# 100 Shell Script Examples [Free Downloads]

Written by**Anonnya Ghosh**

Reviewed by**Md. Ashakul Islam Sowad**

**Last updated:** Apr 18, 2024

A shell script in Linux is a text file containing a sequence of commands written in a scripting language interpreted by the shell. The shell acts as an interface between the user and the operating system, interpreting commands entered by the user or read from a script file. The **GNU Bourne-Again Shell** also known as **bash** is the default shell for most of the **Linux** distributions. It commonly runs in its interactive form which is the **Command Line Interface (CLI).**

In this article, you will find 100 shell script examples along with a basic understanding of **Shell Scripting**.

Table of Contents

## 100 Shell Scripts Free Downloads


### Available Download Options

---

**Most Popular**

PDF (61 Pages)
DOC (61 Pages)

**Combo Pack**

ZIP (100 Scripts)

# Bash Shell Script Examples

Shell scripts are primarily written in scripting languages like Bash (Bourne Again SHell), sh (Bourne Shell), C Shell, Korn Shell (ksh), and others. Among these, **Bash is the default shell** for most Linux distributions. The very first line of a bash script needs to start with the **SheBang (#!)** followed by the path to the bash executable program (**/bin/bash**). Here are 100 examples of basic shell scripts:

## 1. Defining Variables in a Bash Script

Variables in **shell scripting** are containers for storing necessary information. In Bash Script, declare a variable by assigning a value to its reference by `=` operator. Furthermore, print the assigned values using `echo $(VARIABLE_NAME)`.

**The syntax for Variables in Shell Scripting is given below:**

```
VARIABLE_NAME=VALUE
```

Bash
Copy

**The rules for Variables in Shell Scripting are as follows:**

- Use the equal sign (=) to assign values to variable names.
- Variable names are case sensitive i.e. 'A' and 'a' are different.
- To refer to a variable use the **dollar sign ($)** e. **$VARIABLE_NAME**.
- While updating/changing the variable values use only the variable name with the assignment operator(=) i.e. **VARIABLE_NAME= NEW_VALUE**.
- No need to define variable type while declaring variables.
- Enclose multiple words or string values within **Single Quote (' ')** to consider all characters as input.

Here is an example script for defining variable:

```
#!/bin/bash

# Declaration of variables
name=Tom
age=12
# Displaying variables
```

```
echo $name $age
```

Bash

Copy

**Output:**

```
Tom 12
```

Bash

Copy

## 2. Display User Input

You can take user input with the `read` command and store it in a variable. Next, use `echo $(VARIABLE_NAME)` to print the user input. Here's how:

```
#!/bin/bash

read num

echo “The number is: $num”
```

Bash

Copy

## 3. Read User Input With Prompt Message

The `read` command used with option `-p` allows you to prompt a message along with taking user input. You can use `echo $(VARIABLE_NAME)` to display the user input on the screen. Here's an example:

```
#!/bin/bash



read -p "Enter a number:" num

echo “The number is: $num”
```

Bash
Copy

**Output:**

```
Enter a number: 12

The number is: 12
```

Bash
Copy

## 4. Concatenating Multiple Variables

To concatenate multiple variables and store them into a single variable, enclose them with a **double quotation (“ ”)** and then write the variables within `{}` consecutively. Here is an example:

```bash
#!/bin/bash



# Declaration of variables
name='My name is Tom.'
age='My age is 12.'

# Concatenation
info="${name} ${age}"
echo "Result: $info"
```

Bash
Copy

**Output:**

```
Result: My name is Tom. My age is 12.
```

Bash
Copy

## 5. Passing Values to Variables as Command Line Arguments

For passing values as command line arguments, you have to run the script along the values in a sequence. Later access these values using the `$` and input sequence number. Here's how:

```bash
#!/bin/bash

name=$1

age=$2

echo "My name is $name. My age is $age."
```

Bash
Copy

**Run the script using:**

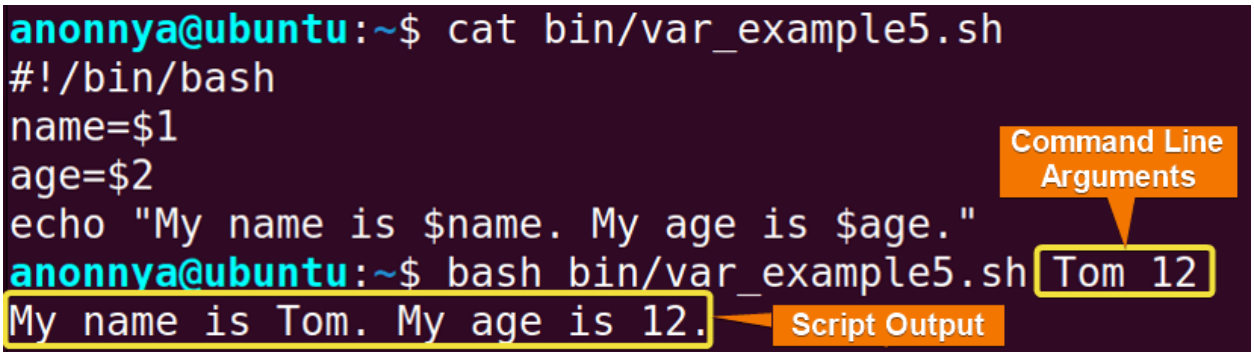
```bash
bash var_example5.sh Tom 12
```

Bash
Copy

**Output:**

```
My name is Tom. My age is 12.
```

Bash
Copy




## 6. Print Environment Variable Using Bash Script

You can store an Environment Variable like other variables and print it using `${!..}` syntax. The example script below shows how:

```
#!/bin/bash

read -p "Enter an Environment Variable name:" var

echo "Environment:${!var}"
```

Bash
Copy

**Output:**

```
Enter an Environment Variable name:

HOME

Environment:/home/anonnya
```

Bash
Copy

## 7. Adding Two Numbers

Run an addition operation using the `+` operator between defined variables and enclose them with `$()`. Here's an example:

```bash
#!/bin/bash


num1=10

num2=20

sum=$(($num1+$num2))

echo "The Sum is: $sum"
```

Bash
Copy

**Output:**

```
The Sum is: 30
```

Bash
Copy

## 8. Subtracting Two Numbers

Subtract two numbers using the `-` operator between defined variables and enclose them with `$()`. Here's an example:

```bash
#!/bin/bash


num1=30

num2=20

dif=$(($num1-$num2))


echo "The difference is: $dif"
```

Bash
Copy

**Output:**

```
The difference is: 10
```

Bash
Copy

## 9. Division of Two Numbers

Run a division using the `/` operator between defined variables and enclose them with `$()`. Here's an example script:

```
#!/bin/bash


num1=30

num2=5

div=$(($num1/$num2))


echo "The division is: $div
```

Bash
Copy

**Output:**

```
The division is: 6
```

Bash
Copy

## 10. Calculating the Remainder of a Division

For generating the remainder of a division use the `%` operator between defined variables and enclose them with `$()`. Here's an example script:

```
#!/bin/bash


num1=30

num2=20

mod=$(($num1%$num2))


echo "The remainder is: $mod"
```

Bash
Copy

**Output:**

```
The remainder is: 10
```

Bash
Copy

## 11. Generate a Random Number Using Bash Script

To generate a random number in bash, use the `RANDOM` function enclosed with `$()`. For example:

```
#!/bin/bash


echo $((RANDOM))
```

Bash
Copy

## 12. Generating a Random Number Between Two Given Numbers

To generate a random number in bash, use the `RANDOM` function with the lower and upper bound in the syntax `(lower + RANDOM % upper)` or `($RANDOM % ($upper - $lower + 1) + $lower )`. Use `read` command to take the upper and lower values from the user. Here's a complete script:

```bash
#!/bin/bash


read -p "Enter minimum range:" min

read -p "Enter maximum range:" max


r_num=$(( $RANDOM % ($max - $min + 1) + $min ))

echo "Random Number: $r_num"
```

Bash
Copy

**Output:**

```
Enter minimum range:10

Enter maximum range:35

Random Number: 24
```

Bash
Copy

## 13. Performing Mathematical Operations Without Storing

To perform multiple mathematical operations without storing them in a separate variable, write the operations inside `echo` . This is how:

```bash
#!/bin/bash


read -p "Enter a number:" num1

read -p "Enter a smaller number:" num2


echo "Addition: $(($num1 + $num2))"

echo "Subtraction: $(($num1 - $num2))"

echo "Multiplication: $(($num1 * $num2))"

echo "Division: $(($num1 / $num2))"
```

