI based text editor.

1. **Create a new file, `hello.sh` and open it with nano.**

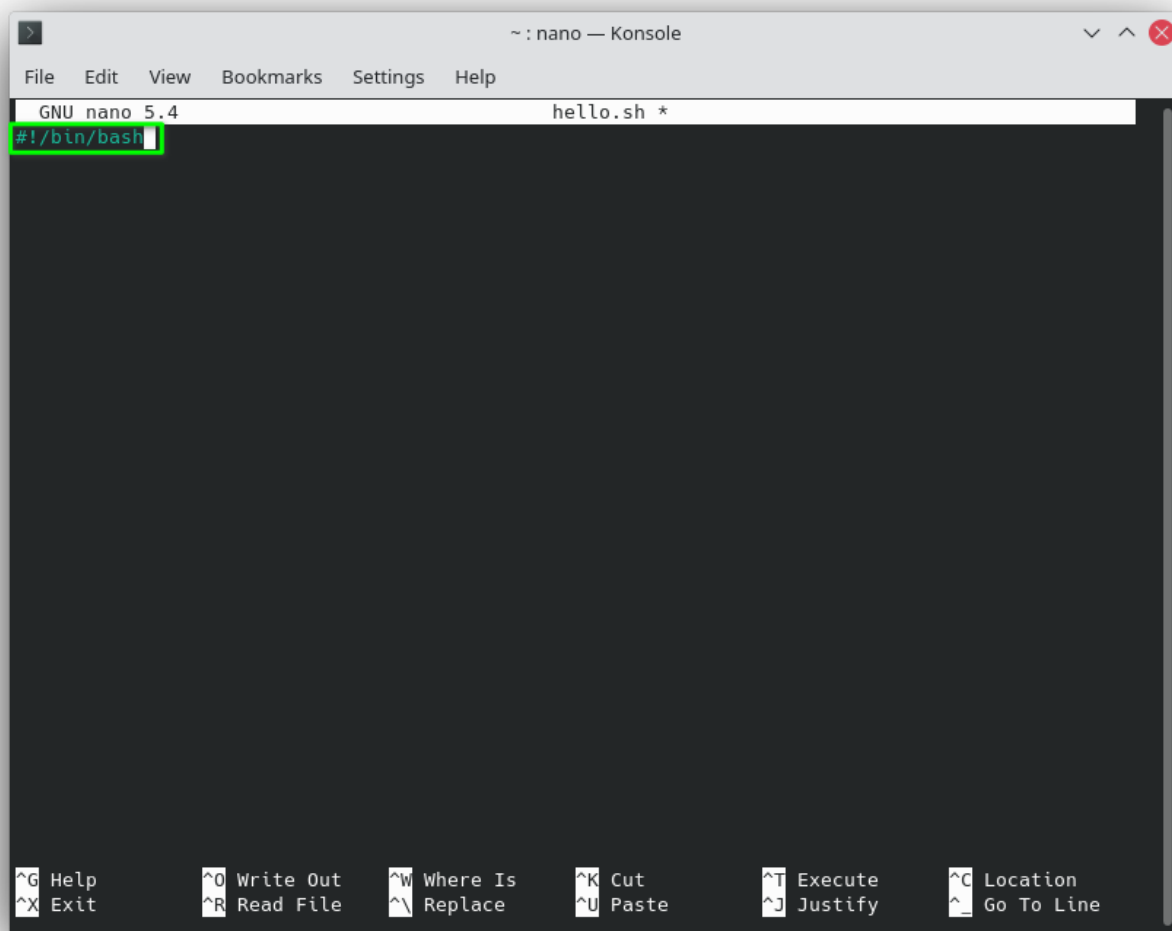
```
nano hello.sh
```



(Image credit: Tom's Hardware)

2. On the first line specify the interpreter to be used in the code. In this case it is Bash. The first part of the line, `#!/`, is called the “shebang” and it indicates the start of a script.

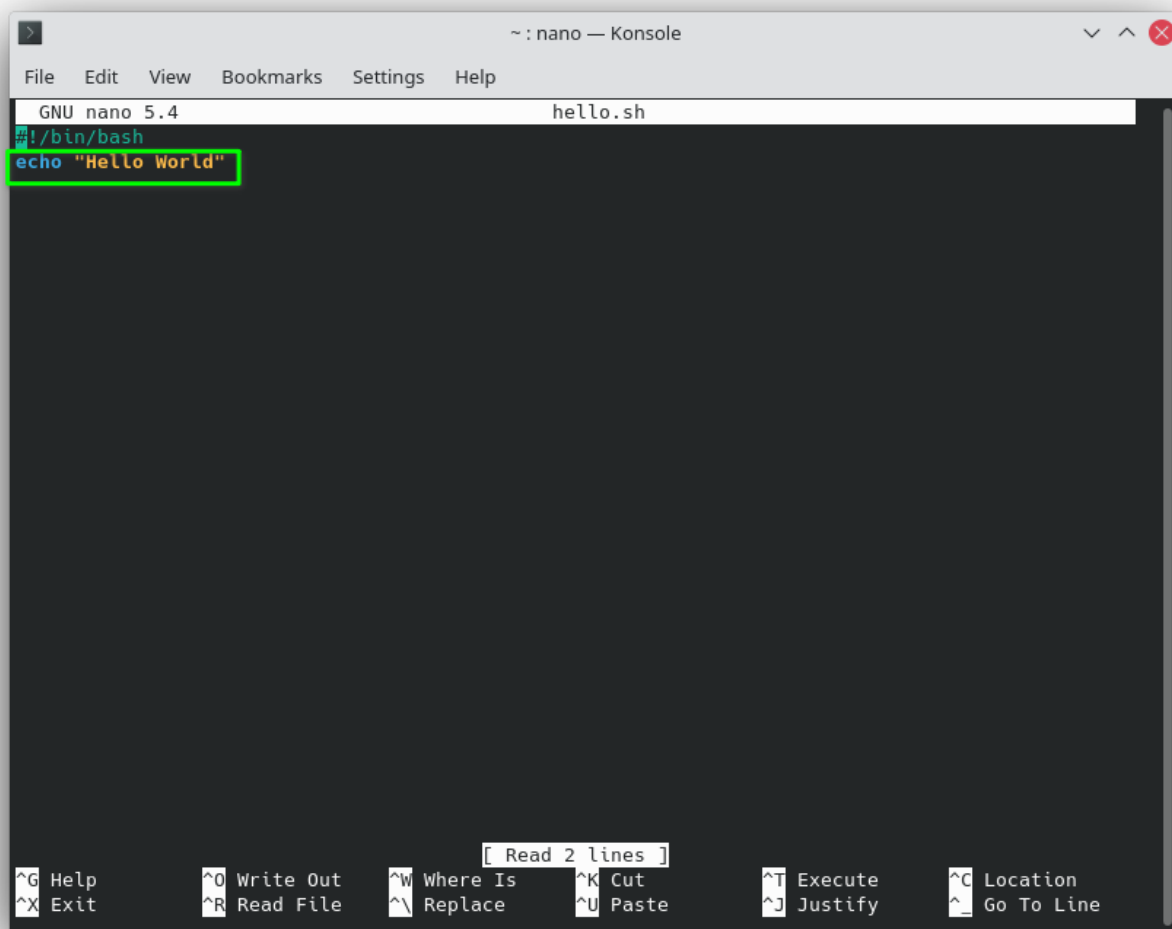
```
#!/bin/bash
```



(Image credit: Tom's Hardware)

3. On a new line **use *echo* to print a string** of text to the screen.

```
echo "Hello World"
```



```
~ : nano — Konsole
File Edit View Bookmarks Settings Help
GNU nano 5.4 hello.sh
#!/bin/bash
echo "Hello World"

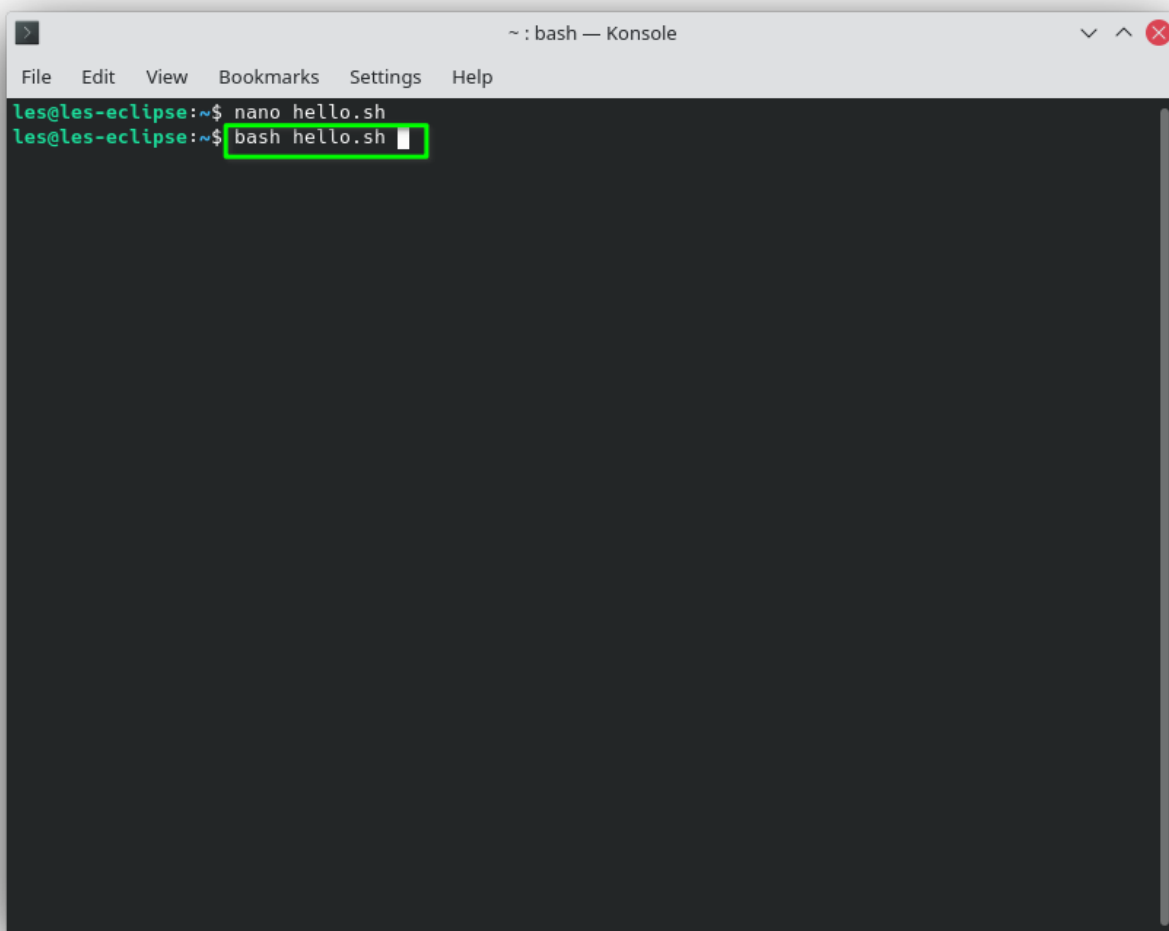
[ Read 2 lines ]
^G Help      ^O Write Out  ^W Where Is  ^K Cut       ^T Execute   ^C Location
^X Exit      ^R Read File  ^\ Replace   ^U Paste     ^J Justify   ^_ Go To Line
```

