Some examples extracted from:
https://linuxconfig.org/bash-scripting-tutorial

https://ryanstutorials.net/linuxtutorial/

```
#!/bin/bash
# declare STRING variable
STRING="Hello World"
#print variable on a screen
echo $STRING
```

```
$ chmod +x hello_world.sh
```

```
./hello_world.sh
```

Spremenljivke:

```
#!/bin/bash
 STRING="HELLO WORLD!!!"
 echo $STRING
```

Globalne in lokalne spremeljivke

```
#!/bin/bash
#Define bash global variable
#This variable is global and can be used anywhere in this bash script
VAR="global variable"
function bash {
#Define bash local variable
#This variable is local to bash function only
local VAR="local variable"
echo $VAR
}
echo $VAR
bash
# Note the bash global variable did not change
# "local" is bash reserved word
echo $VAR
```

Bash Arithmetics

```
#!/bin/bash

echo '### let ###'
# bash addition
let ADDITION=3+5
echo "3 + 5 =" $ADDITION

# bash subtraction
let SUBTRACTION=7-8
echo "7 - 8 =" $SUBTRACTION

# bash multiplication
let MULTIPLICATION=5*8
echo "5 * 8 =" $MULTIPLICATION

# bash division
let DIVISION=4/2
echo "4 / 2 =" $DIVISION

# bash modulus
let MODULUS=9%4
echo "9 % 4 =" $MODULUS

# bash power of two
let POWEROFTWO=2**2
echo "2 ^ 2 =" $POWEROFTWO


echo '### Bash Arithmetic Expansion ###'
# There are two formats for arithmetic expansion: $[ expression ]
# and $(( expression #)) its your choice which you use

echo 4 + 5 = $((4 + 5))
echo 7 - 7 = $[ 7 - 7 ]
echo 4 x 6 = $((3 * 2))
echo 6 / 3 = $((6 / 3))
echo 8 % 7 = $((8 % 7))
echo 2 ^ 8 = $[ 2 ** 8 ]


echo '### Declare ###'

echo -e "Please enter two numbers \c"
# read user input
read num1 num2
declare -i result
result=$num1+$num2
echo "Result is:$result "

# bash convert binary number 10001
result=2#10001
echo $result
```

```
# bash convert octal number 16
result=8#16
echo $result

# bash convert hex number 0xE6A
result=16#E6A
echo $result
```

```
linuxconfig.org$ ./arithmetic_operations.sh
### let ###
3 + 5 = 8
7 - 8 = -1
5 * 8 = 40
4 / 2 = 2
9 % 4 = 1
2 ^ 2 = 4
### Bash Arithmetic Expansion ###
4 + 5 = 9
7 - 7 = 0
4 x 6 = 6
6 / 3 = 2
8 % 7 = 1
2 ^ 8 = 256
### Declare ###
Please enter two numbers 23 45
Result is:68
17
14
3690
linuxconfig.org$
```

Round floating point number

```
#!/bin/bash
# get floating point number
floating_point_number=3.3446
echo $floating_point_number
# round floating point number with bash
for bash_rounded_number in $(printf %.0f $floating_point_number); do
echo "Rounded number with bash:" $bash_rounded_number
done
```

```
linuxconfig.org:~$ ./round.sh
3.3446
Rounded number with bash: 3
linuxconfig.org:~$
```

Bash floating point calculations

```
#!/bin/bash
# Simple linux bash calculator
echo "Enter input:"
read userinput
echo "Result with 2 digits after decimal point:"
echo "scale=2; ${userinput}" | bc
echo "Result with 10 digits after decimal point:"
echo "scale=10; ${userinput}" | bc
echo "Result as rounded integer:"
echo $userinput | bc
```

```
linuxconfig.org:~$ ./simple_bash_calc.sh
Enter input:
10/3.4
Result with 2 digits after decimal point:
2.94
Result with 10 digits after decimal point:
2.9411764705
Result as rounded integer:
2
linuxconfig.org:~$
```

Argumenti

```
#!/bin/bash
# use predefined variables to access passed arguments
#echo arguments to the shell
echo $1 $2 $3 ' -> echo $1 $2 $3'

# We can also store arguments from bash command line in special array
args=("$@")
#echo arguments to the shell
echo ${args[0]} ${args[1]} ${args[2]} ' -> args=("$@"); echo ${args[0]}
${args[1]} ${args[2]}'

#use $@ to print out all arguments at once
echo $@ ' -> echo $@'

# use $# variable to print out
# number of arguments passed to the bash script
echo Number of arguments passed: $# ' -> echo Number of arguments passed:
$#'


/arguments.sh Bash Scripting Tutorial
```

Izvajanje programov

```
#!/bin/bash
# use backticks " ` ` " to execute shell command
echo `uname -o`
# executing bash command without backticks
echo uname -o
```

Branje uporabnikovega inputa

```
#!/bin/bash

echo -e "Hi, please type the word: \c "
read  word
echo "The word you entered is: $word"
```

IF ELSE

```
#!/bin/bash
directory="./BashScripting"

# bash check if directory exists
if [ -d $directory ]; then
        echo "Directory exists"
else
        echo "Directory does not exists"
fi
```

```
linuxconfig.org ~$ ./bash_if_else.sh
Directory does not exists
linuxconfig.org ~$ mkdir BashScripting
linuxconfig.org ~$ ./bash_if_else.sh
Directory exists
linuxconfig.org ~$
```

