

The background is a dark blue gradient with various abstract elements. In the top left, there are large orange and blue circles. A white square outline is positioned below the orange circle. In the top right, a white triangle outline points right towards a blue circle. On the right side, there is a faint cloud outline. In the bottom right, there are two white rectangle outlines. The bottom left features a faint padlock icon and a purple circle. The bottom center has a faint gear icon. The text 'SPICE WORLD 2021 VIRTUAL' is centered in white.

SPICE WORLD 2021 VIRTUAL



PowerShell Scripting Patterns and Practices

<https://github.com/jdhitsolutions/SpiceWorld2021>

SPICEWORLD2021
VIRTUAL



Jeff Hicks

@jeffhicks

<https://jdhitsolutions.com/blog>

<https://github.com/jdhitsolutions>

SPICEWORLD2021
VIRTUAL

Automating with PowerShell

- PowerShell is about automation
 - Increase efficiency
 - Maintain consistency
 - DevOps
- Automation target is irrelevant
 - Microsoft 365
 - Active Directory
 - Azure
- You will use same scripting techniques and concepts





**You can't learn PowerShell
scripting in 30 minutes...**

SPICEWORLD2021
VIRTUAL



**...but I can teach you
some guiding principles**

Terms

- Cmdlet
 - PowerShell core unit of execution
 - Compiled .NET command
- Script
 - PowerShell “batch” file
 - An automated session or control script
 - .ps1 file extension
 - Can also be used to define functions
- Function
 - Self-contained, repeatable blocks of code
 - Your version of a compiled cmdlet
 - Written in PowerShell's scripting language

Terms

- Module
 - PowerShell's packaging mechanism
 - Collection of functions and supporting files
 - Deploy modules via PowerShell Gallery
 - Or your own
 - Module commands
 - Get-Command –noun module

Start with a command

- Run core command or commands in an interactive console session
- Learning something new?
 - Learn from the GUI
 - Then learn the PowerShell commands
- Identify possible parameters
 - Test with variables
- Think about “scripting at scale”



Use the right tools

- Use a PowerShell-aware scripting tool
 - Notepad is not PowerShell-aware
- The PowerShell ISE is deprecated
 - Doesn't support PowerShell 7
- VS Code is the community standard
 - Free
 - Ecosystem of extensions
 - Cross-platform




Avoid Problems

- Make sure your script file will properly run before starting
- These are your prerequisites at the beginning of your file
- #requires –version 5.1
- #requires –RunAsAdministrator
- #requires –Modules Hyper-V
- Help about_requires
- In modules you can define many of these in the manifest


Naming Conventions

- Scripts can have any name
 - Avoid spaces
- Module name should reflect the toolset
 - Use a prefix
- Function names follow Verb-Noun
 - Get-Verb
 - Singular noun
 - Consider a prefix
 - You can alias the function name
- Variable and Parameter names should be meaningful
 - No Hungarian notation
 - Use PowerShell standard names
- Use full cmdlet and parameter names – ***no aliases***




```
Function MakeHomeFolders {
    Param($p,$n,$server)
    Try {
        [void](Test-WSMan $server -ErrorAction Stop)
        icm {
            $strFold = "Data", "Reports", "Public", "Documents"
            $h = ni $using:p -N $using:n -i Directory
            $strFold | % { ni $h.FullName -N $_ -i Directory }
        } -cn $server
    }
    Catch {
        ww "Unable to create home folders on $server. $($_.exception.message)"
    }
}
```

ww is a privately-defined alias for Write-Warning



```
Function New-HomeFolder {
    [cmdletbinding()]
    [alias("Make-HomeFolder")]
    Param($Path, $Name, $Computername)
    Try {
        [void](Test-WSMan -ComputerName $Computername -ErrorAction Stop)
        Invoke-Command -ScriptBlock {
            $Folders = "Data", "Reports", "Public", "Documents"
            # $Home is a built-in variable so don't use that
            $homeFolder = New-Item -Path $using:Path -Name $using:Name -ItemType Directory
            $Folders | ForEach-Object {
                New-Item -Name $_ -Path $homeFolder.FullName -ItemType Directory
            }
        } -ComputerName $Computername
    }
    Catch {
        Write-Warning "Unable to create home folders on $Computername.
        $($_.exception.message)"
    }
}
```



