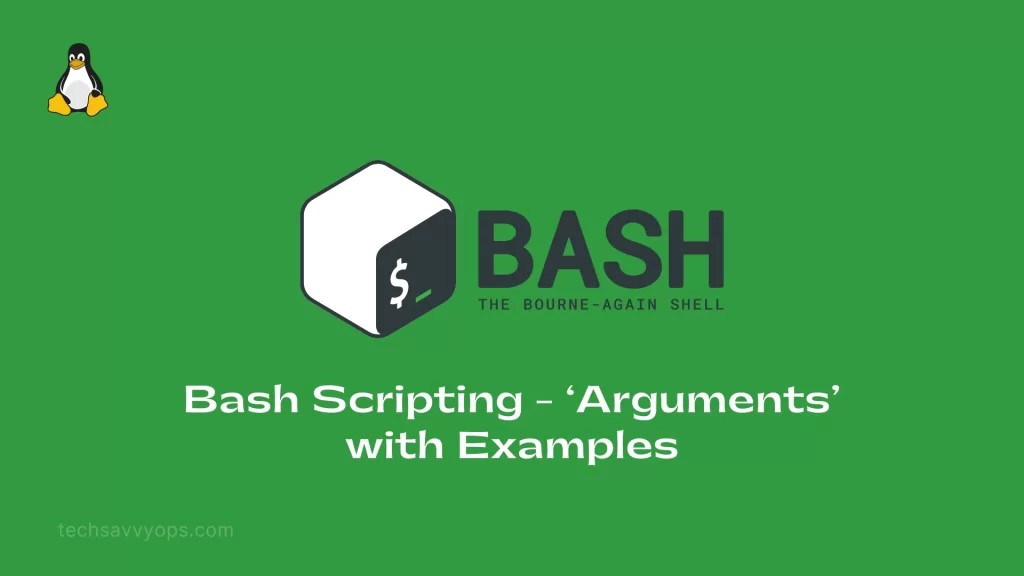
**Arguments in Bash**

[**Pranav T**](https://techsavvyops.com/author/pranavtadakamalla/)



Bash scripting is a cornerstone of automation and system management. A powerful feature of Bash is the ability to pass command-line arguments to scripts, making them versatile and dynamic. This blog will explain how arguments work in Bash using a practical example.

# The Example Script

Here’s a script that takes a directory name as an argument and counts the number of objects ( les and subdirectories) it contains:

1 /bin/bash

2

3 lines=$(ls l $1 | wc l)

4

1. if [ $# ne 1 ]
2. then

## 7 echo "Enter one directory name please try again" 8 exit 1

9 f

10

11 echo "There are $(($lines-1)) objects in $1"

# How the Script Works

$1 represents the first argument passed to the script, which should be the name of a directory.

ls -l $1 lists the directory contents in long format.

wc -l counts the number of lines in the output.

The script subtracts one from the total line count to exclude the header line from ls -l.

if [ $# -ne 1 ] checks if exactly one argument is provided ($# represents the number of

arguments).

If not, the script prints an error message and exits with exit 1 to indicate failure.

If the correct number of arguments is supplied, the script prints the number of objects in the specified directory.

# Usage

1. Save the script to a le, such as count\_objects.sh.
2. Make it executable

## 1 chmod x count\_objects.sh

1. Run the script with a directory name as an argument
   1. ./count\_objects.sh /var
2. If successful, it outputs
   1. There are X objects in /var
3. If no or multiple arguments are provided, it displays
   1. Enter one directory name please try again

# Special Variables

$0: The name of the script itself.

$1, $2, …: Positional parameters representing the first, second, and subsequent arguments.

$#: The number of arguments passed to the script.

$\* or “$@”: All arguments passed to the script.

# Final thoughts

Command-line arguments make Bash scripts flexible and powerful. By understanding how to use special variables and validate input, you can create robust scripts tailored to different tasks. Try enhancing the example script to further improve your Bash scripting skills.