(Image credit: Tom's Hardware)

4. **Save the code** by pressing CTRL + X, then press Y and Enter.

5. **Run the code from the terminal.**

```
bash hello.sh
```

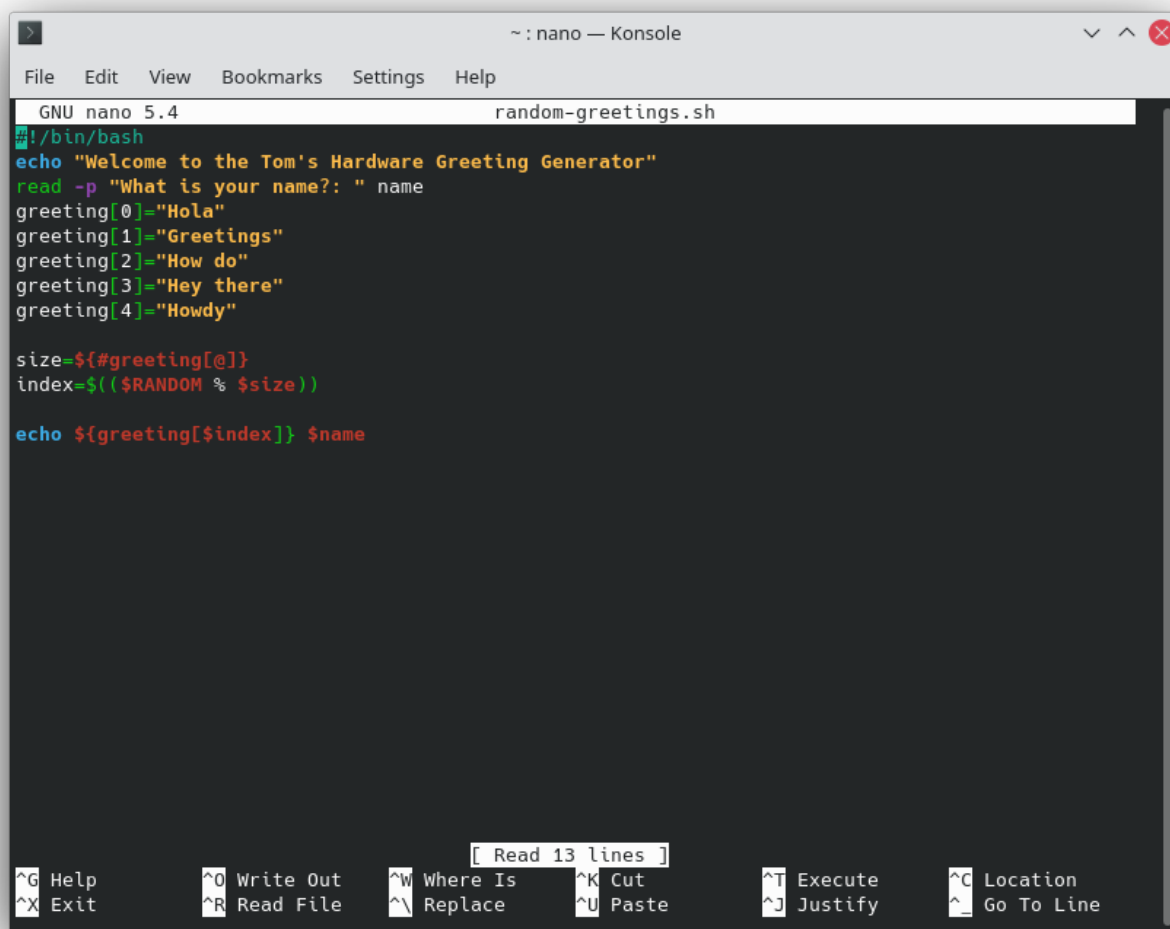
A terminal window titled '~ : bash — Konsole' with a menu bar (File, Edit, View, Bookmarks, Settings, Help). The prompt is 'les@les-eclipse:~\$'. The first command is 'nano hello.sh'. The second command is 'bash hello.sh', which is highlighted with a green box. The terminal background is dark gray.

```
les@les-eclipse:~$ nano hello.sh
les@les-eclipse:~$ bash hello.sh
```

(Image credit: Tom's Hardware)

The output of the command is a single line of “Hello World.” This proves that our script works and we can move on to something a little more interesting.

Capturing User Input in Bash Scripts



```
GNU nano 5.4 random-greetings.sh
#!/bin/bash
echo "Welcome to the Tom's Hardware Greeting Generator"
read -p "What is your name?: " name
greeting[0]="Hola"
greeting[1]="Greetings"
greeting[2]="How do"
greeting[3]="Hey there"
greeting[4]="Howdy"

size=${#greeting[@]}
index=$((RANDOM % $size))

echo ${greeting[$index]} $name
```

(Image credit: Tom's Hardware)

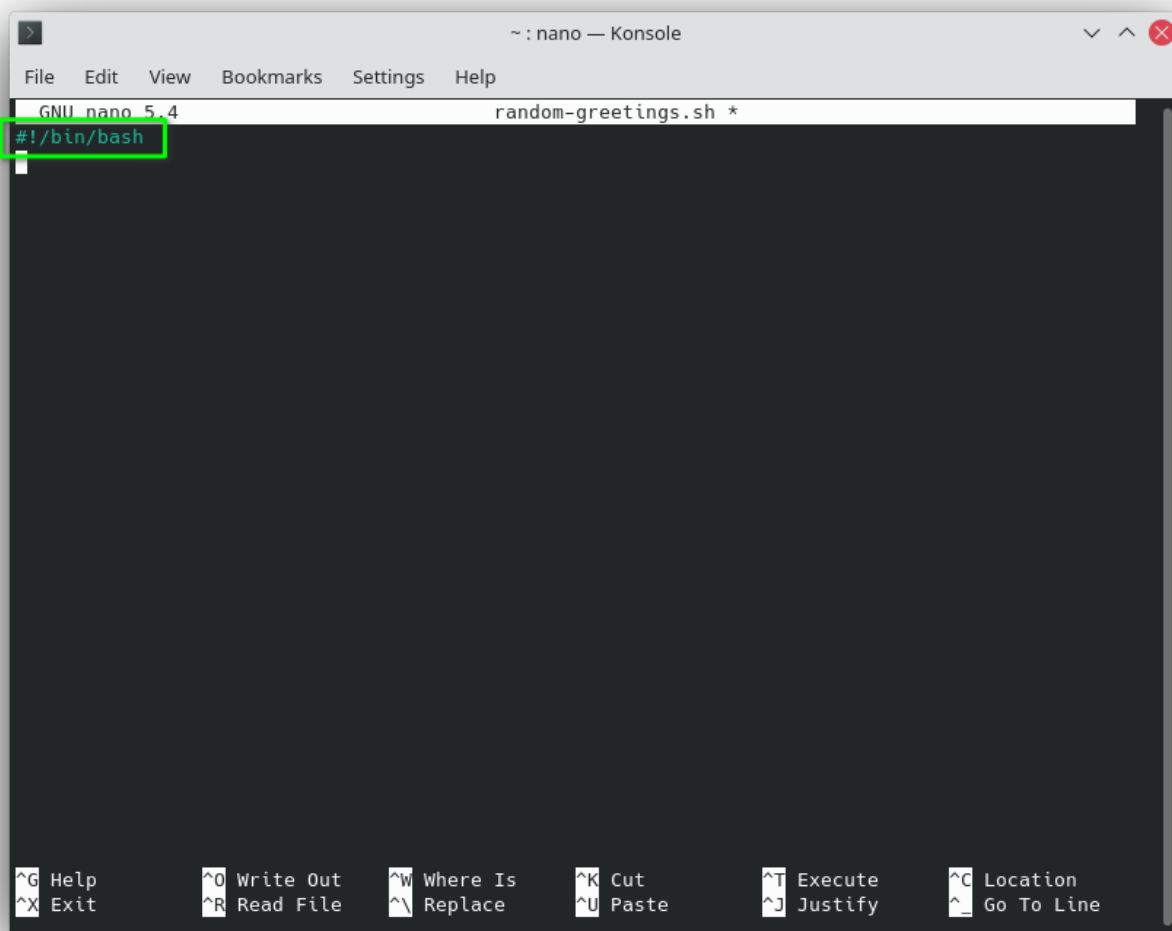
Bash is a fully formed language, and with it we can create interactive tools and applications. In this example we will create a greeting generator which will capture the username and then choose a random greeting from an array of greetings.

