

## Level Up Coding

---

★ Member-only story

# Writing Better Shell Scripts with Lua

A new way to write modern, readable, clean shell scripts by avoiding traditional shell languages



Shalitha Suranga · Subscribed

Published in Level Up Coding · 7 min read · 17 hours ago



14





Moon ("Lua" in Portuguese), Photo by [Linda Xu](#) on [Unsplash](#)

Unix-based and Unix-like operating systems offer command-line and graphical interfaces for users to handle day-to-day operations. Programmers prefer choosing the command-line-based approach to interact with the operating system with the terminal, and they automate most of their day-to-day programming tasks by writing shell scripts.

A shell script typically consists of operating system commands and automation logic written in a shell scripting language like Bourne shell and Bash. The built-in shell interpreters offer a simple way to write shell scripts using command-oriented language syntax. The command-oriented language works well for writing less-complex automation scripts that don't process much data, but writing complex shell scripts and processing data within them is undoubtedly time-consuming due to the lack of readable, general-purpose programming features in command languages.

In this article, we'll learn how to write modern, readable, and clean shell scripts by using Lua instead of traditional, well-known command interpreters like `sh`, `bash`, and `zsh`.

## Why Lua for Shell Scripting?

System administrators, DevOps engineers, and programmers seek alternative languages for shell scripting if they need to write complex automation scripts that process data and handle somewhat complex logic. Writing any script is possible with Bash-like command languages, but using such command languages beyond simple command-based shell scripting often creates complicated, unreadable scripts.

Using a minimal language that offers a simple child process API indeed creates a great productive environment for writing advanced shell scripts. Writing better, readable shell scripts is possible with Python and JavaScript (Node.js), but Lua offers a simpler programming interface with syntactical benefits, especially for system administrators who are only familiar with Bash or other command languages.

Here is why Lua is a better alternative for modern shell scripting:

- Lua is an extremely simple but versatile, fully-featured, productive language, so system administrators can easily adapt to it and do their tasks productively
- Lua offers minimal APIs for general tasks in shell scripting, so programmers can skip triggering costly separate binaries by using built-in Lua standard library functions

- The Lua interpreter is only 279KB and doesn't need any third-party installers for installation, so Lua is a comfortable tool for CI/CD shell scripts
- The Lua developer community created the Luash project to write shell scripts more productively, offering a better wrapper around the child process API, effectively using Lua syntactical features

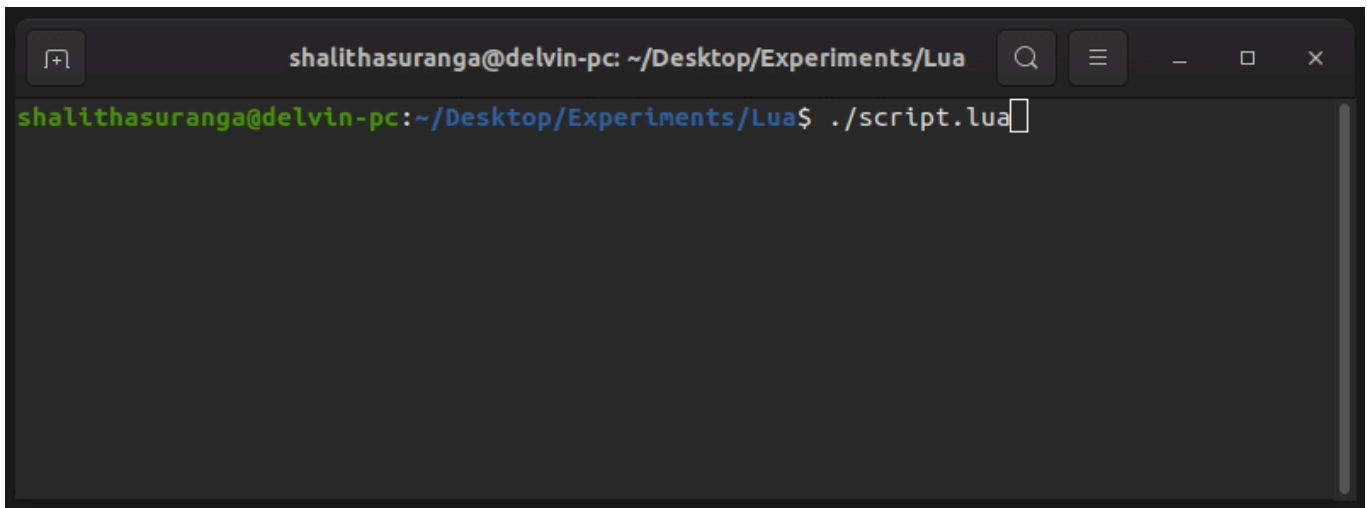
## Writing Simple Shell Scripts With `os.execute()`

