2025/11/02 1/7 bash\_scripts

# **Bash Scripting**

## **Table of Contents**

- Basics
  - Exit codes
  - Variables and Data Types
  - Command substitution
  - Input/output redirection
- Control Structures
  - Conditionals
    - If statements
    - Case statements
  - Loops
    - For loops
    - While loops
  - Functions
- Comparisons and Tests
  - String comparison
  - Numerical comparison
  - Variable existence
  - File checks
  - Permissions checks
- Logical Operations and Combinations
  - Internal combinations
  - External combinations
  - Conditional execution
- Useful Commands and Concepts
  - sleep
  - read
  - set options
  - mktemp
  - trap
  - Arrays
- Best Practices

# **Basics**

#### **Exit codes**

In bash, exit codes indicate the success or failure of a command or script.

• 0 → success

anything else → failure

### **Variables and Data Types**

Bash primarily deals with strings, but can handle numbers and arrays. Example:

```
name="John"
age=30
```

#### **Command substitution**

Allows you to use the output of a command as part of another command. Example:

```
current_date=$(date +%Y-%m-%d)
```

### Input/output redirection

Allows you to control where input comes from and where output goes. Example:

```
echo "Hello" > output.txt # Redirect output to a file
cat < input.txt # Read input from a file</pre>
```

# **Control Structures**

#### **Conditionals**

#### If statements

Used for conditional execution of code. Example:

```
if [[ condition ]]; then
  echo "condition met"
elif [[ condition_2 ]]; then
  echo "condition_2 met"
else
  echo "no condition met"
fi
```

http://2027a.net/ Printed on 2025/11/02

2025/11/02 3/7 bash\_scripts

#### **Case statements**

Used for multiple conditional branches. Example:

```
case "$variable" in
  pattern1) command1 ;;
  pattern2) command2 ;;
  *) default_command ;;
esac
```

#### Loops

#### For loops

Used to iterate over a list of items. Examples:

```
# Regular
my_array=(1 2 3 4 5)
for item in ${my_array[@]}
  echo $item
done
# C-style
for ((i = 0; i < 10; i++)); do
  echo "$i"
done
# Range
for i in {1..10}; do
  echo "$i"
done
# Pattern matching
for item in ./content/*.md; do
  echo "$item"
done
# Command result
for item in $(ls ~/Notes/); do
 echo $item
done
```

#### While loops

Executes a block of code as long as a condition is true. Example:

```
counter=0
while [[ $counter -lt 5 ]]; do
  echo $counter
  ((counter++))
done
```

### **Functions**

Reusable blocks of code. They operate like mini-scripts. Example:

```
greet() {
  echo "Hello, $1!"
}
greet "World"
```

# **Comparisons and Tests**

### **String comparison**

```
val="a"
[[ "$val" == "a" ]]
[[ "$val" != "b" ]]
```

# **Numerical comparison**

```
num=1
[[ "$num" -eq 1 ]] # equal
[[ "$num" -ne 2 ]] # not equal
[[ "$num" -lt 2 ]] # less than
[[ "$num" -le 2 ]] # less than or equal
[[ "$num" -gt 1 ]] # greater than
[[ "$num" -ge 1 ]] # greater than or equal
```

#### Variable existence

```
val=""
```

http://2027a.net/ Printed on 2025/11/02

2025/11/02 5/7 bash scripts

```
[[ -z $val ]] # var is null
[[ -n $val ]] # var is not null
```

#### File checks

```
file="./hello"
[[ -f $file ]] # file exists
[[ -d $file ]] # directory exists
[[ -e $file ]] # file or directory exists
```

#### **Permissions checks**

```
file="./hello"
[[ -r $file ]] # readable
[[ -w $file ]] # writable
[[ -x $file ]] # executable
```

# **Logical Operations and Combinations**

#### Internal combinations

```
[[ $val -gt 5 -a $val -lt 10 ]] # -a -> AND
[[ $val -gt 5 -o $val -lt 3 ]] # -o -> OR
```

#### **External combinations**

```
[[ $val -gt 5 ]] && [[ $val -lt 10 ]] # AND
[[ $val -gt 5 ]] || [[ $val -lt 3 ]] # OR
```

#### **Conditional execution**

```
command1 && command2 # Execute command2 if command1 was successful
command1 || command2 # Execute command2 if command1 failed
```

# **Useful Commands and Concepts**

#### sleep

Pauses the script for a specified amount of time.

#### read

Reads input from the user. Example:

```
read -p "Do you want to continue (Y/n) " resp
if [[ $resp != "Y" ]]; then
  exit 1
fi
echo "Continuing..."
```

### set options

Activates strict mode in bash:

```
set -euo pipefail
```

- set -e: exit on error
- set -u: exit on unset var
- set -o pipefail: exit on pipe fail

# mktemp

Creates a temporary file or directory.

### trap

Sets up a function to be called when the script receives specific signals.

### **Arrays**

Store multiple values in a single variable. Example:

```
fruits=("apple" "banana" "cherry")
echo ${fruits[0]} # Outputs: apple
```

http://2027a.net/ Printed on 2025/11/02

2025/11/02 7/7 bash\_scripts

# **Best Practices**

- Use meaningful variable names
- Comment your code
- Use functions for repeated code
- Always quote variables when using them
- Use set -euo pipefail for safer scripts

From:

http://2027a.net/ - /dev/null

Permanent link:

http://2027a.net/tech/bash\_scripts?rev=1729866053

Last update: 2024/10/25

