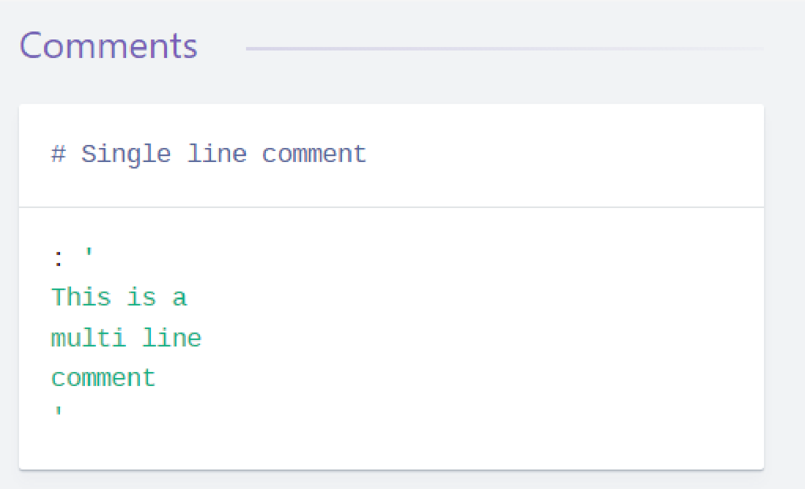
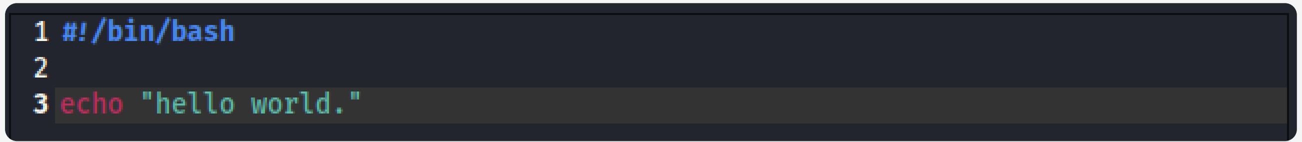
Hello World



Comments

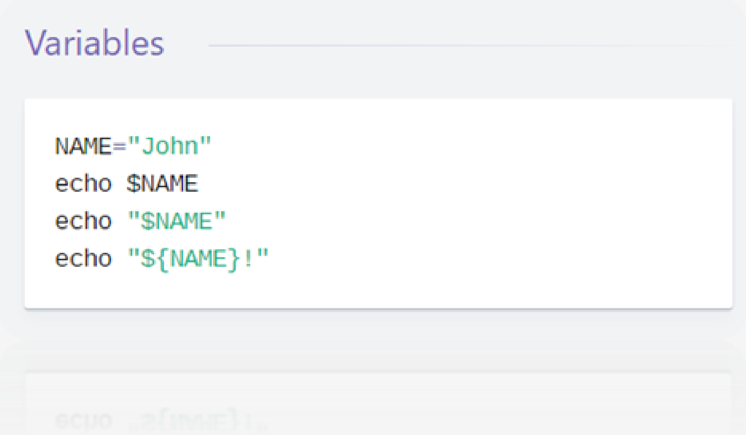
```
# Single line comment
```

```
: '
This is a
multi line
comment
'
```



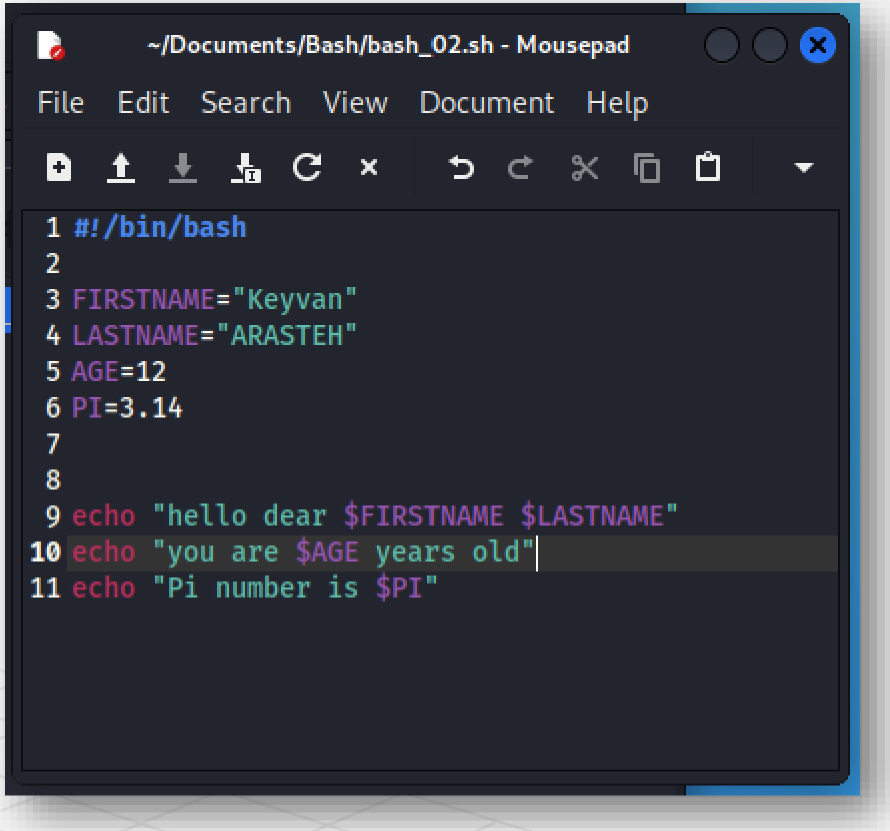
```
#!/bin/bash

echo "hello world."
```

Variables



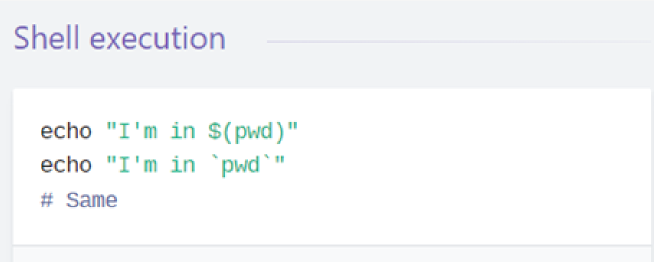
Variables

```
NAME="John"
echo $NAME
echo "$NAME"
echo "${NAME}!"
```



