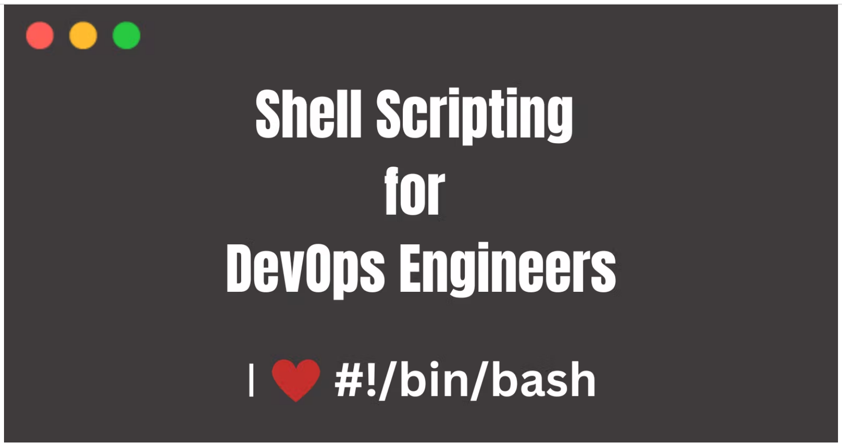
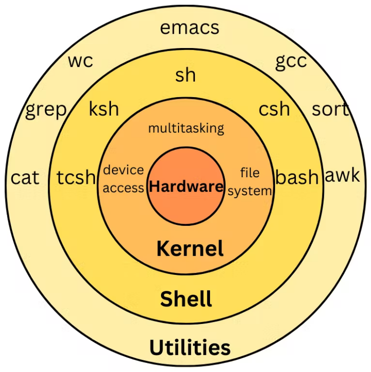


Day 4 Task: Basic Linux Shell Scripting for DevOps Engineers.

If you want to become a DevOps engineer, it's important to learn shell scripting or bash scripting.

Kunal Maurya · Feb 26, 2023 · 8 min read

What is Shell?



In Linux architecture, the shell is a command-line interface that provides a way for users to interact with the operating system. The shell is a program that takes commands from the user and executes them by communicating with the operating system kernel.

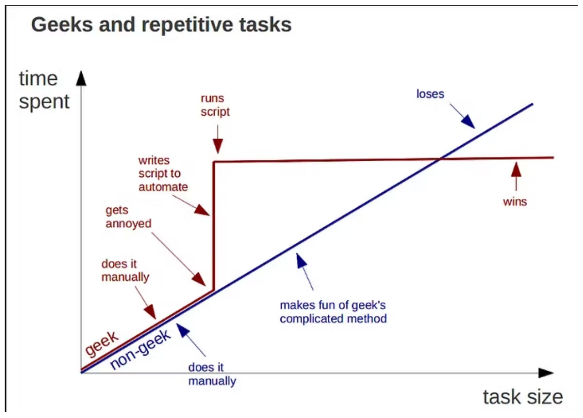
There are several types of shells available in Linux, but the most common one is the Bash (Bourne-Again SHell). Other popular shells include Zsh, Ksh, and Csh.

The shell provides several features such as command-line editing, command history, and the ability to execute commands in batch mode by reading them from a file. It also provides a way to execute complex commands by combining multiple commands and pipes.

Why Shell Scripting?

A script is a program that contains a sequence of commands that are executed by an interpreter. It's like a recipe for the computer to follow. Anything that can be executed from the command line can be put into a script.

Scripts are useful for automating repetitive tasks. If you find yourself performing the same set of commands repeatedly, you can create a script to automate the process.



Importance of Shell Scripting in DevOps

In the context of DevOps, shell scripting is important because it allows for the automation of repetitive tasks that are often required in deploying, managing, and monitoring applications and infrastructure.

By creating shell scripts, DevOps teams can automate tasks such as setting up environments, configuring servers, deploying applications, running tests, and monitoring system health. This not only saves time and effort but also reduces the likelihood of errors that can occur when performing these tasks manually.

Let’s write our first script

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

To run it,

```
$ chmod 755 script.sh
$ ./script.sh
```

If you did not understand the script, mostly the first line. Don’t worry we will see shell scripts in detail in this article.