Bash

Copy

**Output:**

```
Enter a number:35

Enter a smaller number:15

Addition: 50

Subtraction: 20

Multiplication: 525
```

```
Division: 2
```

Bash
Copy

## 14. Performs a Bitwise Operation Based on User Input

The given script performs either of the bitwise AND, OR, NOT operations by `&`, `|`, `!` respectively on the 2 input numbers. If the user enters any other operand as input then the script displays an error message.

```
#!/bin/bash

read -p "Enter two numbers: " num1 num2
read -p "Enter operation to perform (AND, OR, NOT): " op

case $op in

AND) echo "Result: $num1 & $num2 = $((num1&num2))";;
OR) echo "Result: $num1 | $num2 = $((num1|num2))";;
NOT) echo "Result: $num1 ^ $num2 = $((num1^num2))";;
*) echo "Invalid operator.";;

esac
```

Bash
Copy

**Output:**

```
Enter two numbers: 4 5

Enter operation to perform (AND, OR, NOT): AND

Result: 4 & 5 = 4
```

Bash
Copy

## 15. Check If a Number is an Even or Odd

If a number is even, then its remainder after dividing by 2 will be 0. Otherwise, the number is odd. So, use the remainder operator `%` within `if-else` to check if a number is even or odd. Here's how:

```
#!/bin/bash


read -p "Enter a number:" num

if [ $((num%2)) == 0 ]

then

echo "The number is even"

else

echo "The number is odd"

fi
```

Bash
Copy

**Output:**

```
Enter a number:25

The number is odd
```

Bash
Copy

## 16. Perform an Arithmetic Operation Based on User Input

To perform user input-based operations implement the `if-elif-else` condition like the below script:

```
!/bin/bash


read -p "Enter a number:" num1

read -p "Enter a smaller number:" num2

read -p "Enter an operand:" op


if [ $op == + ]

then

echo "$num1 + $num2 = $((num1+num2))"

elif [ $o == - ]

then

echo "$num1 - $num2 = $((num1-num2))"

elif [ $op == * ]

then

echo "$num1 * $num2 = $((num1*num2))"
```

```
elif [ $op == / ]

then

echo "$num1 / $num2 = $((num1/num2))"

else

echo "Operator not listed"

fi
```

Bash
Copy

**Output:**

```
Enter a number:34

Enter a smaller number:14

Enter an operand:+

34 + 14 = 48
```

Bash
Copy

## 17. Performs a Logical Operation Based on User Input

You can perform user-input based operations with the `case` statement as the below script:

```
#!/bin/bash


read -p "Enter two values: "  val1 val2

read -p "Enter an operation(and/or/not) to perform:" op
```

```
case $op in
and)
if [[ $val1 == true && $val2 == true ]]
then
echo "Result: true"
else
echo "Result: false"
fi;;
or)
if [[ $val1 == true || $val2 == true ]]
then
echo "Result: true"
else
echo "Result: false"
fi;;
not)
if [[ $val1 == true ]]
then
echo "Result: false"
else
echo "Result: true"
```

```
fi;;

*) echo "Invalid operator."

esac
```

Bash
Copy

**Output:**

```
Enter two values: true false

Enter an operation(and/or/not) to perform:or

Result: true
```

Bash
Copy

## 18. Check If a Given Input is a Valid Email ID

A valid email can be checked by defining the email syntax `^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$` inside the `if` condition. The below script shows how:

```
#!/bin/bash


read -p "Enter an email ID: " id

if [[ $id =~ ^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$ ]]

then

echo "This is a valid email ID!"

else

echo "This is not a valid email ID!"
```

```
fi
```

Bash
Copy

**Output:**

```
Enter an email ID: tom@gmail.com

This is a valid email ID!
```

Bash
Copy

## 19. Check If a Given Input is a Valid URL

To check a valid URL, use a simple `if-else` condition with the URL pattern `^(http|https)://[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$`. The below script shows how:

```
#!/bin/bash


read -p "Enter a URL: " url

if [[ $url =~ ^(http|https)://[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$ ]]

then

echo " This is a valid URL!"

else

echo "This is not a valid URL!"

fi
```

Bash
Copy

**Output:**

```
Enter a URL: abcdefg1234

This is not a valid URL!
```

Bash
Copy

## 20. Check if a Given Number is Positive or Negative

To check if a given number is positive or negative, use the comparison operators (`-gt`, `-lt`) inside the `if-elif-else` block.

```
#!/bin/bash

read -p "Enter a number:" num

if [ $num -gt 0 ]

then

echo "The number is Positive!"

elif [ $num -lt 0 ]

then

echo "The number is Negative!"

else

echo "The number is Zero!!"

fi
```

Bash
Copy

**Output:**

```
Enter a number:12

The number is Positive!
```

Bash
Copy

## 21. Check If a File is Writable

To check if a file is writable, use `-w` option with the variable that stores the filename inside a `if` block. For example, if `fname` is the variable that stores the filename to check, use `-w $fname` inside if block to check if it is writable. Here's how:

```
#!/bin/bash

read -p "Enter a File Name:" fname

if [ -w $fname ]

then

echo "The File $fname is writable."

else

echo "The File $fname is not writable."

fi
```

Bash
Copy

**Output:**

```
Enter a File Name:file1.txt

The File file1.txt is writable.
```

Bash
Copy

## 22. Check If a File Exists or Not

To check if a file exists, use `-f` option with the variable that stores the filename inside a `if` block. For example, if `fname` is the variable that stores the filename to check, use `-f $fname` inside if block to check if it exists. Here's how:

```bash
#!/bin/bash

read -p "Enter a File Name:" fname

if [ ! -f $fname ]

then

echo "The File $fname does not exist!"

exit 1

fi

echo "The File $fname exists."
```

Bash
Copy

**Output:**

```
Enter a File Name:myfile.txt

The File myfile.txt does not exist!
```

Bash
Copy

## 23. Check If a Directory Exists or Not

To check if a file is writable, use `-d` option with the variable that stores the directory inside a `if` block. For example, if `dir` is the variable that stores the directory to check, use `-d $dir` inside if block to check if it exists. Here's how:

```
#!/bin/bash


read -p "Enter a File Name: " dir

if [ ! -d $dir ]

then

echo "The directory $dir does not exist!"

exit 1

fi

echo "The directory $dir exists."
```

Bash
Copy

**Output:**

```
Enter a File Name: bin

The directory bin exists.
```

Bash
Copy

## 24. Echo With New Line

To echo with a new line, use of the `echo` command with `-e` and `\n` before the message. That's how:

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

```
echo -e 'Hi\nthere!'
```

Bash
Copy

**Output:**

```
Hi

there!
```

Bash
Copy

## 25. Changing Internal Field Separator (IFS)/Delimiter

By changing the **IFS** you will be able to access values separated by your desired delimiter. First, store the default IFS in a variable using `old_IFS = $IFS`. Now, change IFS according to your preference and complete the task. At the end, restore the original **IFS** using `IFS = $old_IFS`. The below script shows how:

```
#!/bin/bash



#store default IFS
old_IFS= $IFS

IFS=,
read val1 val2 val3 <<< "5,60,70"
echo 1st value: $val1
echo 2nd value: $val2
echo 3rd value: $val3

#restore default IFS
IFS= $old_IFS;
```

Bash
Copy

**Output:**

```
1st value: 5

2nd value: 60

3rd value: 70
```

Bash
Copy

## 26. Take Two Command Line Arguments and Calculate their Sum

You can do direct mathematical operations on command line arguments using the `$((..))`.

```
#!/bin/bash



sum=$(( $1 + $2 ))

echo "The sum of $1 and $2 is $sum"
```

Bash
Copy

**Syntax to Run Script >**

```
bash misc_example3.sh 20 30
```

Bash
Copy

**Output:**

```
The sum of 20 and 30 is 50
```

Bash
Copy

## 27. Take Password Input

In bash, you can utilize the `read` command for taking **password-type** inputs. The application of the `read` with `-sp` option hides the input characters when you type them**.**

```
#!/bin/bash


read -sp "Enter your password: " pass

echo -e "\nYour password is: $pass"
```

Bash
Copy

**Output:**

```
Enter your password:

Your password is: linuxsimply
```

Bash
Copy

## 28. Take Timed Input

You can take timed input in bash using the `read` command with `-t` option. The prompt message will disappear if you do not complete entering your values within the specified time.

