Bash printf – How to Print a Variable in Bash

February 24, 2022

BASH

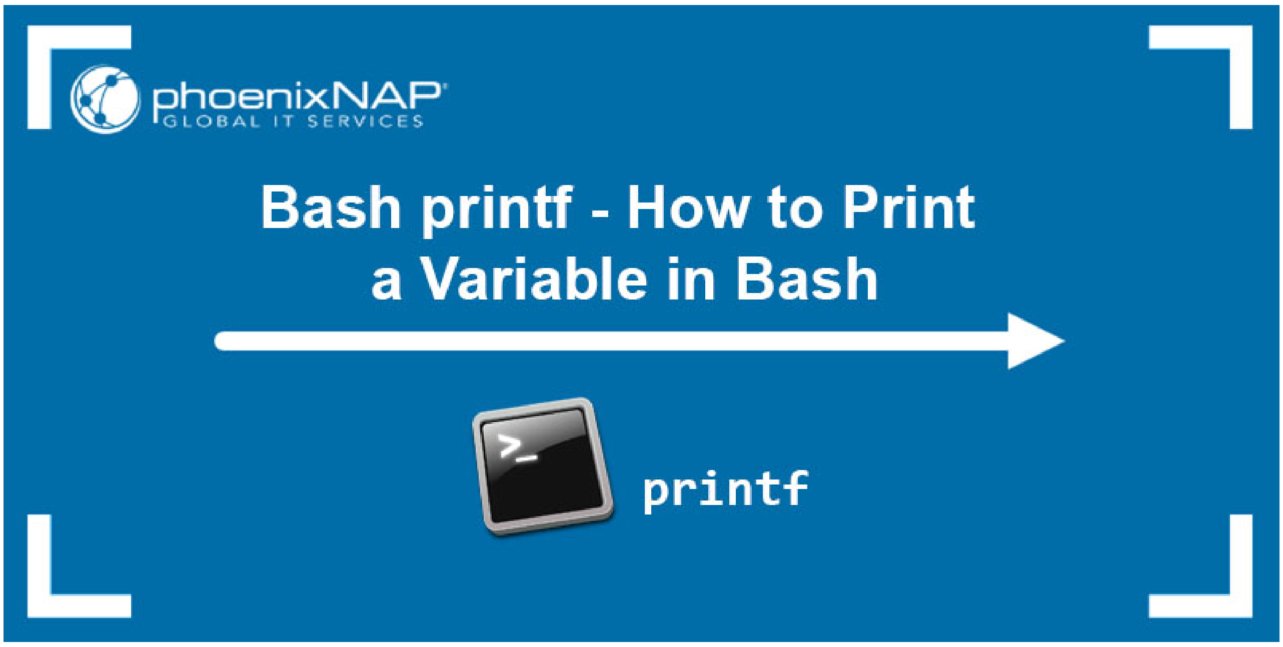
Home » DevOps and Development » Bash printf - How to Print a Variable in Bash

Introduction

The bash **`printf`** command is a tool used for creating formatted output. It is a shell built-in, similar to the **`printf()`** function in C/C++, Java, PHP, and other programming languages. The command allows you to print formatted text and variables in standard output.

Use **`printf`** in bash scripts to increase efficiency and reusability when programming.

In this tutorial, you will learn to use the `printf` command in Linux to print variables and format the output.



Prerequisites

- A machine running Linux
- Access to a terminal (**Ctrl** + **Alt** + **T**)

Note: Another way to print variables and strings in the terminal is to use the echo command. However, while **`echo`** has few or no formatting options, **`printf`** facilitates and offers many text formatting options.

Bash printf Syntax

The **`printf`** command accepts the following syntax:

```
printf [-v var] [format specifiers] [arguments]
```

`[-v var]`

The optional **`-v`** flag assigns the output to the **`[var]`** variable instead of printing it in standard output.

`[format specifiers]`

Format specifiers are strings that determine the methods of formatting specifiers. The following section includes a list of accepted specifiers.

`[arguments]`

Arguments can be any value or variable, and the `[format specifiers]` point to the `[arguments]`. If there are more arguments than format specifiers, the format string is reused until it interprets the last argument.

If there are fewer format specifiers than arguments, number formats are set to zero (`0`), while string formats are set to `null` (empty).

