POWERSHELL CRASH COURSE

PRESENTED BY: JAMES HONEYCUTT

ABOUT ME

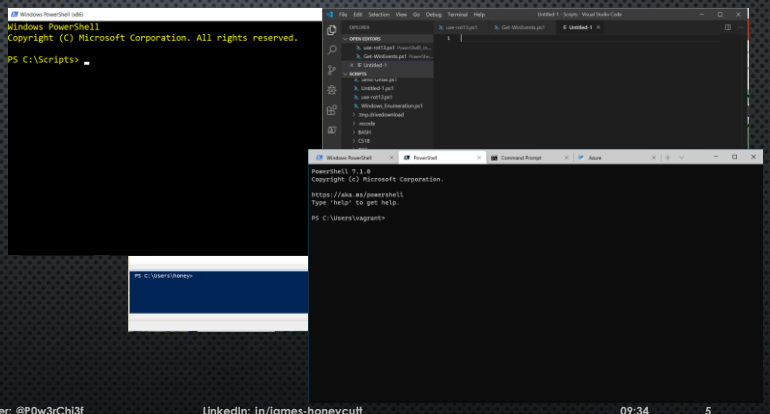
- Husband & Father
- 24 year Soldier
- Trilogy TA – Cyber
- Howard Community College Instructor
- Former Systems Administrator

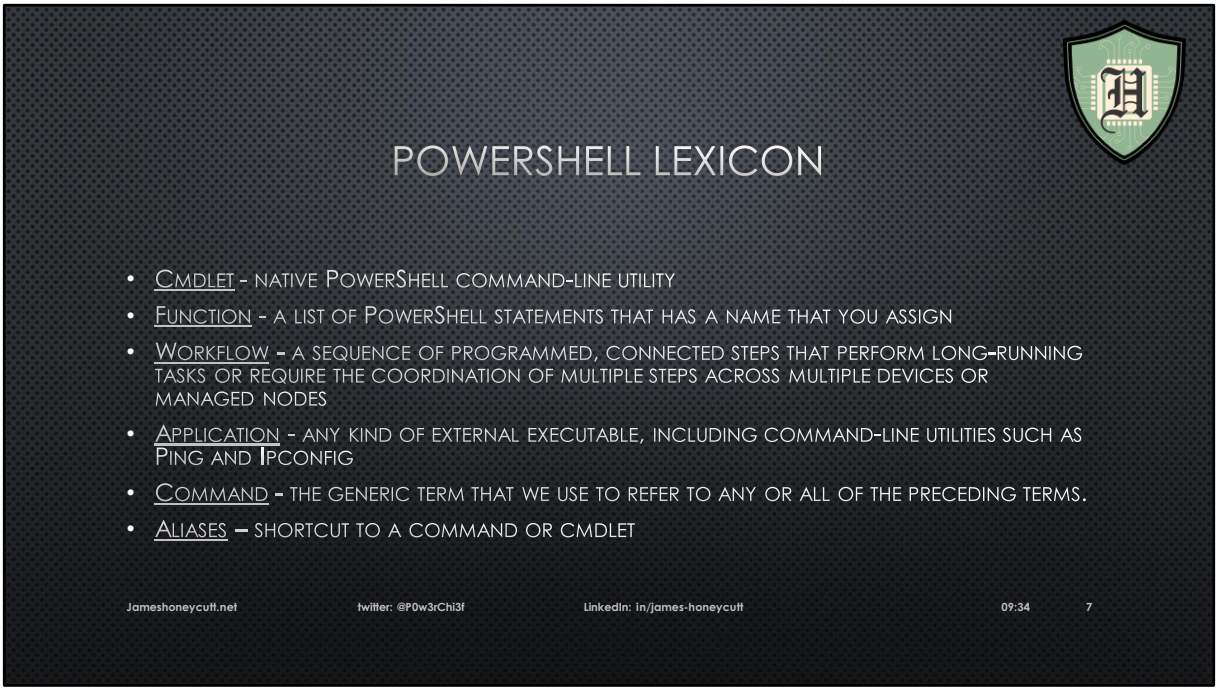
WHY SCRIPTING

- WE HAVE TOOLS
- ONE OFFS
- GUI IS FASTER
- ROUTINE TASKS

ENVIRONMENTS

- PowerShell Console
- PowerShell ISE
- VSCode
- Windows Terminal





A *cmdlet* is a native PowerShell command-line utility. These exist only inside PowerShell and are written in a .NET Framework language such as C#. The word *cmdlet* is unique to PowerShell, so if you add it to your search keywords on Google or Bing, the results you get back will be mainly PowerShell-related. The word is pronounced *command-let*.

A *function* can be similar to a cmdlet, but rather than being written in a .NET language, functions are written in PowerShell's own scripting language.

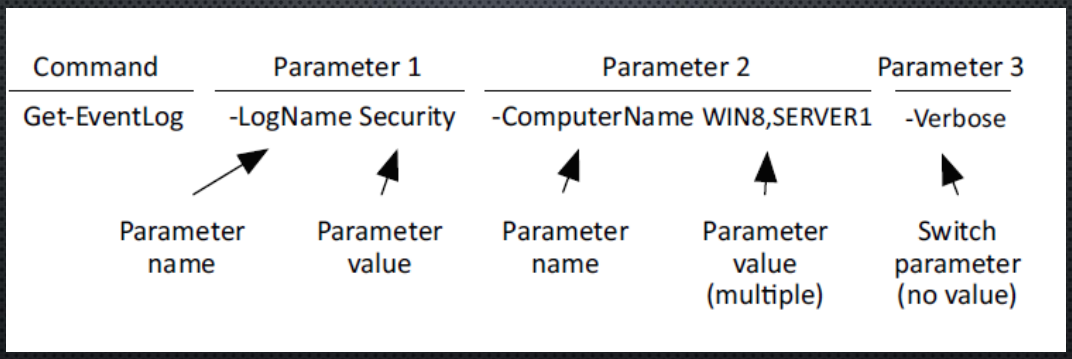
A *workflow* is a special kind of function that ties into PowerShell's workflow execution system.

An *application* is any kind of external executable, including command-line utilities such as Ping and Ipconfig.

Command is the generic term that we use to refer to any or all of the preceding terms.

An alias is a shortcut to a command

COMMAND STRUCTURE



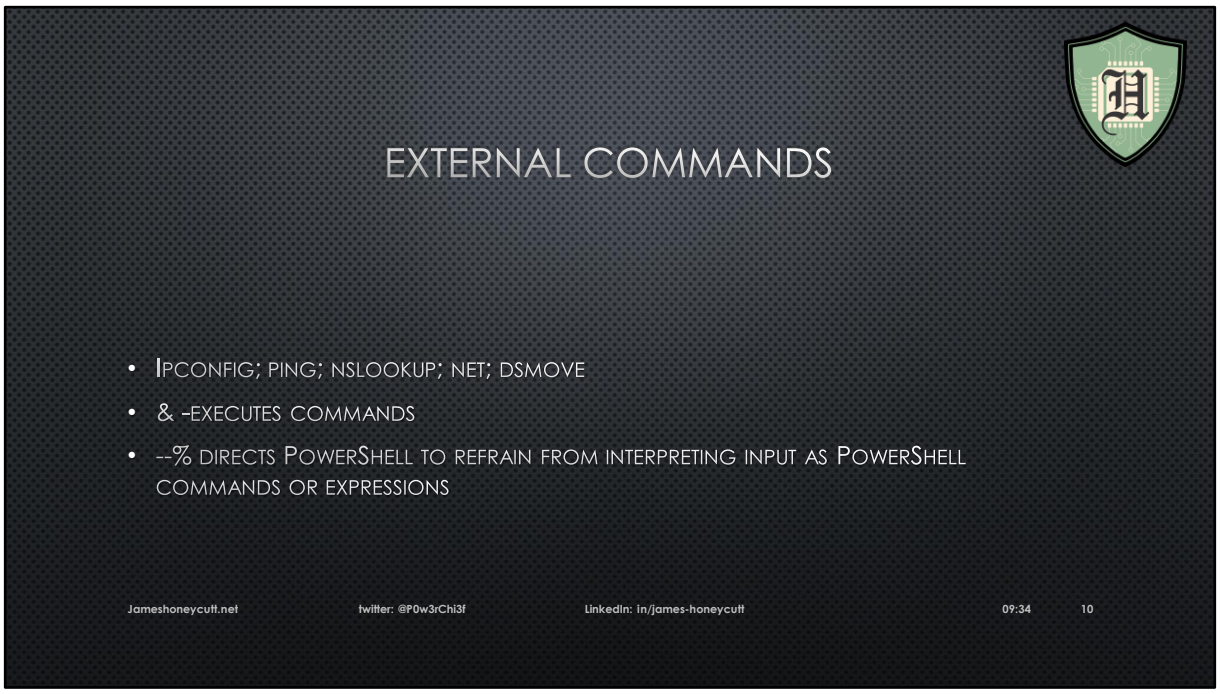
Demo/Try It Now
Get-Help (get-even)
Get-Command
Get-Member
Show-Command
Get-alias

Note
Multiple Parameter sets has unique parameters

Update-Help

get-help *even*
get-help Get-WinEvent
-asstring only works works with computername and list attribute

[[parameter] <type[]>] = optional
[parameter] <type>= mandatory and positional
Parameter <type> = non-positional



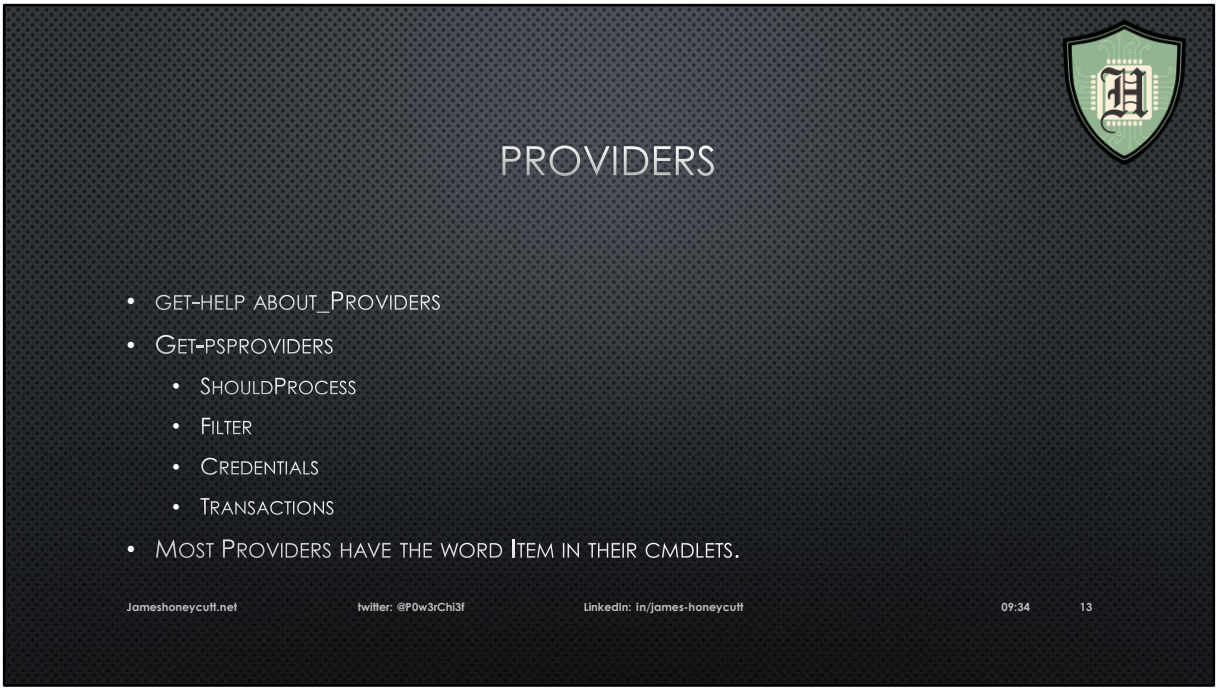
C:\windows\system32\sc.exe --% qc "bits"

DEALING WITH ERRORS

```
PS C:\Scripts> get help
get : The term 'get' is not recognized as the name of a cmdlet, function, script file, or operable program. Check the spelling of
the name, or if a path was included, verify that the path is correct and try again.
At line:1 char:1
+ get help
+ ~~~
    + CategoryInfo          : ObjectNotFound: (get:String) [], CommandNotFoundException
    + FullyQualifiedErrorId : CommandNotFoundException
```

```
PS C:\Scripts> dir C:\Windows /S
dir : Second path fragment must not be a drive or UNC name.
Parameter name: path2
At line:1 char:1
+ dir C:\Windows /S
+ ~~~~~~~~~~~~~~~~
    + CategoryInfo          : InvalidArgument: (C:\Windows:String) [Get-ChildItem], ArgumentException
    + FullyQualifiedErrorId : DirArgumentError,Microsoft.PowerShell.Commands.GetChildItemCommand
```


$error

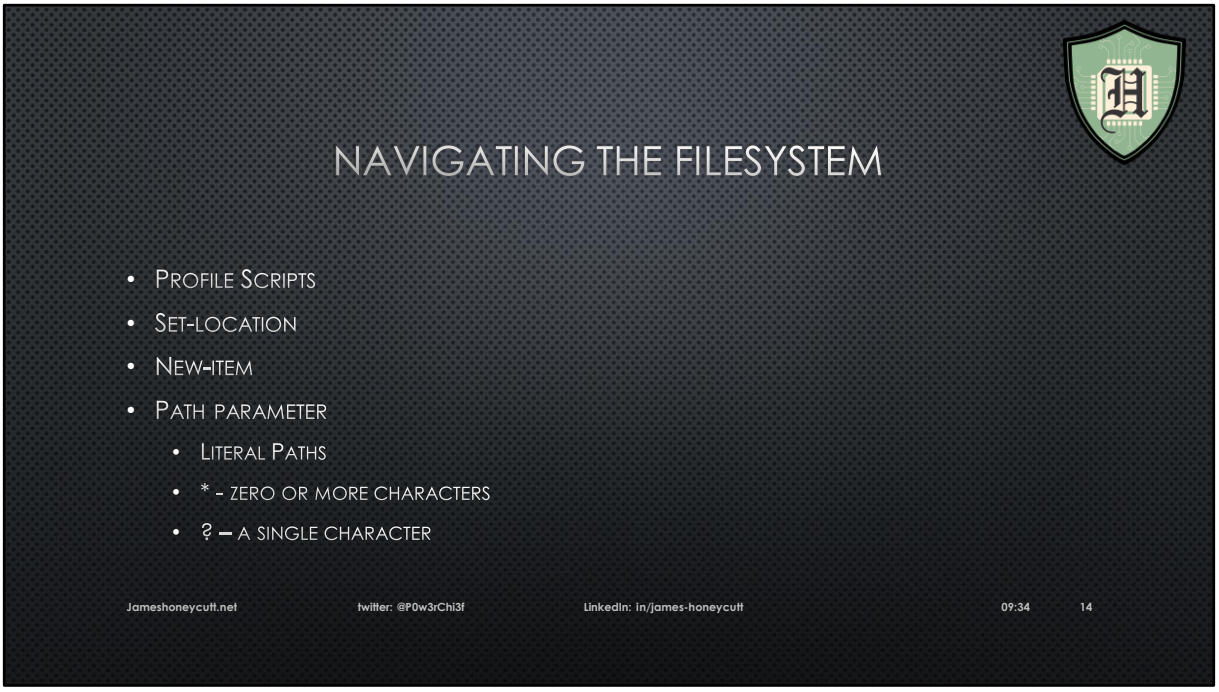


ShouldProcess—The provider supports the use of the -WhatIf and -Confirm parameters, enabling you to “test” certain actions before committing to them.

