PowerShell

ADVANCED FUNCTIONS & MODULES

Microsoft

CONTENT

Learn And Understand Powershell Modules

How To Create And Maintain Powershell Modules

What To Consider While Creating A Powershell Module

How To Install And Use Powershell Modules Effectively

OBJECTIVE

UNDERSTAND

UNDERSTAND

What Is A Module?

Package

Distribute

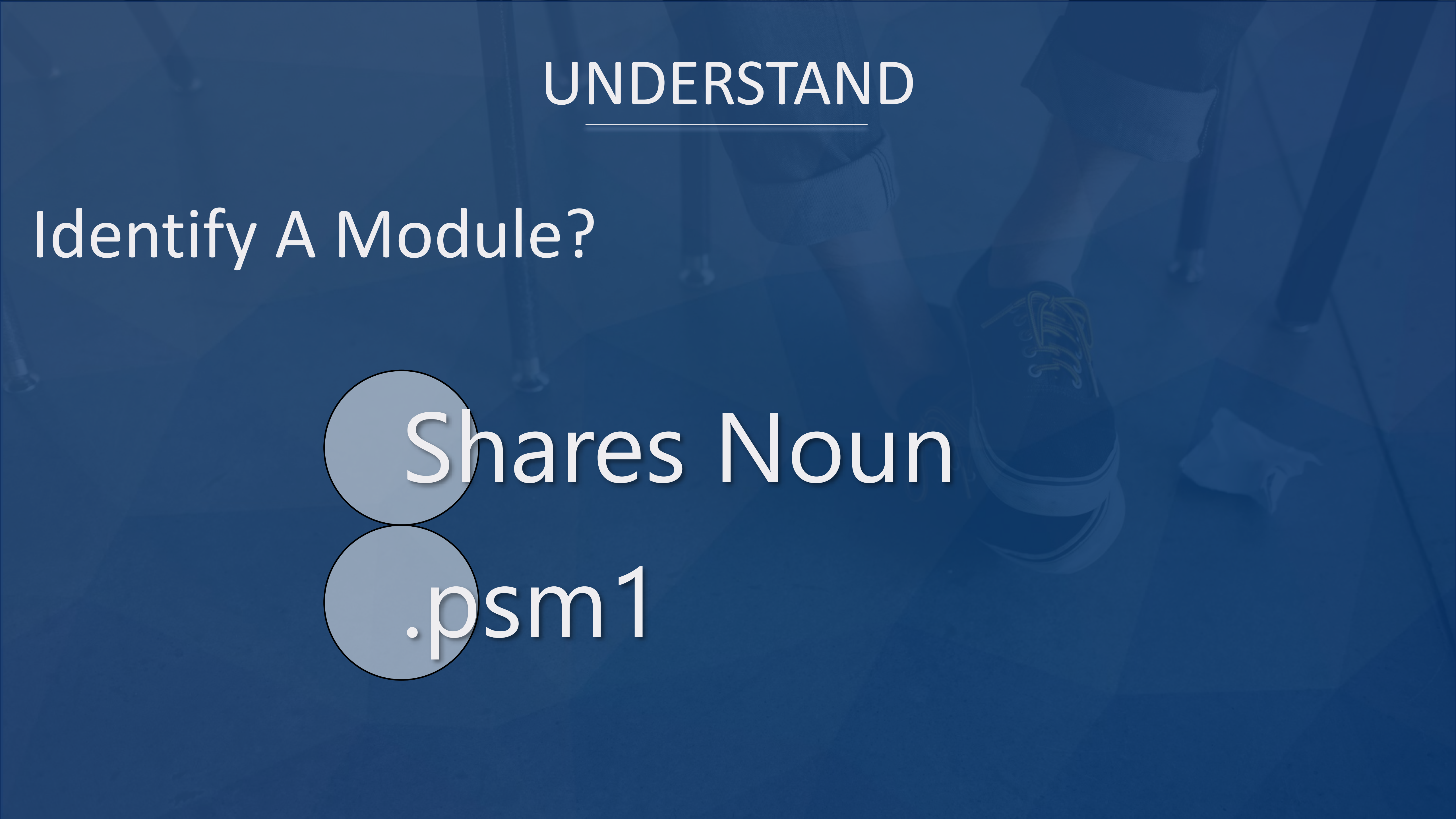
w/o Compilation

partition, organize, and abstract PowerShell code into self-contained, reusable units

UNDERSTAND

Identify A Module?