Bash Comparisons

Arithmetic Comparisons

-lt	<
-gt	>
-le	<=
-ge	>=
-eq	==
-ne	!=

```
#!/bin/bash
# declare integers
NUM1=2
NUM2=2
if [ $NUM1 -eq $NUM2 ]; then
        echo "Both Values are equal"
else
        echo "Values are NOT equal"
fi
```

```
linuxconfig.org:~$ ./statement.sh
Both Values are equal
linuxconfig.org:~$
```

```
#!/bin/bash
# declare integers
NUM1=2
NUM2=1
if [ $NUM1 -eq $NUM2 ]; then
        echo "Both Values are equal"
else
        echo "Values are NOT equal"
fi
```

```
linuxconfig.org:~$ ./statement.sh
Values are NOT equal
linuxconfig.org:~$
```

String Comparisons

=	equal
!=	not equal
<	less then
>	greater then
-n s1	string s1 is not empty
-z s1	string s1 is empty

```
#!/bin/bash
#Declare string S1
S1="Bash"
#Declare string S2
S2="Scripting"
if [ $S1 = $S2 ]; then
        echo "Both Strings are equal"
else
        echo "Strings are NOT equal"
fi
```

```
linuxconfig.org:~$ ./statement.sh
Strings are NOT equal
linuxconfig.org:~$
```

Loops

Bash for loop

```
#!/bin/bash

# bash for loop
for f in $( ls /var/ ); do
        echo $f
done
```

Running for loop from bash shell command line:

```
$ for f in $( ls /var/ ); do echo $f; done
```

```
linuxconfig.org:~$ ./for_loop.sh
backups
cache
lib
local
lock
log
mail
opt
run
spool
tmp
linuxconfig.org:~$
```

Bash while loop

```
#!/bin/bash
COUNT=6
# bash while loop
while [ $COUNT -gt 0 ]; do
        echo Value of count is: $COUNT
        let COUNT=COUNT-1
done
```

Redirections

STDOUT from bash script to STDERR

```
#!/bin/bash

 echo "Redirect this STDOUT to STDERR" 1>&2
```

To prove that STDOUT is redirected to STDERR we can redirect script's output to file:

```
linuxconfig.org$ ./redirecting.sh
Redirect this STDOUT to STDERR
linuxconfig.org$
linuxconfig.org$ ./redirecting.sh > STDOUT.txt
Redirect this STDOUT to STDERR
linuxconfig.org$ cat STDOUT.txt
linuxconfig.org$
linuxconfig.org$ ./redirecting.sh 2> STDERR.txt
linuxconfig.org$ cat STDERR.txt
Redirect this STDOUT to STDERR
linuxconfig.org$
```

STDERR from bash script to STDOUT

```
#!/bin/bash

 cat $1 2>&1
```

To prove that STDERR is redirected to STDOUT we can redirect script's output to file:

```
linuxconfig.org$ ./redirecting.sh /etc/shadow
cat: /etc/shadow: Permission denied
linuxconfig.org$
linuxconfig.org$ ./redirecting.sh /etc/shadow > STDOUT.txt
linuxconfig.org$ cat STDOUT.txt
cat: /etc/shadow: Permission denied
linuxconfig.org$
linuxconfig.org$ ./redirecting.sh /etc/shadow 2> STDERR.txt
cat: /etc/shadow: Permission denied
linuxconfig.org$ cat STDERR.txt
linuxconfig.org$
```

stdout to file

The override the default behavior of STDOUT we can use ">" to redirect this output to file:

```
$ ls file1 > STDOUT

$ cat STDOUT

file1
```

stderr to file

By default STDERR is displayed on the screen:

```
$ ls

file1  STDOUT

$ ls file2

ls: cannot access file2: No such file or directory
```

In the following example we will redirect the standard error (stderr) to a file and stdout to a screen as default. Please note that STDOUT is displayed on the screen, however STDERR is redirected to a file called STDERR:

```
$ ls

file1  STDOUT

$ ls file1 file2 2> STDERR

file1

$ cat STDERR

ls: cannot access file2: No such file or directory
```

stdout to stderr

It is also possible to redirect STDOUT and STDERR to the same file. In the next example we will redirect STDOUT to the same descriptor as STDERR. Both STDOUT and STDERR will be redirected to file "STDERR_STDOUT".

```
$ ls

file1  STDERR  STDOUT

$ ls file1 file2 2> STDERR_STDOUT 1>&2

$ cat STDERR_STDOUT

ls: cannot access file2: No such file or directory

file1
```

File STDERR_STDOUT now contains STDOUT and STDERR.

stderr to stdout

The above example can be reversed by redirecting STDERR to the same descriptor as SDTOUT:

```
$ ls

file1  STDERR  STDOUT

$ ls file1 file2 > STDERR_STDOUT 2>&1

$ cat STDERR_STDOUT

ls: cannot access file2: No such file or directory

file1
```

stderr and stdout to file

Previous two examples redirected both STDOUT and STDERR to a file. Another way to achieve the same effect is illustrated below:

```
$ ls

```

```
file1  STDERR  STDOUT

$ ls file1 file2 &> STDERR_STDOUT

$ cat STDERR_STDOUT

ls: cannot access file2: No such file or directory

file1
```

or

```
ls file1 file2 >& STDERR_STDOUT

$ cat STDERR_STDOUT

ls: cannot access file2: No such file or directory

file1
```