Controlling the flow of PowerShell Functions

Liam Cleary

CEO / MICROSOFT MVP / MICROSOFT CERTIFIED TRAINER

@shareplicity www.shareplicity.com | @helloitsliam www.helloitsliam.com

Overview

Review how to use IF and ELSE Statements

How to use DO-WHILE and DO-UNTIL Looping

When to use SWITCH Statements

Using FOREACH Enumerators

What are IF/ELSE Statements?

Conditional Execution

PowerShell Scripts make decisions and perform different logic based on those decisions. You may have a statement or value to evaluate, then execute a separate code section based on the evaluation. It is called conditional execution.

IF Statement Groups

There are three possible statements to use within an If statement group

IF Statement

This is the first test in the evaluation process. It contains the first statement to check

ELSE Statement

Does not accept any condition. Executes the same action every time the evaluation fails and returns false

ELSEIF Statement

Provides additional conditions, creating multiple outcomes

IF / ELSEIF / ELSE Statement Components

The "IF" Statement

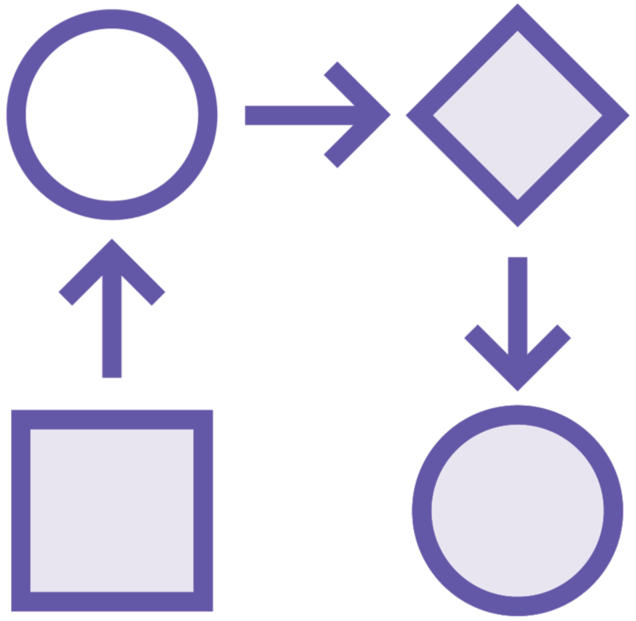
Parentheses Containing Conditions to Evaluate

Any "ELSEIF" Statements

The "ELSE" Statement

Output / Value

IF / ELSEIF / ELSE Flow



The If statement executes

- If the result of the first test equals true, any specified actions or code execute
- If the first test result returns false, all other conditions evaluate until they return true, or they all equal false
- If the result of the subsequent test equals true, any specified actions or code execute
- If all tests equal false, the final else statement executes

IF Syntax

```
$variable = "value"

if($variable)
{
    Write-Output "The $variable check returned true"
}
```

IF / ELSE Syntax

```
$variable = "value"

if($variable)
{
    Write-Output "The $variable check returned true"
}


else
{
    Write-Output "The $variable check returned false"
}
```