- Shares Noun
- .psm1



UNDERSTAND

Module Content

CODE

Functionality

CODE SUPPORT

Assemblies, Help Files, Or Scripts

MANIFEST FILE

Describes, Metadata, Author & Versioning

DIRECTORY

Contains Content

UNDERSTAND

Module Types

- Script Modules
- Binary Modules
- Manifest Modules
- Dynamic Modules

UNDERSTAND

What Is A Metafile?

- Define Prerequisites
- Contains Metadata
- Describes Contents & Attributes
- Component Process
- Processing Directives
- Member Restriction

WRITE

WRITE

```
New-Module
[-ScriptBlock] <ScriptBlock>
[-Function <String[]>]
[-Cmdlet <String[]>]
[-ReturnResult]
[-AsCustomObject]
[-ArgumentList <Object[]>]
[<CommonParameters>]
```

New Module

.psm1

WRITE

```
# Export Function
Export-ModuleMember -Function Verb-Noun

# Export With Aliases
Export-ModuleMember `
-Function Verb-Noun, Get-Test `
-Alias vnn, gtt

# Export No Member
Export-ModuleMember

# Export Variable
Export-ModuleMember -Variable Param
```

Accessibility

Use Export Before Function

Module members include

cmdlets,

functions,

variables &

aliases

WRITE

New Module Manifest

```
PS C:\> New-ModuleManifest -Path Test-Module.psd1 -PassThru

## Module manifest for module 'TestModule'
## Generated by: User01
## Generated on: 1/24/2012
#@{
# Script module or binary module file associated with this manifest
# RootModule = ''
# Version number of this module.ModuleVersion = '1.0'
# ID used to uniquely identify this moduleGUID = 'd0a9150d-b6a4-4b17-a325-e3a24fed0aa9'
# Author of this moduleAuthor = 'User01'
# Company or vendor of this moduleCompanyName = 'Unknown'
# Copyright statement for this moduleCopyright = '(c) 2012 User01. All rights reserved.'
# Description of the functionality provided by this module
# Description = ''
# Minimum version of the Windows PowerShell engine required by this module
# PowerShellVersion = ''
# Name of the Windows PowerShell host required by this module
# PowerShellHostName = ''
# Minimum version of the Windows PowerShell host required by this module
# PowerShellHostVersion = ''
# Minimum version of the .NET Framework required by this module
# DotNetFrameworkVersion = ''
# Minimum version of the common language runtime (CLR) required by this module
# CLRVersion = ''
# Processor architecture (None, X86, Amd64) required by this module
# ProcessorArchitecture = ''
# Modules that must be imported into the global environment prior to importing this module
# RequiredModules = @()
# Assemblies that must be loaded prior to importing this module
# RequiredAssemblies = @()
# Script files (.ps1) that are run in the caller's environment prior to importing this module
# ScriptsToProcess = @()
# Type files (.ps1xml) to be loaded when importing this module
# TypesToProcess = @()
# Format files (.ps1xml) to be loaded when importing this module
# FormatsToProcess = @()
# Modules to import as nested modules of the module specified in RootModule/ModuleToProcess
# NestedModules = @()
# Functions to export from this moduleFunctionsToExport = '*'
# Cmdlets to export from this moduleCmdletsToExport = '*'
# Variables to export from this moduleVariablesToExport = '*'
# Aliases to export from this moduleAliasesToExport = '*'
# List of all modules packaged with this module# ModuleList = @()
# List of all files packaged with this module
# FileList = @()
# Private data to pass to the module specified in RootModule/ModuleToProcess
# PrivateData = ''
# HelpInfo URI of this module
# HelpInfoURI = ''
# Default prefix for commands exported from this module. Override the default prefix using Import-Module -Prefix.
# DefaultCommandPrefix = ''}
```

WRITE

Module Location

```
%systemRoot%\users\<user>\
Documents\WindowsPowerShell\
Modules\<moduleName>
```