Note: Learn more about the awk command, a Linux tool for data processing and manipulating that produces formatted reports.

printf Specifiers

The following table shows common `printf` format specifiers and their descriptions:

Format Specifier	Description
`%c`	Treat the arguments as a single character.
`%d`	Treat the input as a decimal (integer) number (base 10).
`%e`	Treats the input as an exponential floating-point number.
`%f`	Treat the input as a floating-point number.
`%i`	Treat the input as an integer number (base 10).
`%o`	Treats the input as an octal number (base 8).
`%s`	Treat the input as a string of characters.
`%u`	Treat the input as an unsigned decimal (integer) number.
`%x`	Treats the input as a hexadecimal number (base 16).
`%%`	Print a percent sign.
`%Wd`	Print the **W** integer **X** digits wide.

Format Specifier	Description
%(format)T	Outputs a date-time string resulting from using format as a format string for **strftime.** The corresponding argument can be the number of seconds since Epoch (January 1, 1970, 00:00), **-1** (the current time), or **-2** (shell startup time). Not specifying an argument uses the current time as the default value.
\%	Print a percent sign.
\n	Prints a **newline** character.
\t	Print a **tab** character.

Some format specifiers accept format modifiers that modify their actions. Enter the modifiers between the **%** character and the character that specifies the format.

Available format modifiers are:

- **<N>.** Enter a number that specifies a minimum field width. If the output text is shorter, it's padded with spaces. If the text is longer, the field expands.
- **. (dot).** When used with a field width modifier, the field doesn't expand for longer text. Instead, the text is truncated.
- **-.** Left-aligns the printed text. The default alignment is right.
- **0.** Pads the numbers with zeros instead of spaces.
- **<space>.** Pads a positive number with a space, and a negative number with a minus (**-**).
- **+.** Prints all numbers signed (**+** for positive, **-** for negative).
- **'.** For decimal conversions, applies the thousands grouping separator to the integer portion of the output according to the current *LC_NUMERIC* file.

Note: Learn to set environment variables in Linux.

Bash printf Examples

The examples below demonstrate common use cases and different ways of formatting output using **`printf`**.

While the command can print variables in the terminal, it is more beneficial to use it in bash scripts as they allow code reuse and facilitate automation.

Separate Output with a Newline Character

Unlike **`echo`**, using **`printf`** to print a string or variable without any format specifiers outputs the unformatted text in a single line. To add a newline character and separate the entries, specify the **`\n`** format specifier:

A newline character now separates the printed output.

Print Multiple Variables

Use **`printf`** to print multiple variables in standard output. Follow the steps below:

1. Open the terminal (**Ctrl** + **Alt** + **T**) and create a bash script using a text editor such as **vi/vim**:

```
vi script.sh
```

2. Enter the following lines:

```
#!/bin/bash
printf "Enter your name:\n"
read name
printf "Enter your surname:\n"
read surname
printf 'Welcome: %s\n' "$name $surname"
```

The **`%s`** in the format string points to the variables defined in the previous lines, while **`\n`** instructs **`printf`** to add a newline character.

3. Save the script and exit **`vi`** by entering:

```
:wq
```

4. Run the script:

```
bash script.sh
```

5. Input the requested values for the two variables and check the result:

The **`printf`** command stores the input in the two variables and then prints the variables in standard output.

Print Variable in a Mock Table

Automatically add dynamic padding or change the variable output alignment using **`printf`**. Follow the steps below to print a variable in a mock table:

1. Create a script file:

```
vi table.sh
```

2. Enter the following lines and save the script:

```
#!/bin/bash
var='hello'
printf '|%10s|\n' "$var"
```

The **`%10`** ensures there are ten characters within the pipe symbols, padding the variable output with whitespace characters to that length. The **`s`** treats the input variable as a string.

3. Run the script and check the result:

```
bash table.sh
```

The command outputs the variable encased in pipe symbols and aligns it to the right by default. To align the output to the left side, add the `-` modifier after the `%` character in the format string:

```
#!/bin/bash
var='hello'
printf '|%-10s|\n' "$var"
```

Add Decimal Points

Use **`printf`** with the `%f` format specifier to treat the input as a floating-point number and add the precision of two decimal places. Follow the steps below:

1. Create a bash script:

```
vi precision.sh
```

2. Enter the following lines and save the script:

```
#!/bin/bash
printf "Enter a decimal number:\n"
read number
printf "%5.2f\n" "$number"
```

3. Run the script:

```
bash precision.sh
```

The output prints the variable as a floating-point number rounded to two decimal places.

