



# Bash Script

## Cheat Sheet



### Basic Syntax

<code>#!/bin/bash</code>	Shebang at the beginning of a script specifies the interpreter
<code>#!/usr/bin/env bash</code>	Alternative shebang -using environment variable
<code>\$#</code>	Stores the number of argument passes to the Bash script
<code>\$1 , \$2, \$3</code>	Variables that store the values passed as arguments to the Bash script
<code>exit</code>	Exit from the Bash script
<code>CTRL + C</code>	Keyboard shortcut to stop Bash
<code>\$ (command)</code>	Execute a command inside a subshell
<code>sleep</code>	Pause for a specified number of seconds, minutes, hours or days

### Comments

<code>#</code>	Single line comment. The text comes after it will not be executed
<code>: &lt;&lt;' '</code>	Multiple line comment

### Command Execution

<code>command_name</code>	Directly execute the command with specified name
<code>`variable_name =command``</code>	Older version of substituting the output of the command to a specified variable
<code>command &gt; file_name</code>	Redirect the output of a command to a specified file
<code>command &gt;&gt; file_name</code>	Redirect the output of a command to a specified command and append it with the existing content
<code>command1   command2</code>	Use the standard output of command1 as the standard input of command2

### Variables

<code>var_name=val</code>	Assign a value to the specified variable
<code>\$ var_name</code>	Access the value of the specified variable
<code>"\$var_name"</code>	Variables with special bash script character at the beginning must be quoted with double quotes or single quotes
<code>var_name=\$(command)</code>	Assign the output of a command to the specified variable
<code>readonly var_name=val</code>	Prevent the value of a specified variable to be modified
<code>\$HOME, \$PATH, \$USER etc.</code>	Few predefined environment variables
<code>\$0</code>	Predefined variables that stores the name of the script
<code>\$#</code>	Predefined variables that stores the number of command line arguments
<code>#?</code>	Predefined variable that stores the exit status of the last executed command
<code>\$\$</code>	Predefined variable that stores the process ID of the current script
<code>#!</code>	Predefined variable that stores the process ID of the last background command
<code>unset var_name</code>	Delete a variable with specified name

### Input/Output

<code>read -p</code>	Prompt the user for information to enter
<code>command &lt; input_file</code>	Redirect input from a file to a command
<code>command 2&gt; error_file</code>	Redirect standard error to a specified file
<code>command &amp;&gt; file_name</code>	Redirect standard output and standard error to a specified file



### Loops

```
for variable in
list; do
# Code
done
```

Iterate over the list and execute code for each element of the list

```
while condition;
do
# Code
done
```

Execute code repeatedly as long as the condition is true

```
until condition;
do
# Code
done
```

Execute code repeatedly until the condition becomes true

```
select variable
in list; do
# Code
done
```

Execute code based on the choice that the variable takes from the list

**continue** Skip the current iteration of a loop and continue with the next iteration

**break** Terminate a loop based on certain condition

### Data Types

**x=5** Integer or floating point values are treated as Number

### Conditional Statements

```
if [ condition ];
then
#code
fi
```

Test a condition and execute the then clause if it is true

```
if [ condition ];
then
#code
fi
else
#code
fi
```

Execute the then clause if the condition is true, otherwise execute the else clause

```
if [ condition1 ];
then
#code
elif [ condition2
]; then
#code
else
#code
fi
```

Execute the then clause if the condition is true or execute the elif clause if the condition is true, otherwise execute the else clause

```
case variable in
pattern1)
#code
;;
pattern2)
#code
;;
pattern3)
#code
;;
*)
;;
esac
```

Execute code following each pattern if the variable matches the pattern otherwise execute \* if none of the patterns match

**test condition** Returns 0 or 1 indicating whether the condition is true or false

### Arithmetic Operations

**+** Addition

**-** Subtraction



# Bash Script

## Cheat Sheet



### Data Types

<code>is_valid=0</code>	Boolean value represent False
<code>is_valid=1</code>	Boolean value represents True
<code>declare -a var</code>	Declare an indexed array
<code>declare -A var</code>	Declare an associated array
<code>declare -i var</code>	Declare an integer variable
<code>declare -r var</code>	Declare a read only variable
<code>declare -x var</code>	Declare an exported variable

`var_name=""` Absence of value or uninitialized variable

`array=("element1" "element2" "element3"...)` A collection of elements accessed using numerical indices

`declare -A array1`  
`array1["element1"]="value1"` A collection of elements accessed using string indices  
`array2["element2"]="value2"`

`var="Hellow World"` Sequence of characters enclosed in single or double quotes is treated as String

### Boolean Operators

<code>&amp;&amp;</code>	Logical AND operator
<code>  </code>	Logical OR operator
<code>!</code>	NOT equal to operator

### String Comparison Operators

<code>=</code>	equal
<code>!=</code>	not equal
<code>&lt;</code>	less than
<code>&gt;</code>	greater than
<code>-n str1</code>	string str1 is not empty
<code>-z str2</code>	string str2 is empty

### Arithmetic Operations

<code>*</code>	Multiplication
<code>/</code>	Division
<code>%</code>	Modulus or remainder
<code>**</code>	Raise to a power
<code>((i++))</code>	Increment a variable
<code>((i--))</code>	Decrement a variable

### Function

```
function_name(
) {
    # code
    function name
}
```

`function_name` Call a function with specified function name

`local var_name=val` Declare a local variable inside a function

`return` Exit a function and return a value of the calling function

### Arithmetic Conditional Operators

<code>-lt</code>	Equals to mathematical < operator (less than)
<code>-gt</code>	Equals to mathematical > operator (greater than)
<code>-le</code>	Equals to mathematical <= operator (less than equal)
<code>-ge</code>	Equals to mathematical >= operator (greater than equal)
<code>-eq</code>	Equals to mathematical == operator (equal)
<code>-ne</code>	Equals to mathematical != operator (not equal)



### String Manipulation

`concatenated="$str1 $str2"` Concatenate the variables set in str1 and str2

`substring=${str:n}` Extracts a substring from n-th index to till the end of the string that stored in variable str

`substring=${str:0:5}` Extracts substring from 0-th index to 5-th index of the string that stored in variable str

`length=${#str}` Find the length of the string that stored in variable str

`[[ $str == *World* ]]` Returns True if the string stored in variable str contains the word World

`replaced=${str/World/Universe}` Replaces the first occurrence of 'World' with 'Universe' within the string stored in str variable

`trimmed=${str# }` Trims leading whitespace of the string  
`trimmed=${trimmed%* }` Trims trailing whitespaces of the string stored in trimmed variable