Some key points that we are going to discuss in our article:-

- Shebang
- Comments
- Variables
- User Input
- Tests
- Decision Making

- Iterative Statements
- Logic Operations
- Functions
- Wildcards
- Debugging

Shebang

You might have noticed in the above script that it starts with `#!/bin/bash`. This is called shebang. It is basically to refer the path to the interpreter. There are many interpreters, some of them are: bash, zsh, csh and ksh, etc.

- To use bash: `#!/bin/bash`
- To use zsh: `#!/bin/zsh`
- To use ksh: `#!/bin/ksh`
- To use csh: `#!/bin/csh` and so on...

Why Shebang? `#` is often called sharp and `!` is called Bang, hence the name sharp bang, but generally people say it **shebang** instead of a sharp bang.

Comments

Comments are started with a `#` sign, anything after the pound sign on that line is ignored.

Variables

Variables in shell scripting are used to store values that can be used later in the script. Variables can be assigned a value using the assignment operator "=", and the value can be accessed using the variable name.

Here's an example of how to declare a variable in a shell script:

COPY

```
#!/bin/bash
my_var="Hello, World!"
echo $my_var
```

In this example, the variable `my_var` is assigned the value "Hello, World!" using the assignment operator "=".

The value of the variable is then printed to the screen using the `echo` command. The variable is accessed using the variable name `$my_var`.

Rules for Variable assigning:

- Variables are case-sensitive.
- By convention, variables are uppercase.
- To use a variable, just write the variable name followed by the `$` sign.

Here's another example that demonstrates how to use variables in shell scripting:

COPY

```
#!/bin/bash
x=10
```

```
y=20
sum=$((x + y))
echo "The sum of $x and $y is $sum"
```

When this script is executed, the output will be:

```
The sum of 10 and 20 is 300
```

User Input

`read` command accepts STDIN (Standard Input)

```
read -p "PROMPT MESSAGE" VARIABLE
```

Here is an example of how user input can be used in a shell script:

```
#!/bin/bash

echo "Please enter your name: "
read name
echo "Hello, $name! Welcome to my script."
```

In this example, the user is prompted to enter their name using the "echo" command. Then, the "read" command is used to read the user's input and assign it to the variable "name". Finally, the script uses the "echo" command again to print a welcome message that includes the user's name.

```
Please enter your name:
John
Hello, John! Welcome to my script.
```

Tests

Tests are used for decision-making.

1. **File Test Operations**

   ```
   -d FILE_NAM  # True if FILE_NAM is a directory
   -e FILE_NAM  # True if FILE_NAM exists
   -f FILE_NAM  # True if FILE_NAM exists and is a regular file
   -r FILE_NAM  # True if FILE_NAM is readable
   -s FILE_NAM  # True if FILE_NAM exists and is not empty
   -w FILE_NAM  # True if FILE_NAM has write permission
   -x FILE_NAM  # True if FILE_NAM is executable
   ```

2. **String Test Operations**

```
-z STRING  # True if STRING is empty
-n STRING  # True if STRING is not empty
STRING1 = STRIN2 # True if strings are equal
STRING1 != STRIN2 # True if strings are not equal
```

Arithmetic Operators

```
var1 -eq var2  # True if var1 is equal to var2
var1 -ne var2  # True if var1 not equal to var2
var1 -lt var2  # True if var1 is less than var2
var1 -le var2  # True if var1 is less than or equal to var2
var1 -gt var2  # True if var1 is greater than var2
var1 -ge var2  # True if var1 is greater than or equal to var2
```

Decision Making

Decision-making in shell scripting is performed using the "if-then-else" statement. The "if-then-else" statement allows the script to make decisions based on specific conditions.

The syntax of the "if-then-else" statement is as follows:

```
if [ condition ]
then
    # commands to be executed if the condition is true
else
    # commands to be executed if the condition is false
fi
```

Here's an example that uses the "if-then-else" statement to check if a number is even or odd:

```
#!/bin/bash

echo "Enter a number: "
read num

if [ $((num%2)) -eq 0 ]
then
    echo "$num is even"
else
    echo "$num is odd"
fi
```

output:

```
Enter a number:
4
4 is even
```

Iterative Statements: Loops

In shell scripting, a loop is a control structure that allows a set of commands to be executed repeatedly. There are two common types of loops in shell scripting: the "for" loop and the "while" loop.

