Fayssal Elaazouzi

Mar 22 · 5 min read · Listen

Day 4 : Basic Linux Shell Scripting for DevOps Engineers 👍

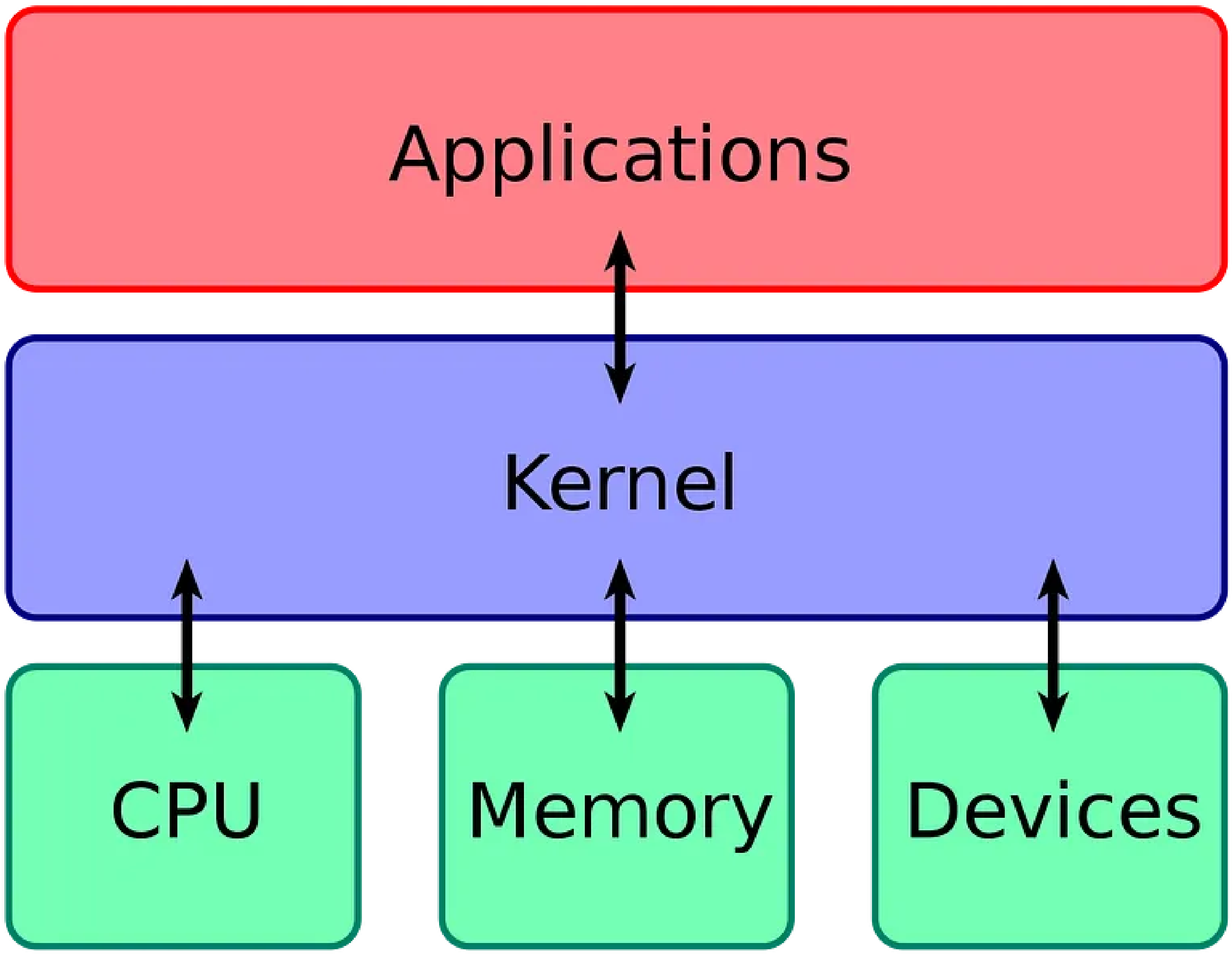
Hi, I'm **Fayssal**, and this is my first day of **#90DaysOfDevops** initiated by **Shubham Londhe**. Today, I'll be discussing Basic linux Shell Scripting for DevOps Engineers.

1. What is Kernel?

In computing, a kernel is the central component of an operating system that manages system resources, controls hardware access, and provides various services to applications running on the system. The kernel acts as a bridge between software and hardware, providing a layer of abstraction that allows software to interact with hardware in a uniform way.

