# Bash Shebang

Posted Jul 23, 2019 • 3 min read


If you are learning Bash scripting by reading other people's code you might have noticed that the first line in the scripts starts with the `#!` characters and the path to the Bash interpreter.

This sequence of characters ( `#!` ) is called **shebang** and is used to tell the operating system which interpreter to use to parse the rest of the file.

## Shebang Interpreter Directive #

The Shebang interpreter directive takes the following form:

```
#!interpreter [arguments]
```

- The directive must be the first line in the script.
- The directive must start with shebang `#!`
- White space after the shebang characters is optional.
- Interpreter is the full path to a binary file (ex: `/bin/sh` , `/bin/bash` ).
- Interpreter arguments are optional.

- `#!/bin/bash` - Uses `bash` to parse the file.
- `#!/usr/bin/env perl` - Uses the `env` command to find the path to the `perl` executable.
- `#!/usr/bin/python` Executes the file using the `python` binary.

## Using Shebang in Bash Scripts

If a shebang is not specified and the user running the Bash script is using another Shell the script will be parsed by whatever the default interpreter is used by that Shell. For example, the default interpreter for `bash` is `bash` and for `zsh` is `sh`. To ensure that your script will always be interpreted with Bash you'll need to specify the executable path using shebang.

There are two ways to use the Shebang directive and set the interpreter.

01. Using the absolute path to the bash binary:

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

02. Using the `env` utility:

```
#!/usr/bin/env bash
```

The advantage of using the second approach is that it will search for the `bash` executable in the user's `$PATH` environmental variable. If there are more than one paths to `bash`, the first one will be used by the script.

When using the first option to add an option to the Bash shell supply pass it to the interpreter. For example, to run the script in a debug mode you would use `#!/bin/bash -x`. If you are using the `env` method then you need to use `set` to declare the option. To enable the debug mode you would add `set -x` after the shebang line.

## Example Script

Let's create a simple script using shebang that will print "Hello World". Open your text editor and

```
$ nano hello_world
```

hello_world

```
#!/bin/bash

echo "Hello, World"
```

To be able to run the script without specifying the interpreter from the command line you'll need to make the file executable :

```
$ chmod +x hello_world
```

Now if you can run the script by typing `./` followed by the script name:

```
$ ./hello_world
```

Output

```
Hello, World
```

## Overriding the Shebang

If for some reason you want to override the interpreter set in the Shebang line you need to run the script by explicitly specifying the wanted shell.

For example to run a script that has `#!/bin/sh` specified in the Shebang line using the `bash` shell you would type:

```
$ bash hello_world
```

Please note, it is not a good idea to override the shell interpreter as it may lead to unexpected behavior of the script.

## Conclusion

By now you should have a good understanding of what is Shebang and how to use it in your Bash scripts.