Bash HereDoc Tutorial With Examples

By Milica Dancuk - Published: March 3, 2022

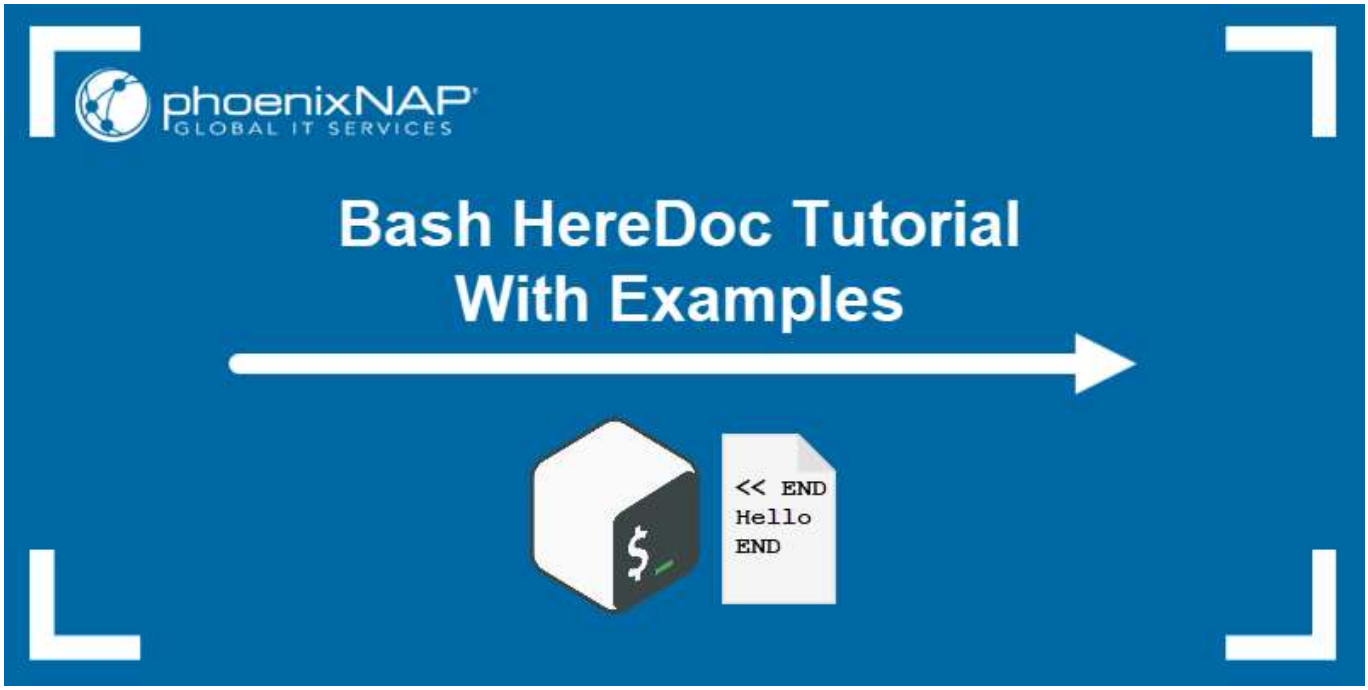
Topics: Bash, Linux

Introduction

A here document (**HereDoc**) is a section of code that acts as a separate file. A HereDoc is a multiline string or a file literal for sending input streams to other commands and programs.

HereDocs are especially useful when redirecting multiple commands at once, which helps make Bash scripts neater and easier to understand.

This article teaches you the basics of using HereDoc notation and some typical use cases.



Prerequisites

- Access to the command line/terminal as a sudo user.
- A text editor to write Bash scripts.
- Basic Linux commands. For a quick reference, grab our Linux commands cheat sheet.