Description	Path
Current user, Current Host	$Home\[My]Documents\WindowsPowerShell\Profile.ps1
Current User, All Hosts	$Home\[My]Documents\Profile.ps1
All Users, Current Host	$PsHome\Microsoft.PowerShell_profile.ps1
All Users, All Hosts	$PsHome\Profile.ps1

WRITE

```
if (!(Test-Path -Path $Profile))
{
   New-Item -ItemType File `
        -Path $Profile –Force
}
```

Description	Name
Current user, Current Host	$Profile
Current User, All Hosts	$Profile.CurrentUserCurrentHost
All Users, Current Host	$Profile.CurrentUserAllHosts
All Users, All Hosts	$Profile.AllUsersCurrentHost

Profile

Create Profile, if not exists

ADVANCED FUNCTIONS

ADVANCED FUNCTIONS

```
PS C:> Get-Command | ? `
   $_.Source -match "Azure"

Name
-------
Get-Azure…
Get-AzureAccount
Set-AzureAclConfig
```

Function Noun

Choose A Noun That Describes The Module & Groups Functions

ADVANCED FUNCTIONS

```
PS C:> Get-Verb
Verb
-------
Get, Set, Find, Format,
Import, Open, Select, Add,
Remove…
```

Function Verb

Will It Change Something?

Choose A Well Known Verb And Use It For What It Is Expected

ADVANCED FUNCTIONS

Prerequisites

```
#REQUIRES -Version 4.0
#REQUIRES -Modules MyModule1,MyModule2
#REQUIRES -RunAsAdministrator
```

Requires Key Word –Attribute Value1, Value2

Prerequisites Check Enforced By Comments

ADVANCED FUNCTIONS

```
<#
  HELP
    .SYNOPSIS
     Short description
    .DESCRIPTION
     Long description
    .PARAMETER
     Specifies a parameter.
    .EXAMPLE
      C:\PS>
      Example of how to use this cmdlet.
    .INPUTS
     Inputs to this cmdlet (if any).
    .OUTPUTS
     Output to this cmdlet (if any).
    .NOTES
     General notes.
    .COMPONENT
     The component this cmdlet belongs to.
    .FUNCTIONALITY
     The functionality that best describes
     this cmdlet.

#>
```

Help

```
Get-Help Get-Command -Examples
```

```
Example 1:
Get cmdlets, functions, and aliases

    PS C:\>Get-Command

This command gets the Windows PowerShell
cmdlets, functions, and aliases that are
installed on the computer.
```

Help Text

Inside Functions

By .EXAMPLE

ADVANCED FUNCTIONS

```
{
  [CmdletBinding(
    ConfirmImpact=<String>,
    DefaultParameterSetName=<String>,
    HelpURI=<URI>,
    SupportsPaging=$False,
    SupportsShouldProcess=$False,
    PositionalBinding=$True)]
}
```

select output from a very large result set

specifies the name of the parameter used as input

CmdletBindingAttribute

Confirmimpact

Function Should Be Confirmed By $ConfirmPreference

HelpUri

Locate Online Version Of Function Help

SupportsShouldProcess

Adds Confirm And Whatif Parameters

PositionalBinding

Parameters Are Positional By Default

ADVANCED FUNCTIONS

Requesting Confirmation

Requesting Confirmation Process for Commands

Discusses the process that cmdlets, functions, and providers must follow to request a confirmation before they make a change to the system.

Users Requesting Confirmation

Discusses how users can make a cmdlet, function, or provider request confirmation when the Overload:System.Management.Automation.Cmdlet.ShouldProcess method is called.

Confirmation Messages

Provides samples of the different confirmation messages that can be displayed.

ADVANCED FUNCTIONS

```
[OutputType([<Type>],
ParameterSetName="<Name>")]

function Show-World {
  [OutputType([String])]
  Param ($Param)
  Hello $Param
}
```

```
PS C:> (Get-Command Show-World).OutputType
Name           Type
----           ----
System.String  System.String
```

OutputTypeAttribute

Value Only Documentation

Not Derived From Code

Not Compared To Actual Output

Returns .Net Types

ADVANCED FUNCTIONS

```
function Show-World ( [String] $Param1 ) {

    Write-Host “Hello $Param1!”
    “Store $Param1”

}
```

```
PS C:\>$out = Show-World “World”
Hello World!
PS C:\>$out
Store World
```

Inline Parameters
Strongly Typed *

Return Statement

Accessibility?