IF / ELSEIF / ELSE Syntax

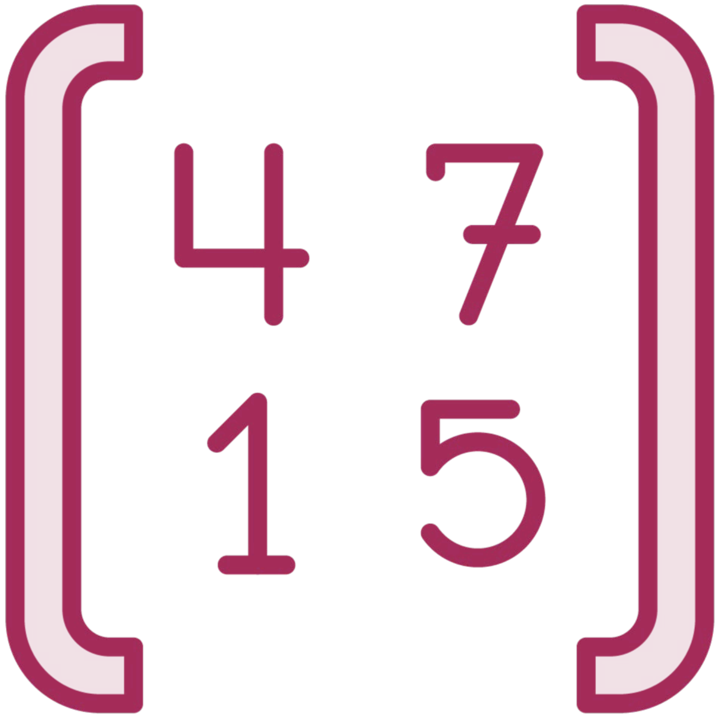
```
$variableone = "value"
$variabletwo = "value"

if($variableone)
{
    Write-Output "The $variableone check returned true"
}


elseif($variabletwo)
{
    Write-Output "The $variabletwo check returned true"
}


else
{
    Write-Output "The $variableone and $variabletwo check returned false"
}
```

The Parentheses



Contains the evaluation condition(s)

Can contain comparison operators

- **Examples:** EQ, NE, GT, LT, LE, LIKE, NOTLIKE, and MATCH

Can contain array of values for comparison

Allows combinations of AND and OR

Support for multiple conditions

Common Conditions Operators

Comparison

Collection

Logical

Arithmetic

Conditions Operators

Assignment

Redirection

Split and Join

Type

Unary

Bitwise

Comparison Operator

The most common use of the "if" statement is comparing two items with each other. PowerShell has multiple operators for comparison scenarios. When you use a comparison operator, you compare the left-hand side's value to the right-hand side's value.

Comparison Operator Syntax

```
# Variables
$variable = "value"
$comparevariable = "value"

# Equal Comparison Operator using "-eq"
if($variable -eq $comparevariable)

# Greater Than Comparison Operator using "-gt"
if($variable -gt $comparevariable)

# Like Comparison Operator using "-like"
if($variable -like $comparevariable)

# Match Comparison Operator using "-match"
if($variable -match $comparevariable)
```

Collection Operator

A comparison operator returns either a true or false value. Collection operators differ slightly, as each item in the collection is evaluated, and the operator returns every value that is true.

Collection Operator Syntax

```
# Variables
$array = 1..10
$comparevariable = 6

# If current array value is greater than compare value
if($array -gt $comparevariable)

# If array contains the compare value
if($array -contains $comparevariable)

# If compare value is within the array
if($comparevariable -in $array)
```

Logical Operator

Logical operators are used to invert or combine other expressions. You can connect multiple statements, allowing you to use a single expression to test for numerous conditions.

Logical Operator Syntax

```
# Variables
$variable = "value"
$comparevariableOne = "value"
$comparevariableTwo = "value"

# Multiple Equal Comparison Operator using "-eq" and "-and"
if(($variable -eq $comparevariableOne) -and ($variable -eq $comparevariableTwo))

# Multiple Equal Comparison Operator using "-eq" and "-or"
if(($variable -eq $comparevariableOne) -or ($variable -eq $comparevariableTwo))
```

Arithmetic Operator

Arithmetic operators calculate numeric values. Using one or more arithmetic operators, you can add, subtract, multiply, and divide values to calculate the remainder of a division operation.

