# Writing Reusable PowerShell Scripts

**Liam Cleary**

CEO / MICROSOFT MVP / MICROSOFT CERTIFIED TRAINER

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

## Overview

Steps to Creating PowerShell Scripts

Understanding Script Signing

Execute Scripts within the PowerShell Console

Creating Functions

Tip and Tricks for Creating Scripts

## Steps to Creating PowerShell Scripts

## Determine the Editor

Windows PowerShell Console

Windows Terminal

Visual Studio Code

## Script Structure

Define any required modules or snap-ins

Declare any variables

Define functions

## Creating a PowerShell Script

Launch selected Editor

Add PowerShell code

Digitally sign, if distributing or for security

Save the file as {Name}.ps1

## Comment PowerShell Scripts

**Single-Line PowerShell Comments**
Begins with the number/hash character (#). Everything on the same line after it is ignored by PowerShell

**Block Comments / Multiline Comments**
Comment blocks in PowerShell begin with "<#" and end with "#>"

**Comment-Based Help**
Collection of keywords and string values wrapped within a block comment

## Commenting Code

```
# Single Line Commenting
# This function returns a simple string
function Invoke-Message() { Write-Host "Some Text" }

# Block Comments / Multiline Comments
<#
    This function returns a simple string
    The string will be displayed in red
#>
function Invoke-Message() { Write-Host "Some Text" -ForegroundColor Red }

# Commenting-out Existing Code
#function Invoke-Message() { Write-Host "Some Text" -ForegroundColor Red }
```

## Understanding Script Signing

# Why Digitally Sign a PowerShell Script?

Separate custom developed, tested and personal scripts

Ensure scripts are not malicious

Validate that the script is doing versus its intended purpose

## Understanding Script Signing

Must sign a script with a code signing certificate

Two types of certificates are suitable for signing a script file: Public Certification Authority and Self-signed

Use a self-signed certificate only to sign scripts that you write for personal use

# Certificate Types

**Public Certification Authority**

Share the script with other computers as they trust the certification authority

**Locally Created Self-Signed**

Self-signed certificate scripts will not execute on other computers, only locally, or computers that trust the self-signed certificate

## Creating Self-signed Digital Certificate

```
# Set the PowerShell Script Path
$script = "C:\Documents\Code\Script.ps1"

# Create Self-signed Code Signing Certificate
New-SelfSignedCertificate `
    -DNSName "script.company.com" `
    -CertStoreLocation Cert:\CurrentUser\My `
    -Type CodeSigningCert `
    -Subject "PowerShell Code Signing Certificate"

# Retrieve the Code Signing Certificate
$certificate = (Get-ChildItem Cert:\CurrentUser\My -CodeSigningCert)[0]

# Set the Code Signing Certificate for the PowerShell Script
Set-AuthenticodeSignature $script -Certificate $certificate

# Validate the Code Signing Certificate
Get-AuthenticodeSignature $script | Format-Table -AutoSize
```

Demo

Digitally Sign a Custom PowerShell Script

## Execute Scripts within the PowerShell Console

## Common PowerShell Consoles

**Windows PowerShell Console**

**Windows Terminal**

**Visual Studio Code**

**PowerShell Integrated Development Environment**

# Executing Scripts

Ensure the Execution Policy is set as Required

Type "&" following by the "**Path to Script File (*.ps1)**"

Press Enter and wait or the script to complete

# Executing Scripts within Visual Studio Code

Integrated Console and Editor

Easy Browse and Load Files

Press "F5" to Execute Entire Script

Select Specific Lines and Execute

## Using the PowerShell IDE

Integrated Console and Editor

Press "F5" to Execute the Entire Script

Select Specific Lines and Execute

## Demo

Execute Script within the PowerShell Console

Execute Script within the PowerShell ISE

Execute Script within the Windows Terminal

Execute Script within the Visual Studio Code

## Creating Functions

## What are Functions?

Functions are the building block of PowerShell Scripts

Reusable throughout the script

Can contain variables, parameters, statements and calls to other functions

## Functions with Arguments and Parameters

**Arguments**

**Parameters**

# Arguments versus Parameters

| Arguments | Parameters |
|---|---|
| Arguments are not specified within a function | A Parameter is variable defined in a function |
| Arguments are populated by passing values as part of the execution | Parameter value is populated when calling the function |
| Values are retrieved by using ID | Parameters have properties |
| | Parameters can be mandatory or optional |