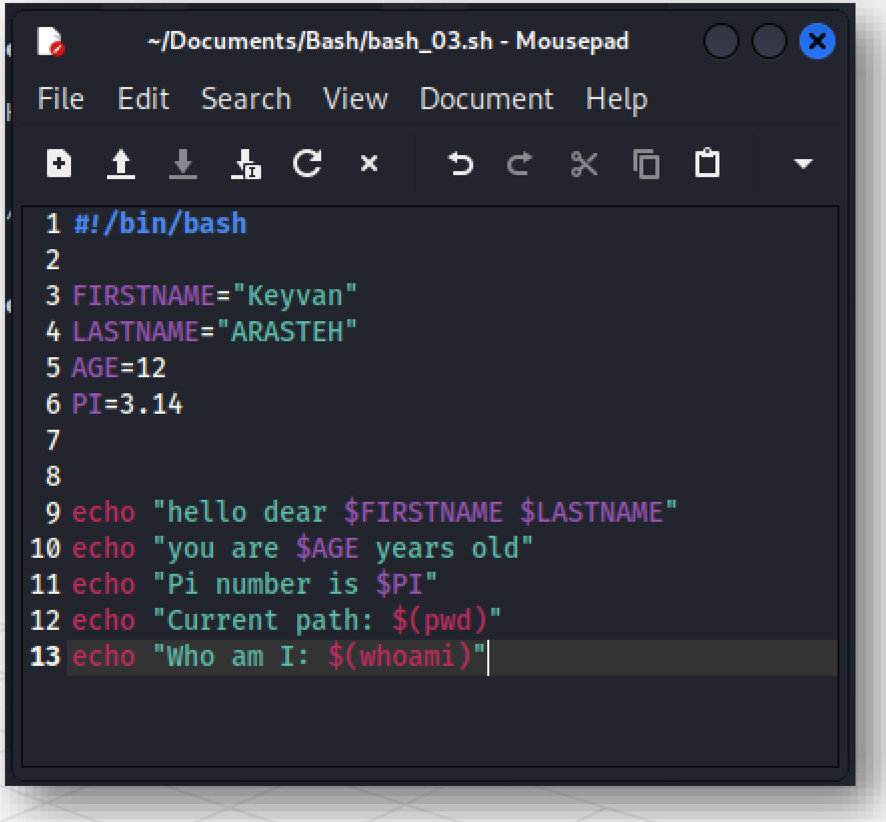
```
#!/bin/bash

FIRSTNAME="Keyvan"
LASTNAME="ARASTEH"
AGE=12
PI=3.14


echo "hello dear $FIRSTNAME $LASTNAME"
echo "you are $AGE years old"
echo "Pi number is $PI"
```

Shell execution



Shell execution

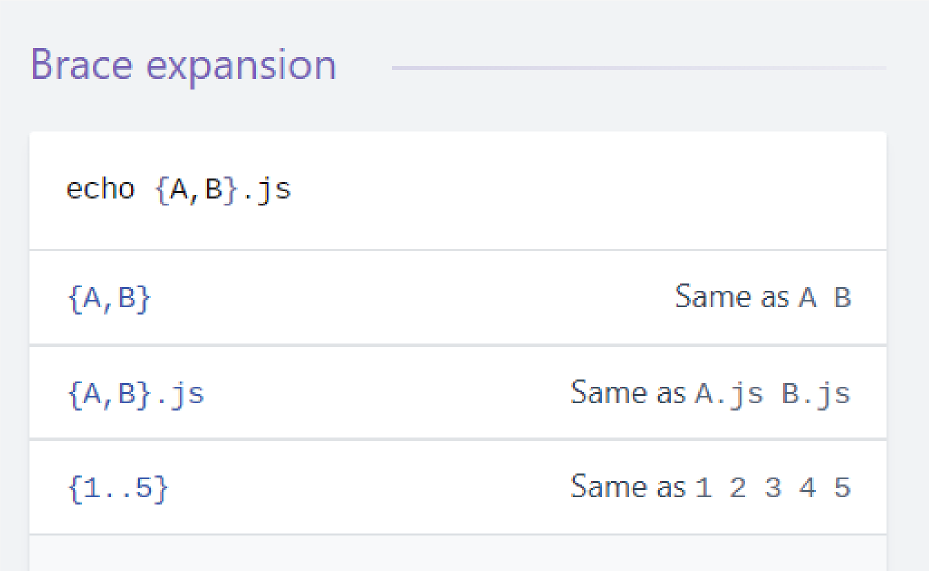
```
echo "I'm in $(pwd)"
echo "I'm in `pwd`"
# Same
```



~/Documents/Bash/bash_03.sh - Mousepad
File Edit Search View Document Help

```
#!/bin/bash

FIRSTNAME="Keyvan"
LASTNAME="ARASTEH"
AGE=12
PI=3.14


echo "hello dear $FIRSTNAME $LASTNAME"
echo "you are $AGE years old"
echo "Pi number is $PI"
echo "Current path: $(pwd)"
echo "Who am I: $(whoami)"
```

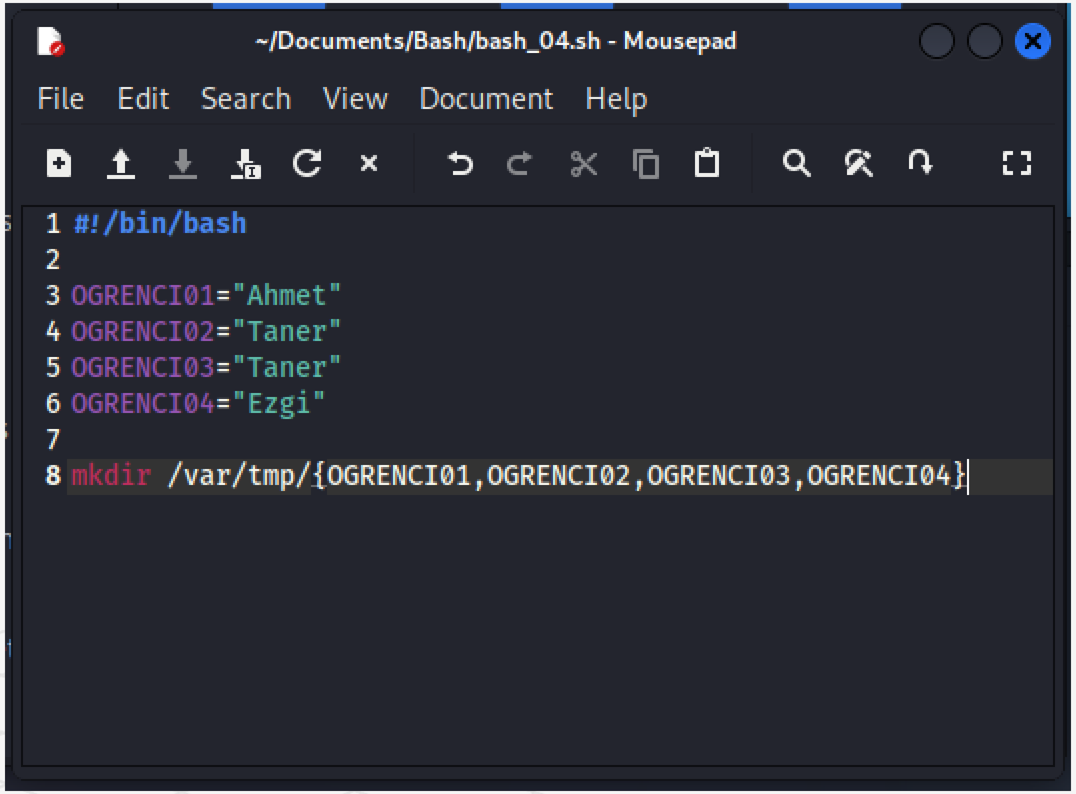
Brace expansion (string list)