Arithmetic Operator Syntax

```
# Variables
$variable = "value"
$comparevariableOne = "value"
$comparevariableTwo = "value"

# Add variables and check if result is greater than variable
if(($comparevariableOne + $comparevariableTwo) -gt $variable)

# Divide variables and check if result is less than variable
if(($comparevariableOne / $comparevariableTwo) -lt $variable)

# Multiply variables and check if result is equal to the variable
if(($comparevariableOne * $comparevariableTwo) -eq $variable)
```

Ternary Operator

PowerShell 7.x introduced new syntax for IF / ELSE using the ternary operator

Condition

The condition expression is evaluated, and the result is converted to a Boolean

IF True

Expression is executed if the condition expression is true

IF False

Expression is executed if the condition expression is false

Ternary Operator Syntax

```
# Variables
$variableOne = "1"
$variableTwo = "3"
$textVariable = $true

# Compare both variable values using "-eq" then return true / false or message
($variableOne -eq $variableTwo) ? $true : $false
($variableOne -eq $variableTwo) ?
        ("$variableOne > $variableTwo ") : ("$variableOne < $variableTwo ")

# Compare text variable values using "-eq" then return true or false
($textVariable) ? "Value is true" : "Value is false"
```

Demo

Review using IF/ELSE Statements

Implement basic logic using IF/ELSE Statements

Using the DO-WHILE and DO-UNTIL Looping

What Is a Loop?

Programming and scripting language construct

Sequence of instructions inside code

Iterate several times as long as the defined condition is met

Loop Types

For

While

Do-While

Do-Until

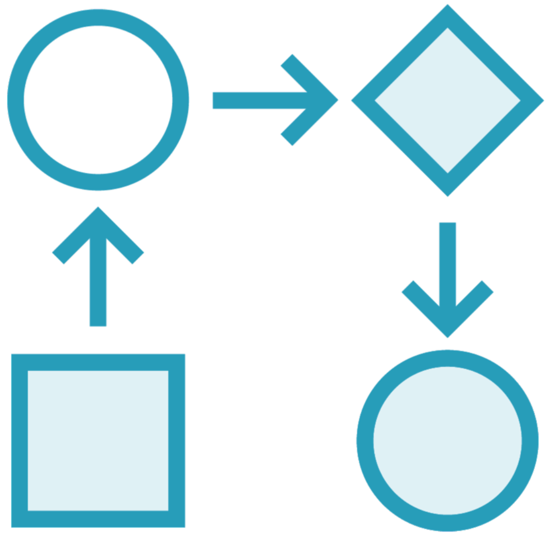
Loop Types

Foreach-Object

Foreach

Foreach()

Entry versus Exit Loops



Entry Loop

- Test condition is checked before entering the loop

Exit Loop

- Test condition is checked after executing the loop

Entry versus Exit Loops

```
# Variable
$number = 1

# Entry Loop – "While Loop"
While ($number -le 10) { $number; $number++ }

# Exit Loop – "Do-While Loop"
Do { $number; $number++ } While($number -le 10)
```

The Do-While Loop

The DO-WHILE loop is almost the same as a regular while loop

It executes the block of statements while the condition returns true

DO-WHILE executes first; then, the condition test is at the end of the loop

Do-While Syntax

```
# Base Syntax
Do { Command sequence } While (<condition>)

# Variable
$number = 1

# Example Syntax
Do { $number; $number++ } While ($number -le 10)
```

While and Do-While Loop Differences

While Loop	Do-While Loop
Entry Controlled Loop	Exit Controlled Loop
Tests the condition before the execution of the first iteration	Tests the condition after the execution of the first iteration
Loop doesn't execute when the condition evaluates too false.	Loop executes at least once even if the condition evaluates too false
It does not require any other syntax	Use the DO keyword at starting of the loop, and the while keyword with the condition at the end of the loop

The Do-Until Loop

DO-UNTIL loops use similar syntax to DO-WHILE loops

The loop repeats until the condition returns true

They are the opposite of DO-WHILE loops

Do-Until Syntax

```
# Base Syntax
Do { Command sequence } Until (<condition is true>)

# Variable
$number = 1

# Example Syntax
Do { $number; $number++ } Until ($number -le 10)
```

Do-While and Do-Until Loop Differences

Do-While Loop	Do-Until Loop
Exit Controlled Loop	Exit Controlled Loop
Uses the **WHILE** keyword for the condition	Uses the **UNTIL** keyword for the condition
Executes while the condition returns true	Continues execution until the condition returns true

Demo

Review Do/While and Do/Until Loops

Implement Do/While and Do/Until Loops within PowerShell

Understanding SWITCH Statements

What are Switch Statements?