Store Data in a Variable

Use the **`-v`** flag with **`printf`** to assign a shell variable to store the data instead of printing it in standard output.

Follow the steps below:

1. Create a bash script:

```
vi variable.sh
```

2. Enter the following lines and save the script:

```
#!/bin/bash
printf -v var 'Welcome'
printf '%s\n' "$var"
```

3. Execute the script:

```
bash variable.sh
```

The output prints the stored variable contents.

Note: Read our tutorial to learn everything about bash functions.

Convert Hexadecimal Numbers

The **%d** format specifier instructs **printf** to treat the input as a decimal (integer) number. Use the specifier to create a script that converts hexadecimal numbers to decimal ones.

Follow the steps below:

1. Create a bash script:

```
vi convert.sh
```

2. Enter the following lines and save the script:

```
#!/bin/bash
printf 'Enter a hexadecimal number:\n'
read number
printf "%d\n" "$number"
```

3. Run the script:

```
bash convert.sh
```

The script converts the input hexadecimal number to a decimal one.

Convert Decimal Numbers to Octal

Use the **%o** format specifier to convert decimal numbers to octal numbers. Follow the steps below:

1. Create a script:

```
vi octal.sh
```

2. Enter the following lines and save the script:

```
#!/bin/bash
printf "Enter a decimal number:\n"
read number
printf "The octal number of "$number" is %o\n" "$number"
```

3. Run the script:

```
bash octal.sh
```

The script converts the inputted decimal value to an octal one.

Print Date

Create a script that utilizes **`printf`** to display the current date. Follow the steps below:

1. Create a script:

```
vi datetime.sh
```

2. Enter the following lines and save the script:

```
#!/bin/bash
printf 'Today is %(%F)T\n' -1
```

The **`%F`** treats the input as a floating-point number, while **`-1`** indicates the current date.

3. Run the script to check if it works:

```
bash datetime.sh
```

The output states the current date and time.

Add Commas to Thousands

When dealing with numbers, use the `'` modifier to make the output more readable by adding commas to denote thousands. Follow the steps below:

1. Create a bash script:

```
vi decimals.sh
```

2. Enter the following lines:

```
#!/bin/bash
printf "Enter a number:\n"
read number
printf "%'d\n" $number
```

3. Run the script:

```
bash decimals.sh
```

After entering a number, the script utilizes **`printf`** to add the commas to the input value automatically.

Get Character ASCII Code

Create a script that uses **`printf`** to output an input character's **ASCII** value. Follow the steps below:

1. Create a script:

```
vi character.sh
```

2. Enter the following lines and then save the script:

```
#!/bin/bash
printf "Enter a character to get character code:\n"
read character
printf "%d\n" "'$character"
```

3. Execute the script and test if it works:

```
bash character.sh
```

The script outputs the input character's ASCII value.

Conclusion

This guide showed how to use the **`printf`** tool to print a variable in Bash and format the output using the available specifiers.

Next, learn to use the bash read command to split a file or input into different fields.

Was this article helpful? Yes No

Bosko Marijan

Having worked as an educator and content writer, combined with his lifelong passion for all things high-tech, Bosko strives to simplify intricate concepts and make them user-friendly. That has led him to technical writing at PhoenixNAP, where he continues his mission of spreading knowledge.

Next you should read

SysAdmin

How to Use the Bash let Command {with Examples}

January 20, 2022

The Bash let command is a built-in utility used for evaluating arithmetic expressions. Learn how to use the Bash let command in this tutorial.

READ MORE

SysAdmin

Bash Export Variable

December 30, 2021

All the variables the user defines inside a shell are local by default. It means that child processes of the shell do not inherit the values of the shell's variables. This tutorial will show you how to export Bash variables.

READ MORE

SysAdmin

Bash declare Statement: Syntax and Examples

December 23, 2021

Although the builtin declare statement does not need to be used to explicitly declare a variable in Bash, the command is often used for more advanced variable management tasks. This tutorial shows how to use the declare command.

DevOps and Development, SysAdmin

Bash case Statement Syntax and Examples

December 15, 2021

The bash case statement simplifies complex conditional statements and tests input strings against specified patterns to execute a command when the condition is met. This tutorial shows how to use the bash case statement.

READ MORE

Live Chat Get a Quote Support | 1-855-330-1509 Sales | 1-877-588-5918

Contact Us
Legal
Privacy Policy
Terms of Use
DMCA
GDPR
Sitemap