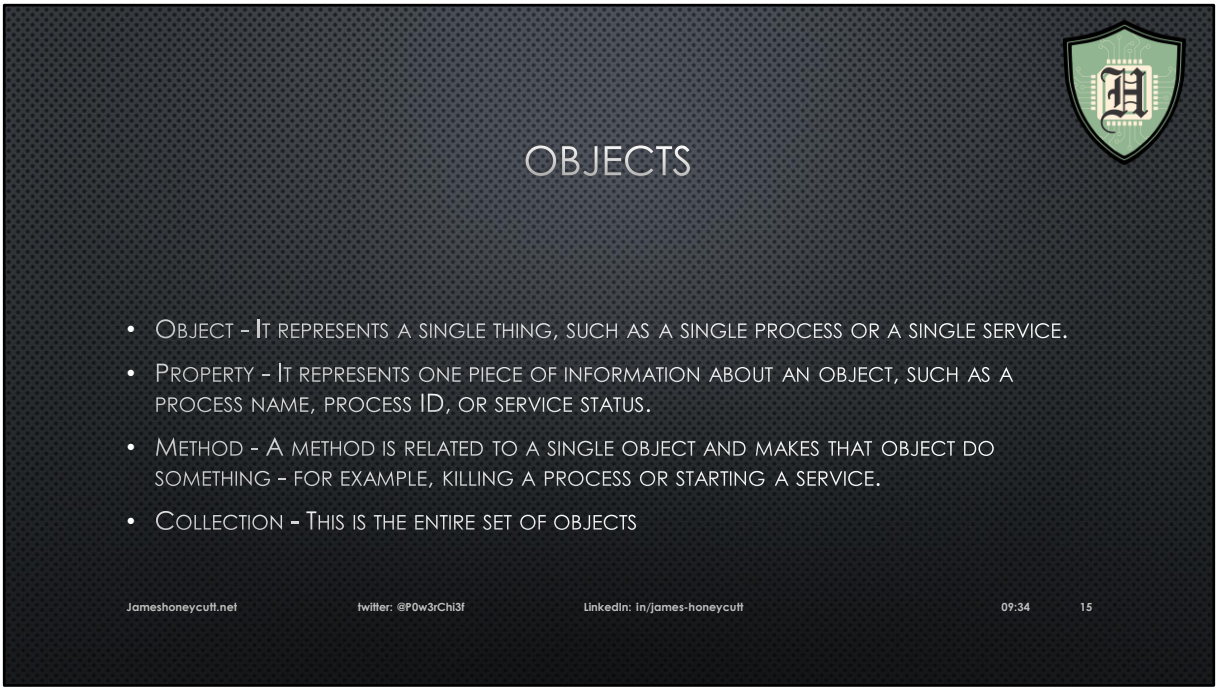
Filter—The provider supports the -Filter parameter on the cmdlets that manipulate providers’ content.

Credentials—The provider permits you to specify alternate credentials when connecting to data stores. There’s a -credential parameter for this.

Transactions—The provider supports the use of transactions, which allows you to use the provider to make several changes, and then either roll back or commit those changes as a single unit.




```
Get-childitem C:\Scripts\*\?????????.ps1
Get-childitem C:\Scripts\*\????.ps1
Get-childitem C:\Scripts\*\*.ps1
Get-childitem C:\*\*.ps1
Get-childitem C:\*\*\*.ps1

Get-childitem HKLM:\Software\Microsoft\Windows\CurrentVersion\Run
Get-childitem HKCU:\Software\Microsoft\Windows\CurrentVersion\Run
Get-childitem HKLM:\Software\Microsoft\Windows\CurrentVersion\RunOnce
Get-childitem HKCU:\Software\Microsoft\Windows\CurrentVersion\RunOnce

Measure-Command -Expression {Get-childitem HKCU:\Software\*\*\*\}
Measure-Command -Expression {Get-childitem
HKCU:\Software\Microsoft\Windows\CurrentVersion}
```



Tasklist pull information, but cannot perform any actions

Get-Process | ConvertTo-HTML | Out-File processes.html

get-process | get-member | where {$_.membertype -eq 'Property'} | group MemberType | sort count –Descending

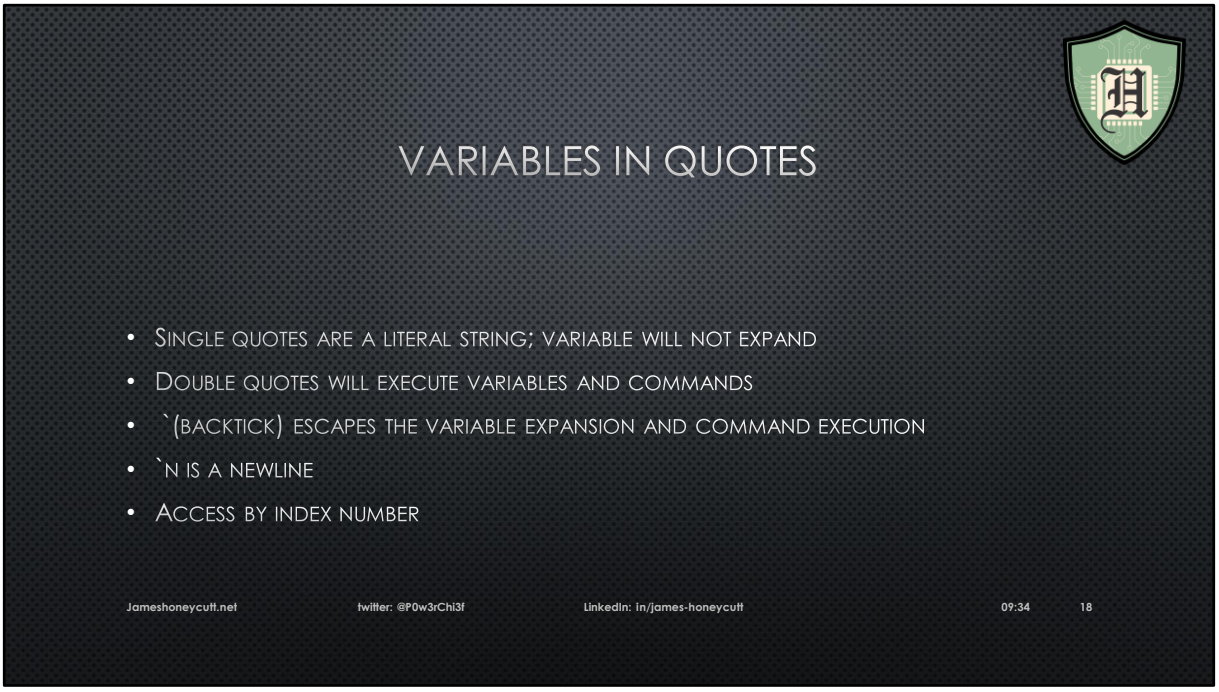
(Get-service –name bits).start()
(Get-service –name bits).stop()

Get-Service * | where {($_.starttype -like "man*") -and ($_.status -eq "Stopped")} | Select-Object -Property Name, Status, StartType
get-service * | where {$_.starttype -like "man*"} | select Name, Status, StartType | group status | sort count –Descending

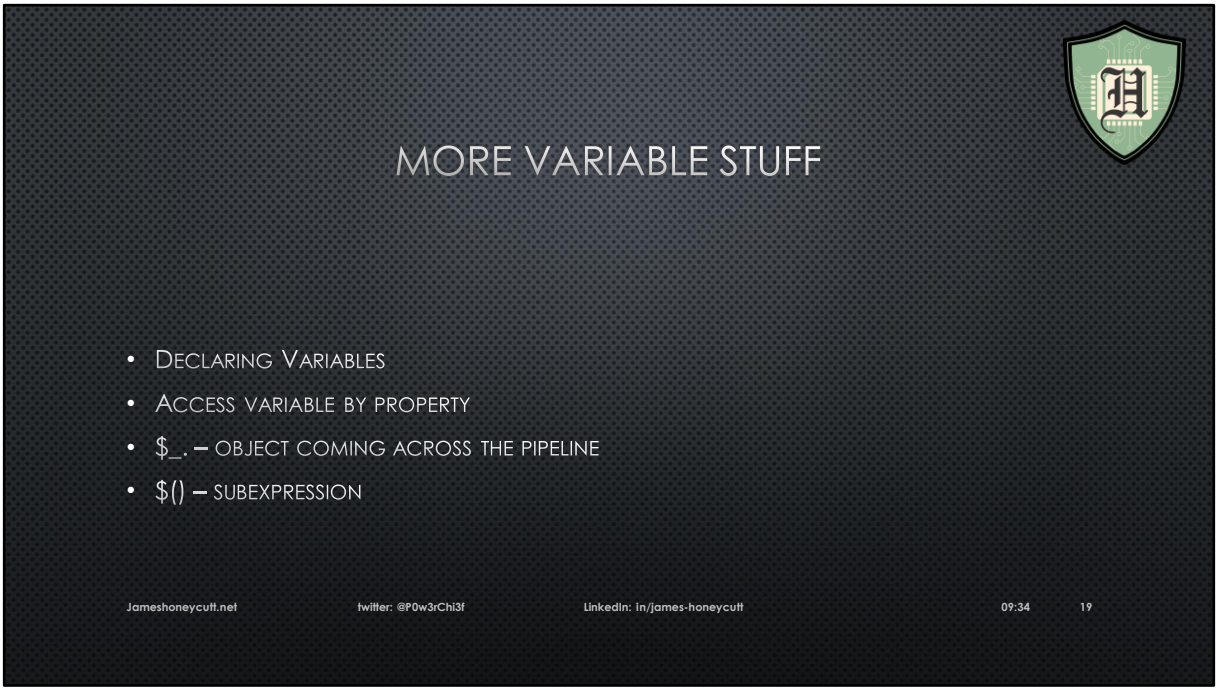
Get-Process | Sort VM -descending | gm
Get-Process | Sort VM -descending | Select Name,ID,VM | gm

VARIABLES

- Place to store your stuff
- Usually contain letters, numbers, and underscores
- Variable names can contain spaces
- Try to make variable names sensible




```
$var = 5
$String = ‘What does $var contain?’
$String2 = “What does $var contain?”
$var = 89; $string2

$string3 = "`$var contains $var“
"`$(get-date) is how you execute a command within quotes“

$object1 = 'Object01 Object02 Object03’
$object2 = 'Object01','Object02','Object03'
$objectList = "Object01 `nObject02 `nObject03“

$ListVariables = 'Object1', 'Object2', 'Object3', 'Object4’
$var[0]
$var[-1] last object
$var[-2] 2nd to last objects
$var.length
$var.touppper()
$var.tolower()                    $var.replace()
```




```
$services = Get-service
$services.name

Get-Service | ForEach-Object { Write-Output $_.Name }

$today = “Today is get-date”
$today = "Today is $($(get-date).DayOfWeek) the $($(get-date).day)th"

$number = Read-host “Enter a number: “
[int]$number = Read-host "Enter a number"
```

VARIABLE TYPES

- [SINGLE] – SINGLE(32)-PRECISION
- [DOUBLE] - DOUBLE(64)-PRECISION FLOATING NUMBERS (NUMBERS WITH A DECIMAL PORTION)
- [STRING]—A STRING OF CHARACTERS
- [CHAR]—EXACTLY ONE CHARACTER [XML]—AN XML DOCUMENT
- [ADSI]—AN ACTIVE DIRECTORY SERVICE INTERFACES (ADSI) QUERY
- [CHAR] - A UNICODE 16-BIT CHARACTER
- [BYTE] - AN 8-BIT UNSIGNED CHARACTER
- [INT] - 32-BIT SIGNED INTEGER
- [LONG] - 64-BIT SIGNED INTEGER

VARIABLE TYPES (CONT.)