The kernel is responsible for tasks such as managing memory, scheduling processes, handling input/output (I/O) requests, and enforcing security policies. It also provides a system call interface that allows applications to request services from the kernel, such as opening a file or creating a network connection.

Different operating systems use different kernel architectures. For example, Linux uses a monolithic kernel architecture, where all kernel services are provided by a single large kernel module, while microkernels, like the one used by QNX, have a smaller kernel that provides only the most essential services, with additional services implemented as user-space processes.



2. What is Shell in Linux?

In Linux, a shell is a command-line interface (CLI) that allows users to interact with the operating system by entering commands. The shell is a program that provides a user interface for accessing the operating system's services and running applications.

The shell acts as an interpreter between the user and the operating system, receiving commands from the user and translating them into instructions that the operating system can understand and execute. The shell also provides features such as command history, command-line editing, and scripting capabilities.

There are several shells available in Linux, such as the Bourne-Again SHell (**bash**), the Z SHell (**zsh**), the C SHell (**csh**), and the Korn SHell (**ksh**), among others. Bash is the most common shell used in Linux, and it is the default shell in many Linux distributions.

Using a shell in Linux can be a powerful way to perform **system administration tasks** and **automate repetitive tasks** through scripting. By entering commands into the shell, users can **manipulate files and directories**, **manage system services**, and perform a wide range of other tasks.

3. Why Linux Shell Scripting?

Linux shell scripting is the process of creating scripts that **automate the execution of multiple commands**. These scripts are written in a shell scripting language (such as **Bash**, the most commonly used shell in Linux), and can be used to perform a variety of tasks, from simple file management to complex system administration tasks.

Shell scripts can be used to **automate repetitive tasks**, **schedule tasks to run at specific times**, or even to **create complex programs that interact with the operating system**. Shell scripting is a powerful tool for system administrators and developers alike, and is an essential skill for anyone working in a Linux or Unix environment.

4. Importance of Shell Scripting in DevOps

Shell scripting plays a crucial role in DevOps because it allows developers and operations teams to automate repetitive tasks, streamline processes, and manage infrastructure at scale. Here are some of the key reasons why shell scripting is important in DevOps:

1. **Automation**: Shell scripting allows DevOps teams to automate repetitive tasks, such as building and deploying applications, configuring servers, and managing backups. This saves time and reduces the risk of errors, while also freeing up team members to focus on more strategic work.

2. **Consistency**: By using shell scripts to automate tasks, DevOps teams can ensure that each task is performed consistently and according to best practices. This helps to reduce the risk of human error and ensures that systems are configured in a consistent and repeatable way.

3. **Scalability**: Shell scripting allows DevOps teams to manage infrastructure at scale, by automating tasks that would otherwise be time-consuming or impossible to perform manually. This is particularly important in cloud environments, where infrastructure can be rapidly scaled up or down based on demand.

4. **Collaboration:** Shell scripting makes it easier for DevOps teams to collaborate on tasks and share knowledge, because scripts can be stored in version control systems and shared across teams. This allows teams to work more efficiently and reduces the risk of siloed knowledge.

5. **Flexibility:** Shell scripting is a flexible and powerful tool that can be used for a wide range of tasks, from simple tasks like file manipulation to complex tasks like building and deploying applications. This makes it a valuable tool for DevOps teams, who need to be able to adapt quickly to changing requirements and environments.

In summary, shell scripting is an essential tool for DevOps teams, because it allows them to automate tasks, ensure consistency, manage infrastructure at scale, collaborate more effectively, and be more flexible and adaptable in their work.

5. What is `#!/bin/bash?`

`#!/bin/bash` is called a shebang or hashbang. It is a special sequence of characters that appears at the beginning of a script or executable file in Unix-like operating systems.