Bash HereDoc Syntax

The syntax for writing a HereDoc is:

```
[COMMAND] <<[-] 'DELIMITER'
  Line 1
  Line 2
  ...
DELIMITER
```

Copy

It consists of the following elements:

- `COMMAND` is optional. Works for any command that accepts redirection.
- `<<` is the redirection operator for forwarding a HereDoc to the `COMMAND`.
- `-` is a parameter for tab suppression.
- `DELIMITER` in the first line defines a HereDoc delimiter token. `END`, `EOT`, and `EOF` are most common, but any multicharacter word that won't appear in the body works. Omit single quotes on the first line to allow command and variable expansion.
- The `DELIMITER` in the last line indicates the end of a HereDoc. Use the same word from the first line without the leading whitespaces.

The HereDoc itself contains any number of lines with **strings**, variables, commands, and other inputs.

Bash HereDoc Examples

This section showcases how to use the HereDoc notation in various situations. The most common use case is with the **cat command**.

Multiline String

Open the terminal and enter the following text, pressing **Enter** after each line:

```
cat << EOF
Hello
World
EOF
```

Copy

The `cat` command reads the HereDoc and writes the contents to the terminal.

Variable Expansion

A HereDoc accepts the use of variables and reads them.

To see how this works, create two variables in the terminal:

```
var1="Hello"
```
Copy

```
var2="World"
```
Copy

Pass the HereDoc to a `cat` command to print the two variables along with an **environment variable**:

```
cat << END
$var1
$var2
$PWD
END
```
Copy

All the variables expand, and their respective values print to the terminal.

Command Expansion

HereDocs accept command substitution. Run the following code in the terminal line by line to see the results:

```
cat << EOF
$(echo Hello)
$(whoami)
EOF
```

Copy

Encompass each command in `$()` to evaluate a statement and fetch the results. Omitting `$()` treats the text as a string.

Ignore Variable and Command Expansion

Add single or double quotes to the first delimiter to ignore variable and command expansion in a HereDoc.

For example:

```
cat << "EOF"
$(echo Hello)
$(whoami)
$PWD
EOF
```

Copy

Adding quotes to the delimiter treats the contents as a HereDoc literal.

Piping and Redirecting

Use piping or redirecting to forward the command results to another command. For example, create a Bash script and add the following contents to pipe a command:

```
#!/bin/bash

cat << EOF | base64 -d
```

Copy

```
SGVsbG8KV29ybGQK
EOF
```

Alternatively, use redirect notation to achieve the same result:

```
#!/bin/bash

(base64 -d) < cat << EOF
SGVsbG8KV29ybGQK
EOF
```

Copy

Run the Bash script to see the results.

In both cases, the output from the `cat` and a HereDoc command pipes (or redirects) to the `base64 -d` command. As a result, the script decodes the message from the HereDoc.

Write to File

HereDoc allows writing multiline documents through one command.

To create a file and save the HereDoc contents, use the following format:

```
cat << EOF > hello_world.txt
Hello
World
EOF
```

Copy

If the document does not exist, the command creates it. Check the file contents to confirm:

```
cat hello_world.txt
```

Copy

The console shows the file contents.

Note: Learn different methods on how to **write to file in Bash**.

Tab Suppression

Add a dash (-) after redirection to suppress leading tabs. For example, create and run the following script:

```
#!/bin/bash

cat <<- EOF
        Hello
        World
EOF
```

Copy

Without tab suppression, the message prints to the console with indentation. Adding the dash removes the tab indent and outputs the message without the leading spaces.

Note: If the text shows up with the leading spaces, press **TAB** instead of copying and pasting the example code.

Inside Statements and Loops

When working with a HereDoc inside statements and loops, keep in mind the following behavior:

- **Code inside statements and loops** is indented. Add a dash after the redirect operator to print messages from a HereDoc without indentation.
- **The ending delimiter** cannot have spaces or indentations before it.

Try the following example code to see how to use a HereDoc inside an **if statement**:

```
#!/bin/bash

if true;
then
        cat <<- "END"
        Hello
        World
END
fi
```

Copy

The dash ensures the indents don't show up when the program runs. The ending delimiter is not indented, and adding spaces causes an error.