1. Create a new file, `random-greetings.sh` using *nano*.

```
nano random-greetings.sh
```

2. On the first line specify the interpreter to be used in the code. In this case it is **Bash**.

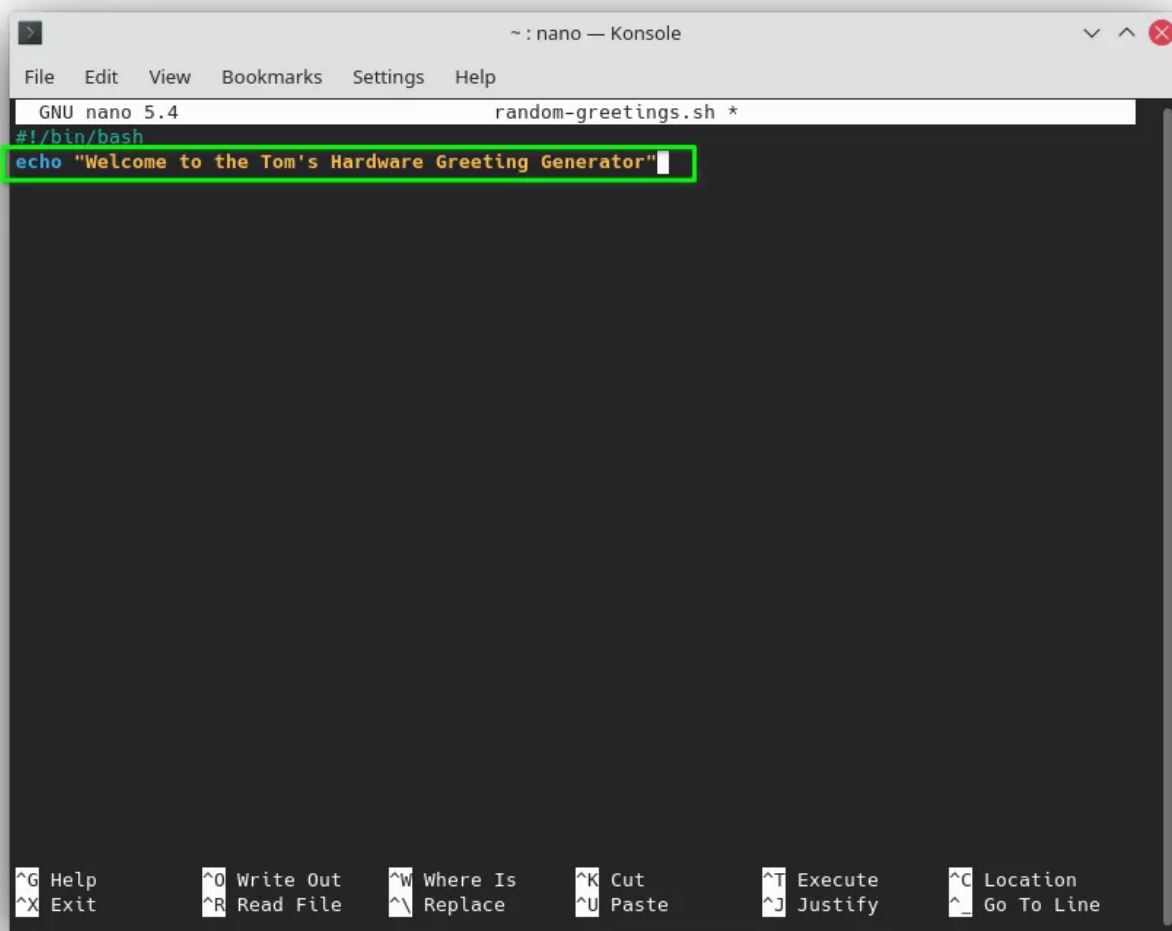
```
#!/bin/bash
```



(Image credit: Tom's Hardware)

3. Add an echo to print a message to the user.

```
echo "Welcome to the Tom's Hardware Greeting Generator"
```


A screenshot of a terminal window titled '~ : nano — Konsole'. The window shows the nano 5.4 editor editing a file named 'random-greetings.sh'. The script content is:

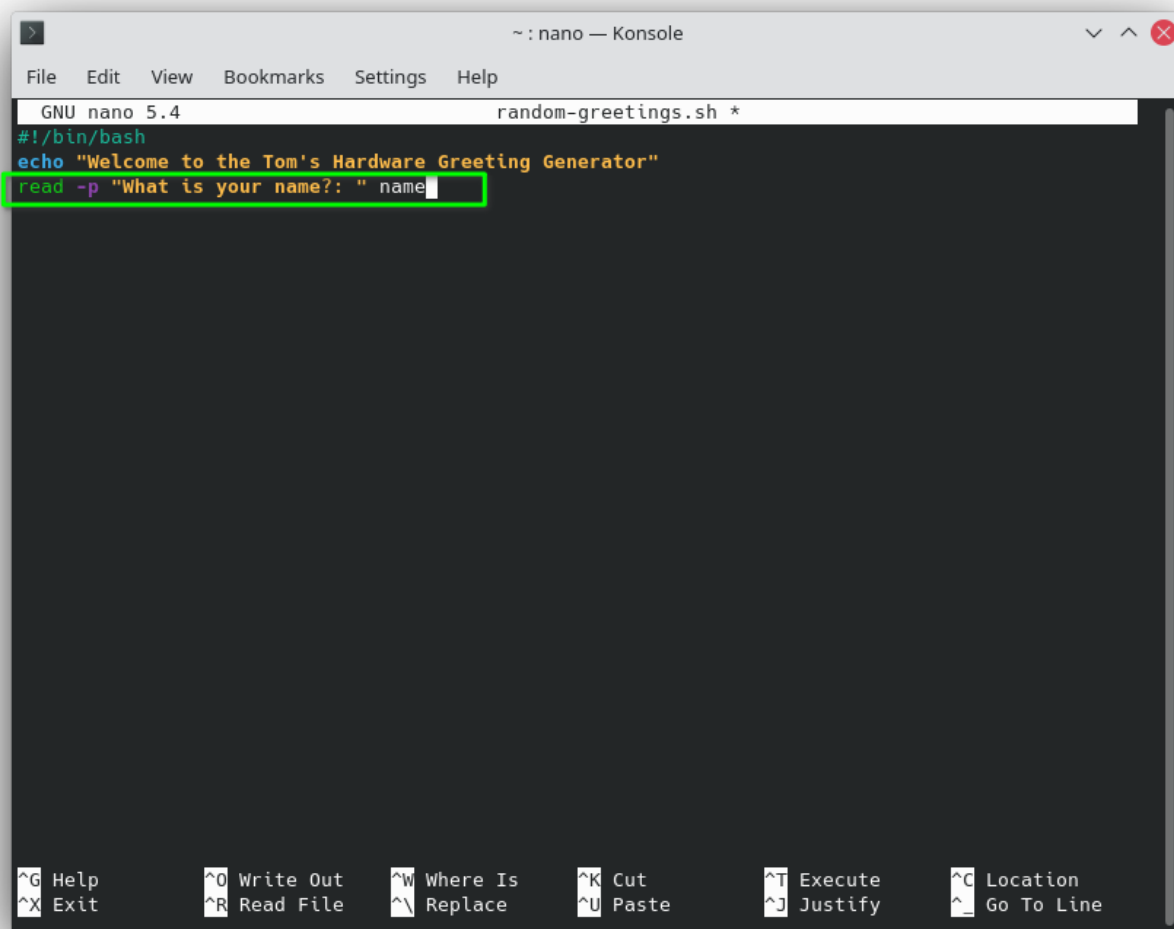
```
#!/bin/bash
echo "Welcome to the Tom's Hardware Greeting Generator"
```

 The second line is highlighted with a green box. The bottom status bar shows various keyboard shortcuts: ^G Help, ^X Exit, ^O Write Out, ^R Read File, ^W Where Is, ^\ Replace, ^K Cut, ^U Paste, ^T Execute, ^J Justify, ^C Location, and ^_ Go To Line.