Brace expansion

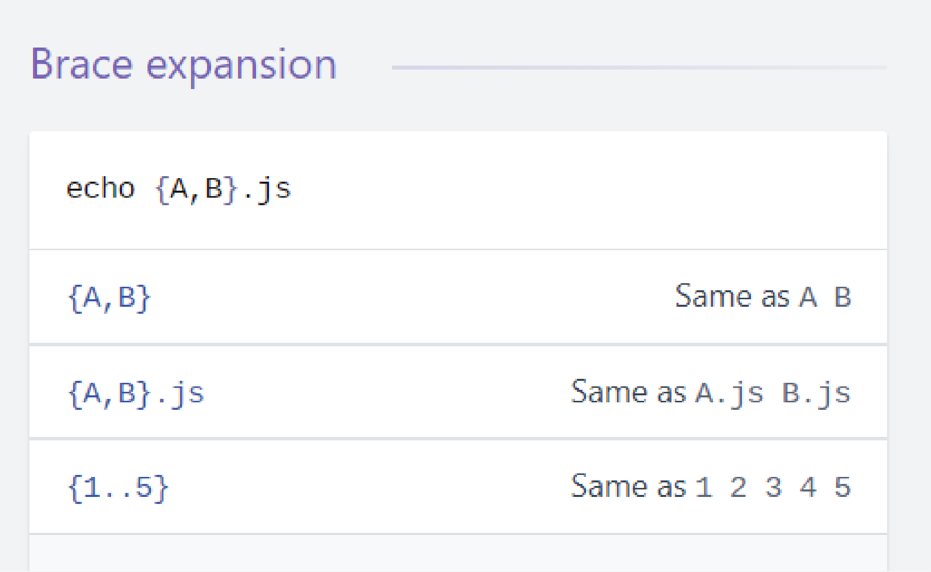
```
echo {A,B}.js
```

`{A,B}`	Same as `A B`
`{A,B}.js`	Same as `A.js B.js`
`{1..5}`	Same as `1 2 3 4 5`



```bash
#!/bin/bash

OGRENCI01="Ahmet"
OGRENCI02="Taner"
OGRENCI03="Taner"
OGRENCI04="Ezgi"

mkdir /var/tmp/{OGRENCI01,OGRENCI02,OGRENCI03,OGRENCI04}
```

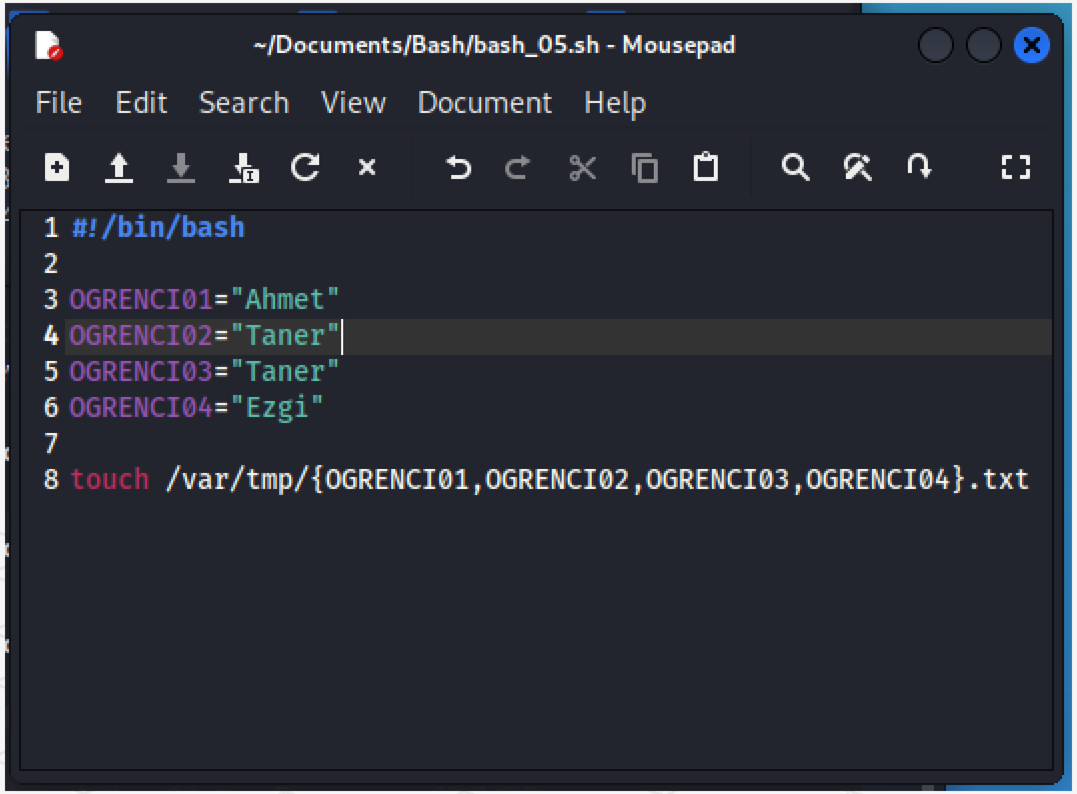
Brace expansion (string list)



Brace expansion

```
echo {A,B}.js
```

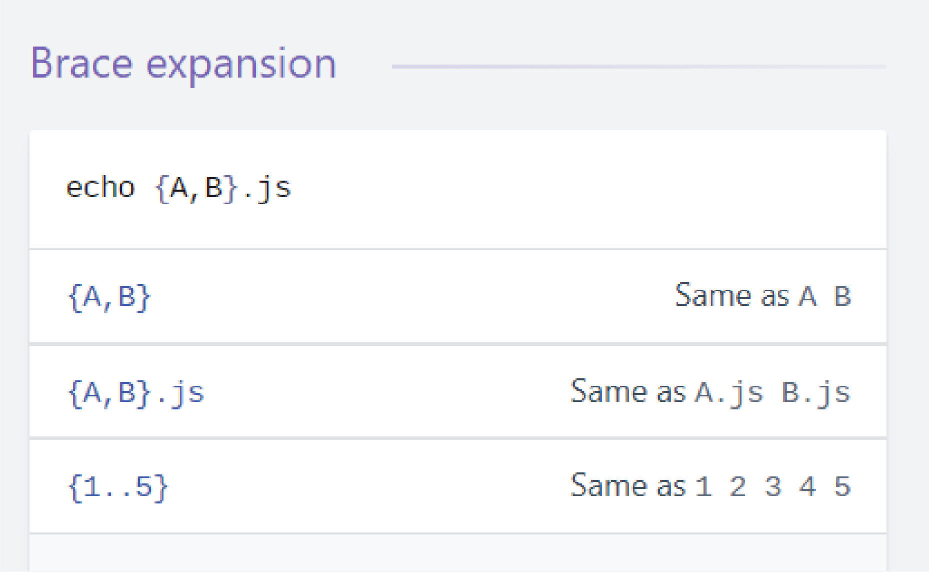
`{A,B}`	Same as `A B`
`{A,B}.js`	Same as `A.js B.js`
`{1..5}`	Same as `1 2 3 4 5`



~/Documents/Bash/bash_05.sh - Mousepad
File Edit Search View Document Help

```bash
#!/bin/bash

OGRENCI01="Ahmet"
OGRENCI02="Taner"
OGRENCI03="Taner"
OGRENCI04="Ezgi"

touch /var/tmp/{OGRENCI01,OGRENCI02,OGRENCI03,OGRENCI04}.txt
```

Brace expansion (range)