Parameters

- You can use parameters in scripts and functions
- Use common and standard parameter names
- Parameter names become variables in your code
- Simple names
 - No spaces, numbers, or special characters
- Define parameter aliases to satisfy corporate culture

Parameters

- Take advantage of parameter validation
- Carefully consider positional vs named parameters
- *Tip*: define a help message
- `help about_functions_advanced_parameters`



Param(

```
[Parameter(Position = 0, HelpMessage = "Specify the username")]  
[ValidatePattern("^\w{2,15}$")]  
[string]$Name,
```

```
[Parameter(HelpMessage = "Specify the top-level path")]  
[ValidateNotNullorEmpty()]  
[string]$Path = "D:\Users",
```

```
[Parameter(HelpMessage = "Enter the server name")]  
[alias("server", "cn")]  
[ValidateSet("SRV1", "SRV2", "SRV3")]  
[string]$Computename = "SRV1",
```

```
[Parameter(Mandatory, HelpMessage = "Enter an alternate credential")]  
[alias("RunAs")]  
[PSCredential]$Credential
```

)

Objects in the pipeline

- Functions write one type of object to the pipeline
- Write-Host is for messaging not output
 - Write-Output is redundant
 - Learn to use Write-Progress
- Do not format your output
- Learn to create your own formatting



```
Function Get-Server {
```

```
#This is a poor scripting exaxmple
```

```
Param($Computername = $env:COMPUTERNAME)
```

```
$os =Get-CimInstance win32_Operatingsystem -CimSession $Computername
```

```
Write-Host ($Computername + " [" + $os.Caption+ "]")
```

```
$t = Get-Process -ComputerName $Computername |
```

```
sort workingset -Descending | select -first 5
```

```
write-host "Top processes"
```


```
$t
```

```
$c = Get-Volume -DriveLetter C -CimSession $Computername
```

```
Write-host ("Free space on C: " + $c.SizeRemaining)
```

```
Write-Host ("Free memory: " + $os.FreePhysicalMemory)
```

```
}
```



```
Function Get-Server {  
    [cmdletbinding()]  
    [outputtype("companyServerInfo")]  
    Param($Computername = $env:COMPUTERNAME)
```


```
    Write-Host "Getting server info for $Computername" -ForegroundColor Green  
    $os = Get-CimInstance win32_Operatingsystem -CimSession $Computername  
    $t = Get-Process -ComputerName $Computername |  
    Sort-Object workingset -Descending | Select-Object -First 5  
    $c = Get-Volume -DriveLetter C -CimSession $Computername  
    [pscustomobject]@{  
        PSTypename           = "companyServerInfo"  
        Computername         = $Computername.ToUpper()  
        OperatingSystem      = $os.caption  
        TopProcesses         = $t  
        FreeDiskGB           = $c.SizeRemaining / 1GB  
        FreeMemoryGB         = $os.FreePhysicalMemory / 1mb  
        ReportDate           = Get-Date  
    }  
}
```



```
PS C:\> Get-Server
```

```
Getting server info for THINKP1
```

```
Computername      : THINKP1
OperatingSystem    : Microsoft Windows 11 Pro
TopProcesses       : {System.Diagnostics.Process (Memory Compression), System.Diagnostics.Process (SamsungMagician),
                     System.Diagnostics.Process (firefox), System.Diagnostics.Process (dwm)...}
FreeDiskGB         : 104.209136962891
FreeMemoryGB       : 9.69895172119141
ReportDate         : 8/3/2021 10:59:51 AM
```



```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration>
  <ViewDefinitions>
    <View>
      <Name>default</Name>
      <ViewSelectedBy>
        <TypeName>CompanyServerinfo</TypeName>
      </ViewSelectedBy>
      <GroupBy>
        <PropertyName>Computername</PropertyName>
      </GroupBy>
      <TableControl>
        <TableHeaders>
          <TableColumnHeader>
            <Label>OperatingSystem</Label>
            <Width>25</Width>
            <Alignment>left</Alignment>
          </TableColumnHeader>
          <TableColumnHeader>
            <Label>TopProcesses</Label>
            <Width>51</Width>
            <Alignment>left</Alignment>
            ...
          </TableColumnHeader>
        </TableHeaders>
      </TableControl>
    </View>
  </ViewDefinitions>
</Configuration>
```



```
PS C:\> Get-Server
```

```
Getting server info for THINKP1
```

```
Computername: THINKP1
```

