



Architecture & Deployment

2025-2026 v0.1.0 on branch main Rev: d1f684699a511826485586b2ea383cf021631add

Shell Scripting

Learn the basics of shell scripting with Bash.

Table of contents

- [What is a script?](#)
 - [How is a script executed?](#)
 - [What can I put in a script?](#)
 - [How do I create a script?](#)
 - [All kinds of scripts](#)
- [What is shell scripting?](#)
- [Shell script basics](#)
 - [Commands](#)
 - [Working directory](#)
 - [Variables](#)
 - [Store the output of commands](#)
 - [Environment variables](#)
 - [Conditionals](#)
 - [The test built-in command](#)
 - [Loops](#)
 - [Special variables](#)
 - [The set built-in command](#)
 - [Functions](#)
 - [Variable scope](#)
- [References](#)

You will need

- A Unix CLI

Recommended reading

- [Command Line Introduction](#)

What is a script?

In a Unix-like operating system, a file that can be executed should be one of the following:

- A **binary file**, which contains machine-readable binary code that has been compiled from source code.
- A **script**, which is a file containing code that is dynamically interpreted.

How is a script executed?

When an executable text file is run, a Unix-like operating system looks for a **shebang** on the first line. A shebang is a line with the following format:

```
#!/interpreter optional-args
```

For example, the following is a valid shebang:

```
#!/bin/bash
```

In this example, it tells the operating system that the **interpreter** which should run this file is `/bin/bash`, meaning that this is a Bash script.

Note

Note that there must not be any space between `#!` and the path of the interpreter `/bin/bash`.

What can I put in a script?

In a bash script, you can put anything you could type in a Bash shell:

```
#!/bin/bash  
echo Hello World
```

In a PHP script, you can put any PHP code you want:

```
#!/usr/bin/php  
<?php  
echo 'Hello World';  
?>
```

Basically, what you can put in a script depends on the interpreter you're using.

How do I create a script?

Simply create your script:

```
$> printf '#!/bin/bash\n echo Hello World' > test.sh
```

More information

The `printf` (**print** format) command is similar to the `echo` command but it has better support for special characters like new lines (`\n`).

Make it executable:

```
$> chmod +x test.sh
```

And run it:

```
$> ./test.sh  
Hello World
```

All kinds of scripts

The following are a few examples of shebangs, but it is nowhere near exhaustive:

Shebang	Script contents
<code>#!/bin/sh</code>	<u>Bourne shell</u> commands
<code>#!/bin/bash</code>	<u>Bash shell</u> commands
<code>#!/bin/zsh</code>	<u>Z shell</u> commands
<code>#!/usr/bin/node</code>	<u>Node.js</u> code
<code>#!/usr/bin/php</code>	<u>PHP</u> code
<code>#!/usr/bin/python</code>	<u>Python</u> code
<code>#!/usr/bin/ruby</code>	<u>Ruby</u> code

Note

Of course, the path to the interpreter must correspond to the actual path of the command used (`sh`, `bash`, `php`, etc). It might differ on your machine. Use `which bash` to find the location of the Bash executable, for example.

What is shell scripting?

Shell scripting is the practice of writing scripts that contain series of shell commands that you want to be able to reuse.

Any script with a shell as the interpreter is a “shell script”.

A script using PHP as the interpreter is still a script, but it’s not a “shell script”. It’s a PHP script.

Shell script basics

A few pointers on writing Bash scripts (compatible with most POSIX shells).

Commands

You can use any shell command in a shell script:

```
#!/bin/bash
echo Hello World
date
ls
```

This script could print:

```
Hello World
Thu Jan 10 23:46:52 CET 2019
file.txt directory ...
```

Working directory

By default, a script executes in the current shell directory.

You can use `cd` to move around to other directories:

```
#!/bin/bash  
pwd  
cd /home  
pwd
```

This script could print:

```
/some/where/over/the/rainbow  
/home
```

Assuming it was executed from the `/some/where/over/the/rainbow` directory.