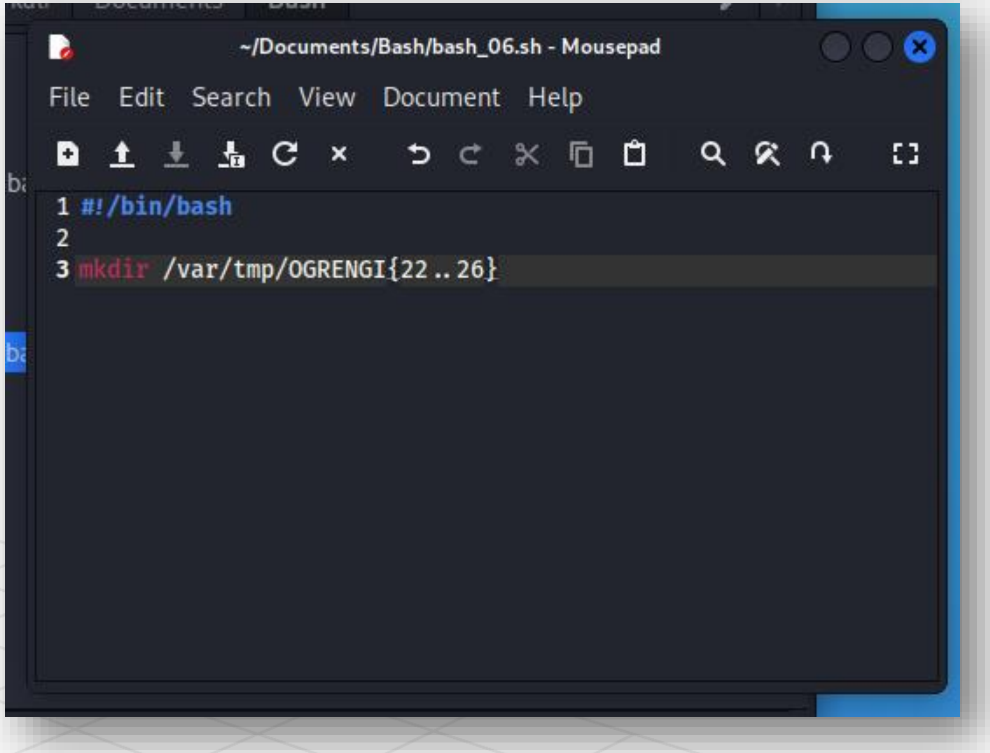


Brace expansion

```
echo {A,B}.js
```

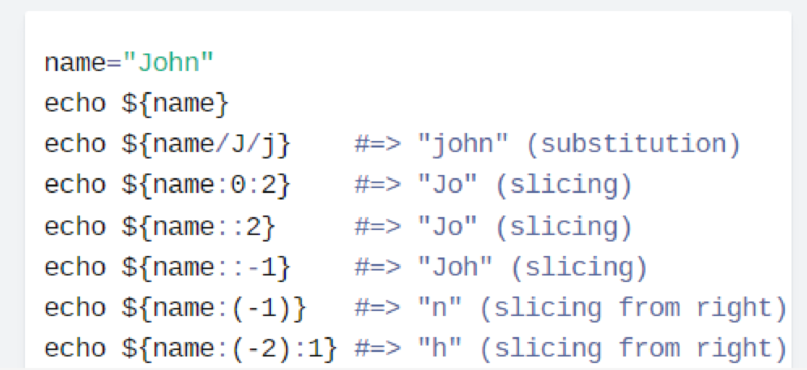
`{A,B}`	Same as `A B`
`{A,B}.js`	Same as `A.js B.js`
`{1..5}`	Same as `1 2 3 4 5`

{<START>..<END>}

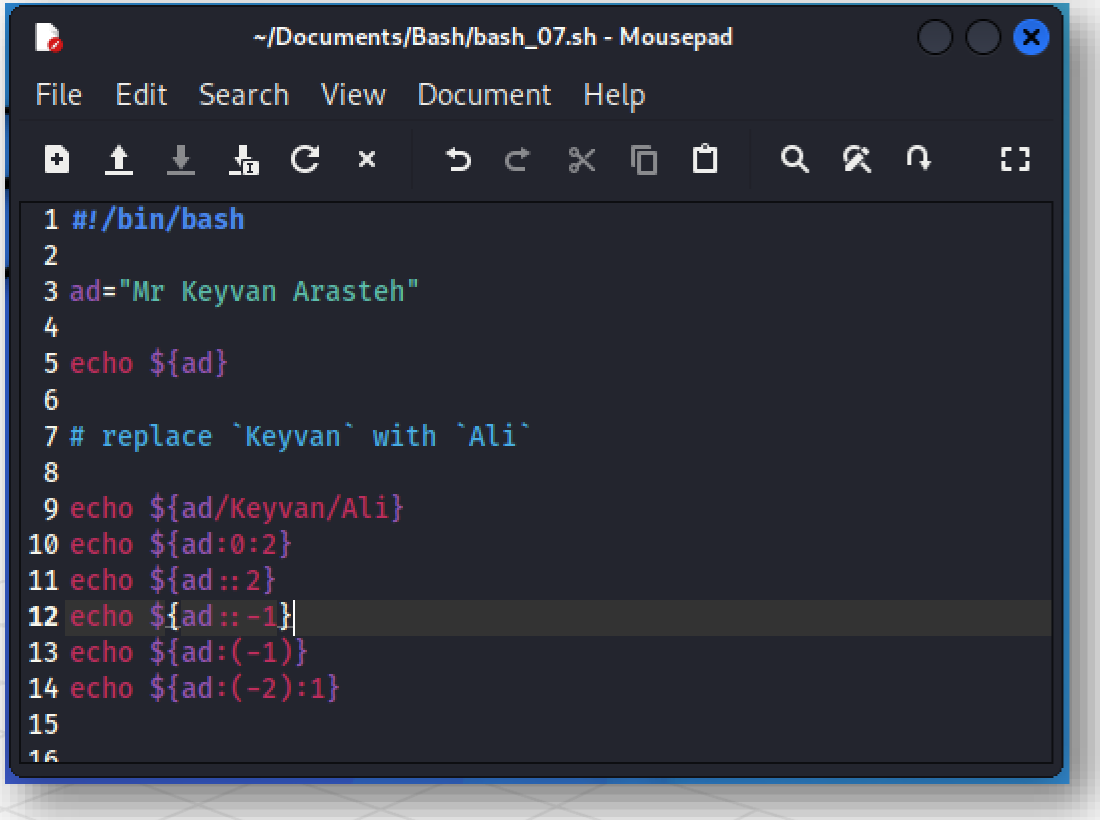


```bash
#!/bin/bash

mkdir /var/tmp/OGRENGI{22..26}
```

Parameter expansions (Basics)

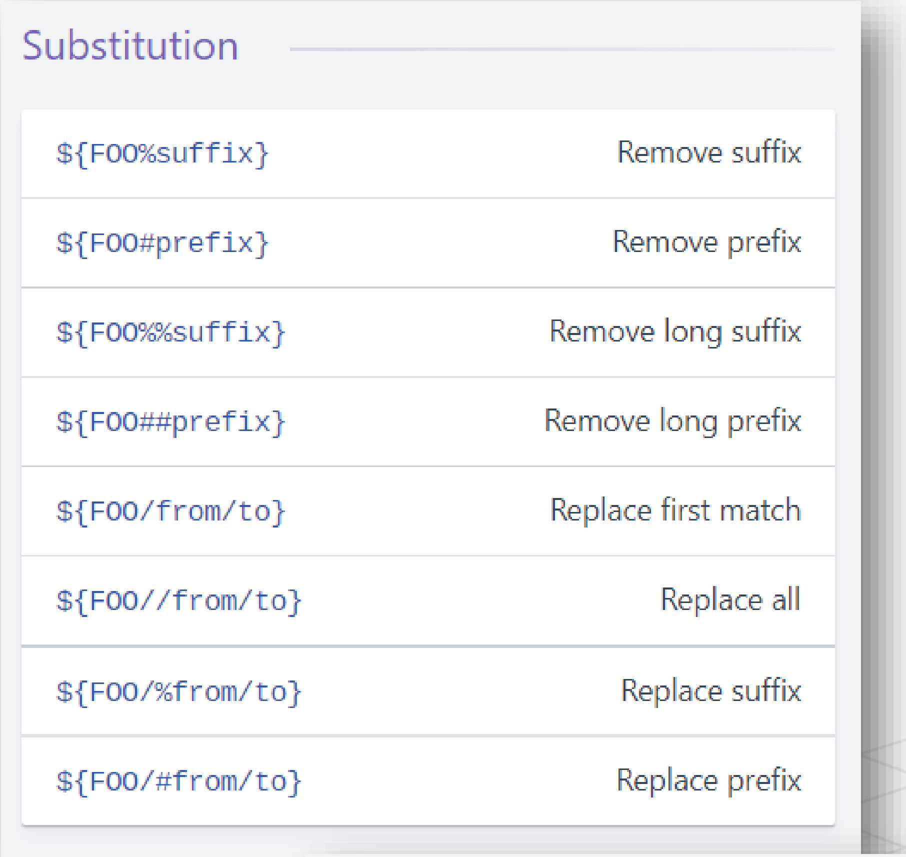


```
name="John"
echo ${name}
echo ${name/J/j}    #=> "john" (substitution)
echo ${name:0:2}    #=> "Jo" (slicing)
echo ${name::2}     #=> "Jo" (slicing)
echo ${name::-1}    #=> "Joh" (slicing)
echo ${name:(-1)}   #=> "n" (slicing from right)
echo ${name:(-2):1} #=> "h" (slicing from right)
```



