Shell Scripting – Shell Variables

Last Updated : 16 May, 2024

Summarize

Comments

Improve

-
-
-

A shell variable is a character string in a shell that stores some value. It could be an integer, filename, string, or some shell command itself. Basically, it is a pointer to the actual data stored in memory. We have a few rules that have to be followed while writing variables in the script (which will be discussed in the article). Overall knowing the shell variable scripting leads us to write strong and good shell scripts.

Rules for variable definition

A variable name could contain any alphabet (a-z, A-Z), any digits (0-9), and an underscore (_). However, a variable name must start with an alphabet or underscore. It can never start with a number. Following are some examples of valid and invalid variable names:

- Valid Variable Names

ABC
_AV_3
AV232

- Invalid variable names

2_AN
!ABD
$ABC
&QAID

Note: It must be noted that no other special character except underscore can be used in a variable name because all other special characters have special meanings in Shell Scripting.

Defining Variables

Syntax

variable_name = <variable data>

Example

```
num="1"
name="Devil"
```

These kinds of variables are scalar variables as they could hold one value at a time.

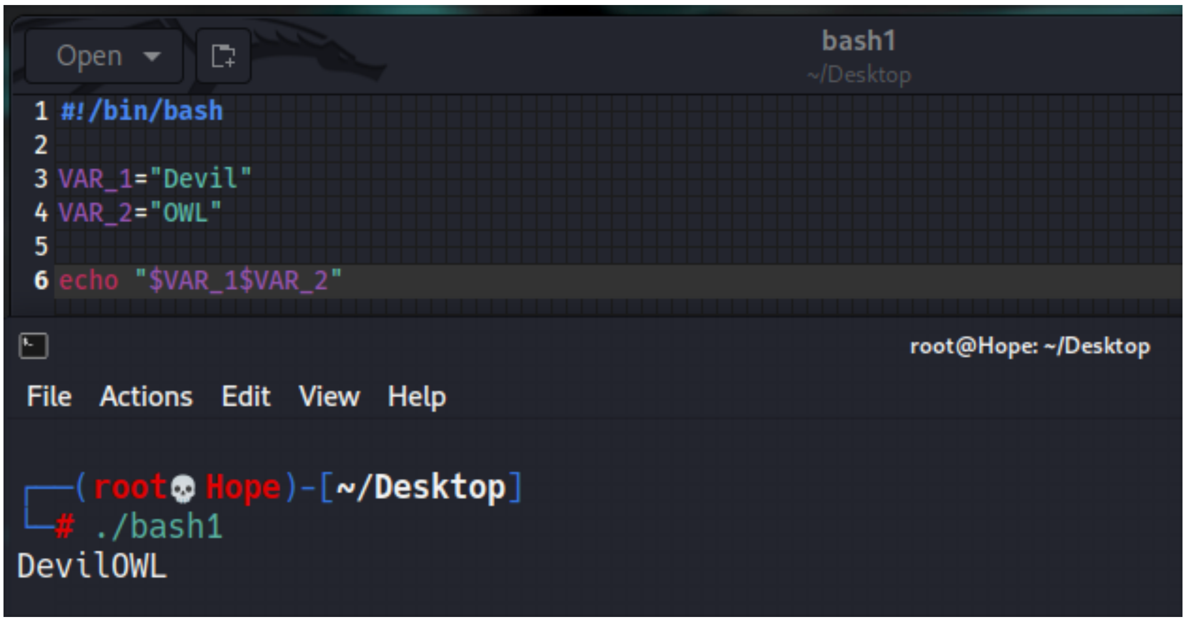
1) Accessing variable

Variable data could be accessed by appending the variable name with '$' as follows:

```
#!/bin/bash

VAR_1="Devil"
VAR_2="OWL"

echo "$VAR_1$VAR_2"
```

Output:

```
DevilOWL
```



```
#!/bin/bash

VAR_1="Devil"
VAR_2="OWL"

echo "$VAR_1$VAR_2"
```

```
┌──(root💀Hope)-[~/Desktop]
└─# ./bash1
DevilOWL
```