1. The For Loop

COPY

```
for VARIABLE_NAME in ITEM_1 ITEM_N
do
  command 1
  command 2
    ...
    ...
  command N
done
```

For example:

COPY

```
#!/bin/bash
for i in 1 2 3 4 5
do
   echo "Number: $i"
done
```

Another way of using for loop:

COPY

```
for (( VAR=1;VAR<N;VAR++ ))
do
  command 1
  command 2
    ...
    ...
  command N
done
```

2. The While Loop

COPY

```
while [ CONNDITION_IS_TRUE ]
do
  # Commands will change he entry condition
  command 1
  command 2
    ...
    ...
  command N
done
```

Here is an example:

```
#!/bin/bash

count=0

while [ $count -lt 5 ]
do
   echo "Count: $count"
   count=$((count+1))
done
```

Logic Operations

Shell scripts support **logical AND** and **logical OR**.

Logical AND = `&&`
Logical OR = `||`

Example: `mkdir tempDir && cd tempDir && mkdir subTempDir` In this example, tempDir is created with `mkdir` command. If it succeeds, then `cd tempDir` is executed, and then `mkdir subTempDir` is executed.

Functions

In shell scripting, a function is a block of code that performs a specific task. Functions are used to simplify code, make it more modular, and easier to maintain.

Syntax:

```
function function_name() {
    command 1
    command 2
    command 3
      ...
      ...
    command N
}
```

Here's an example of a function in shell script:

```
#!/bin/bash

# Define a function
hello() {
  echo "Hello, world!"
}

# Call the function
hello
```

Output:

```
Hello, world!
```

Wildcards

A character or string pattern that is used to match file and directory names is/are called wildcard(s). The process used to expand a wildcard pattern into a list of files and/or directories (or paths) is called Globbing.
Wild Cards can be used with most of the commands that require a file/dir path as an argument. (Example ls, rm, cp etc).

Some Commonly Used Wildcards

*** = Matches zero or more characters** Example:`*.txt` `hello.*` `great*.md`

? = matches exactly one character Example: `?.md` `Hello?`

[ ] = A character class This wildcard is used to match any of the characters included between the square brackets (Matching exactly one character).
Example: `He[loym]`, `[AIEOU]`

[!] = matches characters not included within brackets It matches exactly one character.
Example: To match a consonant: `[!aeiou]`

Debugging

A bug is an error in a computer program/software that causes it to produce an unexpected or incorrect result. Most of the bugs are caused by errors in the code and its design. To fix an error, try to reach to the root of that unexpected behavior.

The bash shell provides some options that can help you in debugging your script. You can use these options by updating the first line of the script.

Some Options:

1. -x option
It prints commands and arguments as they execute. It is called print debugging, tracing or an x-trace. We can use it by modifying the first line `#!/bin/bash -x`

2. -e option It stands for "Exit on error". This will cause your script to exit immediately if a command exits with a non-zero exit status.

3. -v option It prints shell commands/input lines as they are read.

Note* These options can be combined, and more than one option can be used at a time!

```
#!/bin/bash-xe
#!/bin/bash-ex
#!/bin/bash-x-e
#!/bin/bash-e-x
```

Thank you for reading! Hope you find this article helpful.

~Kunal

Subsc letter

Read articles from directly inside your inbox.
Subscribe to the newsletter, and don't miss out.

Enter your email address SUBSCRIBE

Linux Devops shell script shell scripting Devops articles

MORE ARTICLES

Kunal Maurya

Day 3 Task: Basic Linux Commands

Cat Command The cat command is a Unix and Linux utility that stands for "concatenate." It is used to...

Kunal Maurya

Day 2 Task: Basics Linux command

Listing commands Here are some examples of different ls commands in Linux with explanations: ls - T...

Kunal Maurya

Day 1: Getting Started with DevOps

Introduction In today's world, software development has become an integral part of businesses of all...