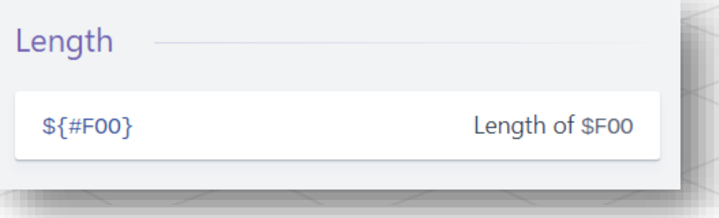
```
#!/bin/bash

ad="Mr Keyvan Arasteh"

echo ${ad}

# replace `Keyvan` with `Ali`

echo ${ad/Keyvan/Ali}
echo ${ad:0:2}
echo ${ad::2}
echo ${ad::-1}
echo ${ad:(-1)}
echo ${ad:(-2):1}
```

Parameter expansions (Substitution)



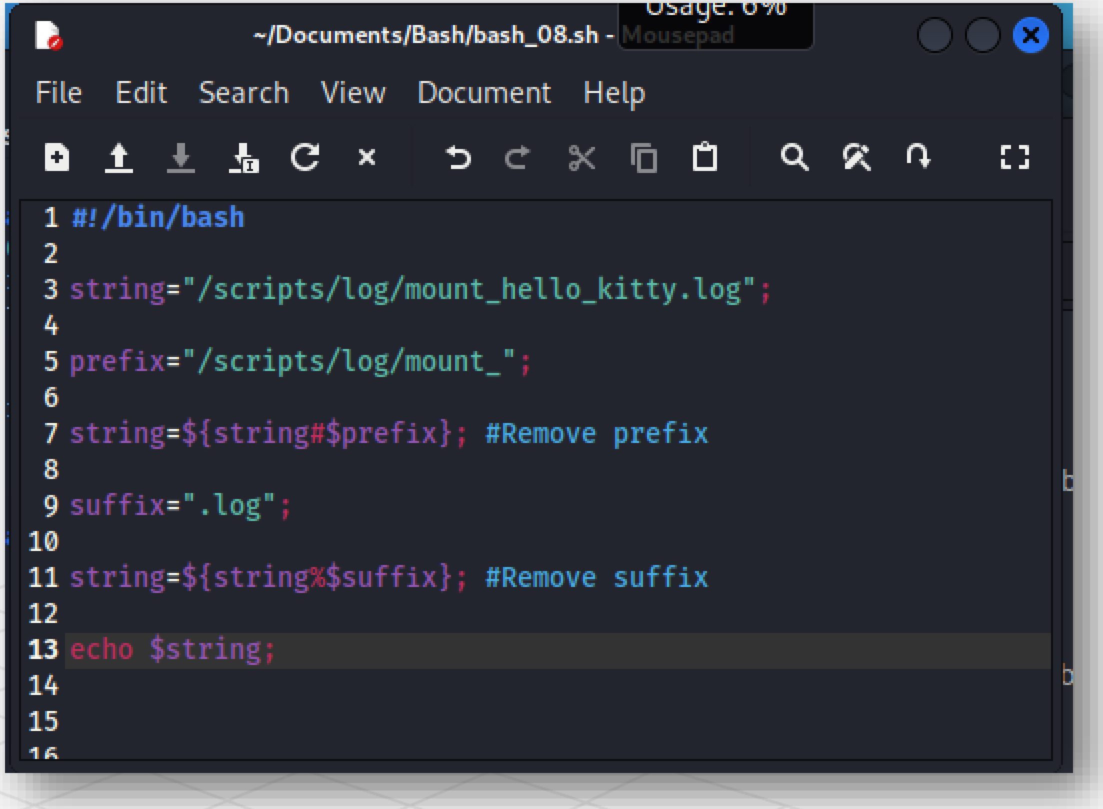
Substitution

`${FOO%suffix}`	Remove suffix
`${FOO#prefix}`	Remove prefix
`${FOO%%suffix}`	Remove long suffix
`${FOO##prefix}`	Remove long prefix
`${FOO/from/to}`	Replace first match
`${FOO//from/to}`	Replace all
`${FOO/%from/to}`	Replace suffix
`${FOO/#from/to}`	Replace prefix



Length

`${#FOO}`	Length of $FOO

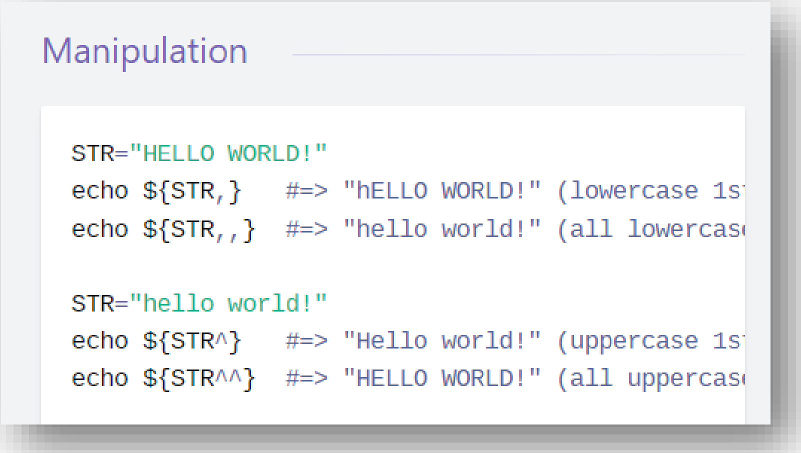


```bash
#!/bin/bash

string="/scripts/log/mount_hello_kitty.log";

prefix="/scripts/log/mount_";

string=${string#$prefix}; #Remove prefix

suffix=".log";

string=${string%$suffix}; #Remove suffix

echo $string;



