Art of PowerShell

# Introductions

Explanation of who we are, what we do

Why we use PowerShell

* Ease repetitive tasks
* Consistent results
* Auditability
* Accessibility of Knowledge
* Makes Delegation Easier
	+ Instead of sharing a KB article to follow the manual steps to perform a task
* Easier scripting than many alternatives
	+ Such as VBScript or .NET
* Glue disparate languages together
	+ Such as command-line DOS tools, MSSQL .NET commands and native PowerShell commands

# Overview of Session

1. Scripting demo
2. Functions demo
3. Helpful Hints
	1. Lessons Learned & Techniques
4. Questions

# Scripting Demo – Bill

Walkthrough of creating a script and what considerations and techniques are useful.

**Case Study:** Retrieving Data Statistics

**Reason:** To aid in data validation and to provide insight into data

**Pseudo-Code:**

1. Retrieve all the properties of the given object (could be a CSV or could just be any object)
	1. $data | Get-Member | Where MemberType –EQ “NoteProperty”
2. Group on the values of each NoteProperty
3. Then output those NoteProperties that have less than X amount of distinct values

$path = ""

$data = Import-CSV $path

$limit = 10

$properties = $data `

 | Select -First 1 `

 | Get-Member -MemberType NoteProperty `

 | Select -ExpandProperty Name

$counts = $properties | %{

 $attribute = $\_

 [Int]$attributeCount = ($data | select -ExpandProperty $attribute | Group-Object).Count

 $attributeName = (Get-Culture).TextInfo.ToTitleCase($attribute.ToLower())

 If ($attributecount -LT $limit) {

 Write-Host "$attributename - Value Breakdown: "

 $data | Select -ExpandProperty $attribute -Unique | %{

 $count = $( $data | Where $attribute -EQ $\_ ).count

 Write-Output "$attribute,$\_,$count"

 }

 }

}

# Functions Demo – Adam

How to build up a function from a script to an advanced function.

**Why would we do this?**

* To reuse the code amongst several scripts
* To use the most up to date code
* Common functionality
* Proper logging

<#

 # Script FileName: func\_Get-VPSADataStatistics.ps1

 # Current Version: A01

 # Description: Retrieve data statistics on a given input.

 # Created By: Adam Listek

 # Version Notes

 # A01 - Initial Release

 #>

#Requires -Version 3.0

Function Get-VPSADataSatistics {

 [CmdletBinding(

 SupportsShouldProcess=$true,

 ConfirmImpact="Low"

 )] # Terminate CmdletBinding

 Param (

 [Parameter(Position=0,

 ValueFromPipeline=$true,

 Mandatory=$true)]$Data,

 [Parameter(Position=1)]$Limit = 10

 ) # Terminate Param

 Begin {

 If ($MyInvocation.BoundParameters.Verbose -match $true) {

 $local:VerbosePreference = "Continue"

 $local:ErrorActionPreference = "Continue"

 $local:verbose = $true

 } Else {

 $local:VerbosePreference = "SilentlyContinue"

 $local:ErrorActionPreference = "SilentlyContinue"

 $local:verbose = $false

 } # Terminate If - Verbose Parameter Check

 If ($MyInvocation.BoundParameters.Debug -eq $true) {

 $local:debug = $true

 } Else {

 $local:debug = $false

 } # Terminate Preferences

 If ($MyInvocation.BoundParameters.WhatIf -eq $true) {

 $local:whatif = $true

 } Else {

 $local:whatif = $false

 } # Terminate Preferences

 } # Terminate Begin

 Process {

 $properties = $data `

 | Select -First 1 `

 | Get-Member -MemberType NoteProperty `

 | Select -ExpandProperty Name

 $properties | %{

 $attribute = $\_

 [Int]$attributeCount = ($data | Select -ExpandProperty $attribute | Group-Object).Count

 $attributeName = (Get-Culture).TextInfo.ToTitleCase($attribute.ToLower())

 If ($attributeCount -LT $limit) {

 $data | Select -ExpandProperty $attribute -Unique | %{

 $attributeData = $data | Where $attribute -EQ $\_

 $count = @($attributeData).count

 [PSCustomObject]@{

 "attribute" = $attribute

 "value" = $\_

 "count" = $count

 }

 }

 }

 }

 } # Terminate Process

 <#

 .SYNOPSIS

 Generates data statistics for a given input.

 .DESCRIPTION

 Given a data input this function will analyze the data based on several criteria and output

 the findings.

 .PARAMETER Data

 The input data to analyze.

 .PARAMETER Limit

 The attribute unique count limit to report on.

 .EXAMPLE

 C:\PS> Get-VPSADataStatistics -Data $data

 attribute value count

 --------- ----- -----

 attribute1 N 3163

 attribute2 Y 6

 .EXAMPLE

 C:\PS> Get-VPSADataStatistics -Data (Import-CSV "C:\InputFile.txt" -Limit 15

 attribute value count

 --------- ----- -----

 attribute1 N 3163

 attribute2 Y 6

 .EXAMPLE

 C:\PS> @($dataSet1, $dataSet2) | Get-VPSADataStatistics

 attribute value count

 --------- ----- -----

 attribute1 N 3163

 attribute2 Y 6

 #>

} # Terminate Function

# Helpful Hints

1. Lessons Learned & Techniques

## Lessons Learned

* **Object Counts**
	+ If you have only one object returned, oftentimes “.count” won’t return a value. You need to wrap that object in a “@($object)” notation to force it to become scaler and then you will get a 1 returned.
* **Hashtables vs. Arrays**
	+ Hashtables are great for doing very fast lookups using the “$hash.ContainsKey()” method but can be cumbersome to use for object manipulation.
	+ Arrays can be very easy to use and work with but can be very slow on lookups over large numbers of items.
	+ In general, use an array first but don’t be afraid to use a Hashtable if the need dictates.
* **[PSCustomObject] vs. Add-Member**
	+ Generating your own custom objects is very powerful and allows you to preciously format your data but using Add-Member allows you to add one or more additional attributes to an existing object being piped in. Both are very useful depending on the use case.
* **Adding whole Objects to Array’s**
	+ Sometimes when you are adding an object to an array such as “$array += $object”, those properties of the object will be adding individually instead of the object as a whole. If you want to add the entire object as an array item, use the notation “$array += ,$object” (note the comma).
* **Enumerating Hashtables**
	+ Make sure to use the method “.GetEnumerator()” which will allow you to use the hashtable rows in a pipeline or loops.

# Questions

Ask away!