Bash Variable Tutorial – 6 Practical Bash Global and Local Variable Examples

by SASIKALA *on* MAY 3, 2010

This article is part of our on-going bash tutorial series. Like any other programming languages, Bash also provides variables.
Bash variables provide temporary storage for information that will be needed during the lifespan of the program.

Syntax:

```
VARNAME=VALUE
```

Note: There should not be any space around "=" sign in variable assignment. When you use VAR=VALUE, shell sees the "=" as a symbol and treats the command as a variable assignment. When you use VAR = VALUE, shell assumes that VAR must be the name of a command and tries to execute it.
Shell does not care about the type of variables. Variables could store strings, integers, or real numbers.

Example.1 Simple Bash Variable Assignment Usage

The following script creates a variable called LIST and assigns the value "/var/opt/bin". To access the variables, just prefix the variable name with $, which will give you the value stored in that variable.

```
$ cat sample.sh

#!/bin/bash

LIST="/var/opt/bin/"

ls -l $LIST
```

Execute the above script, which will list the /var/opt/bin in long format as shown below.

```
$ ./sample.sh

total 8
```

```
drwxrwsr-x 2 bin  bin 4096 Jan 29 06:43 softwares

drwxr-sr-x 5 root bin 4096 Sep  2  2009 llist
```

Bash Variable Scope – Local and Global

In Bash, variables do not have to be declared. But, when you access the variable which is not used so far, you will not get any warning or error message. Instead, it will display a blank value.

Example 2. Blank values in bash variables

```
$ cat var1.sh

#!/bin/sh

echo "Variable value is: $VAR1"

VAR1="GEEKSTUFF"

echo "Variable value is: $VAR1"



$ ./var1.sh

Variable value is:

Variable value is: GEEKSTUFF
```

As shown above, initially the variable will have a blank value, after assigning, you can get your values. export command is used to export a variables from an interactive shell. export shows the effect on the scope of variables.

Example 3. Bash Variables without export

Assign a variable with a value in an interactive shell, and try to access the same in your shell script.

```
$ VAR2=LINUX



$ cat var2.sh

#!/bin/bash

echo "VAR2=$VAR2"

VAR2=UNIX

echo "VAR2=$VAR2"
```

Now, execute the above script as shown below.

```
$ ./var2.sh

VAR2=

VAR2=UNIX
```

Still you will get blank value for variable VAR2. The shell stores variable VAR2 with the LINUX only in the current shell. During the execution of var2.sh, it spawns the shell and it executes the script. So the variable VAR2 will not have the value in the spawned shell. You need to export the variable for it to be inherited by another program – including a shell script, as shown below.

Example 4. Exporting a Bash Variable

```
$ export VAR2=LINUX



$ cat var2.sh

#!/bin/bash
```

```
echo "VAR2=$VAR2"

VAR2=UNIX

echo "VAR2=$VAR2"
```

Now execute the above script.

```
$ ./var2.sh

VAR2=LINUX

VAR2=UNIX

$

$echo $VAR2

LINUX
```

Now, you can notice that after execution of the shell script var2.sh, the value of VAR2 is LINUX. Because the variables will not be passed back to your interactive shell, unless you execute the script in the current shell.

Declaring a Bash Variable

Using declare statement in bash, we can limit the value assigned to the variables. It restricts the properties of variables. Option in a declare statement is used to determine the type of a variable.

Syntax:

```
declare option variablename
```

- declare is a keyword
- option could be:
  - -r read only variable
  - -i integer variable

- -a array variable
- -f for funtions
- -x declares and export to subsequent commands via the environment.

Example 5. Declaration of Bash variable using declare

```
$ cat declar.sh

#!/bin/bash

declare -i intvar

intvar=123 # Assigning integer value.

echo $intvar

intvar=12.3 #Trying to store string type value to an integer variable

echo $intvar



declare -r rovar=281

rovar=212 # Trying to change the readonly variable.
```

From the below execution, you can notice the error message when you assign invalid data to a variable.

```
$ ./declar.sh

123

t.sh: line 6: 12.3: syntax error: invalid arithmetic operator (error token is ".3")

123

t.sh: line 11: rovar: readonly variable
```

Global Bash Variables

Global variables are also called as environment variables, which will be available to all shells. printenv command is used to display all the environment variables.

```
$ printenv

SHELL=/bin/bash

HISTSIZE=1000

SSH_TTY=/dev/pts/1

HOME=/root

LOGNAME=root

CVS_RSH=ssh
```

Local Bash Variables

Local variables are visible only within the block of code. local is a keyword which is used to declare the local variables. In a function, a local variable has meaning only within that function block.

Example 6. Global and Local Bash Variables

```
$ cat localvar.sh

#!/bin/bash

pprint()

{

  local lvar="Local content"

  echo -e "Local variable value with in the function"

  echo $lvar
```

```
  gvar="Global content changed"
  echo -e "Global variable value with in the function"
  echo $gvar
}


gvar="Global content"
echo -e "Global variable value before calling function"
echo $gvar
echo -e "Local variable value before calling function"
echo $lvar
pprint
echo -e "Global variable value after calling function"
echo $gvar
echo -e "Local variable value after calling function"
echo $lvar
```

Execute the above script,

```
$ sh t.sh
Global variable value before calling function
Global content
Local variable value before calling function
```

```
Local variable value with in the function

Local content

Global variable value with in the function

Global content changed

Global variable value after calling function

Global content changed

Local variable value after calling function
```

In the above output, local variables will have only empty value before and after calling the function. Its scope is only with in the function. It got vanished out of the function, whereas the global variable has the updated value even after the function execution.