```

Manipulation

String manipulation

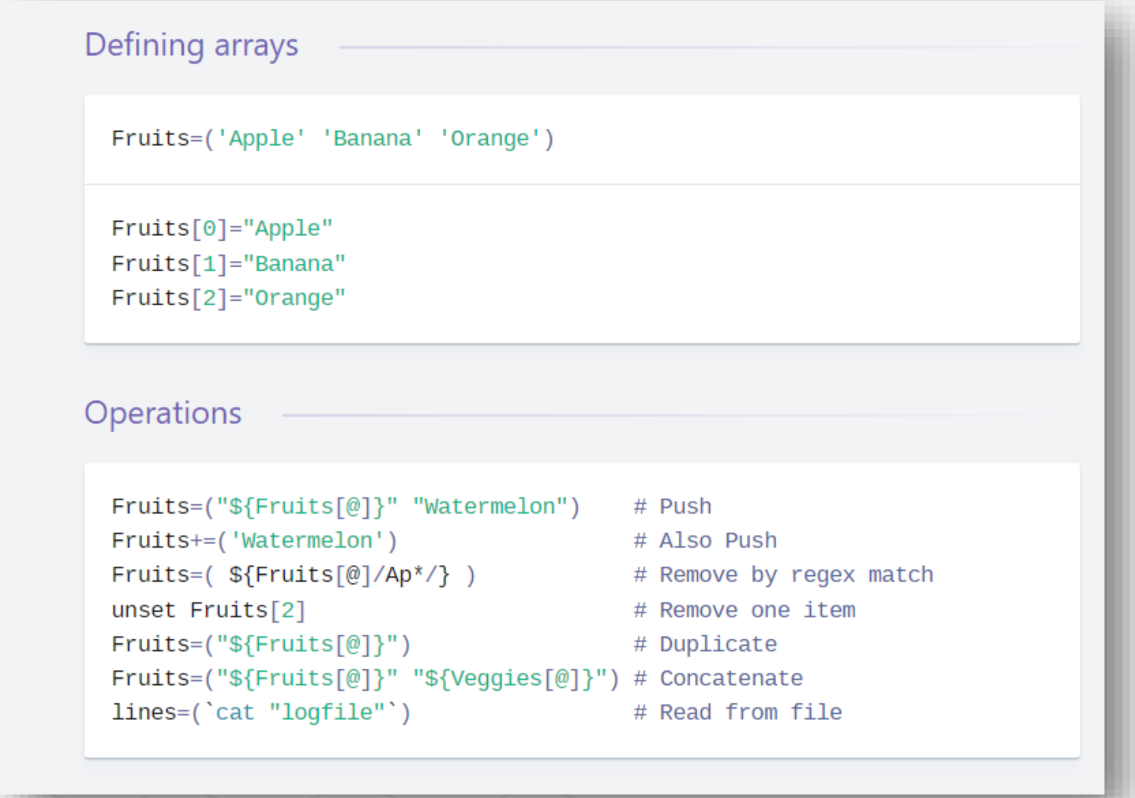
- Upper-case
- First character upper-case
- Lower-case
- First character lower-case



Manipulation

```
STR="HELLO WORLD!"
echo ${STR,}   #=> "hELLO WORLD!" (lowercase 1st
echo ${STR,,}  #=> "hello world!" (all lowercase

STR="hello world!"
echo ${STR^}   #=> "Hello world!" (uppercase 1st
echo ${STR^^}  #=> "HELLO WORLD!" (all uppercase
```

Arrays



Defining arrays

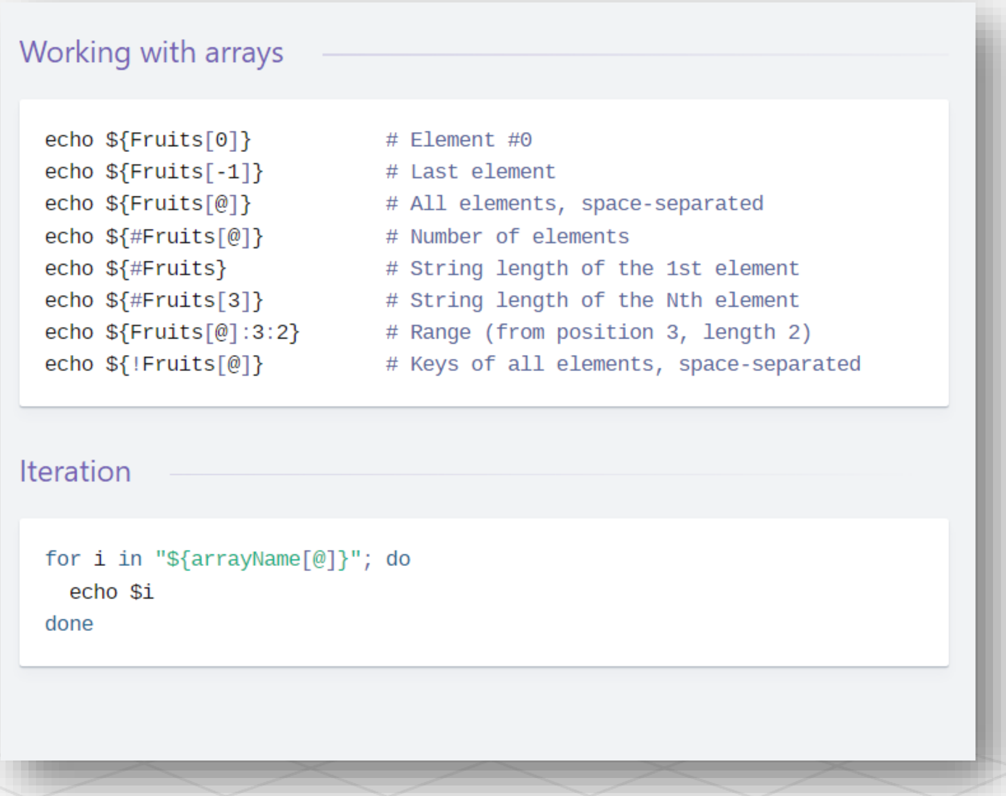
```
Fruits=('Apple' 'Banana' 'Orange')
```

```
Fruits[0]="Apple"
Fruits[1]="Banana"
Fruits[2]="Orange"
```

Operations

```
Fruits=("${Fruits[@]}" "Watermelon")    # Push
Fruits+=('Watermelon')                  # Also Push
Fruits=( ${Fruits[@]/Ap*/} )            # Remove by regex match
unset Fruits[2]                         # Remove one item
Fruits=("${Fruits[@]}")                 # Duplicate
Fruits=("${Fruits[@]}" "${Veggies[@]}") # Concatenate
lines=(`cat "logfile"`)                 # Read from file
```

Arrays II



Working with arrays

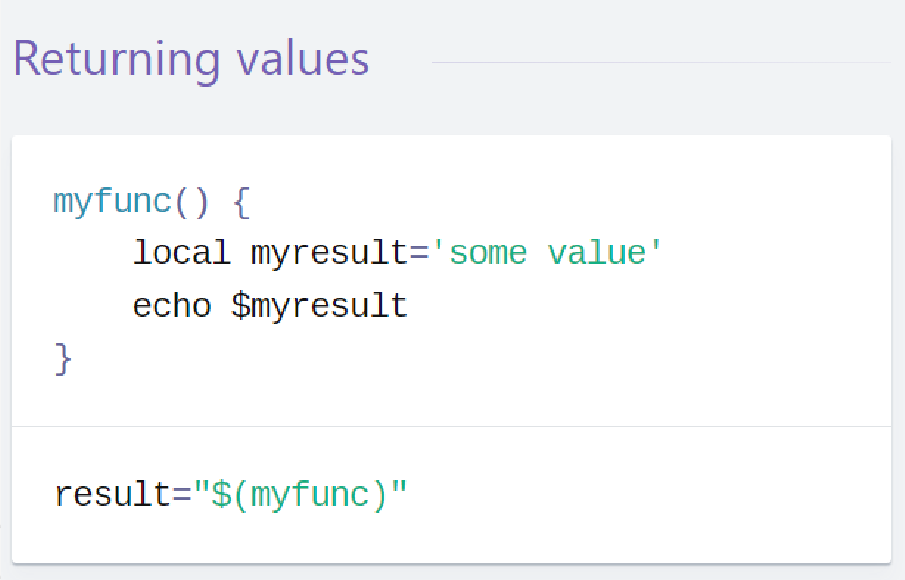
```
echo ${Fruits[0]}           # Element #0
echo ${Fruits[-1]}          # Last element
echo ${Fruits[@]}           # All elements, space-separated
echo ${#Fruits[@]}          # Number of elements
echo ${#Fruits}             # String length of the 1st element
echo ${#Fruits[3]}          # String length of the Nth element
echo ${Fruits[@]:3:2}       # Range (from position 3, length 2)
echo ${!Fruits[@]}          # Keys of all elements, space-separated
```