- [BOOL] BOOLEAN TRUE/FALSE VALUE
- [DECIMAL] A 128-BIT DECIMAL VALUE
- [DATETIME] DATE AND TIME
- [ARRAY] AN ARRAY OF VALUES
- [HASHTABLE] HASHTABLE OBJECT

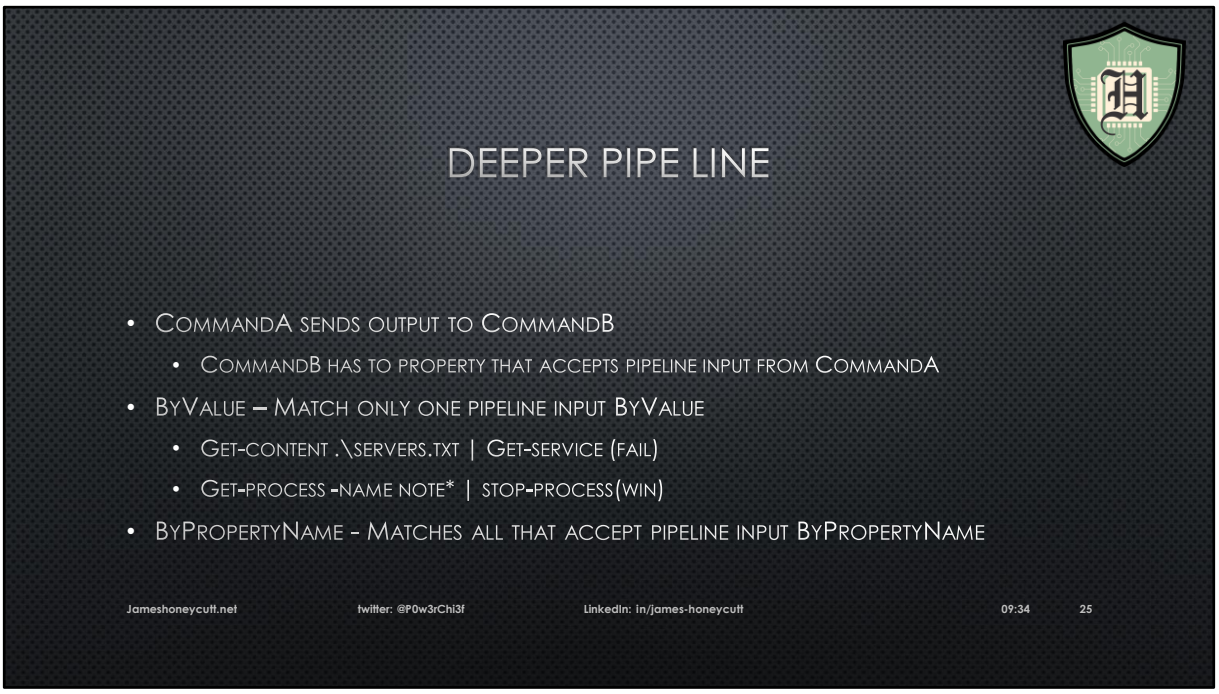
VARIABLE COMANDS

- New-Variable
- Set-Variable
- Remove-Variable
- Get-Variable
- Clear-Variable
- Get-ChildItem Env:

Get-Service | ConvertTo-HTML | Out-File services.html

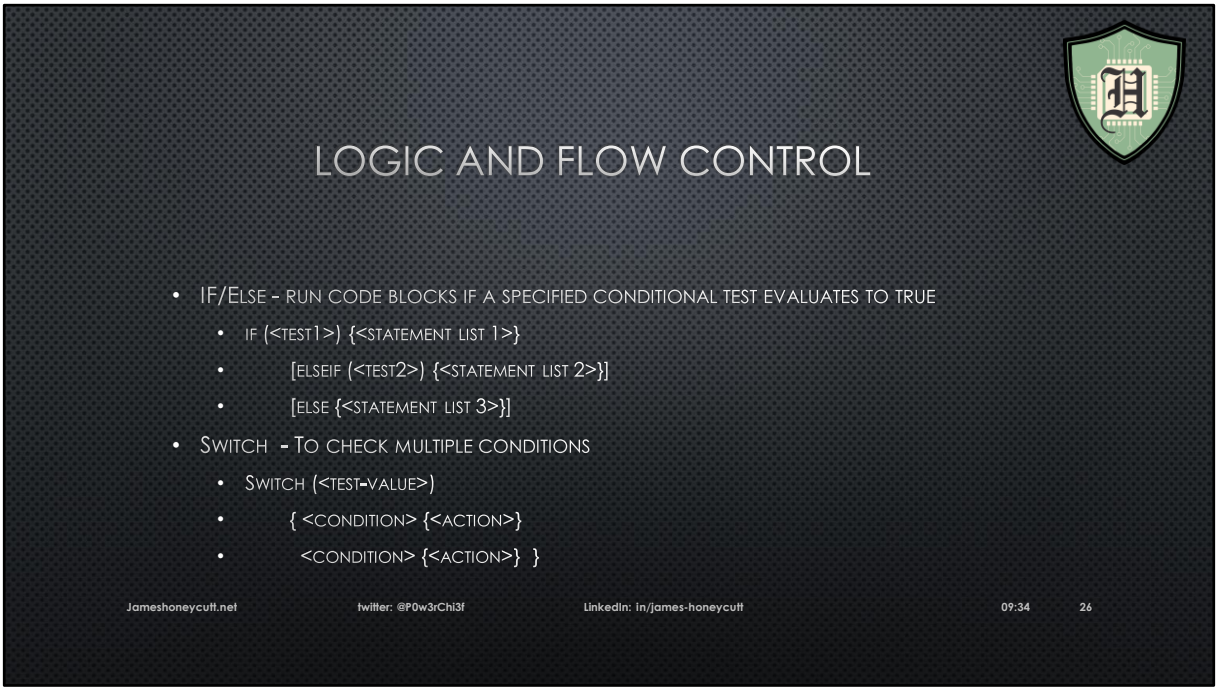
Get-Process -name Notepad | Stop-Process

get-process | export-csv c:\temp\processes.csv -NoTypeInformation



Get-help get-service –full

Get-content .\servers.txt | Get-service (fail)
Get-process -name note* | stop-process(win)



If/Elses

```
 $a = 1
if (($a -gt 2) -and ($a -le 10))
   {Write-Host "The Value $a is greater than 2 less than 10."}
elseif  ($a -le 2)
   {Write-Host "The value $a is less than 2."}
else {Write-Host "The value $a is greater than 10."}
```

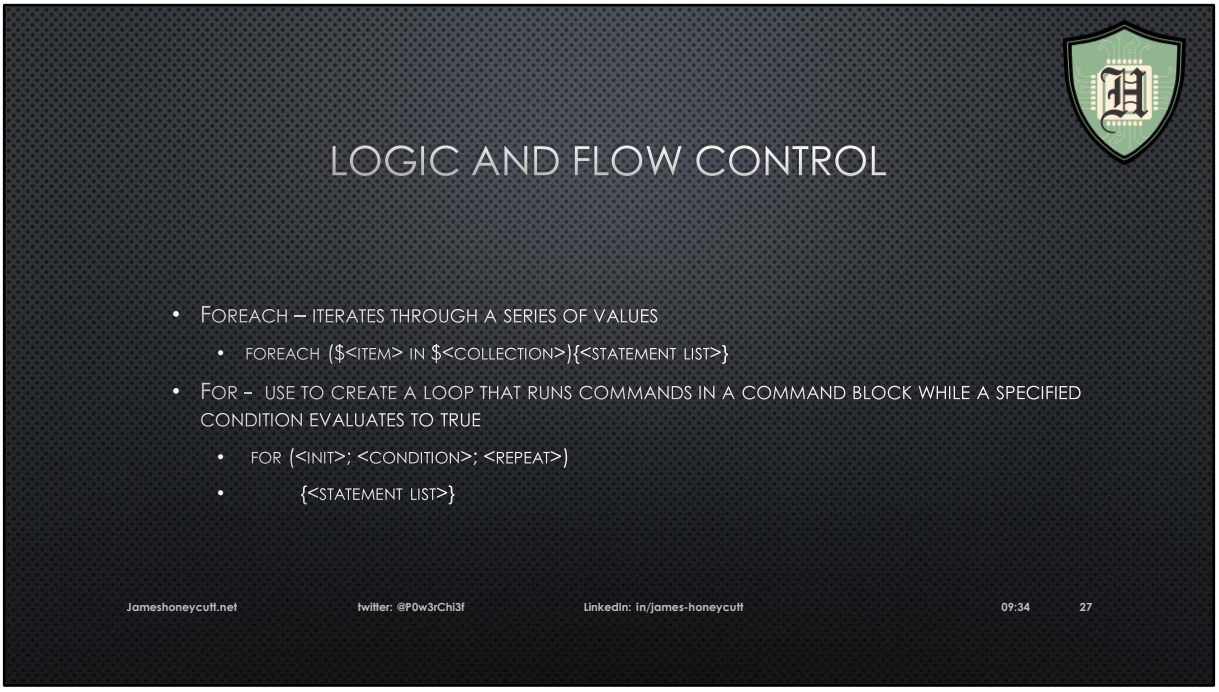
code “C:\Scripts\PowerShell\UserAccounts\Uploaded to GitHub\New-ADUser_Prompted.ps1”

Switch

```
$a = 3
switch ($a)
       {
         1 {"It is one."}
         2 {"It is two."}
         3 {"It is three."; Break}
```

```
        4 {"It is four."}
        3 {"Three again."}
      }
```

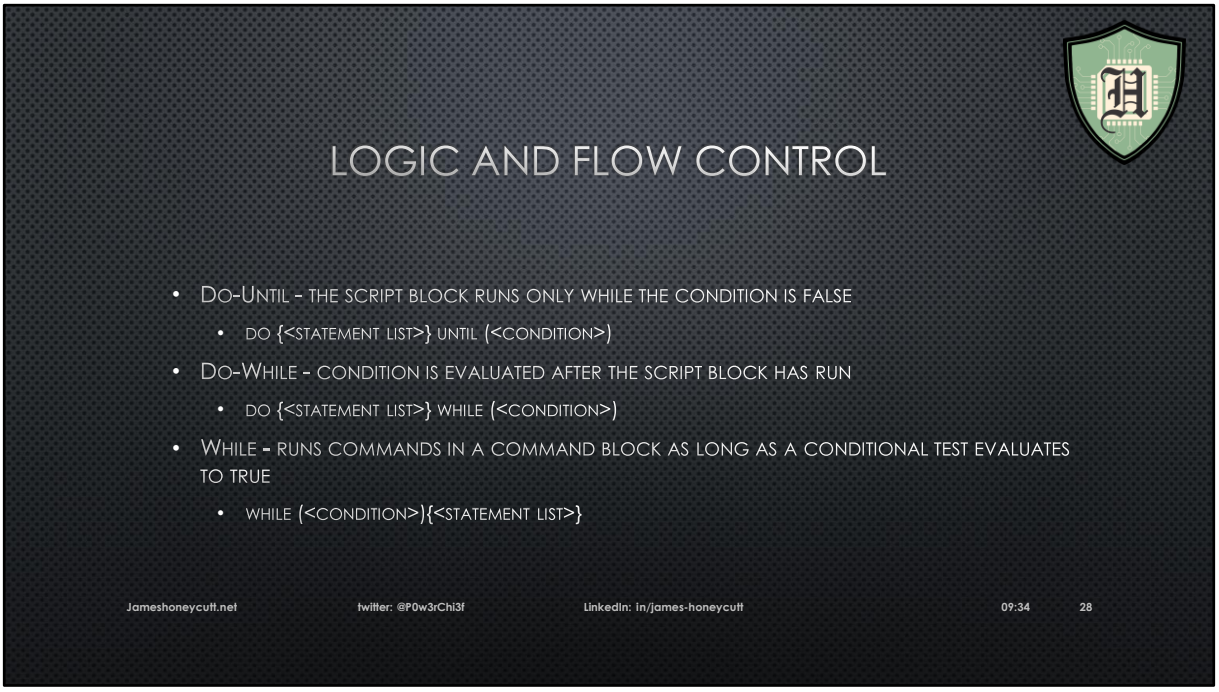
code C:\Scripts\PowerShell\Vmware\ViewVM-SetState.ps1



ForEach
$letterArray = "a","b","c","d“
foreach ($letter in $letterArray) {Write-Host $letter}

code “C:\Scripts\PowerShell\UserAccounts\Uploaded to GitHub\Get-InactiveUsers.ps1”

For
for($i=1; $i -le 100; $i++){Write-Host $i}



Do-Until

code C:\Scripts\PowerShell\UserAccounts\lockaccounts.ps1

Do-While

code C:\Scripts\PowerShell_InProgress\Generate-Files.ps1

While

code C:\Scripts\PowerShell\UserAccounts\Invoke-RandomUser.ps1

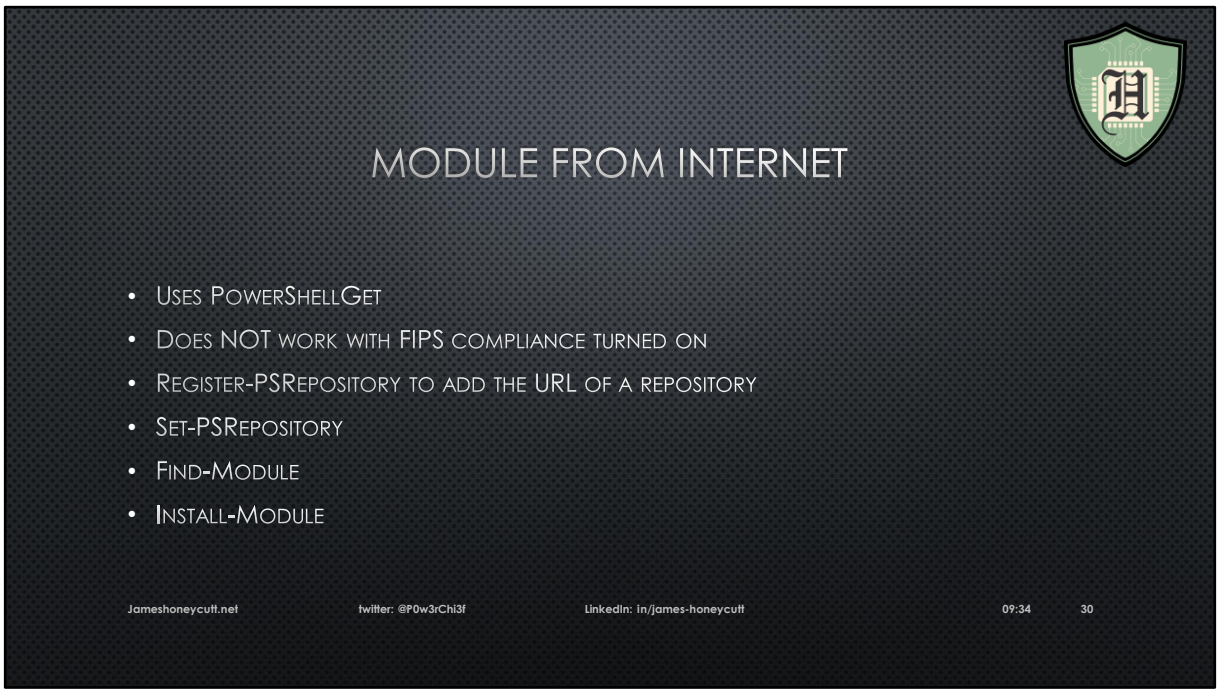
Snapins
Code “C:\Scripts\PowerShell\VMware\Uploaded to GitHub\Refresh-availableVMs.ps1”