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

```
read -t 5 -p "Enter your name within 5 seconds: " name
```

Bash
Copy

**Output:**

```
Enter your name within 5 seconds: Anonnya
```

Bash
Copy

## 29. Find the Length of a String

You can simply use the `${#STRING}` to find the length of a string.

```
#!/bin/bash



str="My name is Tom!"

len=${#str}

echo "The length of the string is: $len"
```

Bash
Copy

**Output:**

```
The length of the string is: 15
```

Bash
Copy

## 30. Check if Two Strings are Equal

**The String operators in Shell Scripting are as follows:**

| String Operators | | |
|---|---|---|
| < (Less than) | == (Equal) | += (Concatenation) |
| > (Greater than) | != (Not equal) | |

Check whether two strings are the same using the `==` **(Equal)** operator inside `if` condition.

```bash
#!/bin/bash


string1="hello"

string2="world"


if [ "$string1" == "$string2" ]; then

echo "The strings are equal."

else

echo "The strings are not equal."

fi
```

Bash

Copy

**Output:**

```
The strings are not equal.
```

Bash

Copy

## 31. Convert All Uppercase Letters in a String to Lowercase

To convert all upper-case letters in a string to lower-case letters, use the `tr` command with the `[:upper:]` and `[:lower:]` classes for conversion. Here is an example shows how to use it:

```bash
#!/bin/bash


read -p "Enter a string: " str

echo "Converted String:" $str | tr '[:upper:]' '[:lower:]'
```

Bash
Copy

**Output:**

```
Enter a string: ABCDefgh

Converted string: abcdefgh
```

Bash
Copy

## 32. Remove All Whitespace From a String

For removing white spaces from a string simply use the `${STRING// /}`. Here's how:

```bash
#!/bin/bash


str="   Hello    from Linuxsimply !   ! "

str=${str// /}

echo "The resultant string: $str"
```

Bash

Copy

**Output:**

```
The resultant string: HellofromLinuxsimply!!
```

Bash
Copy

## 33. Reverse a String

To reverse a string use the `rev` command with `echo` and **Pipe(|)**. Here is a complete script showing how to reverse a string:

```
#!/bin/bash


str="Linuxsimply"

str=$(echo "$str" | rev)

echo "The reversed string: $str"
```

Bash
Copy

**Output:**

```
The reversed string: ylpmisxuniL
```

Bash
Copy

## 34. Reverse a Sentence

You can reverse a sentence by reversing the order of words with the `awk` command. Here's how:

```
#!/bin/bash


sentence="Hello from LinuxsimplY!!"


r_sentence=$(echo "$sentence" | awk '{ for(i=NF;i>0;i--)
printf("%s ",$i); print "" }')

echo "The reversed sentence is: $r_sentence"
```

Bash
Copy

**Output:**

```
The reversed sentence is: LinuxsimplY!! from Hello
```

Bash
Copy

## 35. Capitalize the First Letter of a Word

For capitalizing only the first letter of a word, first cut out the first letter by `${str:0:1}`, then convert it using `tr '[:lower:]' '[:upper:]'`, and finally concatenate it with the rest of the string. Here's how:

```
#!/bin/bash


str="linuxsimply!!"

cap_str=$(echo "${str:0:1}" | tr '[:lower:]' '[:upper:]')${str:1}

echo "The capitalized word is: $cap_str"
```

Bash
Copy

**Output:**

```
The capitalized word is: Linuxsimply!!
```

Bash
Copy

## 36. Replace a Word in a Sentence

You can replace the first occurrence of a word in a string with a given word using the `$(../../..)`.

```
#!/bin/bash


read -p "Enter a sentence: " str1

read -p "Enter the word to be replaced: " str2

read -p "Enter the new word: " str3

echo "Modified sentence: ${str1/$str2/$str3}"
```

Bash
Copy

**Output:**

```
Enter a sentence: I love Linux

Enter the word to be replaced: Linux

Enter the new word: Linuxsimply

Modified sentence: I love Linuxsimply
```

Bash
Copy

## 37. Print Numbers From 5 to 1

You can print a number sequence using the `until` loop in bash. In this case, specify the condition to stop the loop inside `until [ ]`.

```
#!/bin/bash


n=5

until [ $n == 0 ]

do

echo $n

n=$((n-1))

done
```

Bash
Copy

**Output:**

```
5

4

3

2

1
```

Bash
Copy

## 38. Print Even Numbers From 1 to 10

To print only the even number in a range, check the even number condition inside the `for` loop before printing the number.

```bash
#!/bin/bash


for (( i=1; i<=10; i++ ))
do
if [ $((i%2)) == 0 ]
then
echo $i
fi
done
```

Bash
Copy

**Output:**

```
2
4
6
8
10
```

Bash
Copy

## 39. Print the Multiplication Table of a Number

Use the simple `echo` command inside a `for` loop to display the Multiplication Table of a number.

```bash
#!/bin/bash


read -p "Enter a number: " num

for (( i=1; i<10; i++ ))

do

echo "$num x $i = $((num*i))"

done
```

Bash

Copy

**Output:**

```
Enter a number: 12

12 x 1 = 12

12 x 2 = 24

12 x 3 = 36

12 x 4 = 48

12 x 5 = 60

12 x 6 = 72

12 x 7 = 84

12 x 8 = 96

12 x 9 = 108
```

Bash

Copy

## 40. Calculate the Sum of Digits of a Given Number

For calculating the sum of digits of a given number, extract each digit using the `%` operator and store the summation in a fixed variable using a loop.

```bash
#!/bin/bash


read -p "Enter a number: " num

sum=0

while [ $num -gt 0 ]

do

dig=$((num%10))

sum=$((sum+dig))

num=$((num/10))

done

echo "The sum of digits of the given number: $sum"
```

Bash
Copy

**Output:**

```
Enter a number: 1567

The sum of digits of the given number: 19
```

Bash
Copy

## 41. Calculate the Factorial of a Number

Calculate the factorial of a number by running multiplications inside a `for` loop.

```bash
#!/bin/bash


read -p "Enter a number: " num

temp=1

for (( i=1; i<=$num; i++ ))

do

temp=$((temp*i))

done

echo "The factorial of $num is: $temp"
```

Bash
Copy

**Output:**

```
Enter a number: 6

The factorial of 6 is: 720
```

Bash
Copy

## 42. Calculate the Sum of the First “n” Numbers

To calculate the sum of the first **n** numbers run a for loop and addition operation till **n**.

```bash
#!/bin/bash


read -p "Enter a number: " num

sum=0

for (( i=1; i<=$num; i++ ))

do

sum=$((sum + i))

done

echo "Sum of first $num numbers: $sum"
```

Bash
Copy

**Output:**

```
Enter a number: 100

Sum of first 100 numbers: 5050
```

Bash
Copy

## 43. Loop Through Array Elements

For accessing each array element you can use the `for` loop in the following manner. Indicate the desired array using `${ARRAY_NAME[@]}` and access each item stored in the array.

```bash
#!/bin/bash

arr=("mango" "grape" "apple" "cherry" "orange")

for item in "${arr[@]}"; do
```

```
echo $item

done
```

Bash
Copy

**Output:**

```
mango

grape

apple

cherry

orange
```

Bash
Copy

## 44. Find the Smallest and Largest Elements in an Array

To find the smallest and largest element in a given array, first, initialize a small and a large number. Then compare the array elements with these numbers inside any loop. Here's an example script showing how:

```
#!/bin/bash


arr=(24 27 84 11 99)


echo "Given array: ${arr[*]}"
```

```bash
s=100000

l=0

for num in "${arr[@]}"

do

if [ $num -lt $s ]

then

s=$num

fi

if [ $num -gt $l ]

then

l=$num

fi

done


echo "The smallest element: $s"

echo "The largest: $l"
```