Iteration

```
for i in "${arrayName[@]}"; do
  echo $i
done
```

Functions

```
myfunc() {
     echo "hello $1“
 }

 # Same as above (alternate syntax)

 function myfunc() {
     echo "hello $1"
}
```



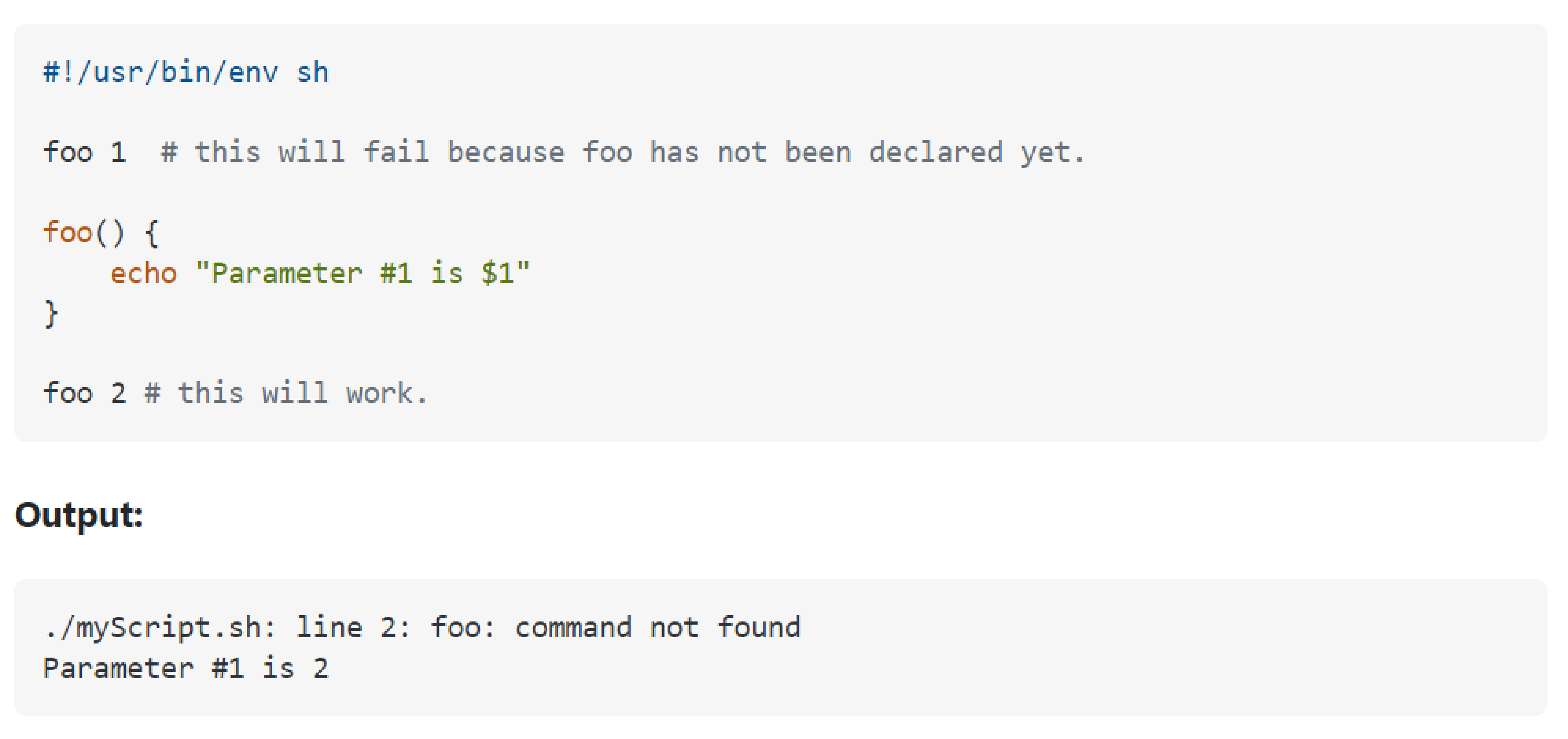
Returning values

```
myfunc() {
    local myresult='some value'
    echo $myresult
}
```

```
result="$(myfunc)"
```

First Define then use

Also, you need to call your function **after** it is declared.



```
#!/usr/bin/env sh

foo 1  # this will fail because foo has not been declared yet.

foo() {
    echo "Parameter #1 is $1"
}

foo 2 # this will work.
```

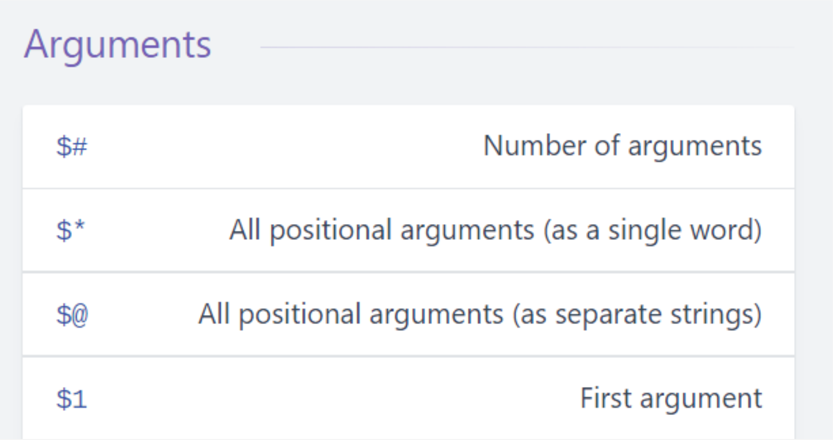
Output:

```
./myScript.sh: line 2: foo: command not found
Parameter #1 is 2
```

Passing Parameters to a bash function

To call a function with arguments:

```
function_name "$arg1" "$arg2"
```



Arguments

$#	Number of arguments
$*	All positional arguments (as a single word)
$@	All positional arguments (as separate strings)
$1	First argument

Functions (examples)

Write a code with 3 functions:

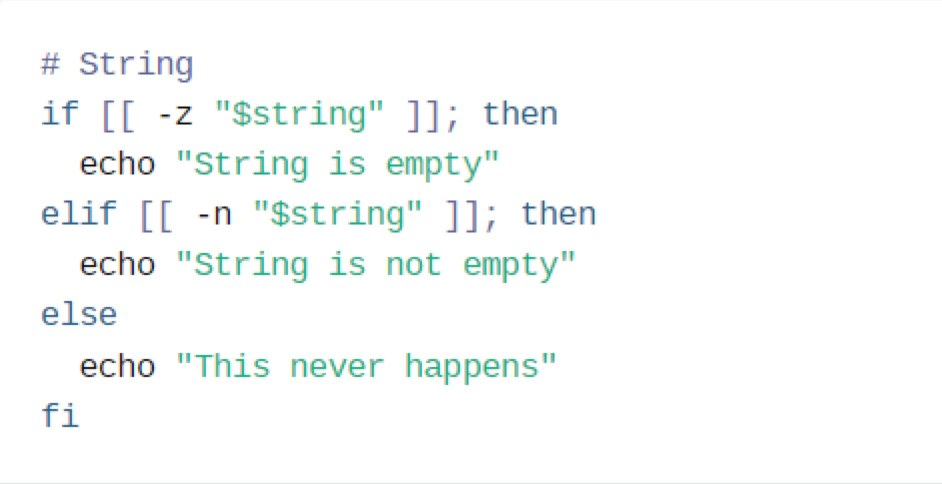
- get inputs
- calculate algorithms
- write outputs

Inputs: First Name, Last Name, Age

Calculation: “Hello FIRST LAST, your are AGE years old”

Output: ….