Modules
Code “C:\Scripts\PowerShell\VMware\Uploaded to GitHub\New-ADUser_Prompted.ps1”
Code “C:\Scripts\PowerShell\VMware\Uploaded to GitHub\Export-OVA.ps1”

Get-module –listavailable
Get-command *host*
Get-help get-host
Get-help get-vmhost

$env:psmodulepath

get-module -ListAvailable VM* | foreach ($Module -in $_.name) { Uninstall-Module -Name $Module}

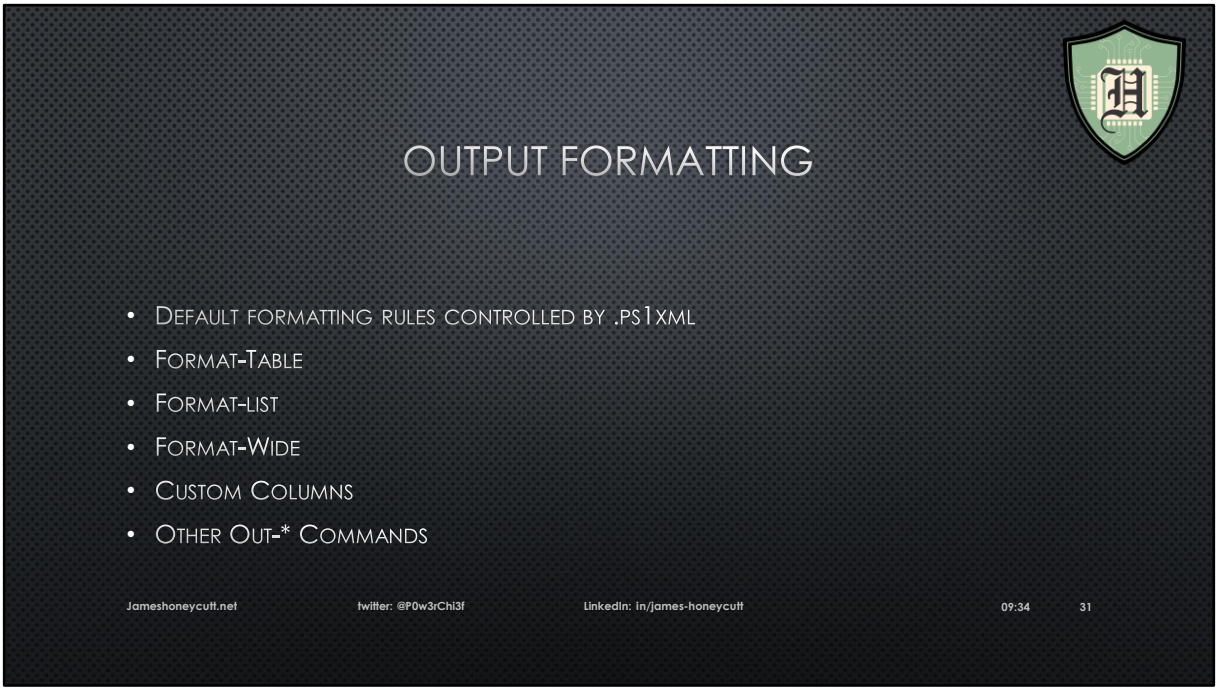


get-command *sql*
Find-module *sql*
Install-module SqlServer

get-command *sql*
get-command -module SqlServer

Set-PSRepository –name “MyLocalStore”

Profile Scripts

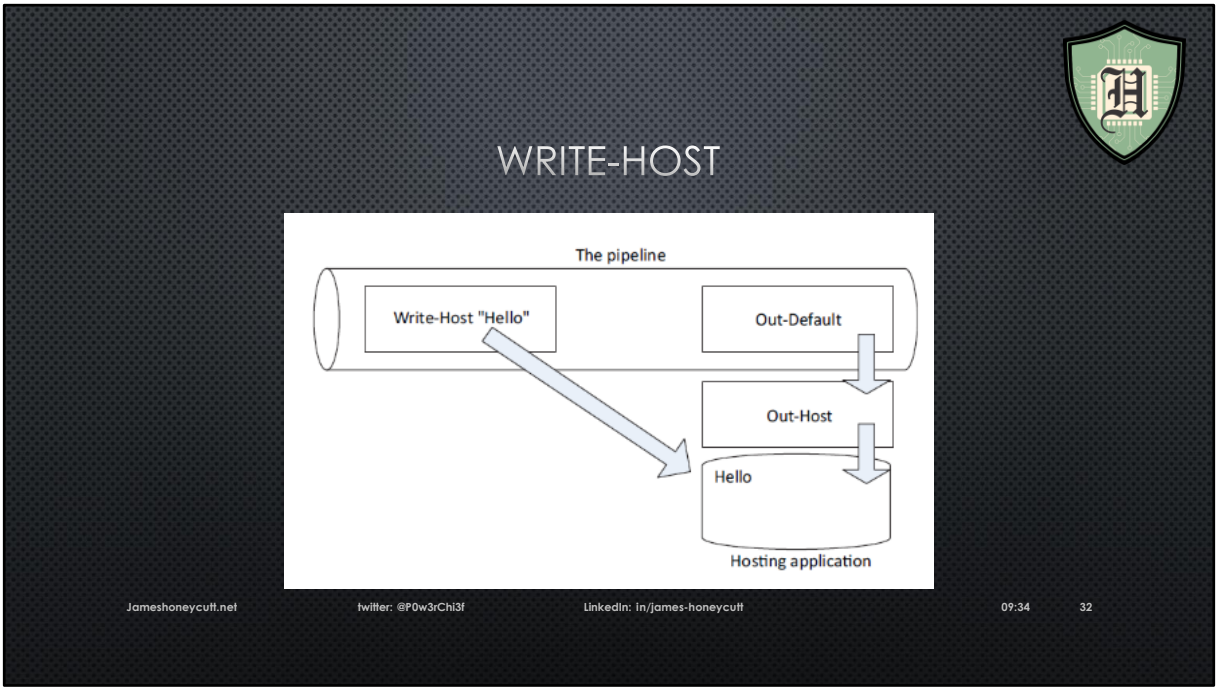


Third formatting rule
i) Four or less properties; format as table
ii) Five or more properties; format as list

Format-Table
-autosize (adjusts to column width)
(1) Get-Process | select -last 15 | Format-Table
ii) -property (instead of piping to select)
(1) Get-Process | select -last 15 | Format-Table -Property MachineName, ID, ProcessName, responding
iii) –groupBy
(1) Get-Service | Sort-Object Status | Format-Table -groupBy Status
iv) –wrap
(1) Get-Service | Format-Table Name,Status,DisplayName -autoSize –wrap

Format-Wide (wide list)
i) Grabs “name” property and creates a 2 column list
ii) -column to change the # of columns
iii) -property select other property instead of name property

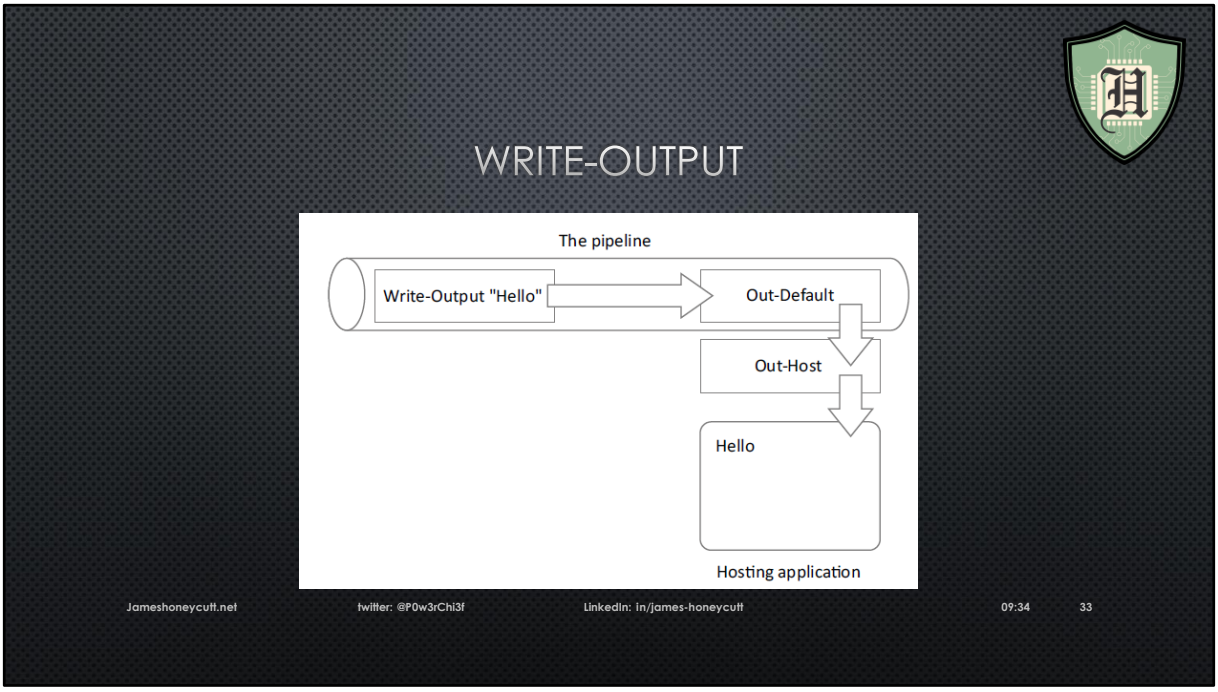
```
Format-list
   -property
      Get-Process | Format-list -Property Name,ID
   -groupBy
      Get-Process | Format-list -Property Name,ID -GroupBy name
   -others

get-command out-*
Get-Service | Group-Object -Property Status | Sort-Object -Property Count -Descending |
ConvertTo-Json | Out-File .\Services.json

Custom Columns
   Get-Process | select -first 5 | Format-Table -Property name,vm
   Get-Process | select -first 5 | Format-Table Name,  @{name='VM(MB)';expression={$_.VM
/ 1MB -as [int]}} –autosize

Code “C:\Scripts\PowerShell\Server\Uploaded to GitHub\Get-ServerRebootStatus.ps1”

$userData = Import-Csv 'C:\Users\honey\Google Drive\Presentations\PowerShell\PowerShell
Crash Course\UserData.csv'
$userData | select -First 5 | Format-Table -property ID,
@{name='FName';expression={$_.first_name}},
@{name='LName';expression={$_.last_name}}, email, gender
$userData | select ID, @{name='FName';expression={$_.first_name}},
@{name='LName';expression={$_.last_name}}, email, gender | export-csv
'C:\Users\honey\Google Drive\Presentations\PowerShell\PowerShell Crash Course\temp.csv'
-NoTypeInformation

get-command out-*
```



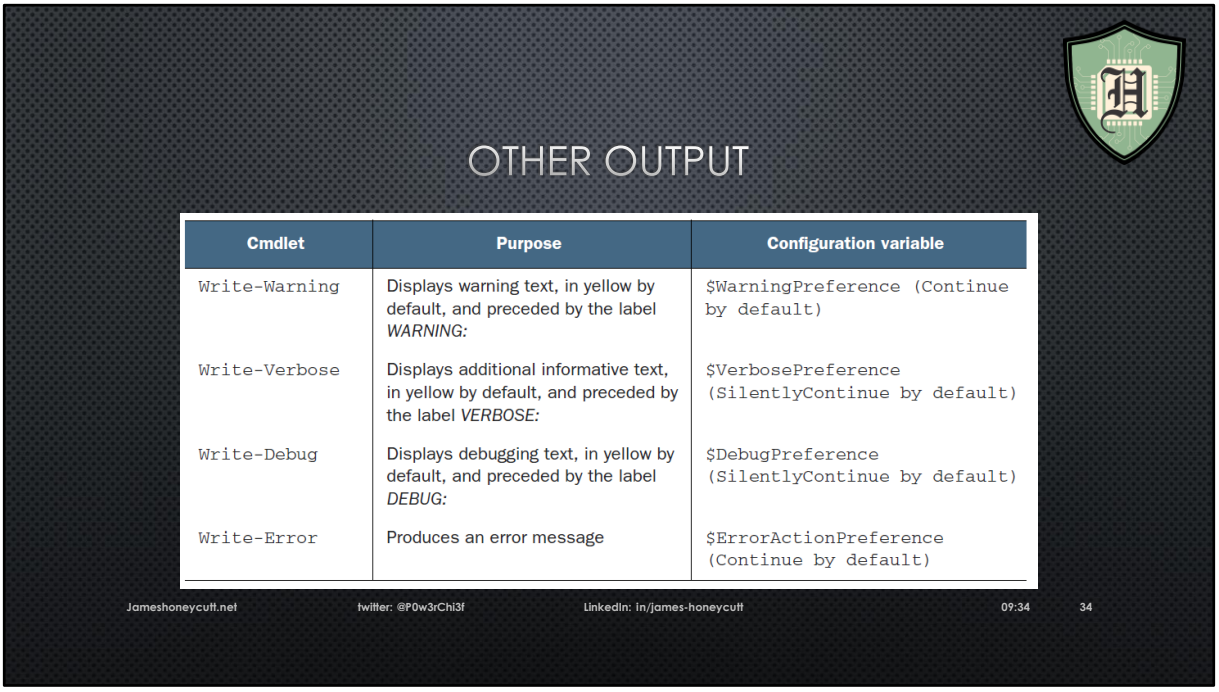
write-host "COLORFUL!" -fore yellow -back magenta