Bash

Copy

**Output:**

```
Given array: 24 27 84 11 99

The smallest element: 11

The largest: 99
```

Bash

Copy

## 45. Sort an Array of Integers in Ascending Order

You can sort an array of integers by converting it into a list of integers using `tr '\n'`. The list of integers is sorted with the `sort -n` command and then converted back into an array.

```bash
#!/bin/bash


arr=(24 27 84 11 99)


echo "Given array: ${arr[*]}"

arr=($(echo "${arr[*]}" | tr ' ' '\n' | sort -n | tr '\n' ' '))

echo "Sorted array: ${arr[*]}"
```

Bash
Copy

**Output:**

```
Given array: 24 27 84 11 99

Sorted array: 11 24 27 84 99
```

Bash
Copy

## 46. Remove an Element From an Array

To remove an element from an array using the pattern substitution concept. The syntax `${arr[@]/$val}` contains all the elements of the original array except for any occurrences of the value `$val`.

```bash
#!/bin/bash


arr=(24 27 84 11 99)


echo "Given array: ${arr[*]}"

read -p "Enter an element to remove: " val

arr=("${arr[@]/$val}")

echo "Resultant array: ${arr[*]}"
```

Bash
Copy

**Output:**

```
Given array: 24 27 84 11 99

Enter an element to remove: 11

Resultant array: 24 27 84  99
```

Bash
Copy

## 47. Inserting an Element Into an Array

For inserting an element into an array, split the array in the given index using `${arr[@]:0:$index}` and insert the element. Here's a complete script showing the process:

```bash
#!/bin/bash


arr=(24 27 84 11 99)


echo "Given array: ${arr[*]}"
read -p "Enter an element to insert: " new_val
read -p "Enter the index to insert the element: " index
arr=("${arr[@]:0:$index}" "$new_val" "${arr[@]:$index}")
echo "The updated array: ${arr[@]}"
```

Bash
Copy

**Output:**

```
Given array: 24 27 84 11 99
Enter an element to insert: 100
Enter the index to insert the element: 3
The updated array: 24 27 84 100 11 99
```

Bash
Copy

## 48. Slicing an Array Using Bash Script

Slice an array in Bash by placing the indices to slice inside the `${arr[@]:$index1:$index2}` pattern.

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

```
arr=(24 27 84 11 99)


echo "Given array: ${arr[*]}"
read -p "Enter 1st index of slice: " index1
read -p "Enter 2nd index of slice: " index2
sliced_arr=("${arr[@]:$index1:$index2}")
echo "The sliced array: ${sliced_arr[@]}"
```

Bash
Copy

**Output:**

```
Given array: 24 27 84 11 99
Enter 1st index of slice: 1
Enter 2nd index of slice: 3
The sliced array: 27 84 11
```

Bash
Copy

## 49. Calculate the Average of an Array of Numbers

Find the sum of array elements using a `for` loop and divide it by the number of elements i.e. `${#arr[@]}`.

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

```bash
echo "Enter an array of numbers (separated by space):"


read -a arr

sum=0

for i in "${arr[@]}"

do

sum=$((sum+i))

done

avg=$((sum/${#arr[@]}))

echo "Average of the array elements: $avg"
```

Bash
Copy

**Output:**

```
Enter an array of numbers (separated by space):

23 45 11 99 100

Average of the array elements: 55
```

Bash
Copy

## 50. Find the Length of an Array

To find the length of an array simply use the syntax: `${#arr[@]}`.

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

```bash
arr=(24 27 84 11 99)


echo "Given array: ${arr[*]}"

len=${#arr[@]}

echo "The length of the array: $len"
```

Bash
Copy

**Output:**

```
Given array: 24 27 84 11 99

The length of the array: 5
```

Bash
Copy

## 51. Check if a String is a Palindrome

You can write the code to check a palindrome inside the function `Palindrome()` and call it by passing the desired string:

```bash
#!/bin/bash


Palindrome () {


s=$1

if [ "$(echo $s | rev)" == "$str" ]

then
```

```
echo "The string is a Palindrome"

else

echo "The string is not a palindrome"

fi

}

read -p "Enter a string: " str

Palindrome "$str"
```

Bash

Copy

**Output:**

```
Enter a string: wow

The string is a Palindrome
```

Bash

Copy

**The rules for Function in Shell Scripting are as follows:**

- Functions must be defined before using/calling them.
- You may pass arguments to functions while calling them.
- To access arguments inside the function, use $1, $2, $3 ... and so on according to the number and sequence of arguments passed.
- The scope of the variables declared inside a function remains within the function.

## 52. Check if a Number is Prime

Create the `Prime()` function that returns whether the parameter passed is prime or not. Here is the script:

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

```bash
Prime () {
num=$1
if [ $num -lt 2 ]
then
echo "The number $num is Not Prime"
return
fi
for (( i=2; i<=$num/2; i++ ))
do
if [ $((num%i)) -eq 0 ]
then
echo "The number $num is Not Prime"
return
fi
done
echo "The number $num is Prime"
}
read -p "Enter a number: " num
Prime "$num"
```

Bash

Copy

**Output:**

```
Enter a number: 2

The number 2 is Prime
```

Bash
Copy

## 53. Convert Fahrenheit to Celsius

Here, the function `Celsius()` runs the necessary formula on the passed temperature value in Fahrenheit to convert it into Celsius. Here is the script:

```
#!/bin/bash


Celsius () {

f=$1

c=$(((($f-32)*5/9))

echo "Temperature in Celsius = $c°C"

}


read -p "Enter temperature in Fahrenheit:" f

Celsius $f
```

Bash
Copy

**Output:**

```
Enter temperature in Fahrenheit:100
```

```
Temperature in Celsius = 37°C
```

Bash
Copy

## 54. Calculate the Area of a Rectangle

Write the formula to calculate the area of a rectangle inside the function `Area()` and call it by passing the height and width.

```
#!/bin/bash


Area() {

width=$1

height=$2

area=$(($width * $height))

echo "Area of the rectangle is: $area"

}


read -p "Enter height and width of the ractangle:" h w

Area $h $w
```

Bash
Copy

**Output:**

```
Enter height and width of the ractangle:10 4

"Area of the rectangle is: 40"
```

Bash
Copy

## 55. Calculate the Area of a Circle

Write the formula to calculate the area of a circle inside the function `Area()` and call it by passing the given radius.

```
#!/bin/bash


Area () {

radius=$1

area=$(echo "scale=2; 3.14 * $radius * $radius" | bc)

echo "Area of a circle with radius $radius is $area."

}


read -p "Enter radius of the circle:" r

Area $r
```

Bash
Copy

**Output:**

```
Enter radius of the circle:4

Area of a circle with radius 4 is 50.24.
```

Bash
Copy

## 56. Calculate Grade with Number

The function `Grade()` runs the necessary conditions to divide the number ranges into grades and returns the resultant grade.

```
Grade() {

score=$1

if (( $score >= 80 )); then

grade="A+"

elif (( $score >= 70 )); then

grade="A"

elif (( $score >= 60 )); then

grade="B"

elif (( $score >= 50 )); then

grade="C"

elif (( $score >= 40 )); then

grade="D"

else

grade="F"

fi

echo "The grade for mark $s is $grade"

}


read -p "Enter a score between 1-100:" s

Grade $s
```

Bash
Copy

**Output:**

```
Enter a score between 1-100:76

"The grade for mark 76 is A"
```

Bash
Copy

## 57. Search for a Pattern Inside a File

The script given below will take a filename and a pattern as user input and search it within the file. If the pattern is found then the lines having the pattern will be displayed on the screen along with line numbers. Otherwise, it will print a message saying the pattern did not match.

```
#!/bin/bash


read -p "Enter filename: " filename

read -p "Enter a pattern to search for: " pattern

grep -w -n $pattern $filename

if [ $? -eq 1 ]; then

echo "Pattern did not match."

fi
```

Bash
Copy

**Output:**

```
Enter filename: poem.txt
```

```
Enter a pattern to search for: daffodils

4:A host, of golden daffodils;

27:And dances with the daffodils.
```

Bash
Copy

## 58. Replace a Pattern in a Fille

The following script will take a file name and a pattern from the user to replace it with a new pattern. Finally, it will display the updated lines on the terminal. If the pattern to replace does not exist, then it will show an error message.

```
#!/bin/bash


read -p "Enter filename: " filename

read -p "Enter a pattern to replace: " pattern

read -p "Enter new pattern: " new_pattern

grep -q $pattern $filename

if [ $? -eq 0 ]; then

sed -i "s/$pattern/$new_pattern/g" $filename

echo "Updated Lines: "

grep -w -n $new_pattern $filename

else

echo "Error! Pattern did not match."

fi
```