## Creating a Basic PowerShell Function

```
# Create a Function
Function Get-Answer()
{
    $question = Read-Host "What is the Capital City in Australia?"

    if($question -eq "Canberra")
    {
        Write-Host "Correct!! You entered $question" -ForegroundColor Green
    }
    else
    {
        Write-Host "Incorrect!! You entered $question" -ForegroundColor Green
    }
}
```

## PowerShell Function with Arguments

```
# Create a Function using Arguments
Function Get-Answer()
{
    $question = Read-Host "Hi $($args[0]), What is the Capital City in Australia?"

    if($question -eq "Canberra")
    {
        Write-Host "Correct!! You entered $question" -ForegroundColor Green
    }
    else
    {
        Write-Host "Incorrect!! You entered $question" -ForegroundColor Green
    }
}
```

# PowerShell Function with Variables

```
# Create a Function using Variables
Function Get-Answer($name)
{
    $question = Read-Host "Hi $name, What is the Capital City in Australia?"

    if($question -eq "Canberra")
    {
        Write-Host "Correct!! You entered $question" -ForegroundColor Green
    }
    else
    {
        Write-Host "Incorrect!! You entered $question" -ForegroundColor Green
    }
}
```

## PowerShell Function with Parameters

```
# Create a Function with Parameters
Function Test-WhatIsCapitalCityofAustralia()
{
    Param(
        [Parameter(Mandatory=$true)]
        [ValidateSet("Canberra","Melbourne","Brisbane","Perth")]
        [string]$city
    )

    if($city -eq "Canberra")
    {
        Write-Host "Correct!! You entered $city" -ForegroundColor Green
    }
    else
    {
        Write-Host "Incorrect!! You entered $city" -ForegroundColor Green
    }
}
```

## Common Function Enhancements

Comments

Error Handling

Console Messages

Loop Statements

Flow Logic

Output Results

## Basic Script – Iterate Files Function

```
# Iterate Files Function
Function Get-Files
{
    Param(
        [Parameter(Mandatory=$true)]
        [string]$FileType
    )

    Get-ChildItem -Path "$path\*.$FileType" -Recurse -Force
}
```

# Basic Script – Create Picker Values

```
# Create Class for Autocomplete Values
class Cities : System.Management.Automation.IValidateSetValuesGenerator
{
    [string[]] GetValidValues()
    {
        $Cities = @('1.4 Million','750 Thousand','2 Million','500 Thousand')
        return $Cities
    }
}
```

# Basic Script – Question Function

```
Function Test-PopulationOfHawaii
{

    param(
        [parameter(Mandatory = $true)]
        [ValidateSet([Cities])]
        [string] $Answer
    )

    if($Answer -eq "1.4 Million")
    {
        Write-Host "Correct!!" -ForegroundColor Green
    }
    else
    {
        Write-Host "Incorrect!!" -ForegroundColor Green
    }
}
```

## Basic Script – Math Function

```
# Define Math Operator Values
[ValidateSet("Add", "Subtract", "Multiply", "Divide")]

# Check the Operator and Perform Specific Sum
if($mathoperator -eq "Add") {
    $answer = Invoke-AddNumbers $numberOne $numberTwo
}
elseif($mathoperator -eq "Subtract") {
    $answer = Invoke-SubstractNumbers $numberOne $numberTwo
}
elseif($mathoperator -eq "Multiply") {
    $answer = Invoke-MultiplyNumbers $numberOne $numberTwo
}
elseif($mathoperator -eq "Divide") {
    $answer = Invoke-DivideNumbers $numberOne $numberTwo
}
```

## Demo

Create a Function within a PowerShell Script

Create a Function with Parameters within a PowerShell Script

Create a Basic Script

## Tip and Tricks for Creating Scripts

# Tip and Tricks for Creating Scripts

Comment the Code

Use Unique Variables

Use Start/Stop Transcript

Use Try/Catch

Use Conditional Logic

Use Loops for Iteration

## Summary

Review the Steps to Creating PowerShell Scripts

Digitally Signed a Sample PowerShell Script

Executed Scripts within Multiple Consoles

Created Functions with and without Parameters

Explained some Tip and Tricks for Creating Scripts