

BASH

TLDR:

- Bash is a programming language



- Our purpose will be to automate commands you'd normally do in the terminal
- Example: setting up a folder structure automatically, by creating folders and files

BASH SCRIPT CONVENTIONS

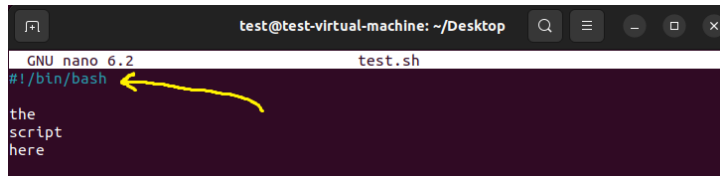
For the purpose of this course, optional points will be stated as **necessities**

- A bash script file will have the **.sh** extension

```
test@test-virtual-machine:~/Desktop/scripts$ touch my_first_script.sh
test@test-virtual-machine:~/Desktop/scripts$ ls
my_first_script.sh
```

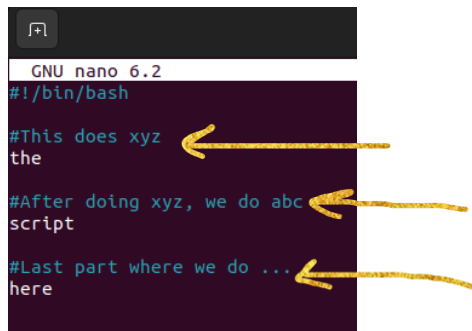
- A bash script file always starts with:

`#!/bin/bash`



```
test@test-virtual-machine: ~/Desktop
GNU nano 6.2 test.sh
#!/bin/bash
the
script
here
```

- Comments in a script file will start with a #, they don't impact the script and only help the reader



```
GNU nano 6.2
#!/bin/bash
#This does xyz
the
#After doing xyz, we do abc
script
#Last part where we do ...
here
```

LOGIC OF AUTOMATION

Context

You're in charge of 90 Linux computers at your local highschool. The schoolboard asks you to create a folder for each course, on EVERY computer.

Are you seriously going to go on all 90 computers and create the folders **manually**?

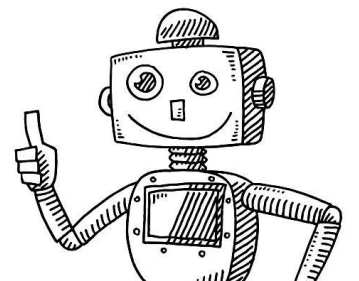


Solution

You can make a **bash script** that creates those folders **for you**. *You would then have the script run for all computers at once (however that part is outside the scope of this course).*

Conclusion

The script does exactly what a human would do, just **faster** and on **command**.



LAYING OUT THE SCRIPT

Before creating the script, let's first do the actions manually at least once to understand

*For the purpose of understanding scripting, the manual commands will be less optimal

1. Let's first go to the Desktop

```
test@test-virtual-machine:~$ cd ~/Desktop
test@test-virtual-machine:~/Desktop$
```

2. We want to create 4 folders: Math, Science, English, History

```
test@test-virtual-machine:~/Desktop$ mkdir Math
test@test-virtual-machine:~/Desktop$ mkdir Science
test@test-virtual-machine:~/Desktop$ mkdir English
test@test-virtual-machine:~/Desktop$ mkdir History
test@test-virtual-machine:~/Desktop$
```

3. We enter Math & Science to create 2 Chapter folders, and only 1 for English & History

```
test@test-virtual-machine:~/Desktop$ cd Math/
test@test-virtual-machine:~/Desktop/Math$ mkdir Chapter_1
test@test-virtual-machine:~/Desktop/Math$ mkdir Chapter_2
test@test-virtual-machine:~/Desktop/Math$ cd ..
test@test-virtual-machine:~/Desktop$ cd Science
test@test-virtual-machine:~/Desktop/Science$ mkdir Chapter_1
test@test-virtual-machine:~/Desktop/Science$ mkdir Chapter_2
test@test-virtual-machine:~/Desktop/Science$ cd ..
test@test-virtual-machine:~/Desktop$ cd English/
test@test-virtual-machine:~/Desktop/English$ mkdir Chapter_1
test@test-virtual-machine:~/Desktop/English$ cd ..
test@test-virtual-machine:~/Desktop$ cd History/
test@test-virtual-machine:~/Desktop/History$ mkdir Chapter_1
test@test-virtual-machine:~/Desktop/History$
```