You don't have to import anything or learn language-specific concepts (e.g., callbacks and promises in JavaScript) to execute a simple command or command combination in Lua. See how you can execute shell commands in Lua with `os.execute()` :

```
#!/usr/bin/lua

function _(cmd)
    os.execute(cmd)
end

_("mkdir -p src/api/fs src/api/os")
_("cd src/api && touch fs/README.md os/README.md")
_("tree src")
```



Executing some shell scripts from Lua using `os.execute()`, a screen recording by the author

You can use your favourite Bash features with Lua to further boost your shell scripting productivity. See how the following script uses the Bash brace expansion feature within Lua:

```
#!/usr/bin/lua

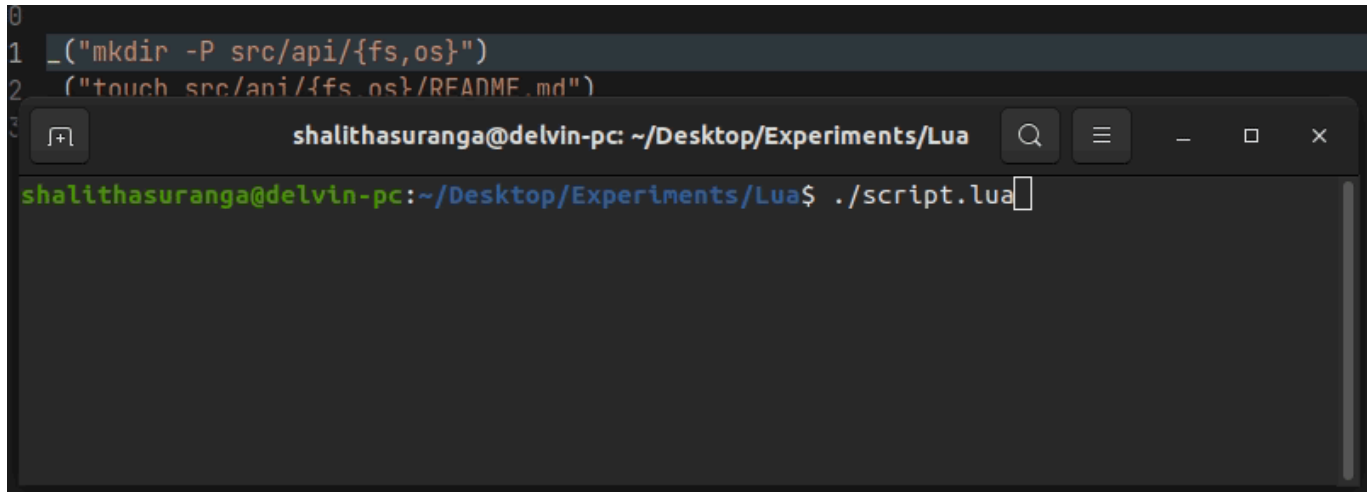
function _(cmd)
    os.execute("/usr/bin/bash -c \"\" .. cmd .. \"\"")
end

_("mkdir -p src/api/{fs,os}")
_("touch src/api/{fs,os}/README.md")
_("tree src")
```

You can exit from Lua scripts if a command execution failure happens by checking the return values of the `os.execute()` function as follows:

```
function _(cmd)
    local success, _, code = os.execute("/usr/bin/bash -c \"\" .. cmd .. \"\"")
    if not success then
        print("Command '" .. cmd .. "' failed: " .. code)
    end
end
```

```
os.exit(code)
end
end
```



```
1 _("mkdir -P src/api/{fs,os}")
2 ("touch src/api/{fs,os}/README.md")

shalithasuranga@delvin-pc: ~/Desktop/Experiments/Lua$ ./script.lua
```

The Lua script stops at command execution failures, a screen recording by the author

Using this approach, you can turn your existing simple command-based shell scripts into Lua scripts in seconds by wrapping them with `_()`.

## Using Lua Logic in Shell Scripts

Executing all your shell script logic in a command language using the `os.execute()` function is not undoubtedly the way to write better shell scripts with Lua — reducing the shell script complexity by balancing the overall automation logic between the shell language and Lua is the right way to do Lua-based shell scripting.