ADVANCED FUNCTIONS

```
function Show-World {
      param(
         [String] $Param1 = “Hello”,
         [String] $Param2 = “World”
      )

    Write-Host “$Param1 $Param2!”
}
```

Advanced Parameter
attributes & arguments

```
PS C:\>Show-World
Hello World!
PS C:\>Show-World “Test” “Out”
Test Out!
```

Default values

ADVANCED FUNCTIONS

```
function Show-World {
      param(
        [parameter(Mandatory=$true,
        ValueFromPipeline=$true)]
        [String] $Param1 = “Hello”,
        [String] $Param2 = “World”
      )

     Write-Host “$Param1 $Param2”
}
```

Advanced Parameter

attributes & arguments

Attributes Of Parameters

ADVANCED FUNCTIONS

```
Param (
     [parameter(
        Argument1=value1,
        Argument2=value2)]
)
```

Attributes Of Parameters

- Position
- Parameter SetName
- ValueFrom Pipeline
- ValueFromPipeline ByPropertyName
- HelpMessage
- Alias
- Validation

ADVANCED FUNCTIONS

```
Param (
   [parameter(Mandatory=$true)]
   [AllowEmptyString()]
   [AllowEmptyCollection()]
   [ValidateCount(1,5)]
   [ValidateLength(1,10)]
   [ValidatePattern("[0-9][0-9][0-9][0-9]")]
   [ValidateRange(0,10)]
   [ValidateScript( { $_ -ge (Get-Date) } )]
   [ValidateSet("Low", "Average", "High")]
   [ValidateNotNull()]
   [String[]] $ComputerName
)
```

Validation

- AllowEmptyString
- AllowEmpty Collection
- ValidateCount
- ValidateLength
- ValidatePattern
- ValidateRange
- ValidateScript
- ValidateSet
- ValidateNotNull
- ValidateNotNull OrEmpty

Use Current Pipeline Object To Access Input

ADVANCED FUNCTIONS

```
Function Verb-Noun
{
    Param ($Parameter1)
    Begin{}
    Process{}
    End{}
}
```

Advanced Methods

BEGIN *

One-Time Preprocessing

PROCESS

Record-By-Record Processing

END *

One-Time Post-Processing

ADVANCED FUNCTIONS

```
function Verb-Noun {

  <#Comment Section#>
  [CmdletBinding()]
  [OutputType([int])]
  param(
   [Parameter(Mandatory=$true)]
   [ValidateNotNull()]
   [string] Param1)

   begin {  }

   process {  }

   end {  }

} Export-ModuleMember -function Verb-Noun
```

Advanced Function

ADVANCED FUNCTIONS

```
function Verb-Noun {
   <#Comment Section#>
   [CmdletBinding()]
   [OutputType([int])]
   param(
    [Parameter(Mandatory=$true)]
    [ValidateNotNull()]
    [string] Param1)
    begin {  }
    process {  }
    end {  }
} Export-ModuleMember -function Verb-Noun
```

Advanced Function

- Name
- Comment & Help
- CmdletBindingAttribute
- OutputTypeAttribute
- Advanced Parameters
- Advanced Methods
- Accessibility

ADVANCED FUNCTIONS

Return Custom Objects

```
$hash = @{
  Param1 = $parm1
  Param2 = $parm2
}
```

Define A Hash With Attributes

```
$cusObj = New-Object PSObject `
      -Property $hash
```

Create Custom Object
New-Object & Pass Hash

```
return $cusObj
```

Return Custom Object Or
Create As Last Statement

ADVANCED FUNCTIONS

```
[PSCustomObject]@{
  Param1 = $parm1
  Param2 = $parm2
}
```

Return Custom Objects

- Define A Hash With Attributes
- Create Custom Object PSCustomObject Class
- Return Custom Object Explicitly

INSTALLING & IMPORTING

INSTALLING & IMPORTING

```
#REQUIRES
#Help
param ($Par)
Import-Module “module.psm1”

  $lPar = “local parameter”
  Verb-Noun –Param1 $Par `
      –Param2 $lPar


Remove-Module
```

```
PS C:\>. ./run.ps1 –Par “Par”
```