(Image credit: Tom's Hardware)

4. Read the user's keyboard input, using a prompt to ask a question to the user. The user's input is saved to a variable *name*.

```
read -p "What is your name?: " name
```



The image shows a terminal window titled '~ : nano — Konsole'. The window contains a nano editor interface with a menu bar (File, Edit, View, Bookmarks, Settings, Help) and a status bar at the bottom. The status bar displays various keyboard shortcuts: ^G Help, ^X Exit, ^O Write Out, ^R Read File, ^W Where Is, ^\ Replace, ^K Cut, ^U Paste, ^T Execute, ^J Justify, ^C Location, and ^_ Go To Line. The editor shows the following code:

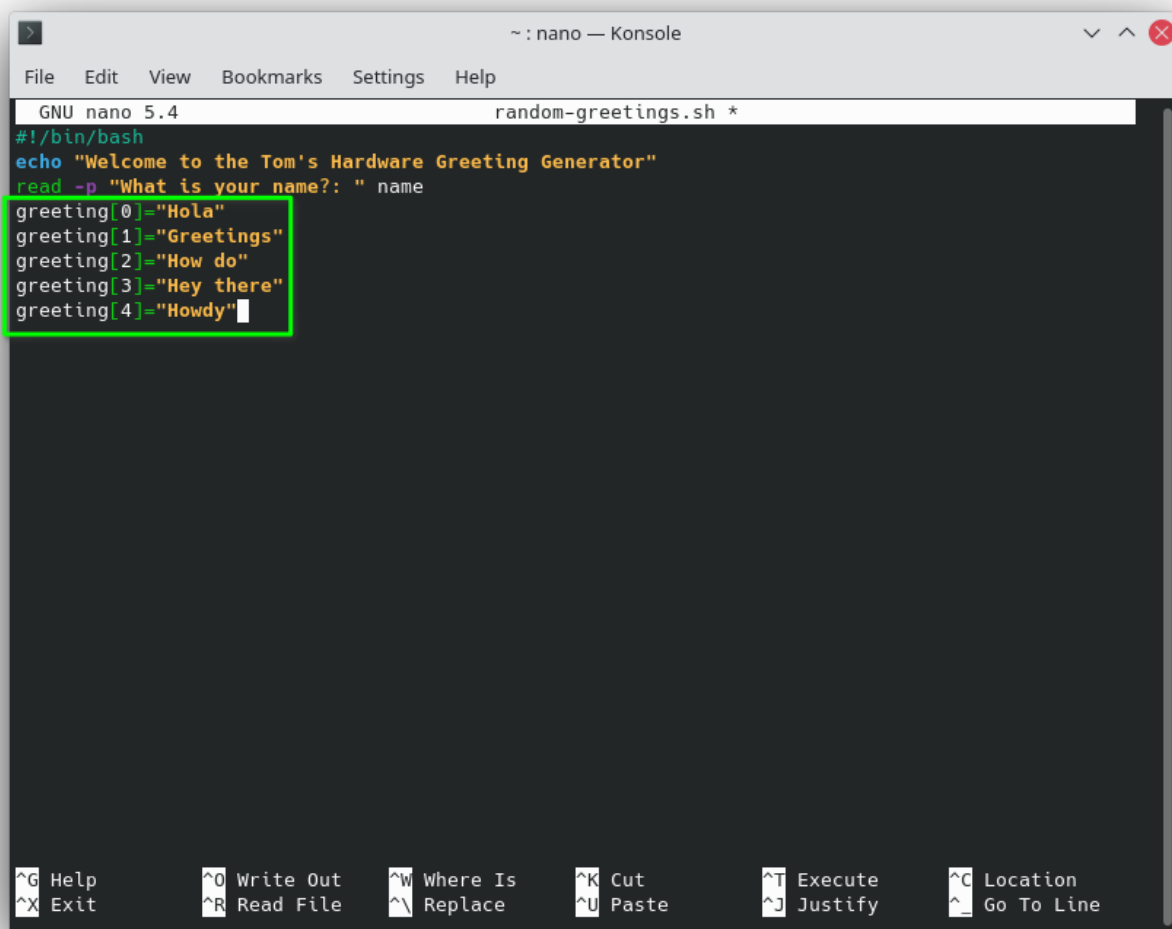
```
GNU nano 5.4 random-greetings.sh *
#!/bin/bash
echo "Welcome to the Tom's Hardware Greeting Generator"
read -p "What is your name?: " name
```

The line `read -p "What is your name?: " name` is highlighted with a green box.

(Image credit: Tom's Hardware)

5. Create an array *greeting* which will store five greetings. An array is a list of items stored with an index number. Using the name of the array and the index number we can pull items (greetings) from the array.

```
greeting[0]="Hola"
greeting[1]="Greetings"
greeting[2]="How do"
greeting[3]="Hey there"
greeting[4]="Howdy"
```



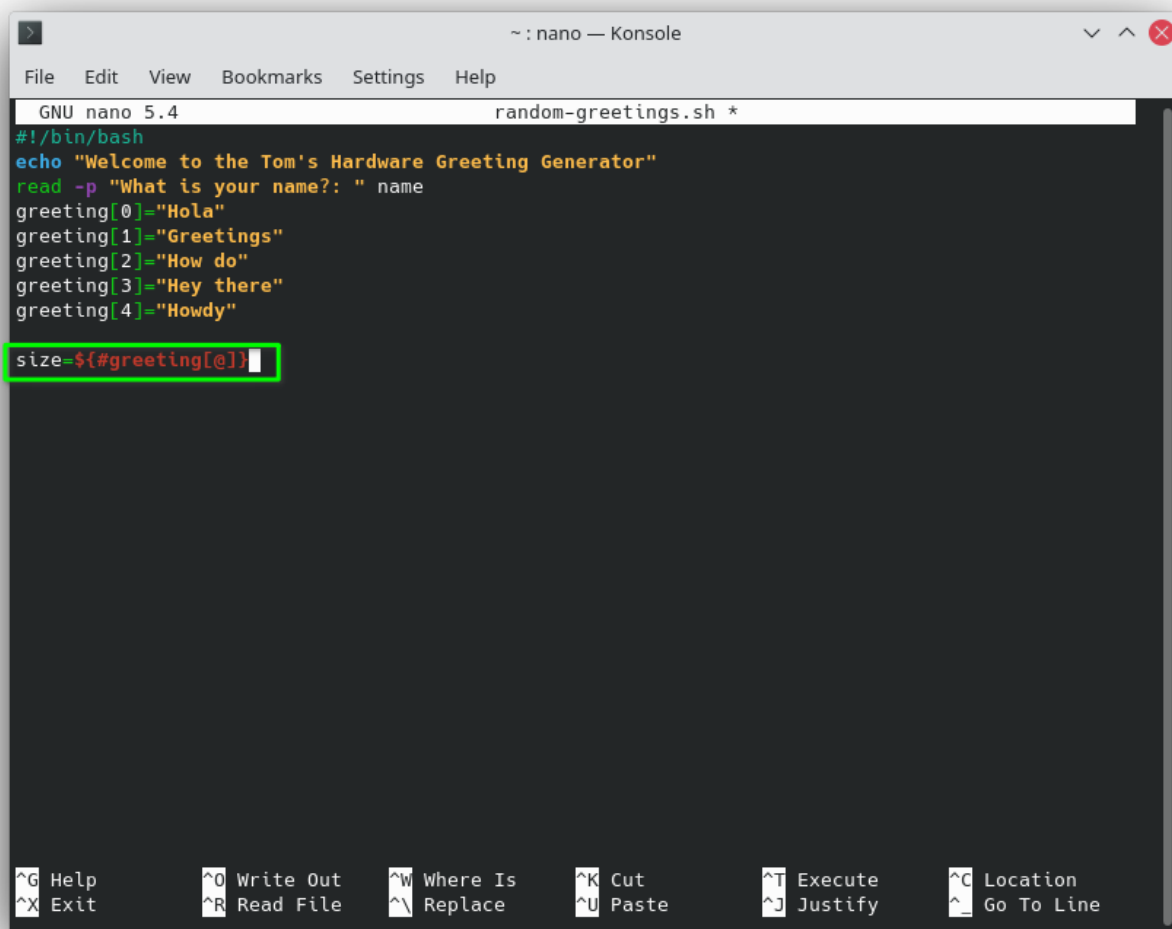
```
GNU nano 5.4 random-greetings.sh *
#!/bin/bash
echo "Welcome to the Tom's Hardware Greeting Generator"
read -p "What is your name?: " name
greeting[0]="Hola"
greeting[1]="Greetings"
greeting[2]="How do"
greeting[3]="Hey there"
greeting[4]="Howdy"

```

(Image credit: Tom's Hardware)