Writing automation script logic in Lua not only gives a clean script — Lua-based shell script logic also improves overall script performance since you no longer spawn a new process to do every little sub-task. For example, you

can write your shell script logic in Lua and trigger external programs only if needed, as shown in the following example script:

```
code = io.open("main.c.template"):read("*a")
print("Template loaded: ")
print(code)

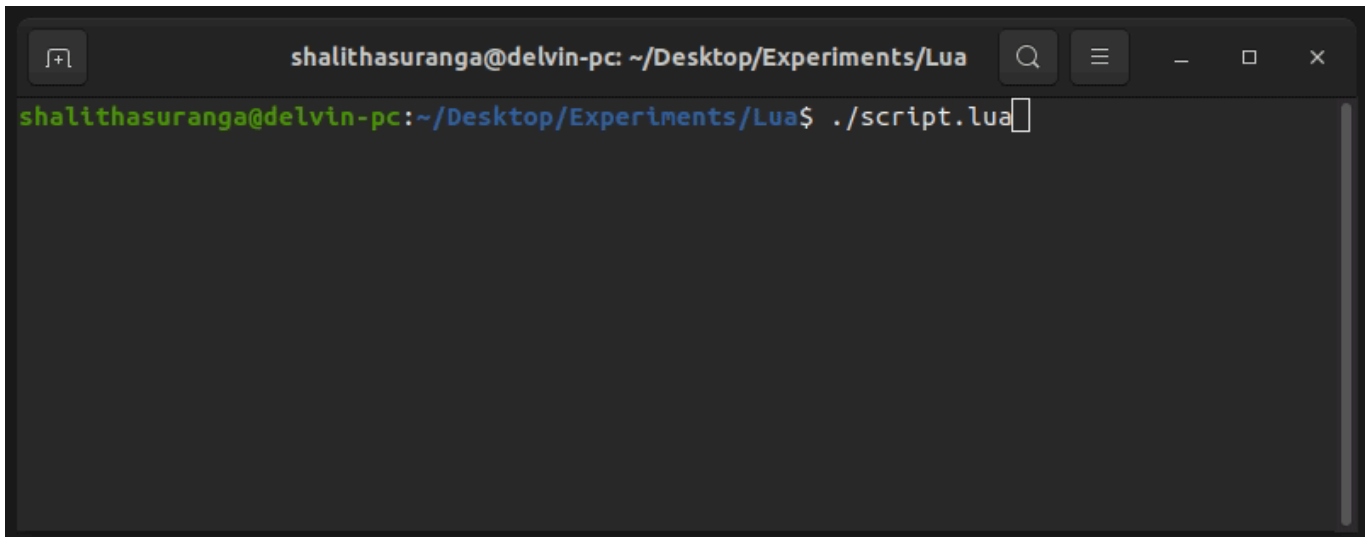
io.write("Enter your name: ")
code = code
      :gsub("{author}", io.read())
      :gsub("{date}", os.date())
io.open("main.c", "w"):write(code):close()

print("Source:")
print(code)

print("Compiling main.c...")
_ ("gcc main.c -o main")

print("Running ./main")
_ ("./main")
```

The above shell script dynamically generates a C program based on a template and executes it by performing file handling, data processing, and terminal I/O with Lua, and spawning a child process only for compilation of the generated source code and program execution:



Demonstrating a Lua shell script that compiles and runs a C program, a screen recording by the author

Several shell languages, like Bourne shell, don't offer built-in string manipulation syntax or commands, and Bash's string replacement syntax is unique and not developer-friendly. Handling strings with Lua's string manipulation functions is undoubtedly productive and faster in shell scripting:

```
# POSIX (sh), spawns a new process
str="Hello World"
echo "$str" | sed -e "s/World/Lua/g"

# Bash, unique, unclear syntax
str="Hello World"
echo ${str/World/Lua}

# Lua, minimal OOP syntax
str = "Hello World"
print(str:gsub("World", "Lua"))
```

The Lua built-in standard library implements file handling, string manipulation, math, and internal operating system features to let you write



advanced data processing and automation logic more productively compared to popular shell interpreter features.

## Using Luash to Write Advanced Shell Scripts

You can write most shell scripts productively using Lua for automation logic and your preferred command language for executing external commands. It's possible to shorten and improve shell scripts using Bash's non-POSIX features (e.g., brace expansion) with Lua too.

What if you need to extract outputs from commands and implement command pipelines within the Lua scripting context? It's not possible with `os.execute()` since it won't provide process I/O handlers.