Use Import-Module

Use $Profile To Add On Session Start

Create Script

- Import, Run & Remove

TIPS & TRICKS

TIPS & TRICKS

General Thoughts

Use Visual Studio Code
Use Blueprints To Auto-Generate
Use Pester To Test Functionality
Use Visual Studio Codes Extensions For PowerShell

Create Small Functions Inside Modules &
Create Scripts To Execute Functions From Modules Directly
Create Tests To Test Module Functions And Explain Functionality

Create Help Comments
Stick To The PowerShell Naming Convention

TIPS & TRICKS

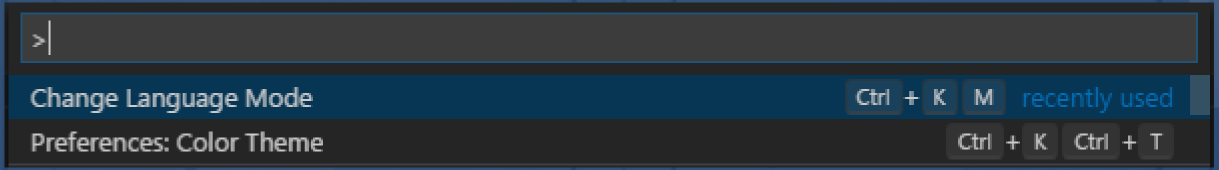
Style And Naming Guidelines

Naming Guidelines

TIPS & TRICKS

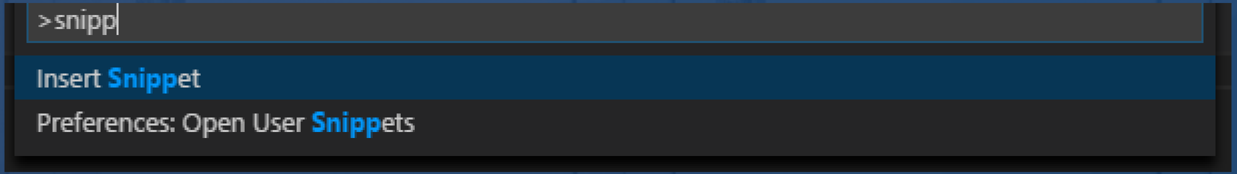
Visual Studio Code

CTRL + Shfit + P



```
>
Change Language Mode                    Ctrl + K  M   recently used
Preferences: Color Theme                    Ctrl + K  Ctrl + T
```

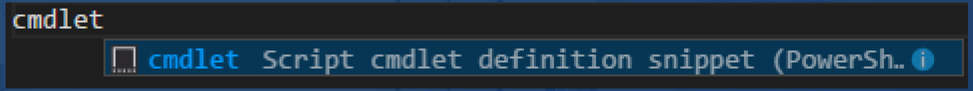
Change Language Mode
-> PowerShell



```
>snipp
Insert Snippet
Preferences: Open User Snippets
```

Perferences: Open User Snippets
-> PowerShell

PowerShell Exentsion



```
cmdlet
    cmdlet  Script cmdlet definition snippet (PowerSh…
```

TIPS & TRICKS

```
"Function advanced": {
    "prefix": "funcadv",
    "body": [
        "function ${verb}-${noun} {",
        "<#.DESCRIPTION",
        "\tDescription",
        ".EXAMPLE",
        "\tExample of how to use this cmdlet",
        "#>",
        "\t[CmdletBinding()]",
        "\t[OutputType([${int}])]",
        "\tparam(",
        "\t\t[Parameter(Mandatory=$${true})]",
        "\t\t[${string}] ${Param1}",
        "\t)",
        "\tbegin {",
        "\t}",
        "\tprocess {",
        "\t\t$0",
        "\t}",
        "\tend {",
        "\t}",
        "}"
    ],
    "description": "Advanced function"
}
```

Visual Studio Code

Blueprint For Advanced Function

Type:

funcadv + Tab

About Automatic Variables

$PSITEM or $_

$PSITEM or $_

$PSITEM   Same as $_.

Contains the current object in the pipeline object. You can use this variable in commands that perform an action on every object or on selected objects in a pipeline.

```
PS C:\> gcm | Where-Object { $PSITEM.Name -match "Azure" }
PS C:\> gcm | Get-Member
PS C:\> gcm | ? { $_.Name -match "Azure" }
```

Enumerate Attributes For $PSITEM

"pipeline character (|) lying on its side"

What the Heck Is $_?