Example of Accessing variable

2) Unsetting Variables

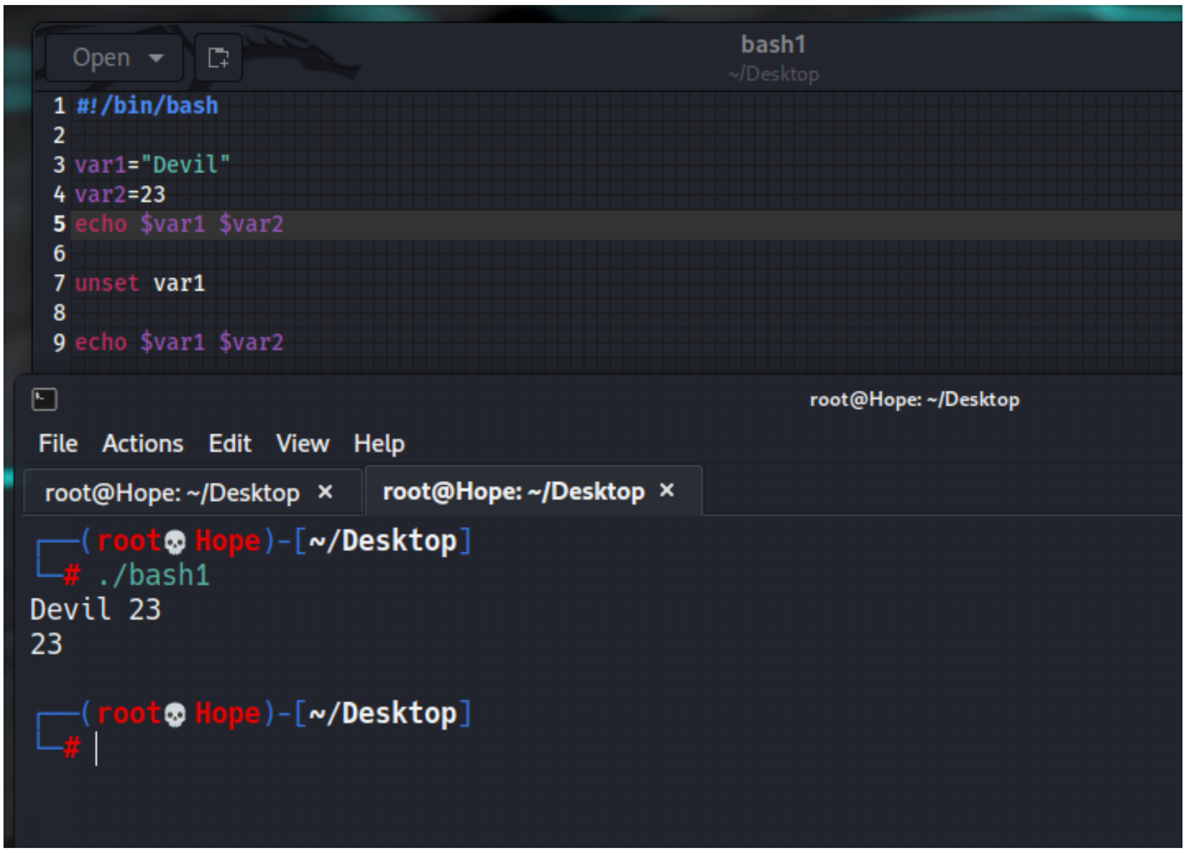
The unset command directs a shell to delete a variable and its stored data from list of variables. It can be used as follows:

```
#!/bin/bash

var1="Devil"
var2=23
echo $var1 $var2

unset var1

echo $var1 $var2
```

Output:

```
DEVIL 23
```

23



```
#!/bin/bash

var1="Devil"
var2=23
echo $var1 $var2

unset var1

echo $var1 $var2
```

```
┌──(root💀Hope)-[~/Desktop]
└─# ./bash1
Devil 23
23

┌──(root💀Hope)-[~/Desktop]
└─# 
```

Example of Unsetting Variables

Note: The unset command could not be used to unset read-only variables.