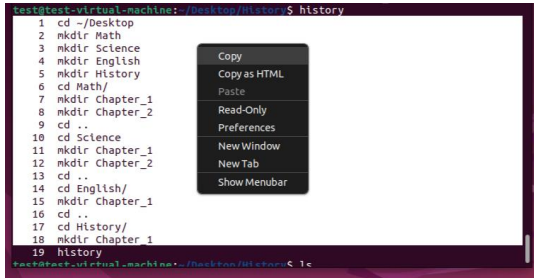
4. Now we can use the **history** command to see the commands we did

```
test@test-virtual-machine:~/Desktop/History$ history
1 cd ~/Desktop
2 mkdir Math
3 mkdir Science
4 mkdir English
5 mkdir History
6 cd Math/
7 mkdir Chapter_1
8 mkdir Chapter_2
9 cd ..
10 cd Science
11 mkdir Chapter_1
12 mkdir Chapter_2
13 cd ..
14 cd English/
15 mkdir Chapter_1
16 cd ..
17 cd History/
18 mkdir Chapter_1
19 history
```

WRITING THE SCRIPT

With the history of commands we did, we can theoretically copy-paste it and turn it into a script

1. Let's **copy** the history of **commands** we did (or just re-write them if your history is riddled with other commands)



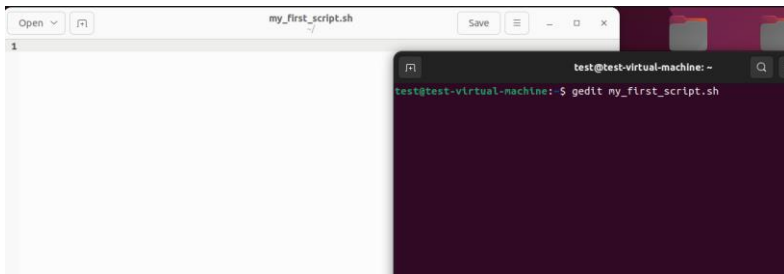
```
test@test-virtual-machine: ~/Desktop/History $ history
1 cd ~/Desktop
2 mkdir Math
3 mkdir Science
4 mkdir English
5 mkdir History
6 cd Math/
7 mkdir Chapter_1
8 mkdir Chapter_2
9 cd ..
10 cd Science
11 mkdir Chapter_1
12 mkdir Chapter_2
13 cd ..
14 cd English/
15 mkdir Chapter_1
16 cd ..
17 cd History/
18 mkdir Chapter_1
19 history
```

A context menu is open over the terminal, showing options: Copy, Copy as HTML, Paste, Read-Only, Preferences, New Window, New Tab, and Show Menubar.

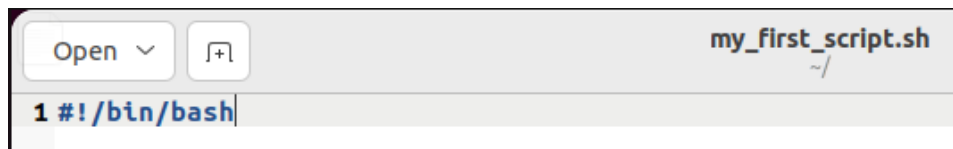
2. Let's go to the Home directory for the purpose of this script and **create** the **script file**

```
test@test-virtual-machine: ~/Desktop $ cd ~
test@test-virtual-machine: ~ $ touch my_first_script.sh
```