RESERVED WORDS

RESERVED WORDS

- assembly
- exit
- process
- base
- filter
- public
- begin
- finally
- return
- break
- for

- sequence
- catch
- foreach
- static
- class
- from (*)
- switch
- command
- function
- throw
- configuration

- hidden
- trap
- continue
- if
- try
- data
- in
- type
- define (*)
- inlinescript
- until

- do
- interface
- using
- dynamicparam
- module
- var (*)
- else
- namespace
- while
- elseif
- parallel

- workflow
- end
- param
- enum
- private

(*) These keywords are reserved for future use.

AUTOMATIC VARIABLES

AUTOMATIC VARIABLES

- $ARGS
- $ERROR
- $EVENT
- $FALSE
- $FOREACH
- $HOME
- $HOST
- $INPUT
- $LASTEXITCODE
- $MATCHES
- $NULL
- $OFS
- $PROFILE
- $PSBOUNDPARAMETERS
- $PSCMDLET
- $PSHOME
- $PSITEM
- $PSSCRIPTROOT
- $PSVERSIONTABLE
- $PWD
- $THIS
- $TRUE

COMMON PARAMETERS

COMMON PARAMETERS

–Debug (db)

–ErrorAction (ea)

–ErrorVariable (ev)

–InformationAction (infa)

–InformationVariable (iv)

–OutVariable (ov)

–OutBuffer (ob)

–PipelineVariable (pv)

–Verbose (vb)

–WarningAction (wa)

–WarningVariable (wv)

Risk mitigation parameters

–WhatIf (wi)

–Confirm (cf)

aliases in parentheses

https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.core/about/about_commonparameters?view=powershell-5.1

RETURN

RETURN

The Return keyword exits a function, script, or script block. It can be used to exit a scope at a specific point, to return a value, or to indicate that the end of the scope has been reached.

Users who are familiar with languages like C or C# might want to use the Return keyword to make the logic of leaving a scope explicit.

In Windows PowerShell, the results of each statement are returned as output, even without a statement that contains the Return keyword. Languages like C or C# return only the value or values that are specified by the Return keyword.

Exits the current scope, which can be a function, script, or script block.

SCRIPT

SCRIPT

```
#REQUIRES & Help
param ($ComputerName = $(throw "required."))
function CanPing {
  $tmp = Test-Connection $ComputerName
  if (!$?) {
   Write-Host "Ping failed: $ComputerName."
   return $false
  } else {
   write-host "Ping succeeded: $ComputerName"
   return $true
  }
}
function CanRemote {
  $s = New-Pssession $ComputerName
  if($s) {
   Write-Host "Remote succeeded: $ComputerName.“
  } else {
   Write-Host "Remote failed: $ComputerName.“
  }
}
if (CanPing $ComputerName){ CanRemote $ComputerName }
```

```
C:\PS> .\test-remote.ps1 -ComputerName Server01
Ping succeeded: Server01
Remote failed: Server01
```

Functions needs to be loaded into memory

Param statement must be the first statement in a script,
except for comments and any #Requires statements.

Parameter values are available to all of the commands in the script

Messy & Noisy

SCRIPT

```
./test-remote.ps1
#REQUIRES & Help
param ($ComputerName = $(throw "required."))
Import-Module Remote.psm1

  if (Try-Ping $ComputerName){
    Try-Remote $ComputerName
  }

Remove-Module Remote.psm1
```

```
C:\PS> .\test-remote.ps1 -ComputerName Server01
ComputerName   Test     Status
------------   ------   ------
Server01       Ping     $True
Server01       Remote   $False
```

```
./Remote.psm1
#REQUIRES
function Try-Ping {
  # HELP
  param()
   …
}
function Try-Remote {
  # HELP
  param()
  …
}
Export-ModuleMember Try-Ping, Try-Remote
```