Anything written to the screen (-host) cannot be captured.
i.e unattended script or remote commands (invoke)

Best practice is to use Write-Verbose
Used for “warm and fuzzy messages”
Connecting to xyz server
Testing connecton



Can send objects into the pipeline
Write-output Hello
write-output "Hello" | where-object { $_.length -gt 3 } | out-default | write-host



Write-Information (v5)

Write-host is a wrapper

May need -informationaction continue

Write-Progress

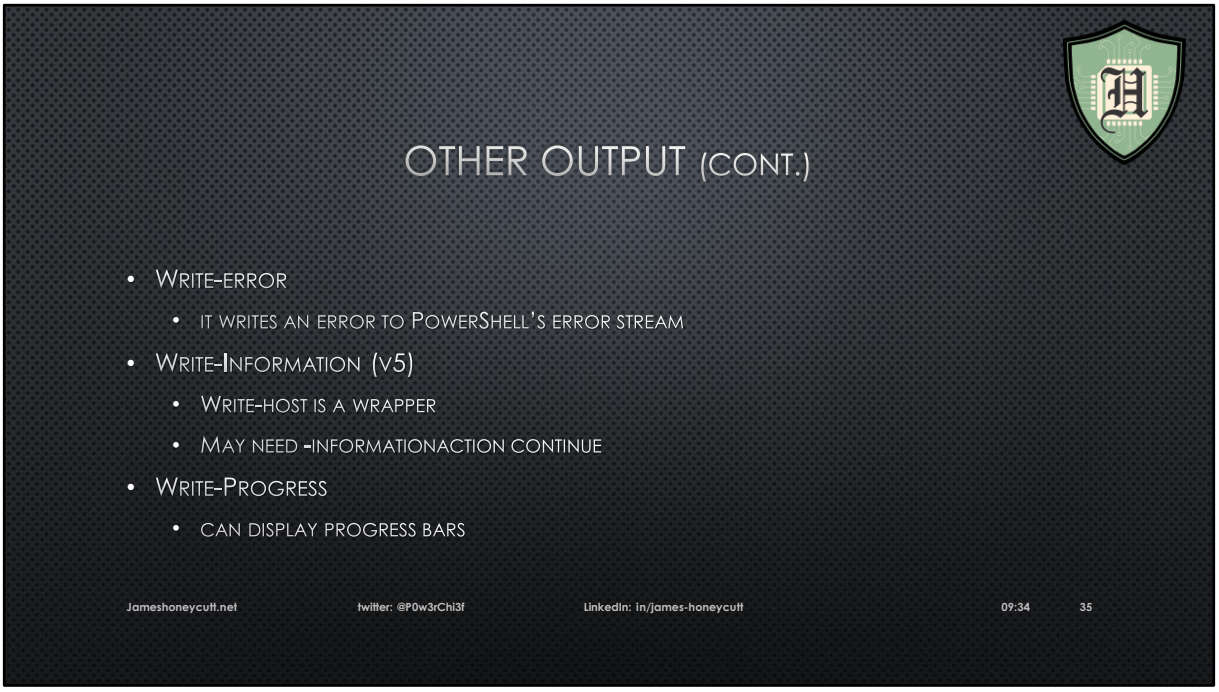
can display progress bars

Write-verbose

Write-Verbose "Test Message" –Verbose

Write-Information

Get-ChildItem C:\Users\honey\Documents*.pdf; Write-Information -MessageData "Here are your PDFs!" -InformationAction Continue

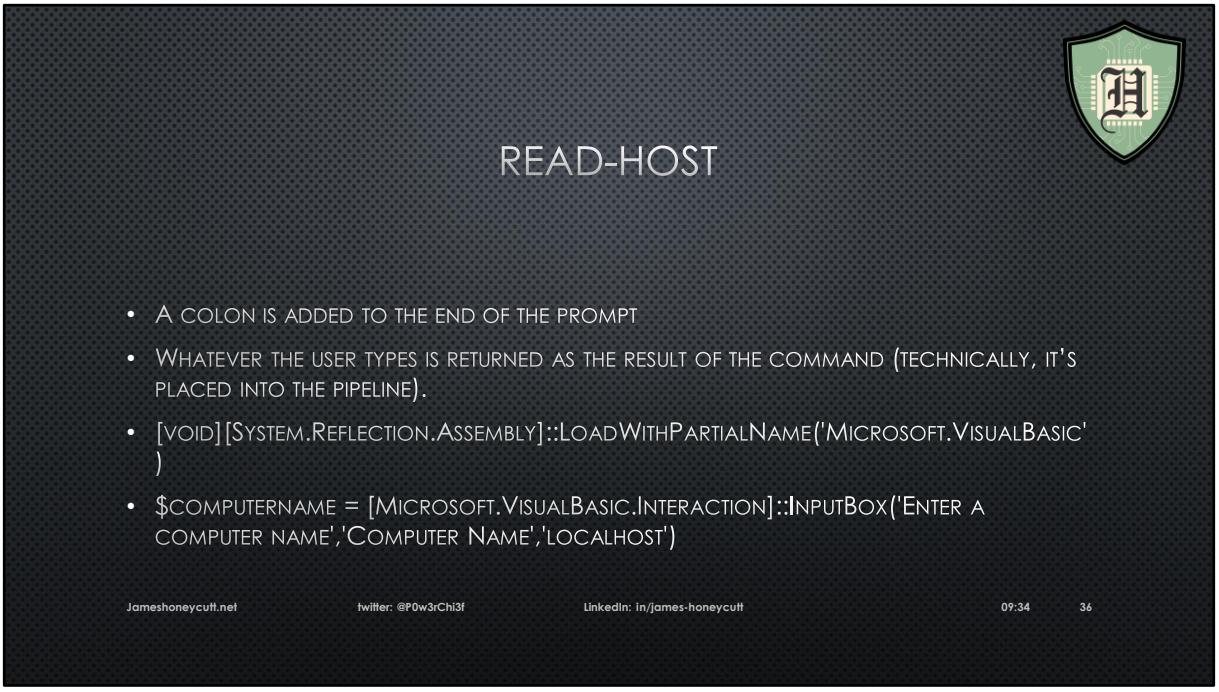


$error

Write-error
it writes an error to PowerShell's error stream

Write-Information (v5)
Write-host is a wrapper
May need -informationaction continue
Get-ChildItem C:\Users\honey\Documents*.pdf; Write-Information -MessageData "Here are your PDFs!" -InformationAction Continue

Write-Progress
can display progress bars



read-host "What is your name?“
Code “C:\Scripts\PowerShell\UserAccounts\Uploaded to GitHub\New-ADUser_Prompted.ps1”

[void][System.Reflection.Assembly]::LoadWithPartialName('Microsoft.VisualBasic’)

[void] - part is converting the result of the command into the void data type.

an object that does not have a value of any **type** data type is a special type that means “throw the result away.”

Another way to do the same thing would be to pipe the result to Out-Null.

[System.Reflection.Assembly] – represents our application

enclosed the type name in square brackets, as if we were declaring a variable to be of that type

we’re using two colons to access a static method of the type

Static methods exist without us having to create an instance of the type.

LoadWithPartialName ()

static method we’re using

accepts the name of the framework component we want to load.

```
$computername = [Microsoft.VisualBasic.Interaction]::InputBox('Enter a computer
name','Computer Name','localhost’)
```

[Microsoft.VisualBasic.Interaction]
: Loaded into memory with the previous command

'Enter a computer name’
: The first parameter is the text for your prompt.

'Computer Name’
: The second parameter is the title for the prompt’s dialog box.

'localhost’
: The default value that you want prefilled in the input box.

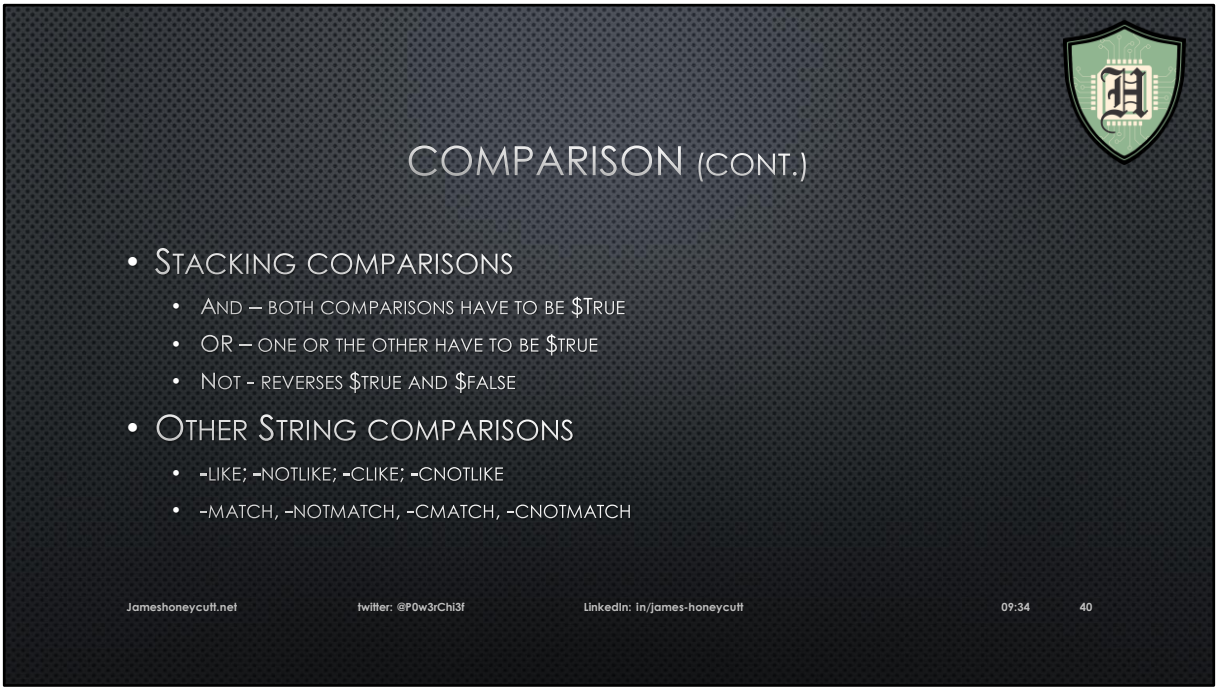

TIME FOR
LUNCH
TIME FOR

FILTERING

- Filter as far left as possible (-filter)
  - Each cmdlet has its own way to filter
  - Get-service can filter -name only
  - Get-adcomputer can filter on any attribute
- Filtering out of pipeline
  - Where-object is used when there is no -filter or you cannot filter on the property you want

COMPARISON

- Comparison Operators
  - -eq (=) = Equal
  - -ne (<>)= Not Equal
  - -ge (>=)and -le (<=) = Greater than or equal to; less than or equal to
  - -gt (>) and -lt (<) = Greater than; less than
  - -ceq, -cne, -cgt, -clt, -cge, -cle = case sensitive



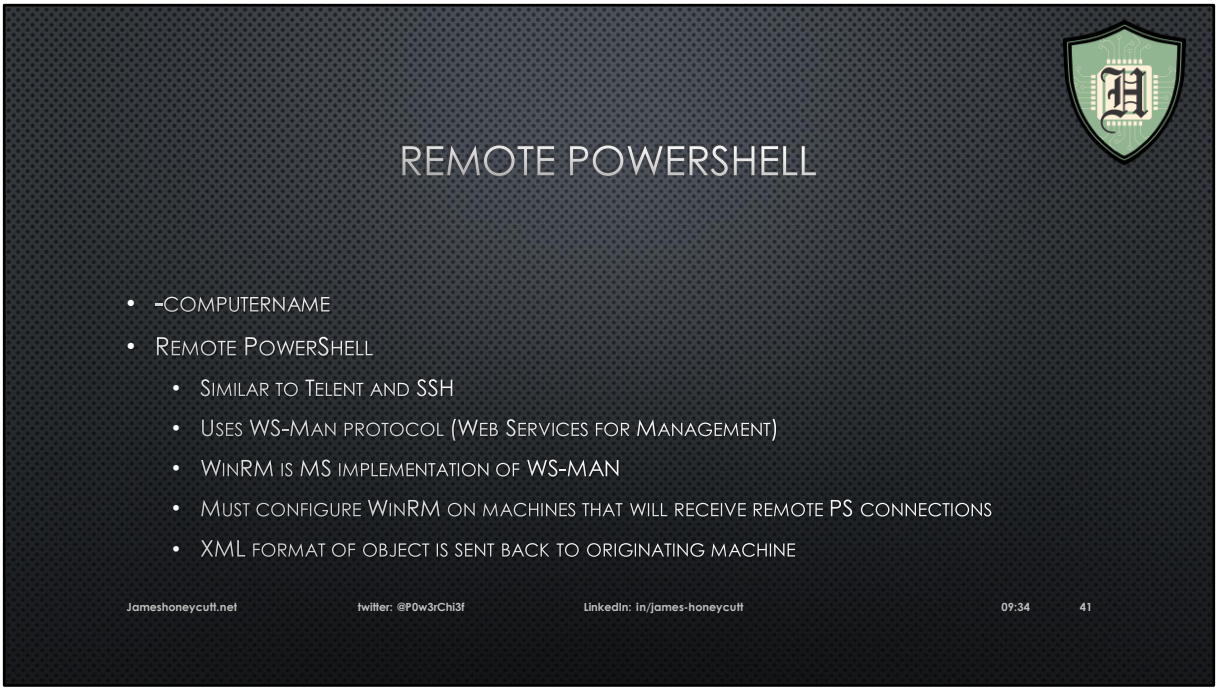
get-service | where-object {$_.status -eq 'running' }

Get-Service | Where Status -eq 'Running'
Simple; new in v3; good when only comparing 1 object

get-service | where-object {$_.status -eq 'running' -AND $_.StartType -eq 'Manual'}
Original syntax; used for multi compare

Get-Process | Where-Object -filter { $_.Name -notlike 'powershell*' } | Sort VM -descending | Select -first 10 | Measure-Object -property VM –sum

Code "C:\Scripts\PowerShell\UserAccounts\Uploaded to GitHub\New-ADUser_Prompted.ps1"



Remote PowerShell

Similar to Telent and SSH

Uses WS-Man protocol (Web Services for Management)

Over http (5985) and https (5986)

WSMan: drive manages remote sessions

Get-ChildItem WSMan:\localhost\Shell

IdleTimeout - specifies the amount of time a session can be idle before it's shut down automatically

MaxConcurrentUsers - specifies the number of users who can have a session open at once

MaxShellRunTime - determines the maximum amount of time a session can be open.