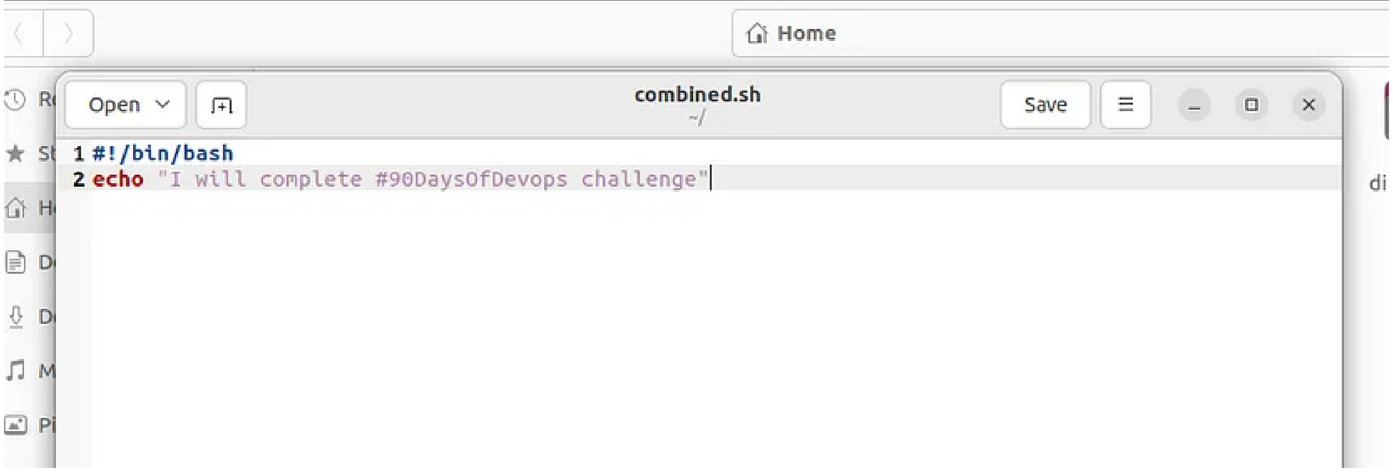
This sequence of characters tells the system which interpreter to use to run the script. In this case, `#!/bin/bash` specifies that the Bash shell should be used to interpret the commands in the script.

For example, if you have a Bash script named "my_script.sh" that begins with `#!/bin/bash`, when you try to execute the script with the command `./my_script.sh`, the system will use the Bash shell to run the commands in the script.

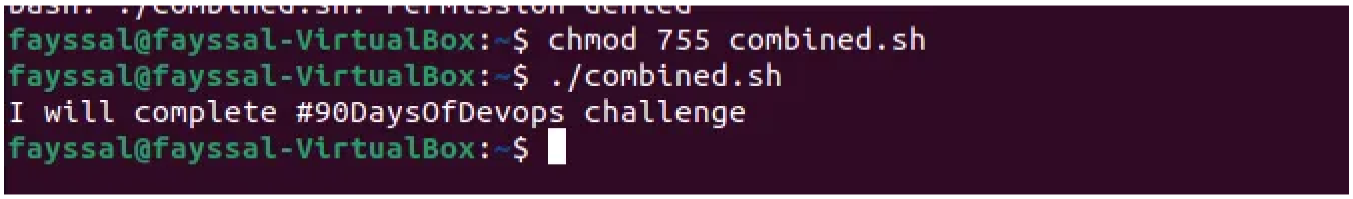
Note that the shebang line must be the first line of the file, and it must begin with the two characters "#!" (hash and exclamation point), followed by the path to the interpreter that should be used to run the script.

6. Examples

Write a Shell Script that prints "I will complete #90DaysofDeVops challenge

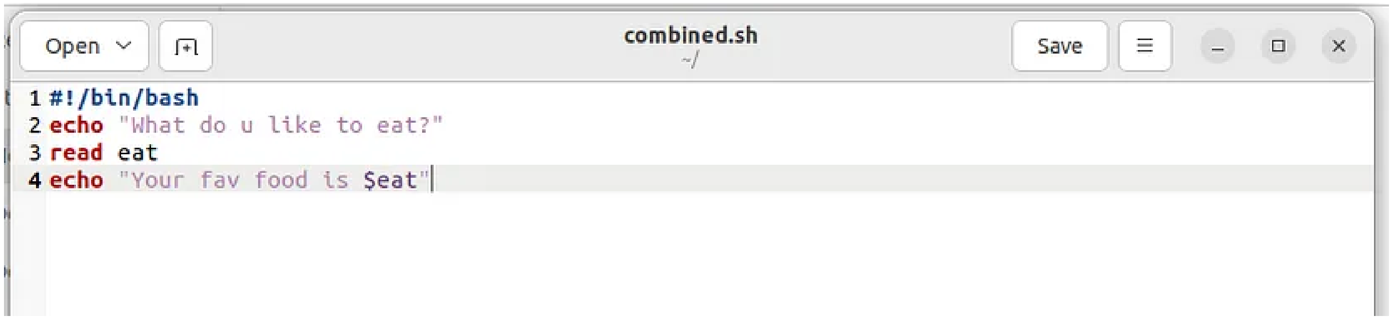


```
#!/bin/bash
echo "I will complete #90DaysOfDevops challenge"
```