Cleaner, Readable & Reusable

HELP

HELP

Cmdlet	Provider	Function	Script	Conceptual ("About")
• Describe cmdlets in a module • XML files • Use command help schema	• Describe providers in a module • XML files • Use the provider help schema	• Describe functions in a module • Can be XML files • Use the command help schema • Or comment-based Help • Topics within the function • or the script • or script module	• Describe scripts in a module • Can be XML files • Use the command help schema • Or comment-based Help • Topics in the script • Or script module	• Describe module and members • To explain how the members can be used together • Are Unicode (UTF-8) encoded Files • File name must use about_<name>.help.txt • about_MyModule.help.txt. • By default, PowerShell includes > 100 of conceptual About Help topics

HELP

CmdletBinding   **HelpUri**

Get-Help Verb-Noun –Examples

HELP

```
TOPIC
    about_<subject or module name>
SHORT DESCRIPTION
    A short, one-line description of the topic
contents.
LONG DESCRIPTION
    A detailed, full description of the subject
or purpose of the module.
EXAMPLES
    Examples of how to use the module or how the
subject feature works in practice.
KEYWORDS
    Terms or titles on which you might expect
your users to search for the information in this
topic.
SEE ALSO
  Text-only references for further reading.
Hyperlinks cannot work in the Windows PowerShell
console.
```

HELP

```
<ModulePath>
  \SampleModule
     \<en-US>
       \about_SampleModule.help.txt
       \SampleModule.dll-help.xml
       \SampleNestedModule.dll-help.xml
    \<fr-FR>
       \about_SampleModule.help.txt
       \SampleModule.dll-help.xml
       \SampleNestedModule.dll-help.xml
```

Splatting

Splatting

```
Get-WmiObject -computername SERVER-R2
-class Win32_LogicalDisk -filter
"DriveType=3" -credential
"Administrator"

@{'key1'='value1';'key2'='value2’}

$parms = @{
  'class'='Win32_BIOS’;
  'computername'='SERVER-R2’;
  'filter'='drivetype=3’;
  'credential'='Administrator’
}

Get-WmiObject @parms
```

You’ll notice a little trick here. The “@” sign is followed by the variable name, which doesn’t include the dollar sign. That’s probably worth a brief explanation. It can be a major “gotcha” in the shell that trips up a lot of people.

PowerShell v5

Classes

Classes

```
Class Car
{
  hidden [String]$vin
  static [int]$numberOfWheels = 4
  [int]$speed
  [datetime]$year
  [String]$model

  function Set-Speed([int] $acc) {
    $this.speed += $acc
 }
}
$chevy = New-Object car
$chevy::numberOfWheels
$chevy
```

Classes

```
using namespace System.Diagnostics
Using module MyModule

function Main
{
  #Using System.Diagnostics.Stopwatch
  $sw = [Stopwatch]::StartNew()
  sleep -Milliseconds 100
  Write-Host (
    'Elapsed: {0} [ms]' -f $sw.ElapsedMilliseconds
  )

}

Main
```

Using statement must appear before any other statements in a script.

Thank You!

Mark Warneke
mark.warneke@microsoft.com
aka.ms/mark/psaf
@mark_mit_k_

Let me know!
aka.ms/mark/psaf/voice

Resources

- Windows PowerShell: Writing Cmdlets in Script
  https://technet.microsoft.com/en-us/library/ff677563.aspx
- About Functions Advanced Methods:
  https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.core/about/about_functions_advanced_methods?view=powershell-5.1
- About Functions Advanced Parameters
  https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.core/about/about_functions_advanced_parameters?view=powershell-5.1
- About Functions CmdletBindingAttribute
  https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.core/about/about_functions_cmdletbindingattribute?view=powershell-5.1
- Scripting Tips and Tricks: CmdletBinding()
  https://blogs.technet.microsoft.com/poshchap/2014/10/24/scripting-tips-and-tricks-cmdletbinding/
- Writing a Windows PowerShell Module:
  https://msdn.microsoft.com/en-us/library/dd878310(v=vs.85).aspx
- How to Write a PowerShell Script Module:
  https://msdn.microsoft.com/en-us/library/dd878340(v=vs.85).aspx
- Building a PowerShell Module:
  ramblingcookiemonster.github.io/Building-A-PowerShell-Module/
- Microsoft Script Center:
  https://technet.microsoft.com/en-us/scriptcenter
- Pester:
  https://github.com/pester/Pester

NEXT: scripts

Scripts

aka.ms/mark/psaf/code

Thanks!

Microsoft