Bash Heredoc

Posted May 9, 2019 • 4 min read

When writing shell scripts you may be in a situation where you need to pass a multiline block of text or code to an interactive command, such as `tee` , `cat` , or `sftp` .

In Bash and other shells like Zsh, a Here document (Heredoc) is a type of redirection that allows you to pass multiple lines of input to a command.

The syntax of writing HereDoc takes the following form:

```
[COMMAND] <<[-] 'DELIMITER'
  HERE-DOCUMENT
DELIMITER
```

- The first line starts with an optional command followed by the special redirection operator `<<` and the delimiting identifier.
  - You can use any string as a delimiting identifier, the most commonly used are EOF or END.
  - If the delimiting identifier is unquoted, the shell will substitute all variables, commands and special characters before passing the here-document lines to the command.
  - Appending a minus sign to the redirection operator `<<–` , will cause all leading tab characters to be ignored. This allows you to use indentation when writing here-documents in shell scripts. Leading whitespace characters are not allowed, only tab.
- The here-document block can contain strings, variables, commands and any other type of input.
- The last line ends with the delimiting identifier. White space in front of the delimiter is not allowed.

Basic Heredoc Examples

In this section, we will look at some basic examples of how to use heredoc.

Heredoc is most often used in combination with the cat command .

In the following example, we are passing two lines of text containing an environment variable and a command to `cat` using a here document.

```
cat << EOF
The current working directory is: $PWD
You are logged in as: $(whoami)
EOF
```

As you can see from the output below, both the variable and the command output are substituted:

```
Output

The current working directory is: /home/linuxize
You are logged in as: linuxize
```

Let's see what will happen if we enclose the delimiter in single or double quotes.

```
cat <<- "EOF"
The current working directory is: $PWD
You are logged in as: $(whoami)
EOF
```

You can notice that when the delimiter is quoted no parameter expansion and command substitution is done by the shell.

```
Output

The current working directory is: $PWD
You are logged in as: $(whoami)
```

If you are using a heredoc inside a statement or loop, use the `<<-` redirection operation that allows you to indent your code.

```
if true; then
    cat <<- EOF
    Line with a leading tab.
    EOF
fi
```

Instead of displaying the output on the screen you can redirect it to a file using the `>`, `>>` operators.

```
cat << EOF > file.txt
The current working directory is: $PWD
You are logged in as: $(whoami)
EOF
```

If the file.txt doesn't exist it will be created. When using `>` the file will be overwritten, while the `>>` will append the output to the file.

The heredoc input can also be piped. In the following example the `sed` command will replace all instances of the `l` character with `e`:

```
cat <<'EOF' |  sed 's/l/e/g'
Hello
World
EOF
```

```
Output
Heeeo
Wored
```

To write the piped data to a file:

```
cat <<'EOF' |  sed 's/l/e/g' > file.txt
Hello
World
EOF
```

Using Heredoc with SSH

Using Heredoc is one of the most convenient and easiest ways to execute multiple commands on a remote system over SSH .

When using unquoted delimiter make sure you escape all variables, commands and special characters otherwise they will be interpolated locally:

```
ssh -T user@host.com << EOF
echo "The current local working directory is: $PWD"
echo "The current remote working directory is: \$PWD"
EOF
```

```
Output
The current local working directory is: /home/linuxize
The current remote working directory is: /home/user
```

You may also want to set up an SSH key-based authentication and connect to your Linux servers without entering a password.

Conclusion

In this guide, you have learned what is `heredoc` and how to use it in your shell scripts.

If you have any questions or feedback, feel free to leave a comment.

bash terminal

Related Articles

DEC 21, 2019

Bash: Append to File

OCT 2, 2019

Writing Comments in Bash Scripts

SEP 21, 2019

Pushd and Popd Commands in Linux

Write a comment