Alternate syntax for doing multiple comparisons with a value

Result of an expression gets compared with several different values

If a value matches, the matching code block executes

Switch Statement Rules

The default statement is optional

The test expression can be a logical or an integer expression

Break statements will terminate the current code execution

Switch Statement Components

The initial test expression

Conditions to match

Switch Statement Syntax

```
# Base Syntax
Switch (<expression>)  {
    <condition1> { <code> }
    <condition2> { <code> }
    <condition3> { <code> }
}

# Variable
$number = 3

# Example Syntax
Switch ($number) {
    5   { Write-Host "Number equals 5" }
    10  { Write-Host "Number equals 10" }
    20  { Write-Host "Number equals 20" }
    Default  { Write-Host "Number is not equal to 5, 10, or 20"}
}
```

Switch Statement Syntax

Test expression can combine operators

Provide support for multiple expression tests

Output can be from multiple comparison code paths

Multiple Expression Switch Statement Syntax

```
# Base Syntax
Switch (<expression1>, <expression2>)  {
     <condition1> { <code> }
     <condition2> { <code> }
     <condition3> { <code> }
}

# Variable
$number1 = 5
$number2 = 11

# Example Syntax
Switch ($number1, $number2) {
     5    { Write-Host "Number equals 5" }
     10   { Write-Host "Number equals 10" }
     20   { Write-Host "Number equals 20" }
     Default  { Write-Host "Number is not equal to 5, 10, or 20"}
}
```

Demo

Review Switch Statement Format

Implement Switch Statement within PowerShell

Using ForEach Enumerators

What is a ForEach Enumerator?

The Foreach statement is a scripting language construct that iterates through collections of items

Can execute operations against each item

Processes each item within the collection unless the execution path changes

What is in a ForEach Loop Statement?

The construct of the ForEach loop is straightforward. It says: ForEach item in a collection, perform the specified tasks

Item

The variable containing the current item

Collection

A collection of objects, such as files, numbers, or other object types

Script Block

The sequence of commands to be executed against each item in the collection

ForEach Syntax

```
# Base Syntax
ForEach (Item In Collection) {
    <code> or <sequence of commands>
}

# Variable
$collection = 1,2,3,4,5,6,7,8,9,10

# Example Syntax
ForEach ($item in $collection) {
    Write-Host "Current Number: $item"
}
```

Example ForEach Syntax

```
# Variables
$collection1 = 1..10
$collection2 = 'A','B','C','D','E','F','G','H'
$path = "C:\Documents"

# Example Number Syntax
ForEach ($item in $collection1) {
    Write-Host "Current Number: $item"
}

# Example Letter Syntax
ForEach ($item in $collection2) {
    Write-Host "Current Letter: $item"
}

# Example Files Syntax
ForEach ($file in Get-ChildItem $path) {
    Write-Host "Current Filename: $file"
}
```

Demo

Review ForEach Structure

Implement ForEach Enumerator within PowerShell

Managing the Flow within PowerShell

IF statements can be nested to create complex execution paths

Loop statements can be nested inside other loops, as well as within IF statements

ForEach is not only a function for enumeration, but is a property on specific objects as a method

All combinations of IF, ELSEIF, ELSE, SWITCH, DO-WHILE, DO-UNTIL, WHILE, FOREACH and SWITCH statements are supported within each other

Summary

Reviewed how and when to use IF and ELSE Statements

Executed various DO-WHILE and DO-UNTIL Loops

Explained and demonstrated when to use SWITCH Statements

Implemented FOREACH Enumerators

Up Next:
Manipulating Data within PowerShell Functions