You can use the `io.popen()` function to send input to processes and read output as follows, using a temporary file to pipe command input:

```
#!/usr/bin/lua

function _(cmd, input)
    if input then
        local tmp = os.tmpname()
        io.open(tmp, "w")
            :write(input)
            :close()
        cmd = cmd .. " <" .. tmp
    end
    return io.popen(cmd, "r")
        :read("*a")
end

print(_("node --version"))

print(_("node", "console.log(100 * 2)"))
```

```
print(_("wc -l",_("grep .lua",_("ls"))))
```



Working with process I/O using the `io.popen()` function, a screen recording by the author

Now, the `_()` function implementation lets us work with basic process I/O and implement pipelines within the Lua scripting context. This approach works well, but it doesn't give us much readable code since we should pass the command as a string to the function.

The Luash project offers a more productive and clean approach to writing any advanced shell script with Lua by letting you run commands using dedicated Lua functions. It internally creates a hook for undefined variables and invokes external commands using [Lua metatables](#).

For example, you can execute the `ls` command from Lua as follows:

```
ls()
```

You can omit parentheses and write more shell-like syntax with Lua as well:

```
ls "-l"
```

You can simply import the `sh.lua` file from your shell script and run shell commands as follows:

```
#!/usr/bin/lua

require("sh")

print(ls "-l")
print(node "--version")
```

Pipelines from the Lua scripting context can also be implemented in a more readable, shell-like way than the previous `_()` function implementation by using Luash's chaining syntax enhancement, as shown in the following example:

Open in app ↗

Medium

Search

Write

13



```
Normal Lua syntax: print(wc(grep(ls(), ".lua"), ".lua"))
-- Previous _() syntax: print(_("wc -l", _("grep .lua", _("ls"))))
```

Sending inputs to a specific command is also possible using the `__input` table key:

```
node {__input = "console.log(100 * 2)"} : print()
```

Using Luash's traditional shell-scripting-like syntax to run shell commands and Lua to write automation logic gives us better, readable shell scripts even if the overall shell script complexity grows.

Let's rewrite the previous C program compilation script with Luash:

```
#!/usr/bin/lua

sh = require("sh")

code = io.open("main.c.template"):read("*a")
print("Template loaded: ")
print(code)

io.write("Enter your name: ")
code = code
      :gsub("{author}", io.read())
      :gsub("{date}", os.date())
io.open("main.c", "w"):write(code)

print("Source:")
print(code)

print("Compiling main.c...")
gcc "main.c -o main"

print("Running ./main")
sh "./main" "" : print()
```

As demonstrated in the above code snippet, Luash seamlessly lets us run shell commands effectively using Lua's syntactical features. Using this

approach, you can get benefits from Lua and its standard library while executing commands in a more shell-like syntax.

There is no Python or Node.js module that offers the same shell scripting experience as Luash

Compare Google Open-Source's [zx project](#) and Python [sh module](#) with Luash to see how it implements more shell-like syntax.

## Conclusion

In this article, we discussed how programmers, system administrators, and DevOps engineers can use Lua for shell scripting. We discussed several approaches for using Lua for shell scripting. The `os.execute()` function lets us execute basic commands and command combinations, and we can improve it using Bash's non-POSIX features. The `io.popen()` function-based technique lets us implement process I/O. Meanwhile, the Luash project syntactically improves the `io.popen()` -based approach and help us write any complex shell script. Moreover, you can modify the Luash script based on your requirements (e.g., using Bash instead of Bourne shell) since it's licensed under the MIT open-source license.

You can choose any Lua shell scripting method based on your preference and write better, modern, readable, and fast shell scripts with Lua by not solely using Bash or similar command languages.

The following story explains several Lua features that you can learn to improve your Lua shell scripts further:

### **Lua: The Easiest, Fully-Featured Language That Only a Few Programmers Know**

Learning Lua is indeed easier than Python, Ruby, and JavaScript

[levelup.gitconnected.com](https://levelup.gitconnected.com)

Thanks for reading.

Programming

Technology

Software Development

Automation

Lua



**Published in Level Up Coding**

223K Followers · Last published 17 hours ago

Follow

Coding tutorials and news. The developer homepage [gitconnected.com](https://gitconnected.com) && [skilled.dev](https://skilled.dev) && [levelup.dev](https://levelup.dev)



**Written by Shalitha Suranga**

6.2K Followers · 42 Following

Subscribed



Programmer | Author of Neutralinojs | Technical Writer

## No responses yet



Alex Mylnikov

What are your thoughts?