Variables

You can declare and reuse variables in scripts:

```
#!/bin/bash  
F00=bar  
echo $F00
```

If your variable contains whitespace (spaces, new lines, etc), be sure to quote it when declaring and using it to avoid issues:

```
#!/bin/bash  
F00="bar baz"  
echo "$F00"
```

Store the output of commands

You can store the result of a command in a variable by wrapping it with backticks:

```
#!/bin/bash
FILES=`ls -l`
NUMBER_OF_FILES=`echo "$FILES" | wc -l`
echo There are $NUMBER_OF_FILES files
```

This script would output 10 if there are 10 files in the current directory.

Environment variables

Environment variables are also available as variables in shell scripts:

```
#!/bin/bash
echo $PATH
```

To set an environment variable, do it like you would in any Bash shell:

```
#!/bin/bash
export F00=bar
```

Note

The `$F00` environment variable in this example will only be set in the context of this script and its child processes.

Conditionals

Bash has a classic `if/then/else` construct:

```
#!/bin/bash
F00="bar"

if [[ "$F00" -eq "foo" ]]; then
    echo F00 is foo
```

```
elif [[ "$F00" -eq "bar" ]]; then
    echo F00 is bar
else
    echo foo is something else
fi
```

More information

The `[[ ]]` syntax is a Bash [test construct](#). Also see Bash [other comparison operators](#).

The `test` built-in command

The `test` command which comes with Bash is another way to write some conditions:

```
#!/bin/bash
```

```
EMPTY_VAR=
```

```
FULL_VAR="full"
```

```
FILE="/path/to/some/file"
```

```
if test -z "$EMPTY_VAR"; then
    echo variable is empty
fi
```

```
if test -n "$FULL_VAR"; then
    echo variable is not empty
fi
```

```
if test -f "$FILE"; then
    echo file exists
else
    echo file does not exist
fi
```

More information

See Bash [file test operators](#) and [other comparison operators](#).

Loops

Bash has a `for` loop:

```
for item in one two three; do
    echo $item
done
```

The above code would print:

```
one
two
three
```

More information

Bash also has `while` and `until`. See [loops & branches](#).

Special variables

Bash has a number of [special variables](#) which are always available:

Variable	Description
<code>\$0</code>	Name of the command being executed.
<code>\$1</code>	First argument passed to the script on the command line (and so on with <code>\$2</code> , <code>\$3</code> , etc).
<code>\$@</code>	All arguments passed to the script.

Variable	Description
----------	-------------

\$?	Exit value of the last executed command.
-----	--

For example, this script says hello to the name passed as the first argument:

```
#!/bin/bash
echo Hello $1
```

The `set` built-in command

The `set` command is specific to Bash and can be used to toggle its option flags.

For example, the `-e` option aborts the script if an error occurs, while the `-x` option prints commands before executing them:

```
#!/bin/bash
set -ex
echo Hello World
cat file-that-does-not-exist
echo Done
```

This script could print:

```
+ echo Hello World
Hello World
+ cat file-that-does-not-exist
cat: file-that-does-not-exist: No such file or directory
```

Note that each command is printed with a leading `+` before being executed, and that the script stops as soon as an error occurs (which is **not the case by default**).

Functions

You can isolate pieces of code in a function. The special argument variables `$1`, `$2`, etc represent the arguments to the function:

```
#!/bin/bash

print_hello() {
    echo Hello $1
}

print_hello World
```

This script would print `Hello World`.

Variable scope

Note that normal Bash variables have no scope, i.e. they are available in the whole file and every function.

To declare a variable that is local to a function, use the `local` keyword:

```
#!/bin/bash

print_hello() {
    local name=$1
    echo Hello $name
}

print_hello World

echo $name
```

This script would print `Hello World` and an empty line, since `$name` is only defined within the `print_hello` function.

References

- [Shell Style Guide](#)
- [Advanced Bash Scripting Guide](#)
 - [Test Constructs](#)
 - [File Test Operators](#)
 - [Other Comparison Operators](#)
 - [Loops & Branches](#)
 - [Local Variables](#)
 - [Special Shell Variables](#)
 - [Starting Off With a Sha-Bang](#)
- [Shell Script](#)

[↑](#) Back to top