6. Create a variable *size* which will store the number of items in the array.

```
size=${#greeting[@]}
```



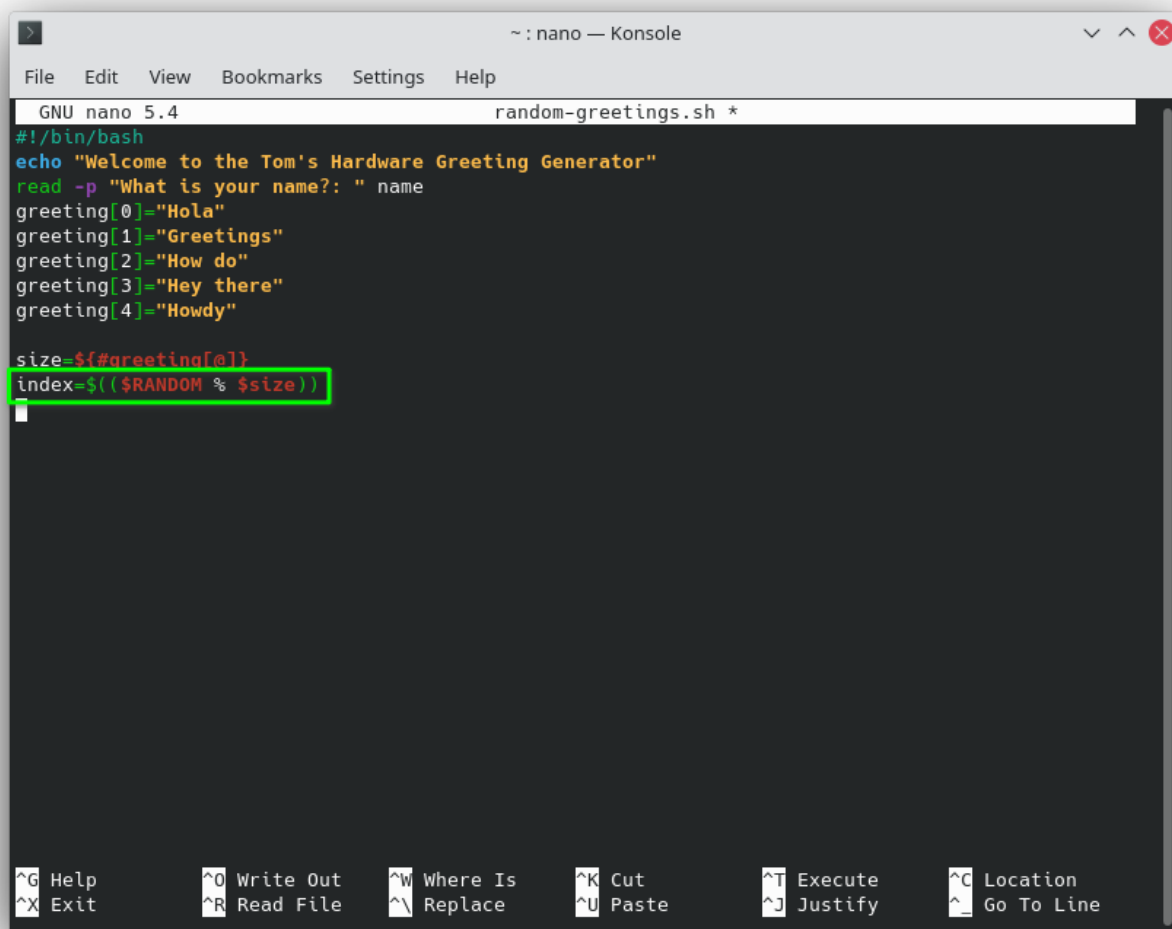
```
~ : nano — Konsole
File Edit View Bookmarks Settings Help
GNU nano 5.4 random-greetings.sh *
#!/bin/bash
echo "Welcome to the Tom's Hardware Greeting Generator"
read -p "What is your name?: " name
greeting[0]="Hola"
greeting[1]="Greetings"
greeting[2]="How do"
greeting[3]="Hey there"
greeting[4]="Howdy"
size=${#greeting[@]}

```

(Image credit: Tom's Hardware)

7. Create a variable *index* and store a randomly chosen number from zero to the size of the array, in this case five items.

```
index=$(( $RANDOM % $size ))
```



The screenshot shows a terminal window with the nano text editor. The title bar reads '~ : nano — Konsole'. The menu bar includes File, Edit, View, Bookmarks, Settings, and Help. The status bar at the bottom shows various keyboard shortcuts: ^G Help, ^X Exit, ^O Write Out, ^R Read File, ^W Where Is, ^\ Replace, ^K Cut, ^U Paste, ^T Execute, ^J Justify, ^C Location, and ^_ Go To Line. The script content is as follows:

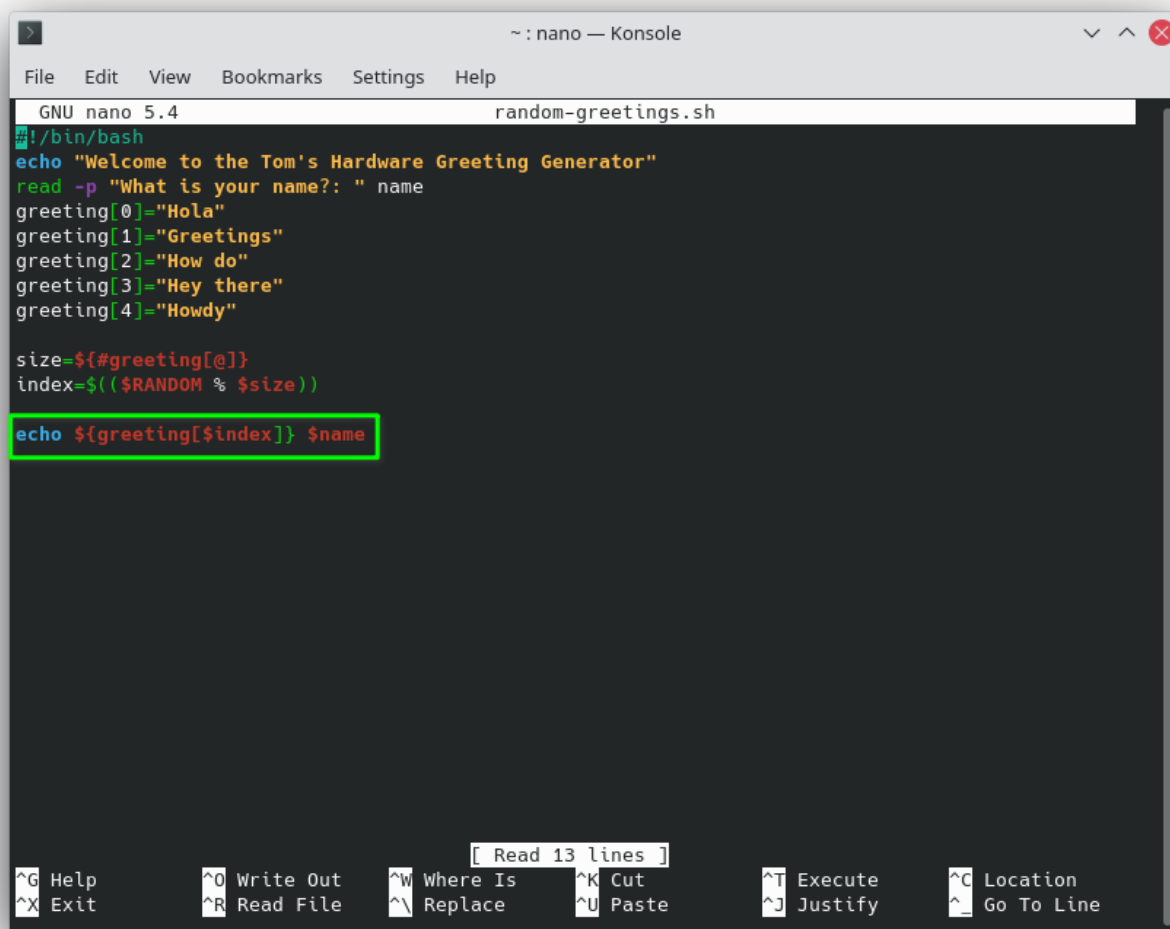
```
GNU nano 5.4 random-greetings.sh *
#!/bin/bash
echo "Welcome to the Tom's Hardware Greeting Generator"
read -p "What is your name?: " name
greeting[0]="Hola"
greeting[1]="Greetings"
greeting[2]="How do"
greeting[3]="Hey there"
greeting[4]="Howdy"

size=${#greeting[@]}
index=$((RANDOM % $size))
```

(Image credit: Tom's Hardware)

8. Use echo to print the randomly chosen greeting along with the user's name.

```
echo ${greeting[$index]} $name
```



The screenshot shows a terminal window titled '~ : nano — Konsole'. The nano editor is open with a file named 'random-greetings.sh'. The script content is as follows:

```
GNU nano 5.4 random-greetings.sh
#!/bin/bash
echo "Welcome to the Tom's Hardware Greeting Generator"
read -p "What is your name?: " name
greeting[0]="Hola"
greeting[1]="Greetings"
greeting[2]="How do"
greeting[3]="Hey there"
greeting[4]="Howdy"

size=${#greeting[@]}
index=$((RANDOM % $size))
echo ${greeting[$index]} $name
```

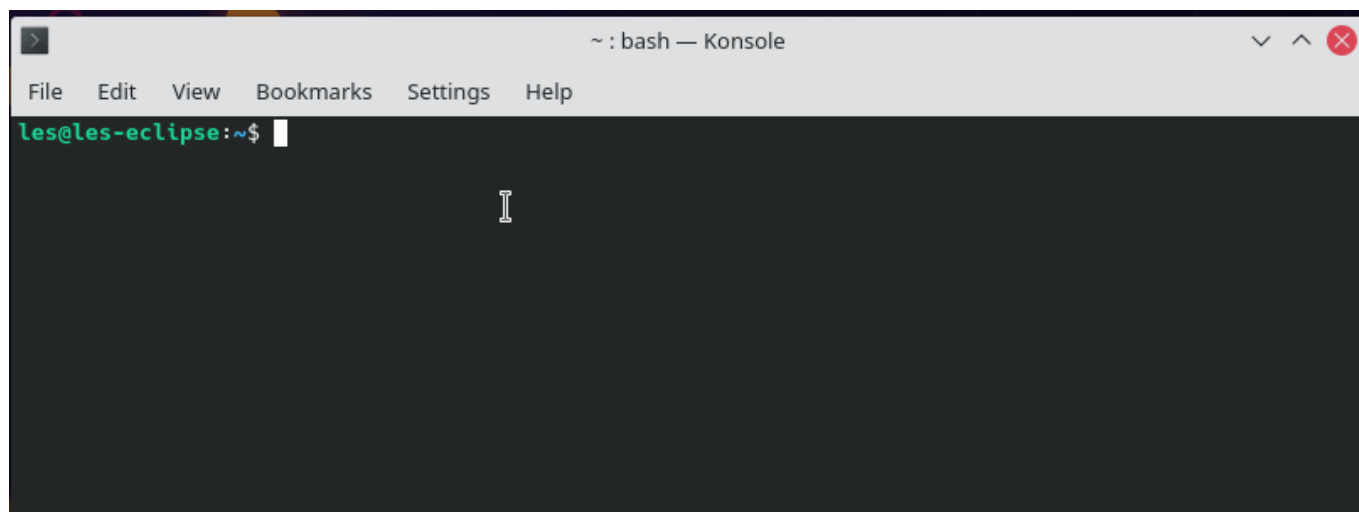
The last line of the script, `echo ${greeting[$index]} $name`, is highlighted with a green box. At the bottom of the window, there is a status bar with various keyboard shortcuts: `^G Help`, `^O Write Out`, `^W Where Is`, `^K Cut`, `^T Execute`, `^C Location`, `^X Exit`, `^R Read File`, `^_ Replace`, `^U Paste`, `^J Justify`, and `^_ Go To Line`. A message '[Read 13 lines]' is also visible in the status bar.

