Function

In Bash scripting, **functions** are blocks of code designed to perform a specific task. They help to structure and reuse code, making scripts more modular and easier to maintain.

Defining a Function:

A function is defined with the following syntax:

```
function_name () {
  # Commands to execute
}
```

Alternatively, you can use the function keyword:

```
function function_name {
  # Commands to execute
}
```

Both forms are valid, but the first style is more commonly used.

Example of a Function:

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

# Calling the function
greet "Shashi"
```

Output:

```
Hello, Shashi!
```

Key Points:

- **Function Name**: greet is the name of the function.
- **Parameters**: $1 refers to the first argument passed to the function (Alice in this case). Bash functions can accept arguments.
- **Calling the Function**: To execute the function, you simply call it by its name (greet "Alice").

```
sum() {
  result=$(( $1 + $2 ))
  echo $result
}

# Calling the function
result=$(sum 3 5)
echo "The sum is: $result"
```

Output:

```
The sum is: 8
```

Function with Arguments:

You can pass multiple arguments to a function, which can be accessed within the function as $1, $2, $3, etc.

```
add_numbers() {
  echo "Sum: $(( $1 + $2 ))"
}

add_numbers 10 20
```

Output

```
Sum: 30
```

Returning an Exit Status:

If you want to return a success or failure status, use the return keyword:

```
check_even_odd() {
  if (( $1 % 2 == 0 )); then
    return 0  # Success, number is even
  else
    return 1  # Failure, number is odd
  fi
}

# Calling the function
check_even_odd 4
echo "Exit status: $?"
```

The special variable $? stores the exit status of the last executed command.

Output

```
Exit status: 0
```

Function Scope:

Local Variables: Variables declared inside a function are local by default. They won't interfere with the global scope. To declare a variable as local, use the local keyword:

```
my_function() {

   local var="This is local"

   echo $var

}

my_function

echo $var  # This will not print anything since `var` is local to the function
```

Global Variables: Variables outside the function are global and accessible inside the function unless shadowed by a local variable.

Why Use Functions in Bash?

- **Modularity**: Break complex scripts into smaller, reusable functions.
- **Maintainability**: Easier to update or change specific logic without modifying the whole script.
- **Reusability**: Functions can be used multiple times, reducing redundancy in your code.

Example:

```
#!/bin/bash


# Function to greet the user

greet_user() {
   echo "Hello, $1! Welcome to Bash scripting."
}

# Function to check if a number is odd or even

check_odd_or_even() {
   if (( $1 % 2 == 0 )); then
      echo "$1 is an even number."
   else
      echo "$1 is an odd number."
   fi
}



# Main Program

# Ask for the user's name and greet them
```

```
echo "Enter your name:"
read user_name
greet_user "$user_name"

# Ask for a number and check if it is odd or even

echo "Enter a number to check if it's odd or even:"
read number
check_odd_or_even "$number"
```

Explanation:

1. greet_user: This function takes the user's name as an argument and prints a personalized greeting.
2. check_odd_or_even: This function checks whether the number passed as an argument is even or odd by using the modulus operator (%).
    1. If number % 2 == 0, the number is even.
    2. Otherwise, it is odd.