3. Let's **open** this in a **text-editor** with a UI for the purpose of this course (gedit)

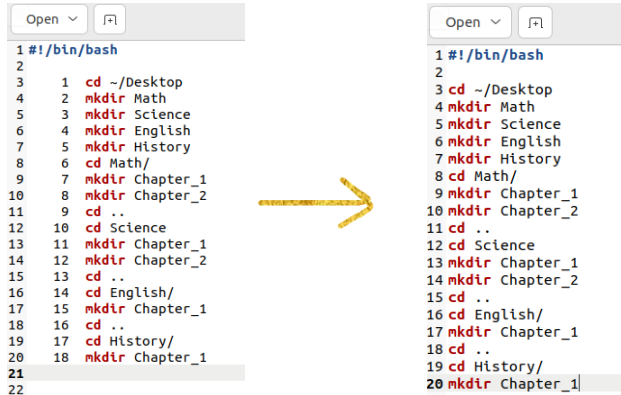


4. Add the necessary line to the top of the file (**#!/bin/bash**)



RUNNING THE SCRIPT

1. Paste the commands into the file, anywhere after the `#!/bin/bash` line (but remove the numbers 1,2,3,4)



```
1 #!/bin/bash
2
3 1 cd ~/Desktop
4 2 mkdir Math
5 3 mkdir Science
6 4 mkdir English
7 5 mkdir History
8 6 cd Math/
9 7 mkdir Chapter_1
10 8 mkdir Chapter_2
11 9 cd ..
12 10 cd Science
13 11 mkdir Chapter_1
14 12 mkdir Chapter_2
15 13 cd ..
16 14 cd English/
17 15 mkdir Chapter_1
18 16 cd ..
19 17 cd History/
20 18 mkdir Chapter_1
21
22
```

2. Press CTRL+S to save or just press the Save button on the top right, then close the text-editor window

3. Before running the script, let's first delete the folders we created to see if it'll actually work

```
test@test-virtual-machine:~$ rm -r Desktop/Math/ Desktop/Science/ Desktop/English/ Desktop/History/
test@test-virtual-machine:~$
```

4. Now we're ready to run the script, and it should have worked:

```
test@test-virtual-machine:~$ bash my_first_script.sh
test@test-virtual-machine:~$
```

or option 2:

```
test@test-virtual-machine:~$ chmod +x my_first_script.sh
test@test-virtual-machine:~$ ./my_first_script.sh
```

BASH SUMMARY

1. Know the commands you want the system to do for you

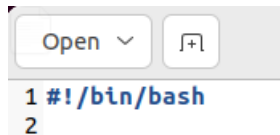
2. Create a script file with a **.sh** extension:

```
touch script_name.sh
```

3. Open the script file with gedit:

```
gedit script_name.sh
```

4. The first line of the script will be:

A screenshot of a code editor window. At the top, there are two buttons: 'Open' with a dropdown arrow and a file icon. Below the buttons, the first line of the script is highlighted in blue and contains the text '#!/bin/bash'. The second line is empty.

```
1 #!/bin/bash
2
```

5. The script will go below that first line, then save and close gedit

6. Run the script with:

```
bash script_name.sh
```

or

```
chmod +x script_name.sh
```

```
./script_name.sh
```

EXERCISE

Script to recreate this part of the ELDEN RING folder structure (create folders & files)

Game/

crashdumps/

attachments/

reports/

settings.dat

EasyAntiCheat/

Certificates/

base.bin

base.cer

easyanticheat_eos_setup.exe

install_script.vdf

settings.json

mods/

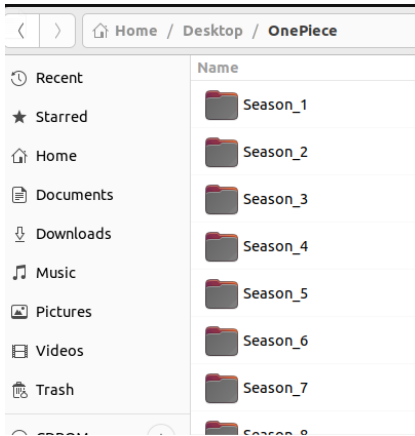
elden_ring_seamless_coop.dll

seamlesscoopsettings.ini

EXTRA – PROGRAMMING (PART 1)

Context

You're asked to create a folder called 'OnePiece' and inside, a folder for each season (20 seasons total) in the following format:



You're clearly not going to type out every single folder name, so let's do it with programming logic.