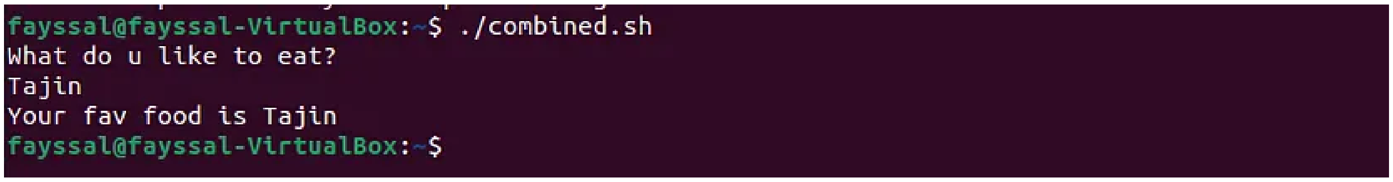


```
fayssal@fayssal-VirtualBox:~$ chmod 755 combined.sh
fayssal@fayssal-VirtualBox:~$ ./combined.sh
I will complete #90DaysOfDevops challenge
fayssal@fayssal-VirtualBox:~$
```

Write a Shell Script to take user input, input from arguments, and print the variables.

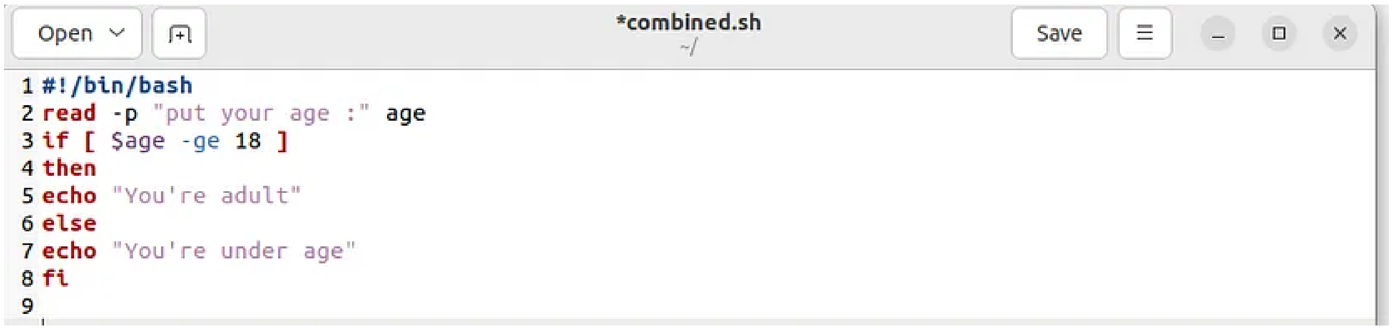


```
#!/bin/bash
echo "What do u like to eat?"
read eat
echo "Your fav food is $eat"
```



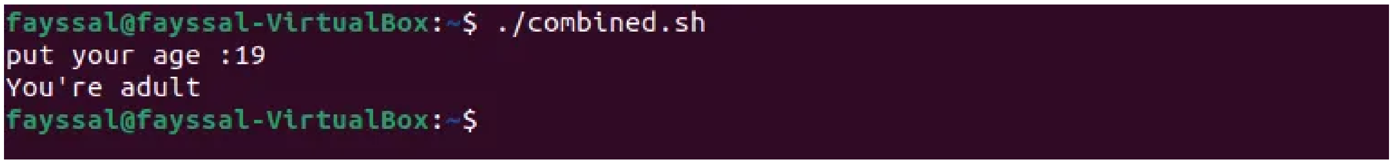
```
fayssal@fayssal-VirtualBox:~$ ./combined.sh
What do u like to eat?
Tajin
Your fav food is Tajin
fayssal@fayssal-VirtualBox:~$
```

Write an Example of If else in Shell Scripting by comparing two numbers



```
#!/bin/bash
read -p "put your age :" age
if [ $age -ge 18 ]
then
echo "You're adult"
else
echo "You're under age"
fi

```



```
fayssal@fayssal-VirtualBox:~$ ./combined.sh
put your age :19
You're adult
fayssal@fayssal-VirtualBox:~$
```

If this post was helpful, please do follow and click the clap 👏 button below to show your support 😄

_ Thank you for reading💚

_Fayssal 👍

Linux Shell DevOps Cloud Computing AWS