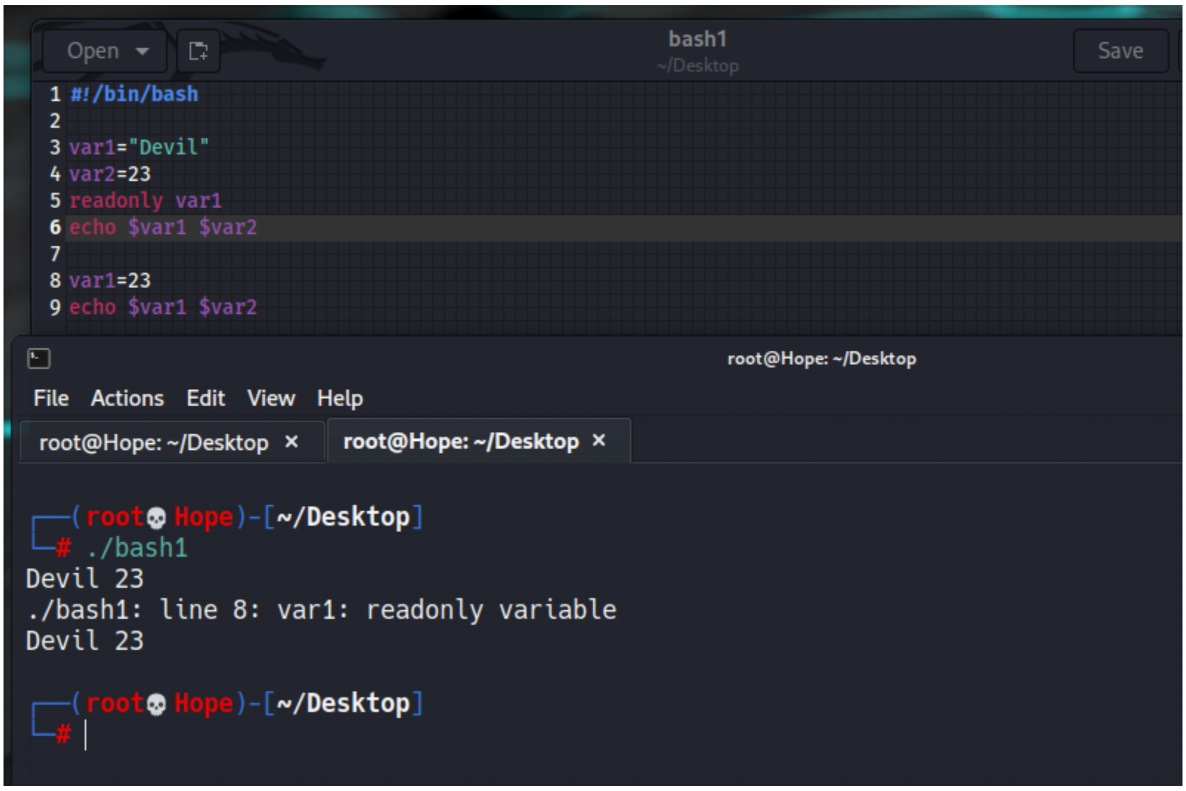
3) Read only Variables.

These variables are read only i.e., their values could not be modified later in the script. Following is an example:

```
#!/bin/bash
var1="Devil"
var2=23
readonly var1
echo $var1 $var2
var1=23
echo $var1 $var2
```

Output:

```
Devil 23
./bash1: line 8: var1: readonly variable
Devil 23
```



```
#!/bin/bash

var1="Devil"
var2=23
readonly var1
echo $var1 $var2

var1=23
echo $var1 $var2
```

```
┌──(root💀Hope)-[~/Desktop]
└─# ./bash1
Devil 23
./bash1: line 8: var1: readonly variable
Devil 23

┌──(root💀Hope)-[~/Desktop]
└─# 
```

Example of Read only Variables.

Now let us see all the above codes in action together. Following is a shell script that includes all the shell variables discussed above.

```
#!/bin/bash
#variable definitions
Var_name="Devil"
Var_age=23

# accessing the declared variables using $
echo "Name is $Var_name, and age is $Var_age."

# read-only variables
var_blood_group="O-"
readonly var_blood_group
echo "Blood group is $var_blood_group and read only."
echo "Error for read only variables, if trying to \
modify them."
echo
var_blood_group="B+"
echo
```