(Image credit: Tom's Hardware)

9. **Save the code by pressing CTRL + X, then press Y and Enter.**

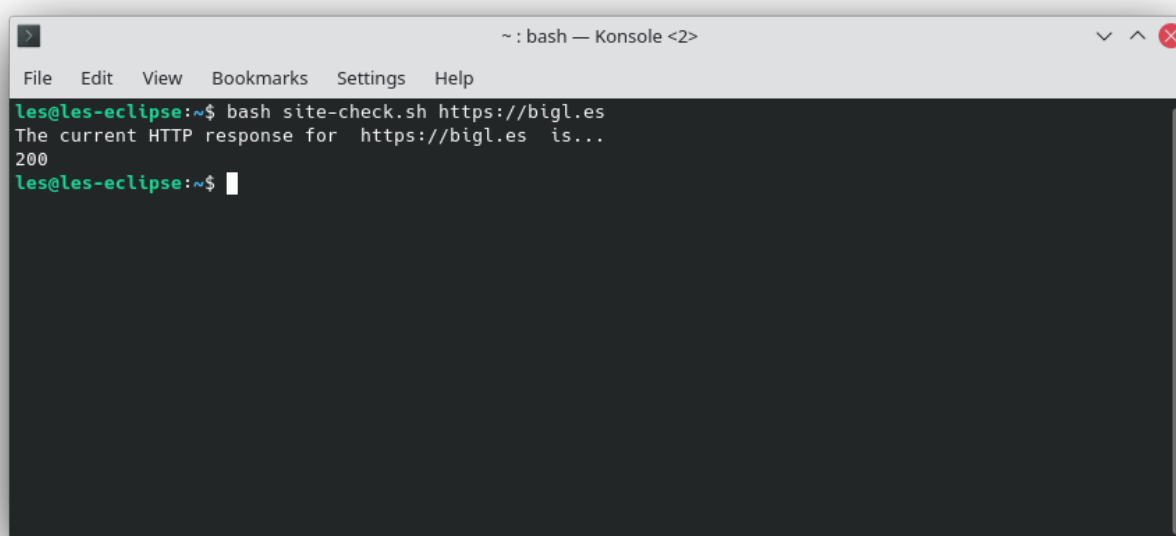
10. **Run the code from the terminal. Follow the instructions to see the greeting.**

```
bash random-greetings.sh
```



(Image credit: Tom's Hardware)

Using Arguments with Bash Scripts



(Image credit: Tom's Hardware)

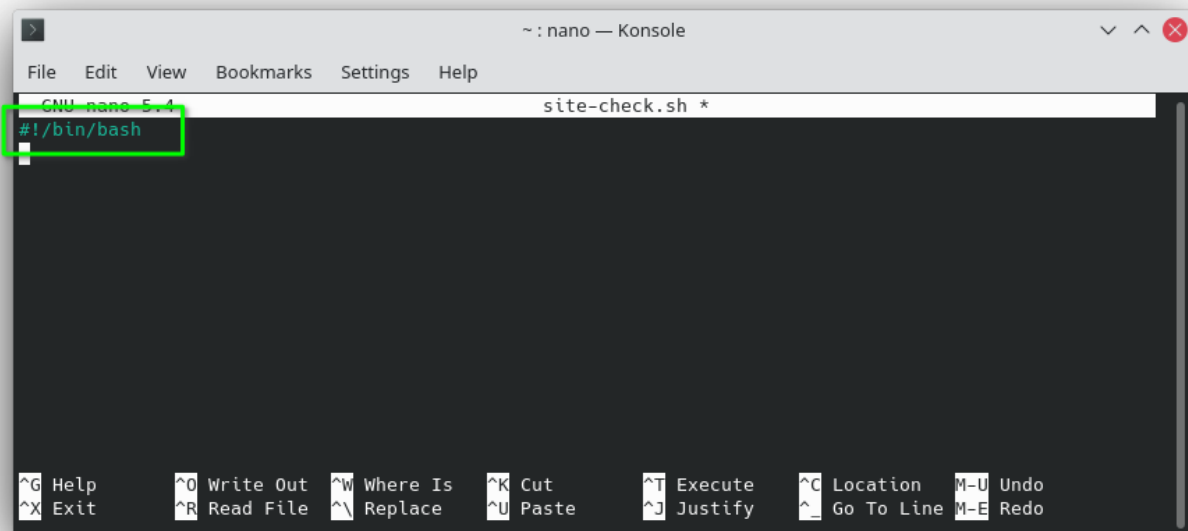
The beauty of the terminal is that we can pass arguments, extra instructions or parameters, to a command. In this example we will write a bash script to check the status of a URL, passed as an argument.

1. Create a new file *site-check.sh* and open *nano*.

```
nano site-check.sh
```

2. On the first line specify the interpreter to be used in the code. In this case it is Bash.

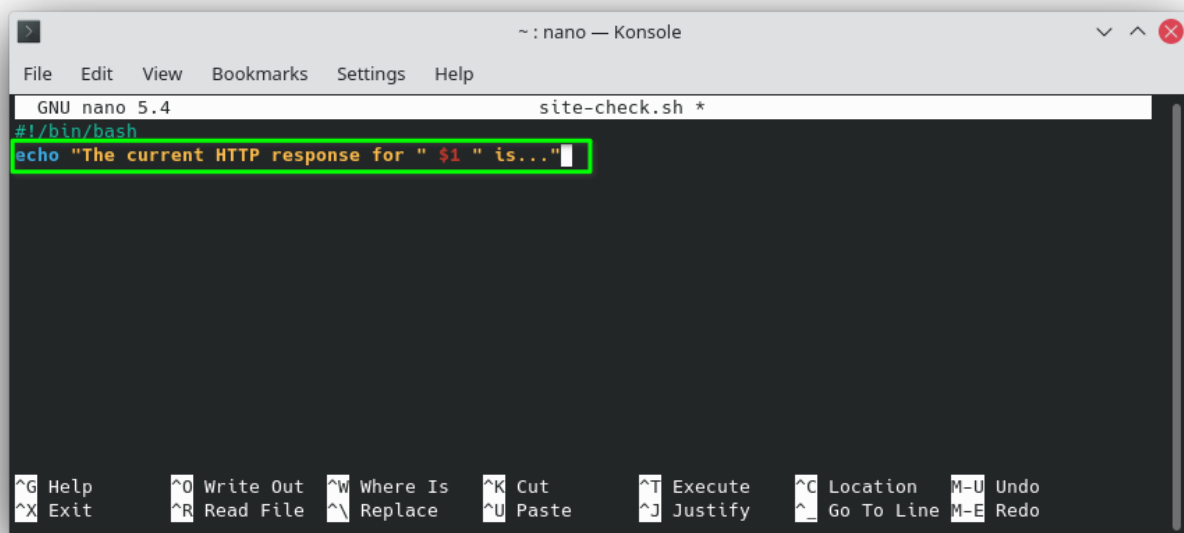
```
#!/bin/bash
```



(Image credit: Tom's Hardware)

3. Using **echo**, write a message to the user to advise them on the current **HTTP response code for the URL**. The value of `$1` is the URL that we specified as an argument.

```
echo "The current HTTP response for " $1 " is..."
```


A screenshot of a terminal window titled '~ : nano — Konsole'. The window shows the GNU nano 5.4 editor editing a file named 'site-check.sh'. The first line of the script is '#!/bin/bash' and the second line is 'echo "The current HTTP response for " \$1 " is..."'. The second line is highlighted with a green box. The bottom of the window shows a status bar with various keyboard shortcuts like ^G Help, ^O Write Out, ^W Where Is, ^K Cut, ^T Execute, ^C Location, M-U Undo, ^X Exit, ^R Read File, ^\ Replace, ^U Paste, ^J Justify, ^_ Go To Line, and M-E Redo.

