Functions

Functions in Bash allow you to group commands into reusable blocks of code, making scripts more modular and easier to maintain.

Defining a Function

1. Using the `function` keyword

```
function my_function {
    echo "Hello from function!"
}
```

2. Without the `function` keyword (Preferred)

```
my_function() {
    echo "Hello from function!"
}
```

Calling a Function

Simply use the function name to invoke it:

```
my_function
```

Passing Arguments to Functions

Arguments passed to a function can be accessed using `$1`, `$2`, etc.

Example:

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

Returning Values from Functions

- Bash functions do **not** return values like in traditional programming languages.
- The `return` statement is used for exit status (0 for success, non-zero for failure).
- To return actual data, use `echo`.

Example: Returning an exit status

```
check_even() {
    if (( $1 % 2 == 0 )); then
```

```
        return 0  # Success
    else
        return 1  # Failure
    fi
}

check_even 4 && echo "Even number" || echo "Odd number"
```

Example: Returning a value using `echo`

```
get_date() {
    echo "$(date)"
}

today=$(get_date)
echo "Today's date: $today"
```

Local vs Global Variables

By default, variables in functions are global. Use `local` to restrict scope.

Example:

```
my_function() {
    local var="I am local"
    echo "$var"
}

my_function
echo "$var"  # This will be empty because 'var' is local
```

Using Functions in Scripts

Functions can be used in scripts to organize code better.

Exa: Simple Calculator

```
add() {
    echo $(($1 + $2))
}
subtract() {
    echo $(($1 - $2))
}

num1=10
num2=5
```

```
sum=$(add $num1 $num2)
diff=$(subtract $num1 $num2)

echo "Sum: $sum"
echo "Difference: $diff"
```