## More from Shalitha Suranga and Level Up Coding



 In Level Up Coding by Shalitha Suranga

### Why Every Programmer Should Learn Lua

Mastering the world's simplest language, Lua teaches lessons that improve your...

★ Mar 4 🖱 351 💬 6



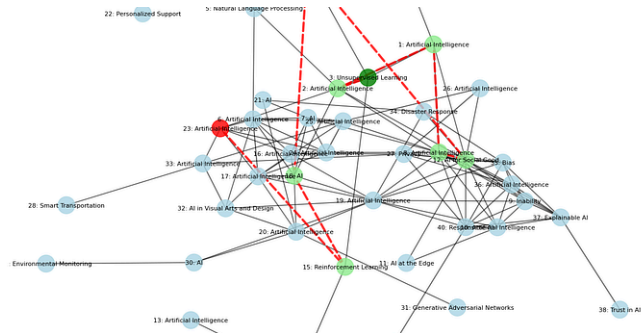
 In Level Up Coding by Dr. Ashish Bamanian 

### Chain-of-Draft (CoD) Is The New King Of Prompting Techniques

A deep dive into the novel Chain-of-Draft (CoD) Prompting that reducing LLM inferenc...

★ Mar 3 🖱 2.4K 💬 35





 In Level Up Coding by Fareed Khan

## Testing 18 RAG Techniques to Find the Best

crag, HyDE, fusion and more!

★ Mar 11 🖱️ 1.5K 💬 25  ⋮

 In Level Up Coding by Shalitha Suranga

## Things Senior Programmers Never Do

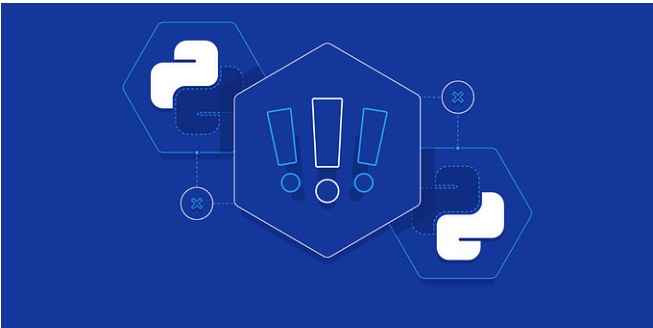
Protect your reputation by not doing these things during programming

★ Dec 29, 2024 🖱️ 1.3K 💬 40  ⋮

See all from Shalitha Suranga

See all from Level Up Coding

## Recommended from Medium







Utsav Madaan



In Coding Nexus by Algo Insights

## 5 Free & Open-Source Tools That Are Total Game Changers for...

Tired of the same old dev tools? 😞 Discover 5 awesome free and open-source gems that...

Mar 29 🖱️ 177 💬 3



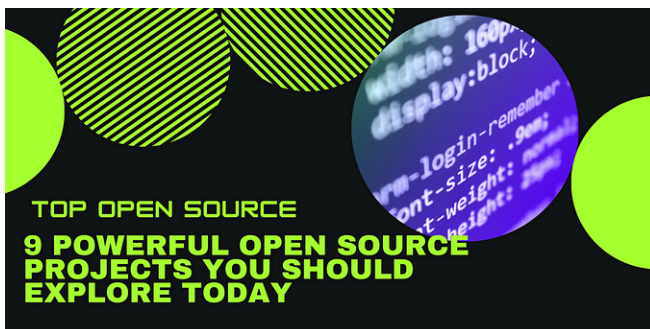
In The Pythoneers by Aashish Kumar

## 10 Production-Grade Python Code Styles I Learned from Real-World...

Discover 10 battle-tested Python code styles that will make your code cleaner, more...



Mar 18 🖱️ 266 💬 1



In Let's Code Future by Let's Code Future

## 9 Powerful Open Source Projects You Should Explore Today 🔥

Must-Read and Save this

## 8 Confusing Python Concepts That Frustrate Most Developers

Python is one of the most beginner-friendly programming languages. But even...



Feb 13 🖱️ 181 💬 3



Devlink Tips

## Top 10 European Open-Source Projects to Watch in 2025:

How Open-Source European Innovation is Changing Digital Sovereignty, Privacy, and...

Mar 19

🖱️ 711 💬 4





Coders Stop

## 8 Terminal Aliases That Will Make Senior Developers Question Your...

We've all been there. It's 2 AM, you're 14 hours into a debugging session, hopped up on...

 6d ago  274  2  

 Mar 30  591  17  

See more recommendations