Bash
Copy

**Output:**

```
Enter filename: poem.txt

Enter a pattern to replace: daffodils

Enter new pattern: dandelions

Updated Lines:

4:A host, of golden dandelions;

27:And dances with the dandelions.
```

Bash
Copy

## 59. Print Contents of Multiple Files

The below script is for reading the contents of multiple files. It will take the file names as user input and display their contents on the screen. If any filename does not exist, it will show a separate error message for that file.

```
#!/bin/bash


read -p "Enter the file names: " files

IFS=' ' read -ra array <<< "$files"

for file in "${array[@]}"

do

if [ -e "$file" ]; then

echo "Contents of $file:"
```

```
cat "$file"

else

echo "Error: $file does not exist"

fi

done
```

Bash
Copy

**Output:**

```
Enter the file names: message.txt passage.txt



Contents of message.txt:



“Merry Christmas! May your happiness be large and your bills be
small.”



Contents of passage.txt:



The students told the headmaster that they wanted to celebrate
the victory of the National Debate Competition.
```

Bash
Copy

## 60. Copy a File to a New Location

You can copy a file to another location using the bash script below. It will read the filename and destination path from the terminal and copy the file if it exists in the current directory. If the file is not there, the script will return an error message.

```bash
#!/bin/bash


read -p "Enter the file name: " file

read -p "Enter destination path:" dest


if [ -e "$file" ]; then

cp $file $dest

file_location=$(readlink -f $dest)

echo "A copy of $file is now located att: $file_location"

else

echo "Error: $file does not exist"

fi
```

Bash
Copy

**Output:**

```
Enter the file name: poem.txt

Enter destination path:/home/anonnya/Documents


A copy of poem.txt is now located at: /home/anonnya/Documents
```

Bash

Copy

## 61. Create a New File and Write Text Inside

The script given below is for creating a new file and writing text inside the file. You will be able to write into the file from the command line. Upon completion, it will show a message saying the file has been created.

```bash
#!/bin/bash


read -p "Enter the file name: " file

echo "Enter text to write:"

read text

echo "$text" > "$file"

echo "-----------------------------------"

echo "The File $file is created!"
```

Bash
Copy

**Output:**

```
Enter the file name: text_file1.txt

Enter text to write:



In English, there are three articles: a, an, and the. Articles
are used before nouns or noun equivalents and are a type of
adjective. The definite article (the) is used before a noun to
indicate that the identity of the noun is known to the reader.
```

```
-----------------------------------


The File text_file1.txt is created!
```

Bash
Copy

## 62. Compare the Contents of Two Given Files

The following bash script takes two file names as user input and compares their contents. If one or either of the files does not exist in the current directory it shows an error to the user. Otherwise prints the result if the files are identical or not.

```
#!/bin/bash


read -p "Enter the 1st file name: " file1

read -p "Enter the 2nd file name: " file2


if [ ! -f $file1 ] || [ ! -f $file2 ]

then

echo "Error! One of the files does not exists."

exit 1

fi

if cmp -s "$file1" "$file2"

then

echo "The Files $file1 and $file2 are identical."
```

```
else

echo "The Files $file1 and $file2 are different."

fi
```

Bash
Copy

**Output:**

```
Enter the 1st file name: article1.txt

Enter the 2nd file name: text_file1.txt



The Files article1.txt and text_file1.txt are identical.
```

Bash
Copy

## 63. Delete a Given File If It Exists

To check if a file exists, use `-f` followed by the variable that holds the filename to check. Then remove it with `rm` command if `-f $file` is true.

```
#!/bin/bash


read -p "Enter the file name for deletion: " file


if [ -f $file ]

then

rm $file
```

```
echo "The file $file deleted successfully!"

else

echo "Error! The file $file does not exist."

fi
```

Bash
Copy

**Output:**

```
Enter the file name for deletion: article1.txt



The file article1.txt deleted successfully!
```

Bash
Copy

## 64. Rename a File

All you have to do is enter the **old filename** and the **new filename**. The script will rename the file if it is available in the directory. If the file is not in the path, then it will display an error message. You can rename an existing file using the script below:

```
#!/bin/bash


read -p "Enter the file name: " file

read -p "Enter new file name: " new_file


if [ -f $file ]
```

```
then

mv "$file" "$new_file"

echo "The file $file has been renamed as $new_file!"

else

echo "Error! The file $file does not exist."

fi
```

Bash
Copy

**Output:**

```
Enter the file name: poem.txt

Enter new file name: daffodils.txt



The file poem.txt has been renamed as daffodils.txt!
```

Bash
Copy

## 65. Check the Permissions of a File

Use `-r` for readability check, `-w` for writeability check, and `-x` for excitability check. The script below checks permissions for the given filename and lists the active permissions of the current user:

```
#!/bin/bash



read -p "Enter the file name: " file
```

```bash
if [ -f $file ]; then

if [ -r "$file" ]; then

echo "Readable"

fi

if [ -w "$file" ]; then

echo "Writable"

fi

if [ -x "$file" ]; then

echo "Executable"

fi

else

echo "Error! The file $file does not exist."

fi
```

Bash
Copy

**Output:**

```
Enter the file name: daffodils.txt

Readable

Writable
```

Bash
Copy

## 66. Sets the Permissions of a Directory for the Owner

Use `-d $dir` to check if the directory exists. Then use `chmod u+rx $dir` to set writing and executing permission of the directory for the user. Below is the complete script of doing the task:

```
#!/bin/bash


read -p "Enter the directory name: " dir


if [ -d $dir ]; then
chmod u+rwx $dir
echo "Directory permissions have been updated!"
else
echo "Error! The directory $dir does not exist."
fi
```

Bash
Copy

**Output:**

```
Enter the directory name: Documents

Directory permissions have been updated!
```

Bash
Copy

## 67. Change the File Owner

Use `chown` command to change the ownership of a file. The script here changes the owner of a file if the file exists in the directory. Since changing ownership

requires administrator permissions, you will need to give the **sudo password** while running the script below:

```bash
#!/bin/bash

read -p "Enter the file name: " file
read -p "Enter file owner name: " owner

if [ -f $file ]; then
sudo chown $owner $file
echo "Changed file owner to $owner!"
else
echo "Error! The file $file does not exist."
fi
```

Bash

Copy

**Output:**

```
Enter the file name: daffodils.txt
Enter file owner name: tom

[sudo] password for anonnya:

Changed file owner to tom!
```

Bash

Copy

## 68. Change the Overall Permissions of a File

You can change the permissions of an existing file using the script below. All you have to do is enter the filename, the permissions in **absolute mode,** and the **sudo** password to activate administrative privileges. The script will update the file permissions if it is available in the directory. If the file is not in the path, then it will display an error message. Here is the script:

```
#!/bin/bash


read -p "Enter the file name: " file

read -p "Enter new permissions in Absolute Mode: " permissions


if [ -f $file ]; then

sudo chmod $permissions $file

echo "Permissions for the file $file has been changed!"

else

echo "Error! The file $file does not exist."

fi
```

Bash

Copy

**Output:**

```
Enter the file name: daffodils.txt

Enter new permissions in Absolute Mode: 777
```

```
[sudo] password for anonnya:


Permissions for the file daffodils.txt has been changed!
```

Bash
Copy

## 69. Check a Remote Host for its Availability

The following script checks the network status of a remote host. You will need to enter the host IP address as input and it will return a message saying if the host is up or down.

```
#!/bin/bash


read -p "Enter remote host IP address:" ip


ping -c 1 $ip
if [ $? -eq 0 ]
then
echo "-----------------------"
echo "Host is up!"
echo "-----------------------"
else
echo "-----------------------"
```

```
echo "Host is down!"

echo "----------------------"

fi
```

Bash
Copy

**Output:**

```
Enter remote host IP address:192.168.0.6


PING 192.168.0.6 (192.168.0.6) 56(84) bytes of data.

64 bytes from 192.168.0.6: icmp_seq=1 ttl=64 time=4.10 ms

--- 192.168.0.6 ping statistics ---


1 packets transmitted, 1 received, 0% packet loss, time 0ms


rtt min/avg/max/mdev = 4.095/4.095/4.095/0.000 ms

----------------------

Host is up!

----------------------
```