MaxShellsPerUser - sets a limit on the number of sessions a single user can have open at once

Get-ChildItem WSMan:\localhost\Service\

MaxConnections - sets the upper limit on incoming connections to the entire remoting infrastructure.

WinRM is MS implementation of WS-MAN

Installed and enabled by default on Servers 2012r2 and up

Installed and disabled by default on Win7 and up

XML format of object is sent back to originating machine

Serialization – to XML format

Deserialization – from XML format

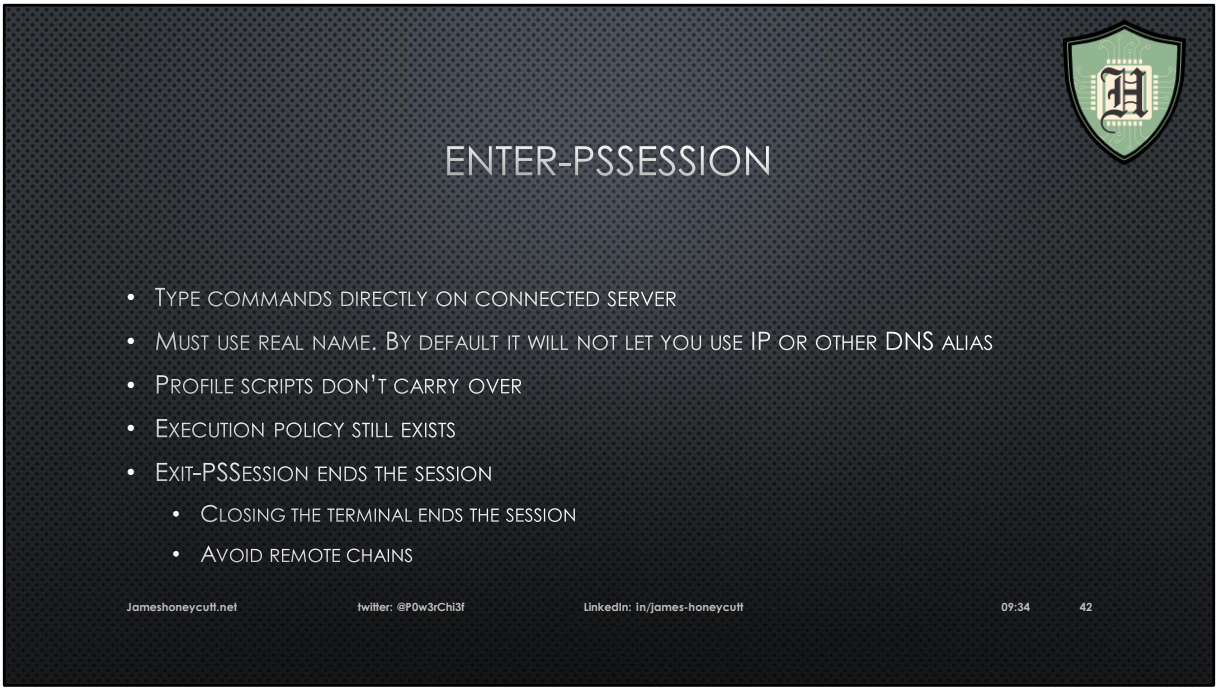
Only a snapshot

Must configure WinRM on machines that will receive remote PS connections

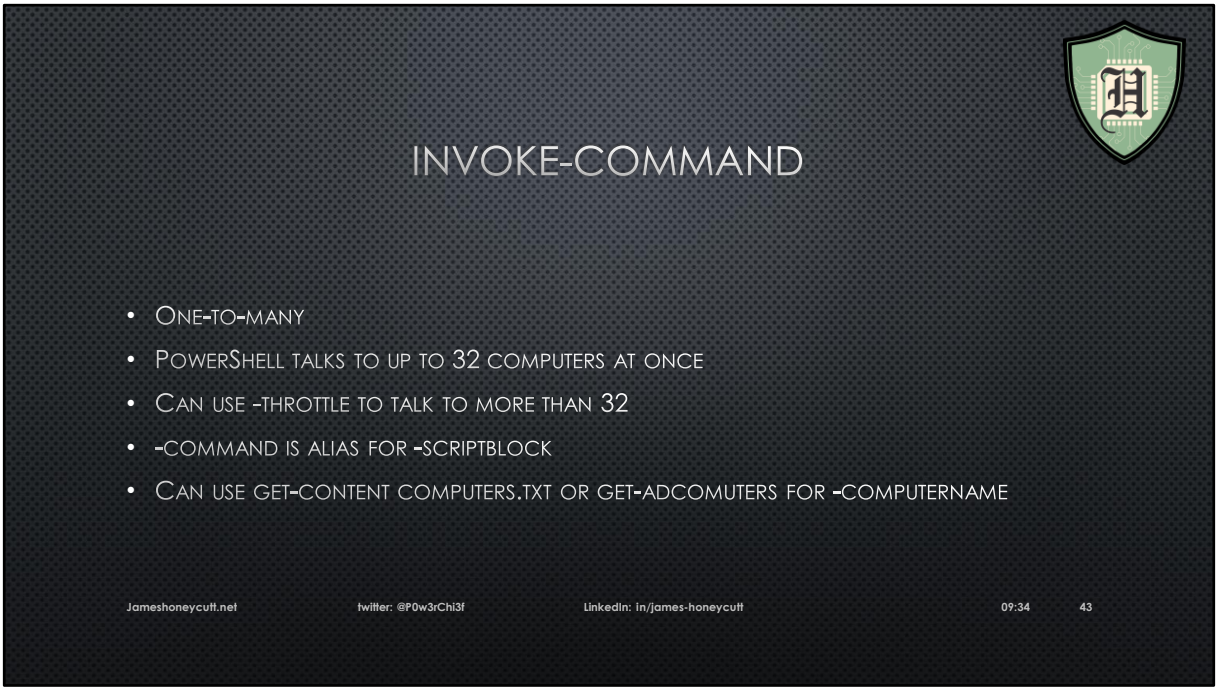
Enable-PSRemoting (call Set-WSManQuickConfig; starts and sets WinRM service to autostart, Registers PowerShell as an endpoint; creates needed firewall rules.

Any adapter set to Public can't have Windows Firewall exceptions

Test-WSMan
Get-PSSessionConfiguration
Enable-PSRemoting -SkipNetworkProfileCheck -Verbose



enter-pssession localhost
enter-pssession DESKTOP-52INA1A -Credential desktop-52ina1a\honey

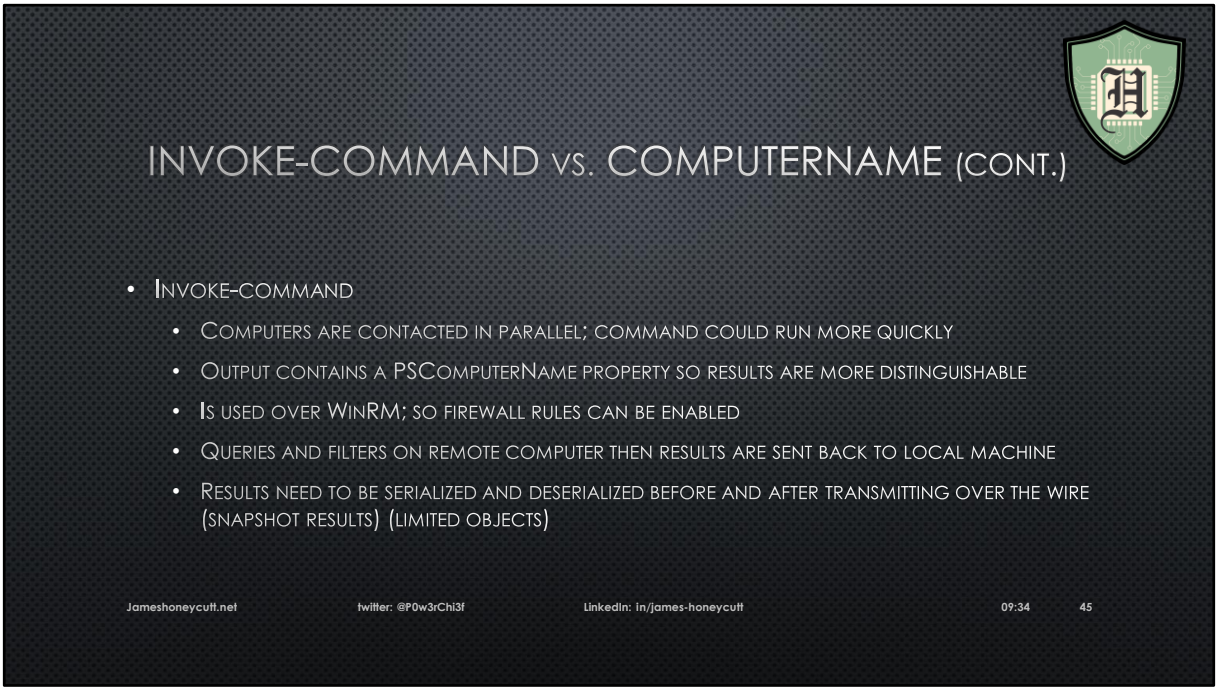


Invoke-Command -computerName localhost, DESKTOP-52INA1A -Credential $creds -command { Get-EventLog Security | where {$_.eventID -eq 4826}}

Code “C:\Scripts\PowerShell\Server\Uploaded to GitHub\Get-ServerRebootStatus.ps1”

INVOKE-COMMAND vs. COMPUTERNAME

- ComputerName
  - Computers are contacted sequentially and could take longer
  - Does not contain a PSComputerName property so results my be hard to separate
  - Connection is not made with WinRM
  - Processing is done on local computer; so all records are brought across the wire then filtered
  - Results are live and don't need to be serialized or deserialized (fully functional Objects)



get-service XblGameSave | start-service

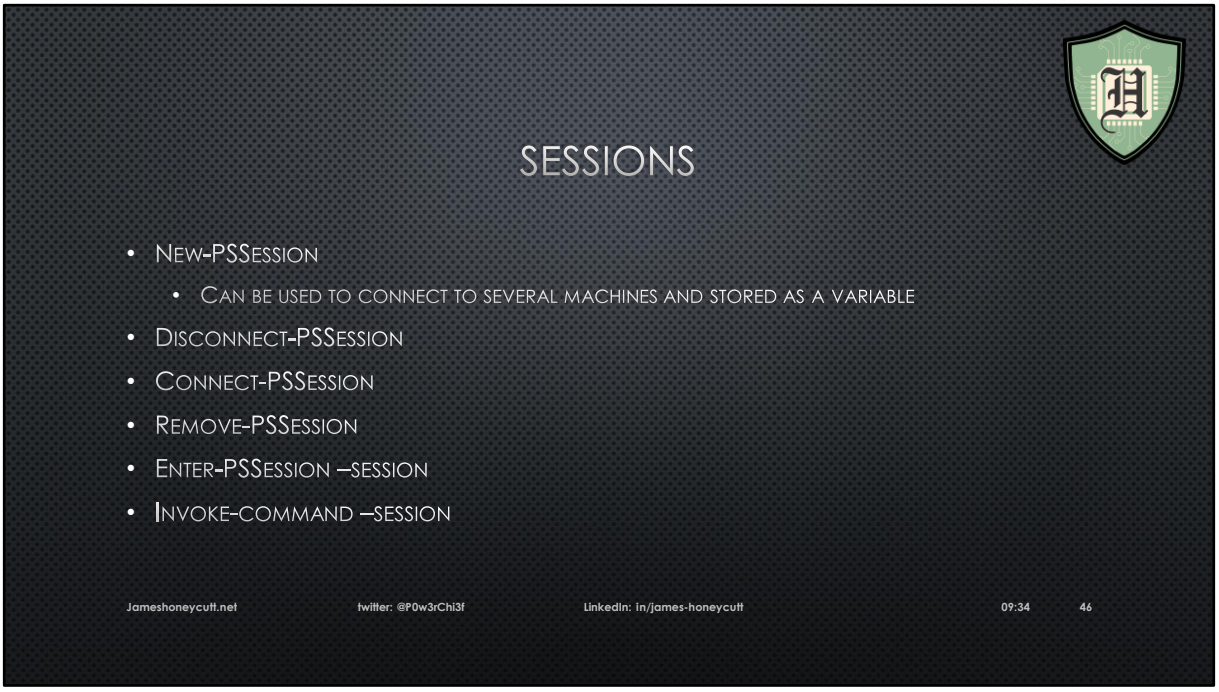
Invoke-Command -computerName localhost, DESKTOP-52INA1A -Credential desktop-52ina1a\honey -command {get-process –name notepad | stop-process}