OperatingSystem	TopProcesses	FreeDiskGB	FreeMemGB
-----	-----	-----	-----
Microsoft Windows 11 Pro	Memory Compression,powershell,SamsungMagician,fi...	104.3874	8




```
PS C:\> $s | select *
```

```
Computername      : THINKP1
OperatingSystem    : Microsoft Windows 11 Pro
TopProcesses       : {System.Diagnostics.Process (Memory Co
                    System.Diagnostics.Process (powershell
FreeDiskGB         : 104.222946166992
FreeMemoryGB       : 8.9820442199707
ReportDate        : 8/3/2021 11:07:29 AM
```

PS C:\> \$s.TopProcesses

Handles	NPM(K)	PM(K)	WS(K)	CPU(s)	Id	SI	ProcessName
-----	-----	-----	-----	-----	--	--	-----
0	0	3916	1603392	162.42	3852	0	Memory Compression
781	578	130564	1246260	53.48	15752	1	SamsungMagician
1186	104	5039184	4865980	2,839.17	12384	1	powershell
618	100	447488	423136	52.52	21832	1	firefox
1738	397	5230604	406964	4,792.97	1808	1	dwm




```
Function Get-FolderReport {  
    Param($Folder)  
  
    dir $folder -file | Group extension | Select Count,Name,  
    @{N="Size";E={$_.group | measure length -sum | select -expand sum}} |  
    Sort Size -Descending | Ft -auto  
}
```

```

Function Get-FolderReport {
    [cmdletbinding()]
    [alias("gfr")]
    [outputtype("companyFolderReport")]
    Param(
        [Parameter(Position = 0, HelpMessage = "Enter the folder path")]
        [ValidateScript( { Test-Path $_ })]
        [string]$Path = "."
    )

    $groups = Get-ChildItem -Path $Path -File | Where-Object {$_.Extension} |
    Group-Object -Property extension
    foreach ($item in $Groups) {
        $size = $item.Group | Measure-Object -Property length -Sum
        [pscustomobject]@{
            PSTypename = "companyFolderReport"
            Path        = (Convert-Path $Path)
            Extension   = $item.Name.substring(1)
            Count       = $item.Count
            Size        = $size.sum
            AuditDate    = Get-Date
        }
    }
}

```



```
Path      : C:\work
Extension : iso
Count     : 1
Size      : 1694584141
AuditDate : 8/3/2021 4:45:28 PM
```



```
PS C:\> Get-FolderReport c:\work | Sort size -Descending | Select -first 10
```

Directory: C:\work

Count	Extension	SizeKB
-----	-----	-----
1	iso	1654867.3252
10	xml	66328.6562
1	zip	13177.1904
8	html	566.5801
8	json	357.584
24	txt	312.5029
7	csv	183.6768
1	pdf	78.8125
4	png	46.4824
2	jpg	23.2686

```
PS C:\> |
```

Functions as cmdlets

- Use [cmdletbinding()]
 - Common parameters
 - Pipeline input
 - Whatif
 - ParameterSets
- Insert Write-Verbose during development
- Begin/Process/End scriptblocks
- help about_functions_advanced*



Advanced Function Demo

SPICEWORLD2021
VIRTUAL



Tips for better PowerShell code

- Document your code
- Use cmdlets over .NET code
- Learn to use Write-Progress
- Leverage splatting



Tips for better PowerShell code

- Handle Errors
- Properly format your code
- Who is your code for?
- Don't let Google write your code