Bash
Copy

## 70. Test if a Remote Port is Open

Takes a host address and port number as the input. If the connection to the host through the port number is successful then it verifies that the port is open. The script below checks the network connection in a system port:

```
#!/bin/bash


read -p "Enter host address:" HOST

read -p "Enter port number:" PORT


nc -z -v -w5 "$HOST" "$PORT"


if [ $? -eq 0 ]; then

echo "---------------------------------------------"

echo "Port $PORT on $HOST is open"

echo "---------------------------------------------"

else

echo "---------------------------------------------"

echo "Port $PORT on $HOST is closed"

echo "---------------------------------------------"

fi
```

Bash

Copy

**Output:**

```
Enter host address:192.168.0.107
```

```
Enter port number:80


Connection to 192.168.0.107 80 port [tcp/http-alt] succeeded!

-----------------------------------------------

Port 80 on 192.168.0.107 is open

-----------------------------------------------
```

Bash

Copy

## 71. Checking Network Connectivity

Use the `ping` command to check network connectivity. The below script checks network connectivity to a remote host via the internet. If there is a successful connection then it returns the status “Internet connection is up”. Otherwise, returns “Internet connection is down”.

```
#!/bin/bash


read -p "Enter a host address:" HOST


if ping -q -c 1 -W 1 $HOST >/dev/null; then

echo "-----------------------------------------------"

echo "Internet connection is up"

echo "-----------------------------------------------"

else
```

```
echo "---------------------------------------------"

echo "Internet connection is down"

echo "---------------------------------------------"

fi
```

Bash
Copy

**Output:**

```
Enter a host address:192.168.0.107

---------------------------------------------

Internet connection is up

---------------------------------------------
```

Bash
Copy

## 72. Automating Network Configuration

The following bash script configures a network IP address and subnet mask. Upon configuration, it sets up the gateway and DNS server at the given addresses. All four IP addresses are passed as user input. It will return an error message if it is unsuccessful at running any of the commands.

```
#!/bin/bash


echo "Enter network configuration variables:"

read -p "Enter an IP address: " ip_addr

read -p "Enter a subnet mask: " subnet_mask
```

```
read -p "Enter a Gateway address: " gateway

read -p "Enter a DNS address: " dns



# Configure network interface
sudo ifconfig eth0 $ip_addr netmask $subnet_mask up
if [ $? -eq 0 ]; then

# Set default gateway
sudo route add default gw $gateway
if [ $? -eq 0 ]; then

# Set DNS servers
sudo echo "nameserver $dns" > /etc/resolv.conf
if [ $? -eq 0 ]; then
echo "------------------------------------------------"
echo "Network configuration completed"
echo "------------------------------------------------"
else
echo "------------------------------------------------"
echo "Error while setting the DNS server."
fi
else
echo "------------------------------------------------"
echo "Error while setting the default gateway."
fi
else
echo "------------------------------------------------"
echo "Network Configuration Failed."
fi
```

Bash

Copy

**Note:** If **ipconfig** is not installed in your system, install it using:

```
sudo apt install ipconfig
```

Bash

Copy

**Output:**

```
Enter network configuration variables:

Enter an IP address: 192.168.0.108

Enter a subnet mask: 255.255.255.0

Enter a Gateway address: 192.168.0.1

Enter a DNS address: 8.8.8.8

---------------------------------------------

Network configuration completed

---------------------------------------------
```

Bash

Copy

## 73. Check if a Process is Running

Use `pgrep` to check if a process is currently running on your system or not. The below script shows an example how to use it:

```
#!/bin/bash


read -p "Enter process name: " process


if pgrep $process > /dev/null

then

echo "Process is running."

else

echo "Process is not running."
```

```
fi
```

Bash
Copy

**Output:**

```
Enter process name: bash

Process is running.
```

Bash
Copy

## 74. Start a Process if It’s Not Running

Check if the process is not running using `! pgrep`. Then pass the process name as user input to start it. The script below to start a not running process:

```
#!/bin/bash


read -p "Enter process name: " process


if ! pgrep $process > /dev/null

then

/path/to/process_name &

echo "The Process $process has now started."

else

echo "The Process is already running."

fi
```

Bash
Copy

**Output:**

```
Enter process name: bash

The Process is already running.
```

Bash
Copy

## 75. Stop a Running Process

Use `pkill` command to stop a running process. The script below can stop a process if it runs in the system:

```
#!/bin/bash


read -p "Enter process name: " process


if pgrep $process > /dev/null

then

pkill $process

echo "The Process $process has stopped."

else

echo "The Process $process is not running."

fi
```

Bash
Copy

**Output:**

```
Enter process name: nslookup

The Process nslookup has stopped.
```

Bash
Copy

## 76. Restart a Process

Use the following script aims to take a process name as input and then restart it:

```
#!/bin/bash


read -p "Enter process name: " process


pid=$(pgrep -f $process)

if [ -n "$pid" ]; then

kill $pid

sleep 5

if pgrep -f $process> /dev/null; then

echo "Process did not exit properly, force killing..."

kill -9 $pid

fi

fi

process_path=$(which $process)
```

```
$process_path & echo "Process restarted."
```

Bash
Copy

If the process is already running then the script kills the process and starts over. After the first kill command, it waits for **5** seconds. If by then the process does not terminate then it will force kill that process before restarting.

**Output:**

```
Enter process name: firefox

Process restarted.
```

Bash
Copy

## 77. Monitor a Process and Restart If It Crashes

The script here takes a process name as input from the user and checks for its status every 5 seconds:

```
#!/bin/bash


read -p "Enter process name: " process


process_path=$(which $process)

while true

do

if pgrep $process > /dev/null

then
```

```
echo "The Process $process is running."

sleep 5

else

$process_path &

echo "The Process $process restarted."

continue

fi

done
```

Bash
Copy

**Output:**

```
Enter process name: firefox



The Process firefox is running.

The Process firefox is running.
```

Bash
Copy

## 78. Display the Top 10 CPU-Consuming Processes

The script below lists the top 10 CPU-consuming processes and prints the Process ID, the percentage of CPU usage along with the command that runs each process:

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

```
echo "The current top 10 CPU-consuming processes: "

ps -eo pid,%cpu,args | sort -k 2 -r | head -n 11
```

Bash
Copy

## 79. Display the Top 10 Memory-Consuming Processes

The given script lists the top 10 memory-consuming processes and prints the Process ID, percentage of memory usage as well as the commands for running each process:

```
#!/bin/bash


echo "The current top 10 Memory-consuming processes: "

ps -eo pid,%mem,args | sort -k 2 -r | head -n 11
```

Bash
Copy

## 80. Kill Processes of a Specific User

Use `pkill -u` followed by the **username** to kill all process of the user. The following script is created to kill all the processes of a specific user:

```
#!/bin/bash


read -p "Enter username: " user


sudo pkill -u $user
```

```
echo "All processes of user $user have been terminated."
```

Bash
Copy

**Output:**

```
Enter username: tom

[sudo] password for anonnya:



All processes of user tom have been terminated.
```

Bash
Copy

## 81. Kill All Processes That are Consuming More Than a Certain Amount of CPU

Use the script below to kill all processes that are consuming more than a certain amount of CPU:

```
#!/bin/bash



read -p "Enter CPU usage threshold: " threshold



if [ "$(ps -eo pid,%cpu | awk -v t=$threshold '$2 > t {print $1}' | wc -c)" -gt 0 ]; then

for pid in $(ps -eo pid,%cpu | awk -v t=$threshold '$2 > t {print $1}')

do
```

```
kill $pid

done

echo "All processes consuming more than $threshold% CPU killed."

else

echo "There are no process consuming more than $threshold% CPU."

fi
```

Bash

Copy

This script takes a CPU usage percentage as user input and terminates all the running processes that are consuming more than the entered CPU threshold. If there is no process above that threshold, then it returns a message saying there are no such processes.