It should work as such:

```
test@test-virtual-machine:~$ bash script.sh
test@test-virtual-machine:~$ ls
Desktop  Downloads  Music      OnePiece  Public    snap      test.sh
Documents LinuxPractice my_first_script.sh Pictures  script.sh Templates Videos
test@test-virtual-machine:~$ cd OnePiece/
test@test-virtual-machine:~/OnePiece$ ls
Season_1 Season_11 Season_13 Season_15 Season_17 Season_19 Season_20 Season_4 Season_6 Season_8
Season_10 Season_12 Season_14 Season_16 Season_18 Season_2 Season_3 Season_5 Season_7 Season_9
test@test-virtual-machine:~/OnePiece$
```

It created the OnePiece folder and the 20 folders inside

EXTRA – PROGRAMMING (PART 2)

Here's how the script looks:

```
Open ▾ [+]  
1 #!/bin/bash  
2  
3 mkdir OnePiece  
4  
5 cd OnePiece  
6  
7 for number in {1..20}  
8 do  
9     mkdir Season_$number  
10 done
```

Let's break it down:

1. line #1 is always `#!/bin/bash`

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

2. We start by creating the OnePiece folder

```
mkdir OnePiece
```

3. We then enter the OnePiece folder

```
cd OnePiece
```

4. We create a **loop**, meaning a repetition that follows a condition
"I will repeat doing X as long as I do not meet the condition to stop"

"number" will start at the value 1 and go up to the value 20

The condition is therefore to stop when "number" reaches the value 20

We can translate this as: *let the variable called "number" take every value within the range 1 to 20*

```
for number in {1..20}
```

5. We inform the script that there's a block of action(s) to **do**:

```
do
```

6. That action is to **create a folder** (mkdir) that will **take the name...**
Season_[value of the variable "number" has at that point in time]
\$number means "take the value of the variable number"

```
mkdir Season_$number
```

7. We inform the script that once that is satisfied, the block of action is **done**

```
done
```

EXTRA – PROGRAMMING (PART 3)

To expand on the **loop**...

for **number** **in** {1..20}

let...

as in “let X take
the value of ...”

within the following set of values...

it's letting you know that the
variable will become the host
of the upcoming set of values

this variable become the host...

the name attributed
to the host that will
be infused with the value

It's him that will become 1,
then 2, then 3, then ... 20

he's the chosen one

starting FROM 1, every WHOLE NUMBER, TO 20

{ 1 .. 20 }

TRANSLATION: today, we have chosen that **number** will be the chosen host to hold the **values** contained within 1 to 20 every time **number** will finish a full loop with one value (ex: 3), it will become the next one (ex: 4)...until it has reached 20

EXTRA – PROGRAMMING (PART 4)

Analogy for the loop...

```
for number in {1..20}  
do  
  mkdir Season_$number  
done
```

variable

Let



become every Pokemon in this list:



set of values

he will become the next Pokemon in line, starting from becoming Oddish up until becoming Meowth...

...ONLY if accomplishes what is required of him (ex: winning a battle)

*In that case, a **loop** would be Ditto **becoming** the first Pokemon in line and **winning** a battle.*

*If that's accomplished, we enter the **2nd iteration** of the loop (2nd time it's repeated), and Ditto becomes the next Pokemon in line and wins another battle.*

*If that's accomplished, we enter the **3rd iteration** of the loop (where he becomes Gastly), etc. etc. until he becomes Meowth and wins the battle, then it stops.*

Same logic for the Season_\$number loop. “number” becomes 1, we create Season_1 folder, then we jump to “number” becomes 2, create Season_2 folder... until we reach “number” becomes 20, create Season_20 folder. Everytime “number” takes a new number, it's a new iteration of the loop.