Multiline Comments

A HereDoc with the **null command (`:`)** creates the effect of **block comments** in Bash scripts.

For example:

```
#!/bin/bash

: << 'END'
```

Copy

```
This is a comment
END
```

Using HereDoc notation as a block comment is unconventional. In general, Bash does not support block commenting.

Escape Characters

To avoid character interpretation, add a backslash (\) before a character:

```
cat << EOF
\$100
EOF
```

Alternatively, avoid character interpretation completely by escaping the delimiter:

```
cat << \EOF
$100
EOF
```

Using quotation marks on the delimiter is equivalent in this case.

Functions

Add parameters to a **function** by forwarding information through a HereDoc. For example, create a function to **read** lines and add information through the HereDoc:

```
#!/bin/bash

readLines(){
        read greeting
        read name
}

readLines << EOF
Hello
$USER
EOF

echo $greeting
echo $name
```

The function stores the information provided by the HereDoc into variables.

Run the script to print the variable values to the terminal.

HereDoc and SSH

A HereDoc is convenient for executing multiple commands on a remote machine. Pass a HereDoc to the **SSH connection** to run multiple commands.

For example:

```
ssh username@host << EOF
echo "Local user: $USER"
echo "Remote user: \$USER"
EOF
```

Copy

The command prints the local and remote users to the console.

HereDoc and SFTP

SFTP helps transfer data securely via the **SSH protocol**. Forward a HereDoc to run multiple **SFTP commands** automatically:

```
sftp username@host << EOF
put test.sh
EOF
```

Copy

The code uploads a sample file to the remote machine.

Note: Learn how to **read files line by line in Bash** using `here` strings.

Conclusion

After going through the examples in this guide, you know how to use HereDoc notation in various situations. HereDoc helps pass multiple commands at once as input for different commands.

Next, learn about advanced text processing with the **AWK command**.

Next you should read

DevOps and Development, SysAdmin

Bash printf - How to Print a Variable in Bash

DevOps and Development, SysAdmin

How to Write a Bash Script with Examples

DevOps and Development

How to Comment in Bash

Contents

Subscribe to our newsletter

SUBSCRIBE

CONTACT US

- Get a Quote
- Support (1-855-330-1509)
- Sales (1-877-588-5918)

COLOCATION

Phoenix

Ashburn

Amsterdam

Atlanta

Belgrade

Singapore

Shipping Instructions

RECENT POSTS

How to Install Vim on Ubuntu

Remote Desktop Connection from Mac to Ubuntu

How to Install .deb Files (Packages) on Ubuntu

How to Install Nmap on Ubuntu

How to Install Jenkins on Ubuntu

SERVERS

Bare Metal Cloud

Dedicated Servers

Database Servers

Virtualization Servers

High Performance Computing (HPC) Servers

Dedicated Streaming Servers

Dedicated Game Servers

Dedicated Storage Servers

SQL Server Hosting

Dedicated Servers in Amsterdam

Cloud Servers in Europe

CLOUD SERVICES

Data Security Cloud

Managed Private Cloud

Object Storage

SOLUTIONS

Disaster Recovery

Web Hosting Reseller

SaaS Hosting

COMPLIANCE

HIPAA Ready Hosting

Big Memory Infrastructure

BUY NOW

BMC PORTAL NEEDS

Disaster Recovery Solutions

High Availability Solutions

Cloud Evaluation

INDUSTRIES

Web Hosting Providers

Legal

MSPs & VARs

Media Hosting

Online Gaming

SaaS Hosting Solutions

Ecommerce Hosting Solutions

PCI Compliant Hosting

Privacy Center

Do not sell or share my personal information

COMPANY

About phoenixNAP

IaaS Solutions

Customer Experience

Platform

Schedule Virtual Tour

Open Source Community

Resource Library

Press

Events

Careers

PROMOTIONS