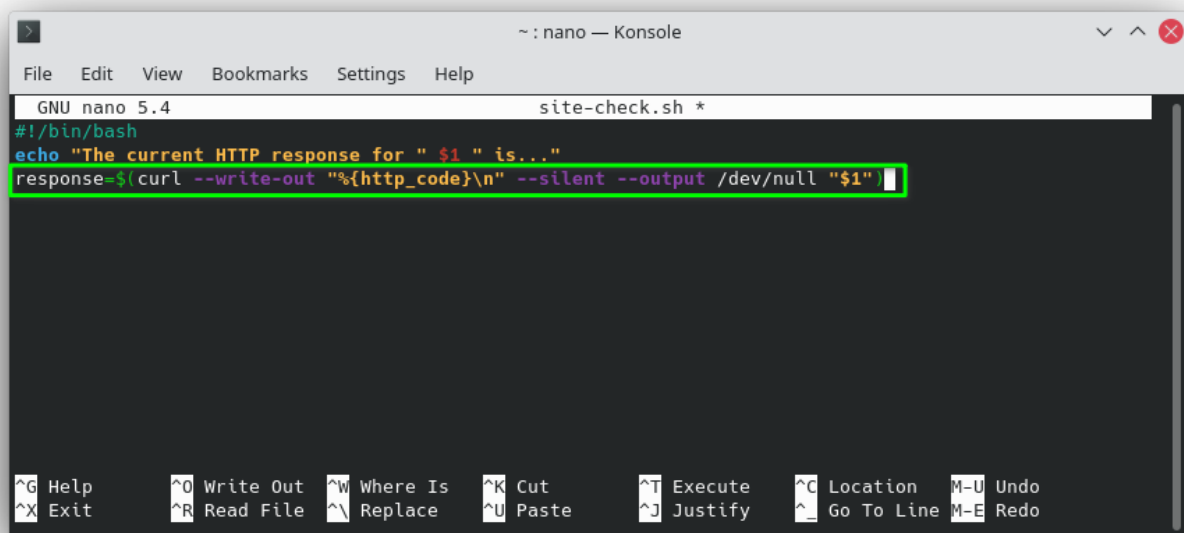
```
~ : nano — Konsole
File Edit View Bookmarks Settings Help
GNU nano 5.4 site-check.sh *
#!/bin/bash
echo "The current HTTP response for " $1 " is..."
^G Help      ^O Write Out ^W Where Is  ^K Cut       ^T Execute  ^C Location  M-U Undo
^X Exit      ^R Read File ^\ Replace   ^U Paste     ^J Justify  ^_ Go To Line M-E Redo
```

(Image credit: Tom's Hardware)

4. Create a variable, *response* to store the output of the *curl* command.

Using the *-write-out* argument we specify that we want to see the HTTP response code. Another argument, *-silent* will ensure that we do not see all of the output from the command spill onto the screen. We then send any residual output such as errors to */dev/null*, essentially a black hole for data. Lastly we use the *\$1* argument (our URL to check) to complete the command.

```
response=$(curl --write-out "%{http_code}\n" --silent --output /dev/null "$1")
```

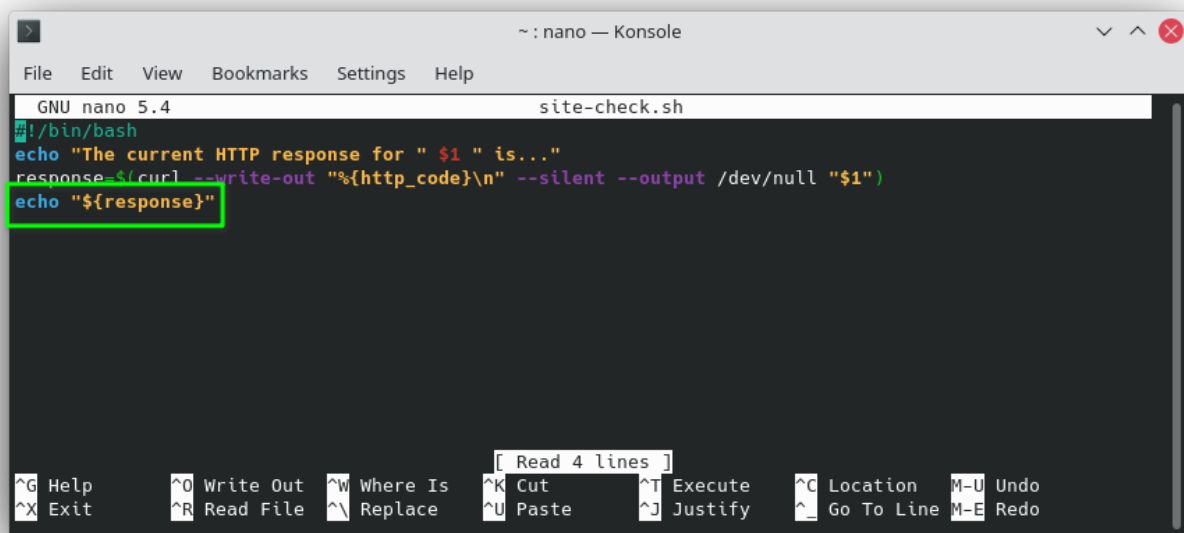


```
~ : nano — Konsole
File Edit View Bookmarks Settings Help
GNU nano 5.4 site-check.sh *
#!/bin/bash
echo "The current HTTP response for " $1 " is..."
response=$(curl --write-out "%{http_code}\n" --silent --output /dev/null "$1")
```

(Image credit: Tom's Hardware)

5. Use *echo* to print the HTTP response code to the terminal.

```
echo "${response}"
```



```
~ : nano — Konsole
File Edit View Bookmarks Settings Help
GNU nano 5.4 site-check.sh
#!/bin/bash
echo "The current HTTP response for " $1 " is..."
response=$(curl --write-out "%{http_code}\n" --silent --output /dev/null "$1")
echo "${response}"
```

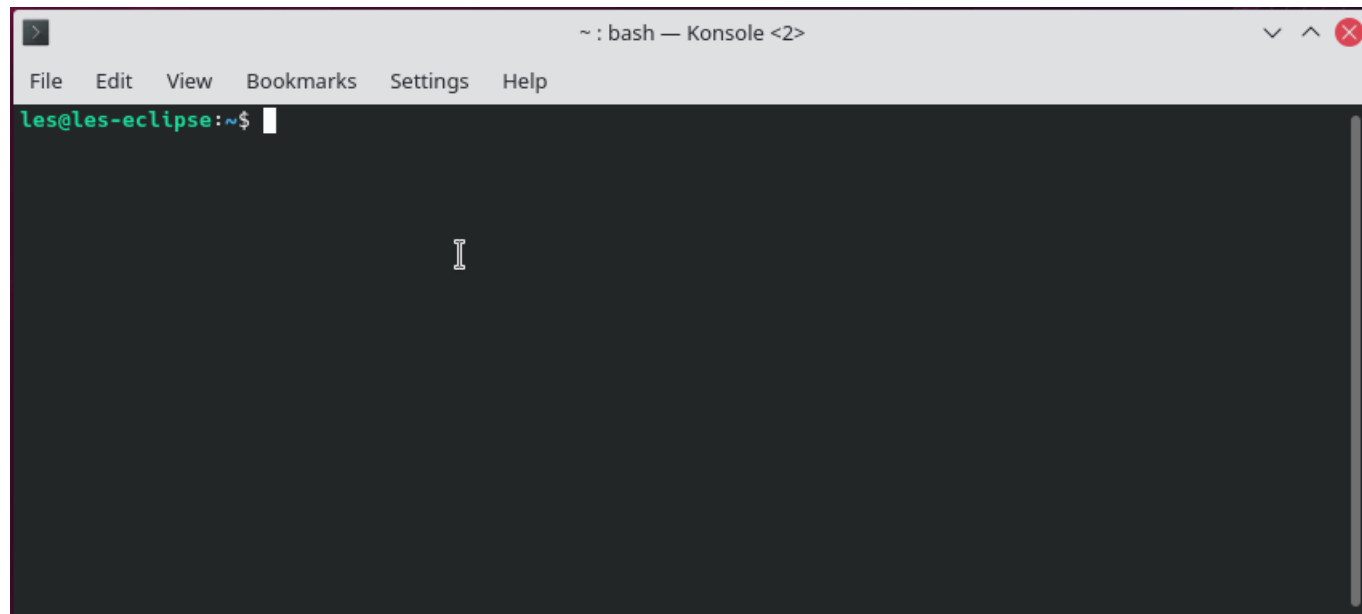
(Image credit: Tom's Hardware)

6. Save the code by pressing CTRL + X, then press Y and Enter.

7. Run the code from the terminal; remember to include a full domain name

to check.

```
bash site-check.sh https://tomshardware.com
```



(Image credit: Tom's Hardware)

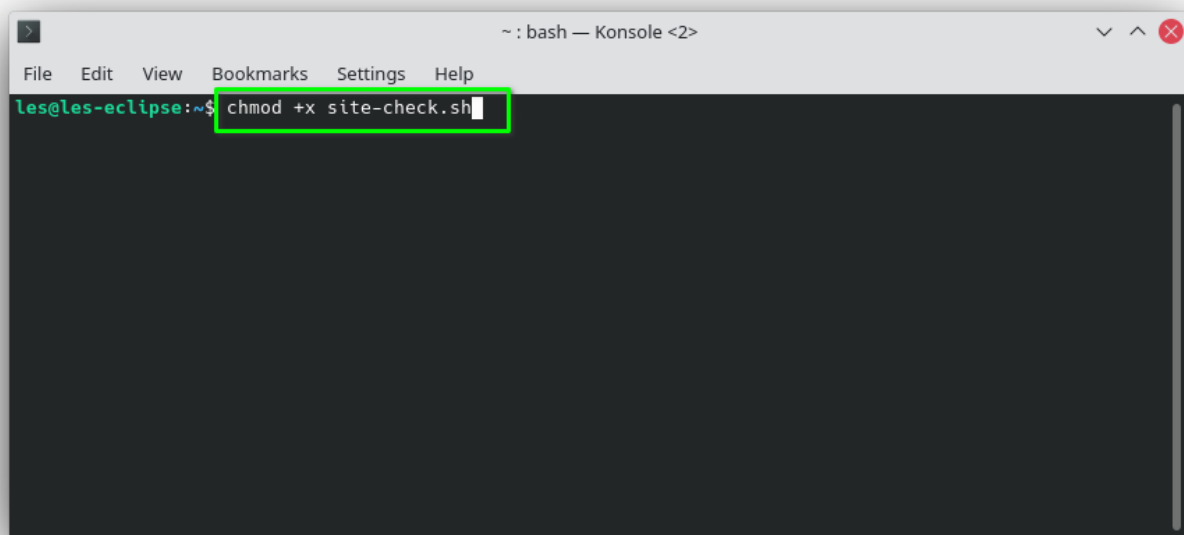
Creating a system-wide executable Bash Script

Right now our scripts work well, but they are not available system-wide as an executable file. Let's change that. For this final part we shall convert the site-check.sh script into an application that we can use from any location on the drive.