## 82. Kill All Processes That are Consuming More Memory

Use the script below to kill all processes that are consuming more than a certain amount of memory:

```
#!/bin/bash



read -p "Enter memory usage threshold (in KB): " threshold



if [ "$(ps -eo pid,%mem | awk -v t=$threshold '$2 > t {print $1}'
| wc -c)" -gt 0 ]; then
```

```bash
for pid in $(ps -eo pid,%mem | awk -v t=$threshold '$2 > t {print $1}')

do

kill $pid

done

echo "All processes consuming more than $threshold KB memory killed."

else

echo "There are no process consuming more than $threshold KB memory."

fi
```

Bash

Copy

This script takes a memory space percentage as user input and terminates all the running processes that are consuming more than the entered space threshold. If there is no process above that threshold, then it returns a message saying there are no such processes.

## 83. Check the Number of Logged-in Users

Use `who` command to check logged-in users. Then use a pipe with `wc -l` to count the number of users. Here's how:

```bash
#!/bin/bash



users=$(who | wc -l)

echo "Number of currently logged-in users: $users"
```

Bash
Copy

## 84. Check Operating System Information

The following script displays information regarding the machine's operating system:

```bash
#!/bin/bash



os_name=$(uname -s)

os_release=$(uname -r)

os_version=$(cat /etc/*-release | grep VERSION_ID | cut -d '"' -f 2)

os_arch=$(uname -m)



echo "OS Name: $os_name"

echo "OS Release: $os_release"

echo "OS Version: $os_version"

echo "OS Architecture: $os_arch"
```

Bash
Copy

It retrieves and lists the os name, release, version as well as system architecture.

**Output:**

```
OS Name: Linux

OS Release: 5.19.0-38-generic
```

```
OS Version: 22.04

OS Architecture: x86_64
```

Bash
Copy

## 85. Check the System's Memory Usage

The script given below calculates the percentage of memory being used:

```
#!/bin/bash


mem=$(free -m | awk 'NR==2{printf "%.2f%%", $3*100/$2}')

echo "Current Memory Usage: $mem"
```

Bash
Copy

The `$3*100/$2` expression converts the usage into percentages and displays the output with two decimal places.

## 86. Check the System's Disk Usage

Use `df` command to check disk usage. The following script displays the percentage of disk space used on the root (/) file system:

```
#!/bin/bash


disk=$(df -h | awk '$NF=="/"{printf "%s", $5}')

echo "Current Disk Usage: $disk"
```

Bash
Copy

## 87. Check the System’s Network Information

Use the script below to get the network information of your system:

```bash
#!/bin/bash


echo " System’s network information:-"


ip=$(hostname -I)

echo "IP Address: $ip"

gw=$(ip route | awk '/default/ { print $3 }')

echo "Gateway: $gw"

dns=$(grep "nameserver" /etc/resolv.conf | awk '{print $2}')

echo "DNS Server: $dns"
```

Bash
Copy

It lists the system’s IP address, Gateway address, and DNS server address.

**Output:**

```
System’s network information:-


IP Address: 192.168.0.109

Gateway: 192.168.0.1

DNS Server: 127.0.0.53
```

Bash
Copy

## 88. Check the Uptime

Use `uptime` command to find out the uptime of the system. Here's an example script:

```bash
#!/bin/bash


uptime | awk '{print $1,$2,$3}' | sed 's/,//'

echo "Uptime: $uptime"
```

Bash
Copy

The given script can be used to find out the uptime of the system. It will return two values. The first one is the current time, and the second one is the uptime i.e. for how long the system has been running.

## 89. Check the System Load Average

The following script returns the system's Load Average:

```bash
#!/bin/bash


loadavg=$(uptime | awk '{print $10,$11,$12}')

echo "Load Average: $loadavg"
```

Bash
Copy

It will extract the load averages for the past 1, 5, and 15 minutes from the system's uptime and display their average on the screen.

## 90. Check the System Architecture

To determine your current machine's architecture, use `uname -m`. Here's how:

```
#!/bin/bash


arch=$(uname -m)

echo "System Architecture: $arch"
```

Bash
Copy

**Output:**

```
System Architecture: x86_64
```

Bash
Copy

## 91. Count the Number of Files in the System

You can use the script below to find the available number of files on your machine:

```
#!/bin/bash


count=$(find / -type f | wc -l)

echo "Number of files in the system: $count."
```

Bash
Copy

It runs the find command to check every file on the system and returns the total file count.

## 92. Automated Backup with Bash

The following script creates a backup file of a given directory:

```bash
#!/bin/bash



read -p "Enter path of the directory to backup: " source_dir

read -p "Enter destination path for backup: " backup_dir



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

backup_file="backup-$date.tar.gz"



# Create backup directory if it doesn't exist
if [ ! -d "$backup_dir" ]; then
mkdir -p "$backup_dir"
fi

# Create backup archive
tar -czf "$backup_dir/$backup_file" "$source_dir"
echo "Completed Creating backup at: $backup_dir."
```

Bash

Copy

Here, The source directory path and the destination directory path are user inputs. The backup file is named along with the current date for keeping track. Upon completion of the task, it returns the path where the backup archive resides.

**Output:**

```
Enter path of the directory to backup: /home/anonnya/Documents

Enter destination path for backup: /home/anonnya/Desktop

tar: Removing leading `/' from member names

Completed Creating backup at: /home/anonnya/Desktop.
```

Bash
Copy

## 93. Generate Alert if Disk Space Usage Goes Over a Threshold

The script below generates an alert if the disk space usage goes over a threshold. It takes the threshold and a filename from the user. The alert is then generated in that file along with the disk space usage. If the space consumed is less than the threshold then the file remains empty.

```
#!/bin/bash



read -p "Enter filename to write alert: " file

touch $file

read -p "Enter disk space threshold: " threshold

df -H | grep -vE "^Filesystem|tmpfs|cdrom" | awk '{ print $5 " "
$1 }' | while read output;

do

usage=$(echo $output | awk '{ print $1}' | cut -d'%' -f1)

if [ $usage -ge $threshold ]; then
```

```
partition=$(echo $output | awk '{ print $2 }')

echo "Alert for \"$partition: Almost out of disk space $usage% as
on $(date) " >> $file

break

fi

done

cat $file
```

Bash
Copy

## 94. Create a New User and Add It to the Sudo Group

Use `useradd` command with `-m -s` option to create a new user and add it to the sudo group. he below script shows how:

```
#!/bin/bash



read -p "Enter username: " username

read -p "Enter password: " password



useradd -m -s /bin/bash -p $(openssl passwd -1 $password)
$username

if [ $? -eq 0 ]; then

usermod -a -G sudo $username



mkdir /home/$username/mydir
```

```
chown -R $username:$username /home/$username/mydir

usermod -d /home/$username/mydir $username


echo "$username ALL=(ALL) NOPASSWD:ALL" >> /etc/sudoers


echo "User $username created successfully!"

echo "User $username added to sudo group!"

else

echo "Error while creating user!"

fi
```

Bash

Copy

The script will take the username and password as input to create the user. It will also create a **home directory for the user** besides adding the account to the sudo group.

**Output:**

```
Enter username: Jim

Enter password: linuxsimply


User Jim created successfully!

User Jim added to sudo group!
```

Bash

Copy

## 95. Monitor Network Traffic

The following script monitors the receiving (RX) and transmitting(TX) packets over a network. Enter the interface name that you want to monitor. Then every 10 seconds, it will display the total packets received and transmitted and their size in KB.

```
#!/bin/bash


read -p "Enter network interface to monitor traffic (ex. eth0): "
net



while true

do

rx=$(ifconfig $net | grep "RX packets" | awk '{print $3 $6 $7}')

tx=$(ifconfig $net | grep "TX packets" | awk '{print $3 $6 $7}')

echo "$(date) RX: $rx, TX: $tx"

sleep 10

done
```

Bash
Copy

**Output:**

```
Enter network interface to monitor traffic (ex. eth0): ens33



Wed May 10 16:55:40 +06 2023 RX: 342(40.4KB), TX: 171(18.4KB)
```

```
Wed May 10 16:55:51 +06 2023 RX: 355(41.6KB), TX: 178(19.0KB)

Wed May 10 16:56:01 +06 2023 RX: 361(42.0KB), TX: 178(19.0KB)

Wed May 10 16:56:11 +06 2023 RX: 361(42.0KB), TX: 178(19.0KB)
```