Conditions (if)

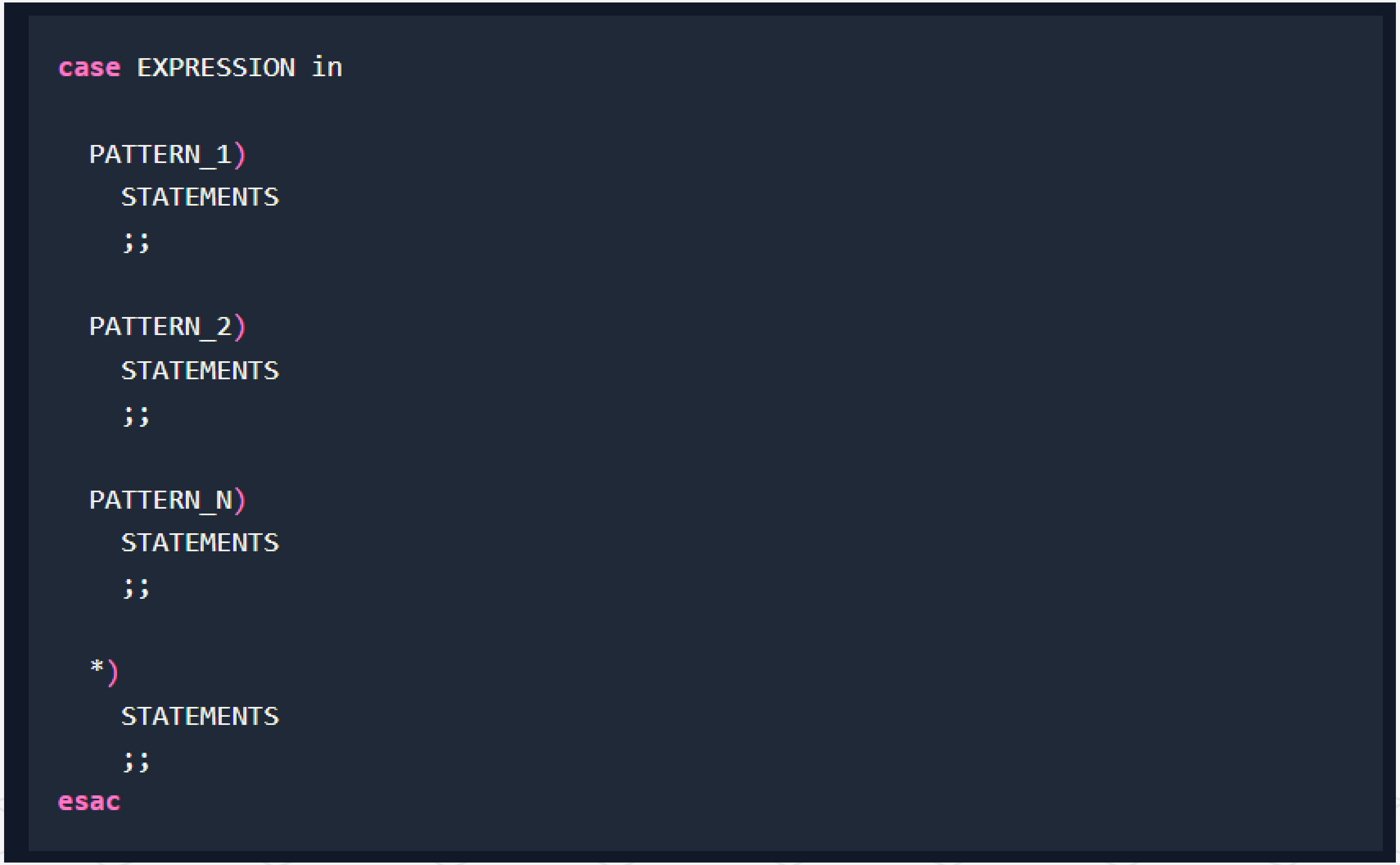


```
# String
if [[ -z "$string" ]]; then
  echo "String is empty"
elif [[ -n "$string" ]]; then
  echo "String is not empty"
else
  echo "This never happens"
fi
```

[[! EXPR]]	Not
[[X && Y]]	And
[[X || Y]]	Or

[[-z STRING]]	Empty string
[[-n STRING]]	Not empty string
[[STRING == STRING]]	Equal
[[STRING != STRING]]	Not Equal
[[NUM -eq NUM]]	Equal
[[NUM -ne NUM]]	Not equal
[[NUM -lt NUM]]	Less than
[[NUM -le NUM]]	Less than or equal
[[NUM -gt NUM]]	Greater than
[[NUM -ge NUM]]	Greater than or equal
[[STRING =~ STRING]]	Regexp

Conditions (switch)



```
case EXPRESSION in

  PATTERN_1)
    STATEMENTS
    ;;

  PATTERN_2)
    STATEMENTS
    ;;

  PATTERN_N)
    STATEMENTS
    ;;

  *)
    STATEMENTS
    ;;
esac
```

Conditions (samples)

Modify previous code with checking parameters with empty value.

Print error if parameters is empty.

```
Email Regex:
       ^[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,4}$
```

Loops

Basic for loop

```
for i in /etc/rc.*; do
  echo $i
done
```

C-like for loop

```
for ((i = 0 ; i < 100 ; i++)); do
  echo $i
done
```

Ranges

```
for i in {1..5}; do
    echo "Welcome $i"
done
```

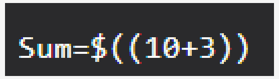
With step size

```
for i in {5..50..5}; do
    echo "Welcome $i"
done
```

Forever

```
while true; do
  ...
done
```

Math Operations



```
Sum=$((10+3))
```

Operator	Name	Use
+	Addition	It adds two operands
–	Subtraction	It subtract second operand from first one
*	Multiplication	Multiply two operands
/	Division	Return the quotient after diving first operand from second operands
%	Modulo	Return remainder after dividing first operand from second operand
+=	Increment by constant	Increment value of first operand with given constant value
-=	Decrement by constant	Decrement value of first operand with given constant value
*=	Multiply by constant	Multiply the given operand with the constant value
/=	Divide by constant	Divide the operand with given constant value and return the quotient
%=	Remainder by dividing with constant	Divide the operand with given constant value and return the remainder
**	Exponentiation	The result is second operand raised to the power of first operand.

Loops (Samples)

Write a code to run in 3 different modes:

- h: Get names and say hello
- a: Get birth year and calculate age
- b: write current date to output:
    * use date command
- q: quit

Date command switchs:

Date +’%Y’

date +’%M’

date +’%D’

Loops & Arrays (Samples)

Write a code which gets inputs from user and pushes to array

code switches:

- remove from index
- check for email
- check for empty

Date command switchs:

Date +’%Y’

date +’%M’

date +’%D’

Read files

We can use the following syntax to take a print of the contents of the file to a terminal.

```
value=`cat file_name`
```

```
Value=$(<file_name)
```

Examples:

```
#!/bin/bash

value=`cat read_file.txt`
echo "$value"
```

```
#!/bin/bash

value=$(read_file.txt)
echo "$value"
```

Read line-by-line

```
#!/bin/bash

file='read_file.txt'

i=1

while read line; do

        #Reading each line
        echo "Line No. $i : $line"
        i=$((i+1))
done < $file
```

Write files

To write the output of Bash commands to a file, we may use

- right angle bracket sign (>)

  or

- double right-angle sign (>>):

```
output=output_file.txt

ls > $output
ls >> $output
```

```
#Appending the system information
uname -a >> $output
```