Get-process -ComputerName desktop-52ina1a -Name notepad

Decentralized objects (demo

Get-service | get-member

Invoke-command -computername DESKTOP-52INA1A -Credential desktop-52ina1a\honey -scriptblock {get-service} | get-member



$sessions = New-PSSession -ComputerName localhost, DESKTOP-52INA1A -Credential desktop-52ina1a\honey

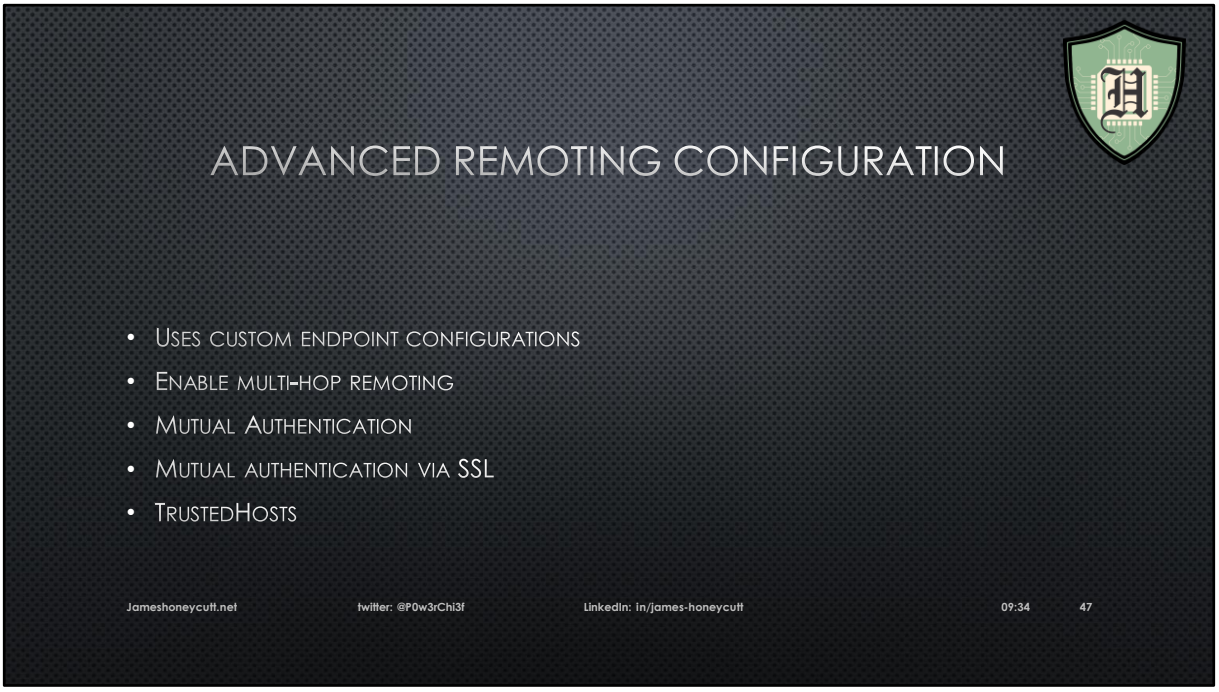
Disconnect-PSSession
Disconnects from the session, but leaves the connection
Same domain admin can see the connection of a different computer
Admin creates session on Comp1 to Comp2, then disconnects session, logs into Comp3 and checks sessions on Comp2, sees his disconnected session
Connect-PSSession
Remove-PSSession

Enter-PSSession -Session $sessions [0]
Enter-PSSession -Session ($sessions |where { $_.computername -eq 'localhost' })
Enter-PSSession -Session (Get-PSSession | where {$_.computername -eq 'localhost'})

$s_server1,$s_server2 = new-pssession -computer localhost, DESKTOP-52INA1A -Credential desktop-52ina1a\honey

Invoke-Command -Command { Get-WmiObject -Class win32_process } -Session $sessions | Format-Table -Property PSComputerName, processname, ProcessID, ParentProcessID

```
invoke-command -command { get-wmiobject -class win32_process } -session $sessions | Select-Object ProcessName, PSComputerName, Path | Group-Object ProcessName | Sort-Object Count -Descending | Format-Table -AutoSize
```



Advanced remoting configuration

Uses custom endpoint configurations (reference my JEA Talk)

Enabling multihop remoting

Second Hop Problem

Enable-WSManCredSSP -Role Client -DelegateComputer $computer (2nd Hop Computer)

Enable-WSManCredSSP -Role Server (ran on middle man computer

Digging deeper into remoting authentication

PowerShell remoting employs mutual authentication

Mostly take care of in a domain environment

The name must resolve to an IP address.

The name must match the computer's name in the directory.

Mutual authentication via SSL

you need to obtain an SSL digital certificate for the destination machine

you need to create an HTTPS listener

https://leanpub.com/secretsofpowershellremoting

TrustedHosts

Shuts off Mutual Authentication

The TrustedHosts item can contain a comma-separated list of computer names, IP addresses, and fully-qualified domain names.

Wildcards are permitted.

Get-Item wsman:\localhost\Client\TrustedHosts

Set-Item wsman:localhost\client\trustedhosts -Value *

Code “C:\Scripts\PowerShell\UserAccounts\RemoveUser_Prompted.ps1”

PowerShell creates a temporary local module with shortcuts to the commands on the remote server
Results brought back through the session are decentralized and do not have methods

ROOT\CIMV2\Win32_LogicalDisk

Get-WmiObject -Namespace root\CIMv2 -list | where {$_.name -like '*dis*'}
Get-WmiObject -class win32_desktop -filter "name='DESKTOP-52INA1A\\honey'"

CIM_ Class are often base classes and access directly
Communicates over RPC – If firewall supports stateful inspection

Win32_ are Windows specific
Communicates over WS-MAN (WinRM)

COMMON INFORMATION MODEL (CIM)

- SIMILAR TO GET-WMIOBJECT
- CIM CMDLETS ARE WRAPPERS FOR WMI COMMANDS
- -CLASSNAME INSTEAD OF -CLASS
- -LIST NOT AVAILABLE; MUST USER -NAMESPACE INSTEAD
- -CREDENTIAL NOT AVAILABLE; MUST USE INVOKE-COMMAND

Start-job –scriptblock
Local job has support -computername
Requires remote powershell scripting

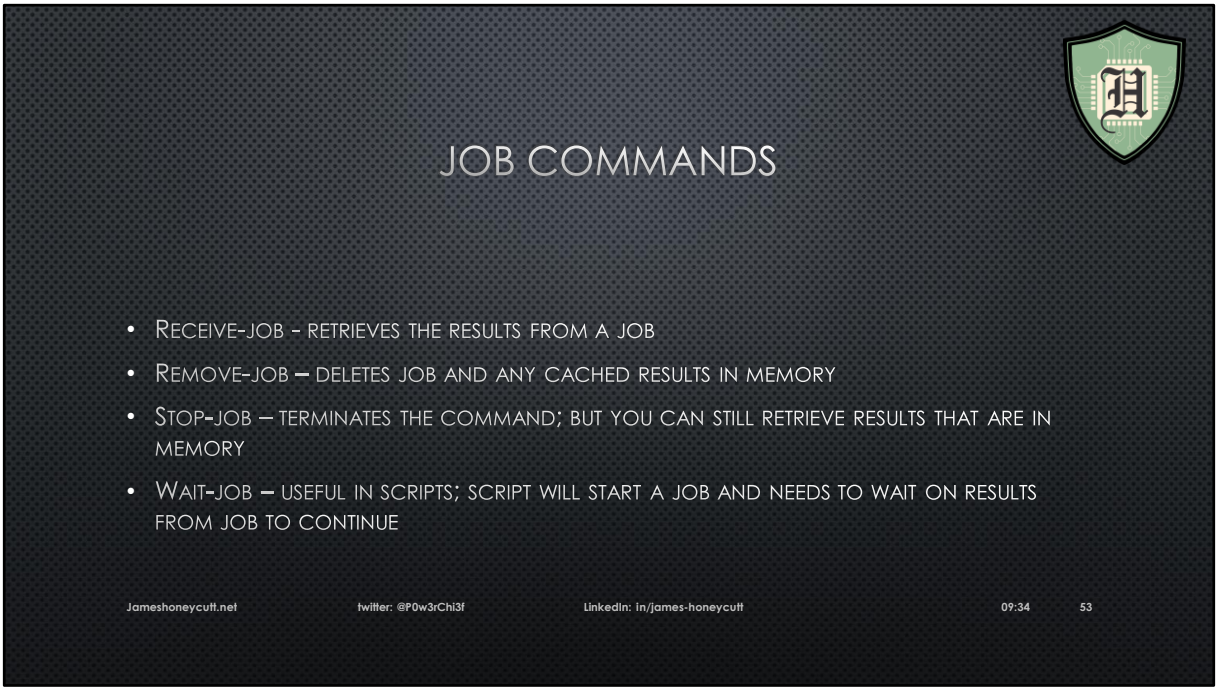
start-job -ScriptBlock {Get-Service}
start-job -scriptblock {get-eventlog security -computer localhost, DESKTOP-52INA1A}

WMI as a Job
Used -asjob parameter
Creates a child job for every computer in the list
Uses normal WMI communications
Get-ciminstance requires start-job or invoke-command with get-ciminstance in scriptblock

start-job -scriptblock {get-eventlog security -computer localhost, DESKTOP-52INA1A } -Credential $creds

Remoting as a job
Used invoke-command –asjob
Requires PSv2 or higher with remoting enabled

Has -jobname parameter



Receive-job

By name or job ID

Receive-jobs clears them out of cache and cannot be retrieved a second time; must use -keep or out-cliXML

Results are decentralized; can be piped into export-xml, sort-object, format-list, convertto-html; out-file

Has more data property

Will be false if there is no data in memory (if you viewed and did not use the -keep parameter)

Will be true if there is data in memory

Get-jobs | receive-jobs

Code C:\Scripts\PowerShell_InProgress\Create-EvilTask.ps1

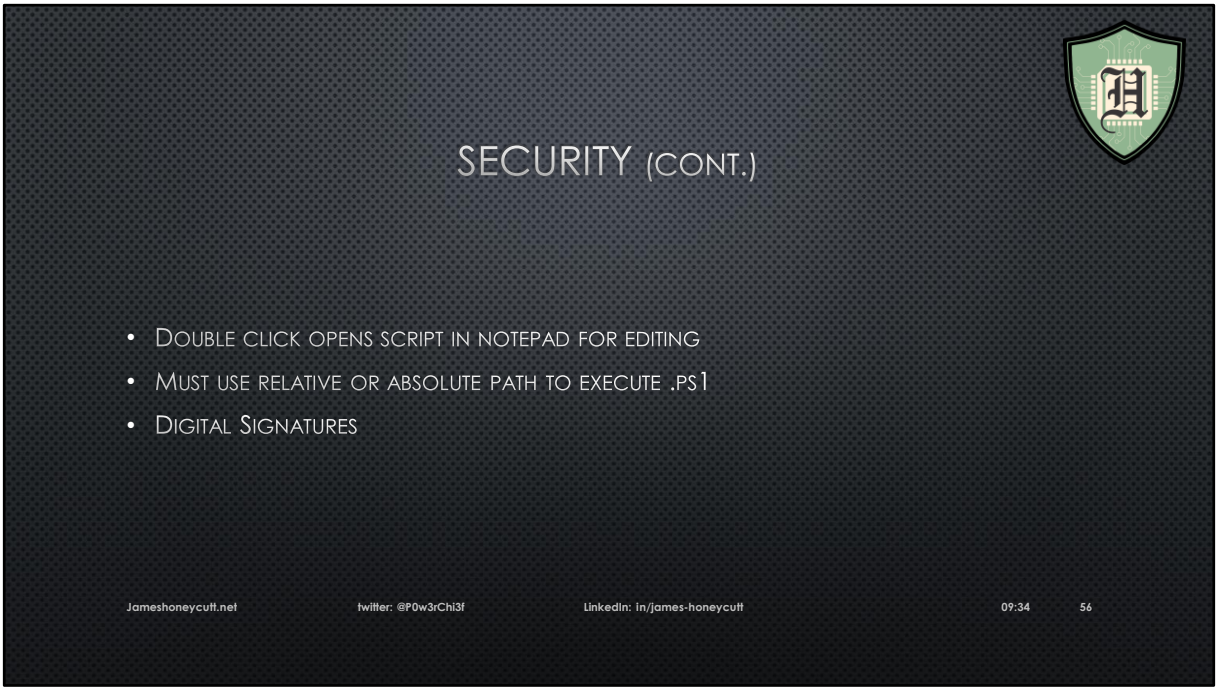
Scheduled task cannot produce output

Scheduled Jobs are a hybrid of background jobs and scheduled tasks

job and scheduled job cmdlets, you can use the Task Scheduler UI and scheduled task cmdlets to manage scheduled jobs, but you can't use the job or scheduled job cmdlets to manage scheduled tasks.