Bash
Copy

## 96. Monitor CPU and Memory Usage

The script below can be used to monitor the CPU and Memory usage of a system. It extracts the CPU and Memory usage information every 10 seconds and converts them into a percentage for displaying on the screen.

```
#!/bin/bash



while true

do

cpu=$(top -bn1 | grep "Cpu(s)" | sed "s/.*, *\([0-9.]*\)%*
id.*/\1/" | awk '{print 100 - $1"%"}')

mem=$(free -m | awk 'NR==2{printf "%.2f%%", $3*100/$2 }')

echo "$(date) CPU Usage: $cpu, Memory Usage: $mem"

sleep 10

done
```

Bash
Copy

## 97. Creating a Script and Adding It to PATH

You can use the script below to customize another script and make it runnable. The script here will take another script name and the commands to write within this new script as user inputs. After receiving the input values, it will update the permission modes of the desired script and add it to the `$PATH` variable to make the new script runnable. After creation, you can run this new script with the `bash` keyword.

```bash
#!/bin/bash



read -p "Enter a name for the command: " my_comm

echo "Enter commands to write on script:"

read comm

read -p "Enter path to the directory containing the command: "
comm_path



# Create script for custom command
echo "#!/bin/bash" > $my_comm.sh
echo "$comm" >> $my_comm.sh

# Make script executable
chmod +x $my_comm.sh

# Add script to PATH
export PATH="$PATH$comm_path/$my_comm.sh"

echo "A script called $my_comm has been created."
```

Bash

Copy

**Output:**

```
Enter a name for the command: echo_hello
```

```
Enter commands to write on script:



echo "Hello from custom command!!"



Enter path to the directory containing the command:
/home/anonnya/bin



A script called echo_hello has been created.
```

Bash
Copy

## 98. Running a Command At Regular Intervals

Use `crontab` to run a command at regular intervals. The script given below runs a command at regular time intervals:

```
#!/bin/bash



read -p "Enter command to run: " com

command_to_run=$(which $com)

read -p "Enter interval for running the command (m h dom mon
dow): " interval
```

```
# Add command to crontab
(crontab -l ; echo "$interval $command_to_run") | sort - | uniq -
| crontab -
echo "Command added to crontab and will run at $interval"
```

Bash

Copy

To achieve this task the user has to enter the desired command and the interval for running that command. The interval passed as input must be in the following format: m h dom mon dow.

## 99. Downloading Files From a List of URLs Using Bash

Use `curl -o` command followed by the filename and URL to download files form the URL. Here's an example script:

```
#!/bin/bash


read -p "Enter the filename containing URLs: " url_file


while read -r url; do

filename=$(basename "$url")
```

```
curl -o "$filename" "$url"

echo "Completed Download $filename"

done < "$url_file"

echo "-----------------------------------------------------------------
--------------------------------"

echo "All files downloaded successfully!"
```

Bash

Copy

The following script takes a filename as input where a list of URLs should be stored. The script will iterate through the list of URLs and download the available contents on the link. It displays each download information on the terminal along with the "Completed Download" message. Upon downloading files from all the URLs, it shows another message saying "All files downloaded successfully!".

## 100. Organize a Directory Based on File Types

The script given below organizes files in a directory depending on their type. The user needs to give a destination directory path to organize the files along with the source directory path.

```
#!/bin/bash



# Specify the source and destination directories
read -p "Enter path to the source directory: " source_dir
read -p "Enter path to the destination directory: " dest_dir

# Create the destination directories if they don't exist
mkdir -p "${dest_dir}/Documents"
mkdir -p "${dest_dir}/Images"
mkdir -p "${dest_dir}/Music"
mkdir -p "${dest_dir}/Videos"
```

```bash
mkdir -p "${dest_dir}/Others"

# Move files to the appropriate directories based on their extensions
for file in "${source_dir}"/*; do
if [ -f "${file}" ]; then
extension="${file##*.}"
case "${extension}" in
txt|pdf|doc|docx|odt|rtf)
mv "${file}" "${dest_dir}/Documents"
;;
jpg|jpeg|png|gif|bmp)
mv "${file}" "${dest_dir}/Images"
;;
mp3|wav|ogg|flac)
mv "${file}" "${dest_dir}/Music"
;;
mp4|avi|wmv|mkv|mov)
mv "${file}" "${dest_dir}/Videos"
;;
*)
mv "${file}" "${dest_dir}/Others"
;;
esac
fi
done
echo "Files organized successfully!"
```

Bash

Copy

This script will create five directories: 1) Documents, 2) Images, 3) Music, 4) Videos, and 5) Others only if they do not already exist on the destination path. Then, it will check all the files and their extension and move them to the corresponding directory. If there is any unknown file extension, then the script will move the file to the Others Directory.

# Conclusion

This article covers **100 shell script examples** that a user can frequently use. These examples range from basic to advanced topics along with the preliminary **concepts of script writing and configurations**. Furthermore, the examples are divided into sections and subsections depending on their topic and level of understanding. Therefore, it is a proper guide for users of every category.

# People Also Ask

## What is $1 shell script?

In a shell script, $1 refers to the **first command-line argument** passed to the script when executed. It allows the script to access and use the value of the first argument provided by the user.

## What is the popular shell script?

One of the popular and widely used shell scripts is the **Bash** shell script. Bash (Bourne Again SHell) is the default shell for most Unix-like operating systems, including Linux. Many scripts are written in Bash due to its versatility and widespread adoption. It is commonly used for automating tasks, managing system configurations, and creating custom utilities on Unix-based systems.

## How many shell scripts are there?

There are many different shell scripting languages such as (Bourne Again SHell) and **PowerShell**. Apart from these, there are other shells like **sh** (Bourne Shell), **csh** (C Shell), **ksh** (Korn Shell), and more. However, Bash and PowerShell are among the most popular and widely used in their. respective environment.

## Is shell script a language?

**No**, a shell script is not a programming language. It is a script written in a shell language, such as Bash, used for automating tasks and executing commands in a command-line interface.

## How to write a shell script?

To write a shell script:

1. Use a text editor to create a new file, e.g. nano script.sh.
2. Add #!/bin/bash at the top to specify the shell.
3. Write your commands.
4. Save and exit the file.
5. Make it executable with chmod +x script.sh.
6. Run the script with ./script.sh.

## Is shell script faster than Python?

The execution speed of a shell script compared to Python depends on the complexity of the tasks. Generally, Python tends to be faster for more complex or CPU-intensive tasks, while simple shell scripts may execute quickly. The choice between them often depends on the specific requirements of the task at hand.

## Which shell is best for scripting?

**Bash** (Bourne Again Shell) is widely considered one of the best shells for scripting, offering a balance of features, compatibility, and widespread use on Unix-like systems, including Linux.

Categories[Bash Programming](#)Tags[bash scripting examples](#)

**Anonnya Ghosh**

Hello there! I am Anonnya Ghosh, a Computer Science and Engineering graduate from Ahsanullah University of Science and Technology (AUST). Currently, I am working as a Linux Content Developer Executive at SOFTEKO. The strong bond between Linux and cybersecurity drives me to explore this world of open-source architecture. I aspire to learn new things further and contribute to the field of CS with my experience. Read Full Bio

# 1 thought on “100 Shell Script Examples [Free Downloads]”

1. **Annpure Bhaskar**
   December 18, 2024 at 9:40 am

   Dear.

   Anonnya Ghosh.

   I am trying to learn linux as beginner and I want to appear in rhel linux administrator exam. I was looking for bash scripting where all examples of bash scripts are available.

   I got your web page really I appreciate it. I am practicing it. Now i am very much confident to slove many scripts.

   Thank you very much.

   Bhaskar A.

   Reply

## Leave a Comment

Comment

NameEmailWebsite

Save my name, email, and website in this browser for the next time I comment.

# Related Articles

- Advanced Shell Script With Examples [Free Downloads]
- Basic Shell Script Examples [Free Downloads]
- 19 Examples of Variables in Shell Script [Free Downloads]
- 16 Examples of For Loop in Shell Script [Free Downloads]

LinuxSimply serves as an informational repository about the Linux operating system.

## Get In Touch!

Subscribe

Built with Kit

## Company

- About Us
- Contact Us
- Career
- All Contributors

## Services

- Pricing
- System Administration
- Server Management
- Bash Automation

## Resources

- Courses
- Help Forum
- Cheat Sheets
- Free Downloads

## Legal Corner

- Disclaimer
- Privacy Policy
- Editorial Policy
- Terms & Conditions