Resources

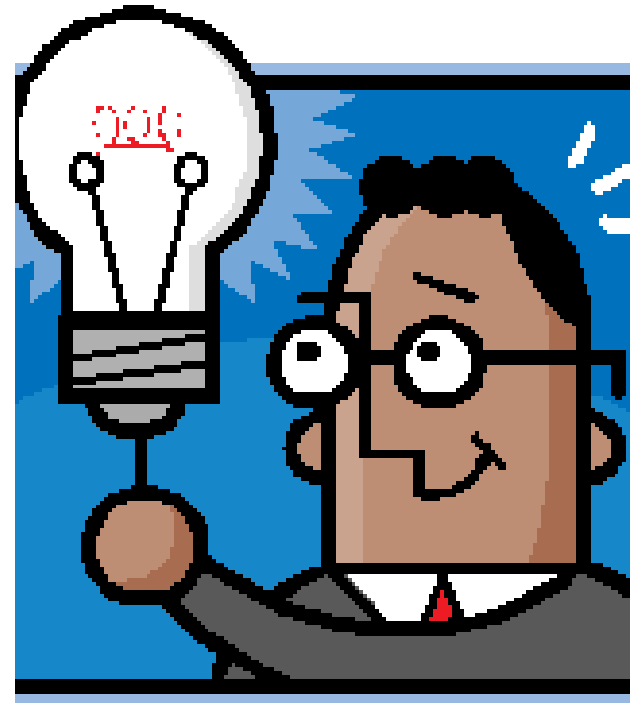
<https://docs.microsoft.com/powershell/>

<https://jdhitsolutions.com/blog>

PowerShell.org

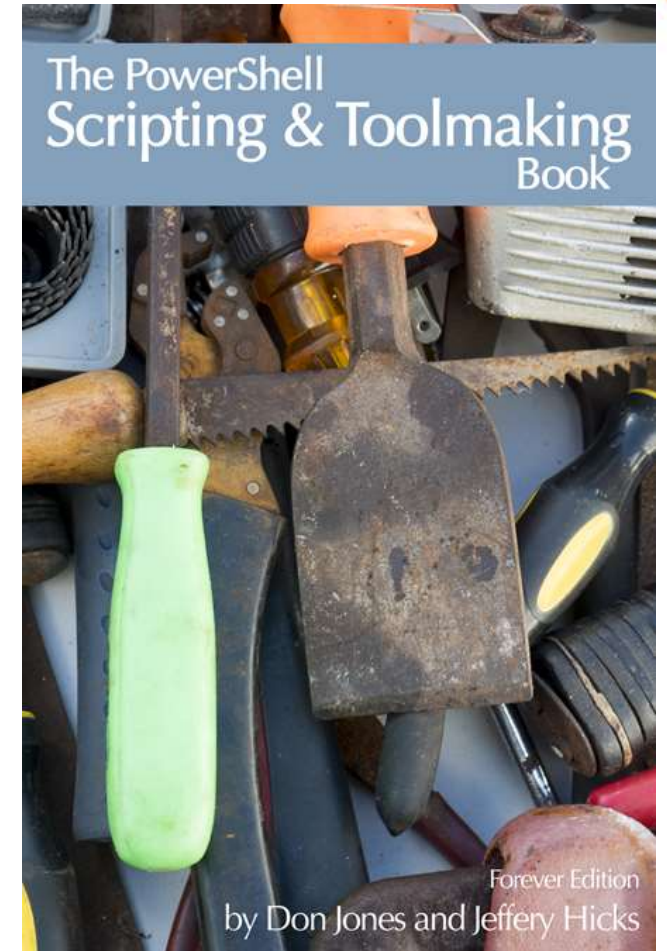
#PowerShell

Pluralsight.com



Resources

- You can start with Learn PowerShell Scripting in a Month of Lunches (Manning)
- <https://leanpub.com/powershell-scripting-toolmaking>
- Check out other PowerShell resources on Leanpub





Questions and answers coming up

<https://github.com/jdhitsolutions/SpiceWorld2021>

SPICEWORLD2021
VIRTUAL



Thank you.

SPICEWORLD2021
VIRTUAL