First we need to make the file executable, then we can copy the file to `/usr/bin` to create an executable available anywhere.

1. **Open a terminal window and navigate to site-check.sh.**
2. **Use `chmod` to set the file as executable. The `+x` argument specifies that the file should be executable.**

```
chmod +x site-check.sh
```



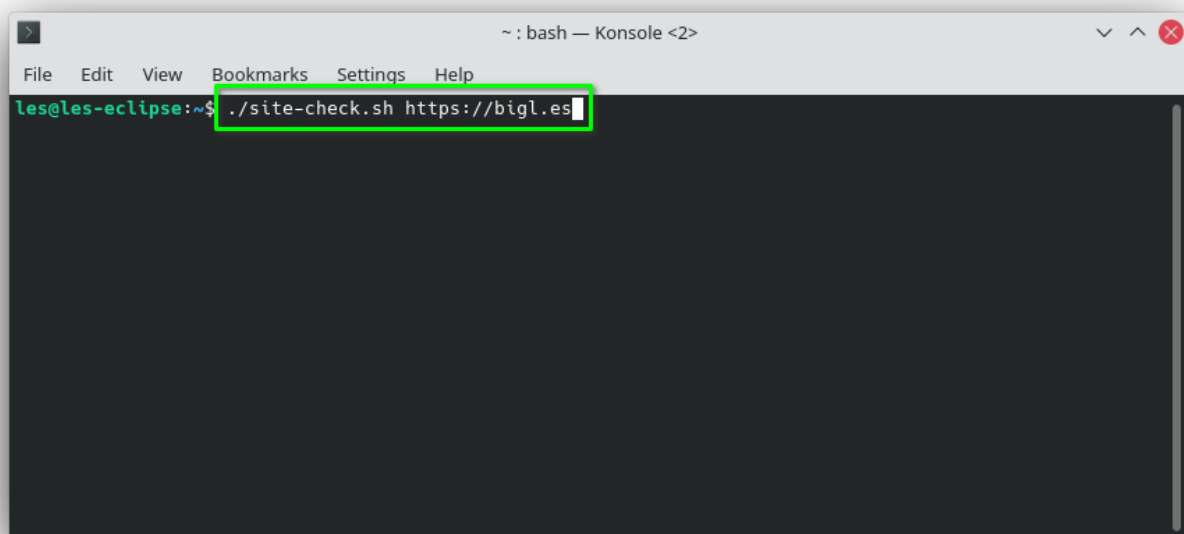
```
les@les-eclipse:~$ chmod +x site-check.sh
```

A terminal window titled '~ : bash — Konsole <2>' with a menu bar (File, Edit, View, Bookmarks, Settings, Help). The prompt is 'les@les-eclipse:~\$' and the command 'chmod +x site-check.sh' is entered, highlighted by a green box.

(Image credit: Tom's Hardware)

3. Run the script. Using `./` we instruct the terminal to run an executable file in the current directory.

```
./site-check.sh https://bigl.es
```



```
les@les-eclipse:~$ ./site-check.sh https://bigl.es
```

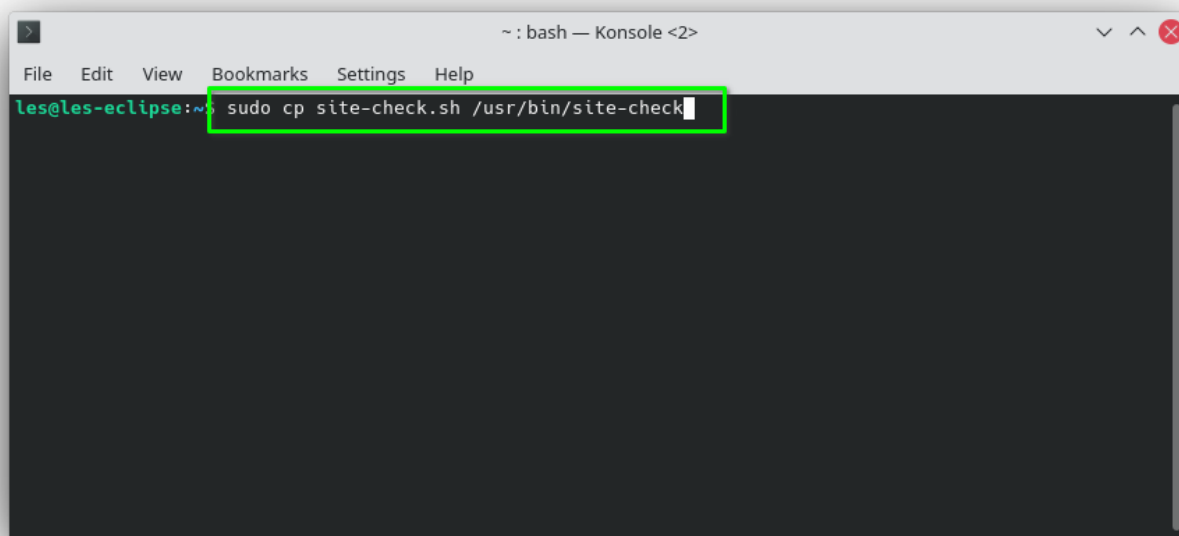
A terminal window titled '~ : bash — Konsole <2>' with a menu bar (File, Edit, View, Bookmarks, Settings, Help). The prompt is 'les@les-eclipse:~\$' and the command './site-check.sh https://bigl.es' is entered, highlighted by a green box.

(Image credit: Tom's Hardware)

4. Copy `site-cehck.sh` to the `/usr/bin/` directory, and change the target

filename to remove the .sh extension. Note that we will need to use `sudo` as the `/usr/bin` directory is not owned by our user. Take a look at our [tutorial on user, file and folder permissions for more information](#).

```
sudo cp site-check.sh /usr/bin/site-check
```



(Image credit: Tom's Hardware)

5. Run ***site-check*** to test that the command works as expected. Remember to include a URL to check.



(Image credit: Tom's Hardware)

**Les Pounder**

Les Pounder is an associate editor at Tom's Hardware. He is a creative technologist and for seven years has created projects to educate and inspire minds both young and old. He has worked with the Raspberry Pi Foundation to write and deliver their teacher training program "Picademy".

MORE ABOUT...**LATEST**

How to Check Your CPU Temperature ►

AMD Zen 4 Ryzen 7000 Specs, Release Date Window, Benchmarks, and More ►

SEE MORE LATEST ►

TOPICS

[LINUX](#)[OPERATING SYSTEMS](#)

 **SEE ALL COMMENTS (0)**

NO COMMENTS YET

COMMENT FROM THE FORUMS ►

MOST POPULAR

How to Download a Windows 11 ISO File and Do a Clean Install

By [Avram Piltch](#) 28 days ago

How to Find a Windows 10 or 11 Product Key

By [Avram Piltch](#) about 1 month ago

How to Overclock Your Graphics Card

By [Jarred Walton](#) about 1 month ago



How to Install Windows 11 22H2 Update Right Now

By [Brandon Hill](#) about 1 month ago

SIGN ME UP

MOST SHARED



How to Use Nohup to Run Linux Scripts Unattended

By [Les Pounder](#) about 2 months ago

How To Use a Multimeter to Measure Voltage, Current and More

By [Jo Hinchliffe](#) about 2 months ago

2 AMD Gaming Chairs Are Now Available For \$579

3 Intel Kills Optane Memory Business Entirely

4 Keyboard Shortage Unfolds in Russia

5 BMW's 3,854-Variable Problem Solved in Six Minutes With Quantum Computing

How To Find Large Files on Linux

By [Jo Hinchliffe](#) about 2 months ago

[Terms and conditions](#)

[Privacy policy](#)

[Cookies policy](#)

[Accessibility Statement](#)

[Advertise](#)

[About us](#)

[Coupons](#)

[Careers](#)

[Do not sell my info](#)

© Future US, Inc. Full 7th Floor, 130 West 42nd Street, New York, NY 10036

How to Check Your Graphics Card Temperature and Other Settings

By [Jarred Walton](#) about 2 months ago

How To Use Resistors in a Project

By [Les Pounder](#) about 2 months ago

How to Build Your Own 3D Printed Raspberry Pi Robot

By [Kevin McAleer](#) about 2 months ago