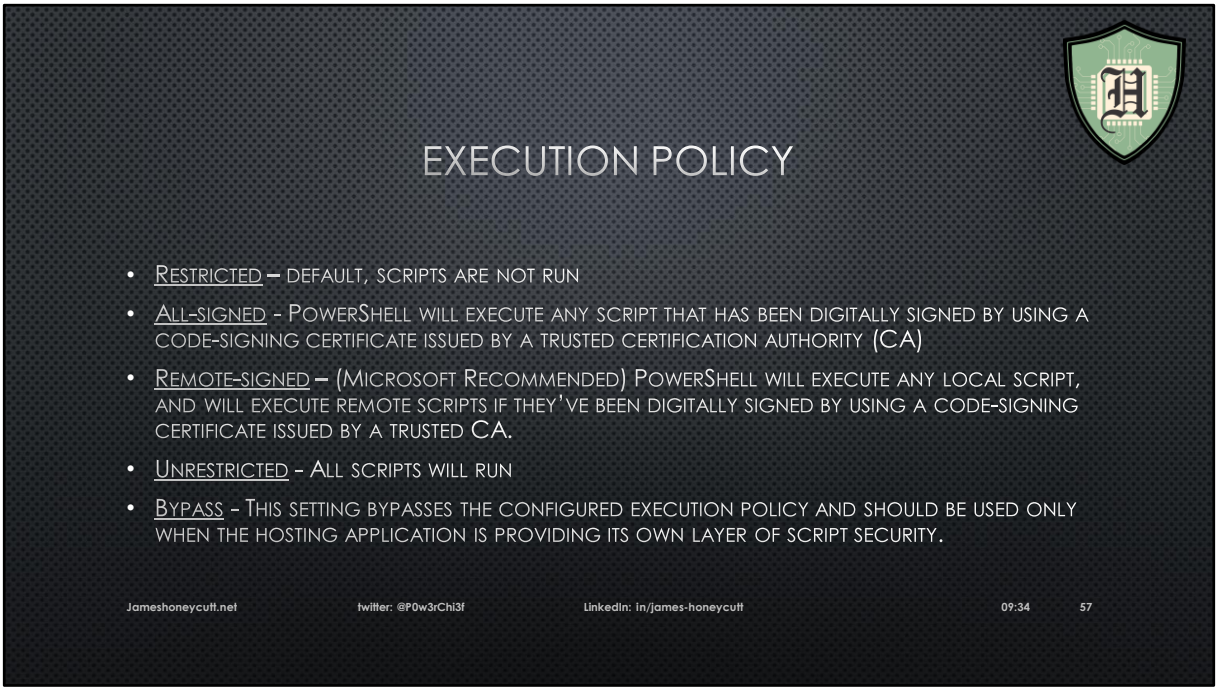
SECURITY

- PowerShell does give any additional permissions
- If you cant do it in the GUI then you cant do it in a script
- Does not defend against malware
- "Even though apiece of malware might use PowerShell to do harm, that doesn't make that malware PowerShell's problem"



Must use relative or absolute path to execute .ps1
Protects from command hijacking

Digital Signatures
Set-AuthenticodeSignature "C:\Scripts\Publish\PowerShell\Files\Backup-Files.ps1" (get-childitem Cert:\CurrentUser\My -CodeSigningcert)

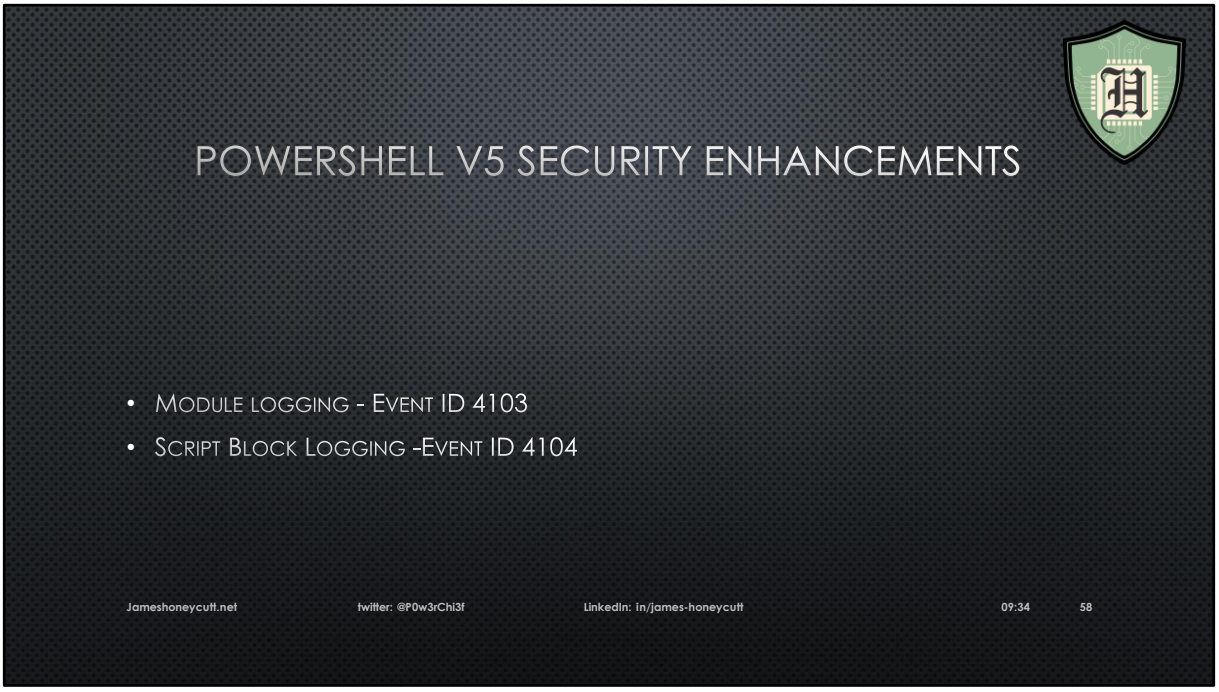


Changed by:
Set-executionpolicy
Must have rights to change HKEY_LOCAL_MACHINE of the registry

GPO
Configuration > Policies > Administrative Templates > Windows Components >Windows PowerShell

Powershell.exe – executionpolicy (overwrites any local or group policy)

Digital Signatures
Set-AuthenticodeSignature "C:\Scripts\Publish\PowerShell\Files\Backup-Files.ps1" (get-childitem Cert:\CurrentUser\My -CodeSigningcert)



Module logging - Event ID 4103

Logs PowerShell pipeline execution details during execution including variable initialization, and command invocation

Able to record some de-obfuscated scripts, and also some output data

Available in v3

Get-WinEvent -LogName application | where {$_.ID -eq 4104}

Script Block Logging -Event ID 4104

Logs and records all blocks of PowerShell code as they are executing

Captures all de-obfuscated code

Available in v5

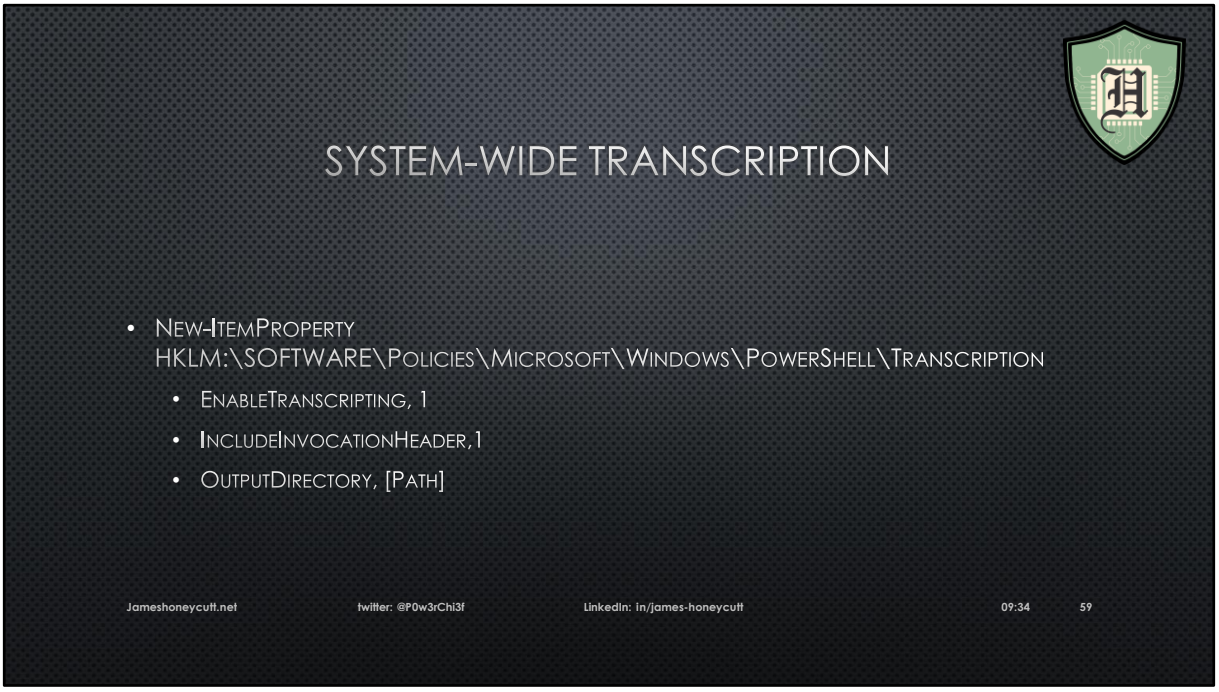
HKLM:\Software\Policies\Microsoft\Windows\PowerShell\ScriptBlockLogging

Create Registry key

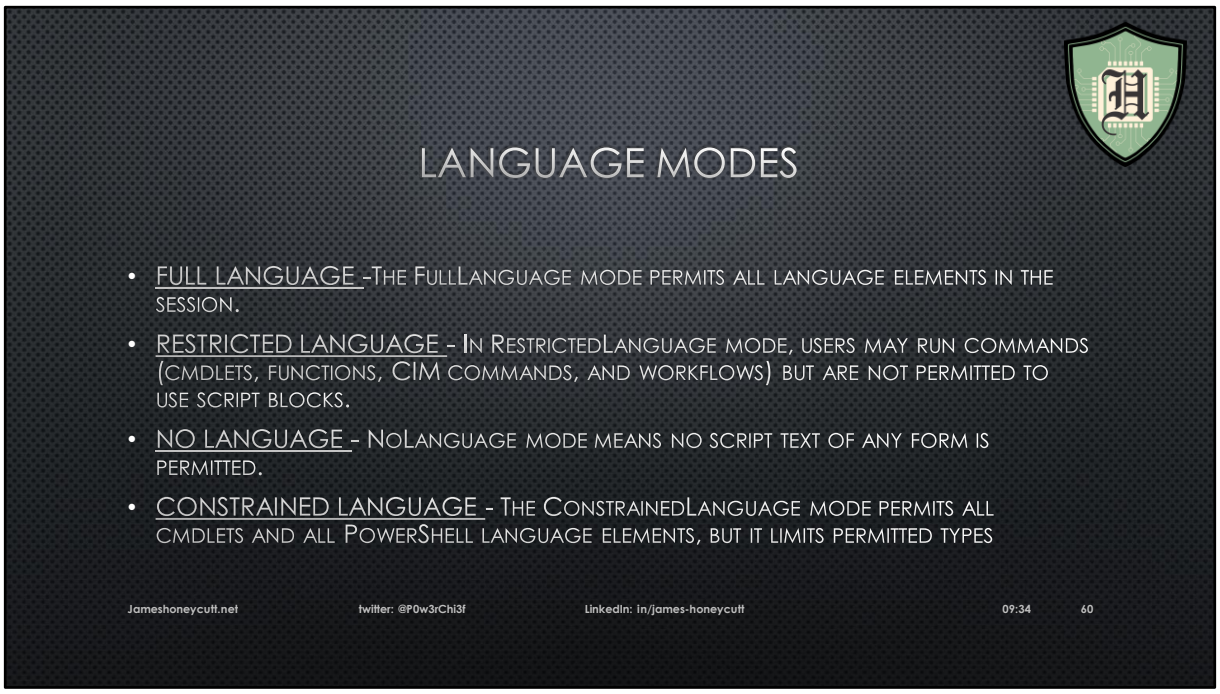
New-Item HKLM:\SOFTWARE\Policies\Microsoft\Windows\PowerShell\ScriptBlockLogging –Force

New-ItemProperty HKLM:\SOFTWARE\Policies\Microsoft\Windows\PowerShell\ScriptBlockLogging -Name "EnableScriptBlockLogging" -PropertyType "DWORD" -Value 1

```
Get-WinEvent -LogName Microsoft-Windows-PowerShell/Operational | where {$_.Id -eq
"4103"} | select -first 5 | format-list
```




```
New-ItemProperty
HKLM:\SOFTWARE\Policies\Microsoft\Windows\PowerShell\Transcription -Name
"EnableTranscripting" -PropertyType "DWORD" -Value 1
New-ItemProperty
HKLM:\SOFTWARE\Policies\Microsoft\Windows\PowerShell\Transcription -Name
"IncludeInvocationHeader" -PropertyType "DWORD" -Value 1
New-ItemProperty
HKLM:\SOFTWARE\Policies\Microsoft\Windows\PowerShell\Transcription -Name
"OutputDirectory" -PropertyType "DWORD" -Value 1
```



Disable-WindowsOptionalFeature -Online -FeatureName MicrosoftWindowsPowerShellV2
$ExecutionContext.SessionState.LanguageMode

Code C:\Scripts\Publish\PowerShell\Files\Backup-Files.ps1

SCRIPTING (CONT.)

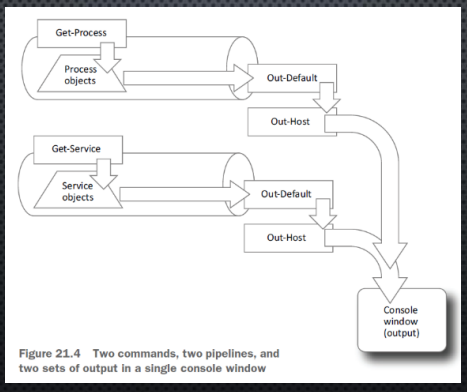


Figure 21.4 Two commands, two pipelines, and two sets of output in a single console window

Get-Process
Get-Service

SCRIPTING (CONT.)

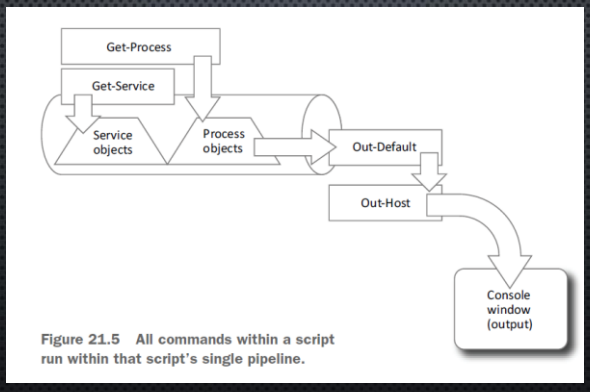
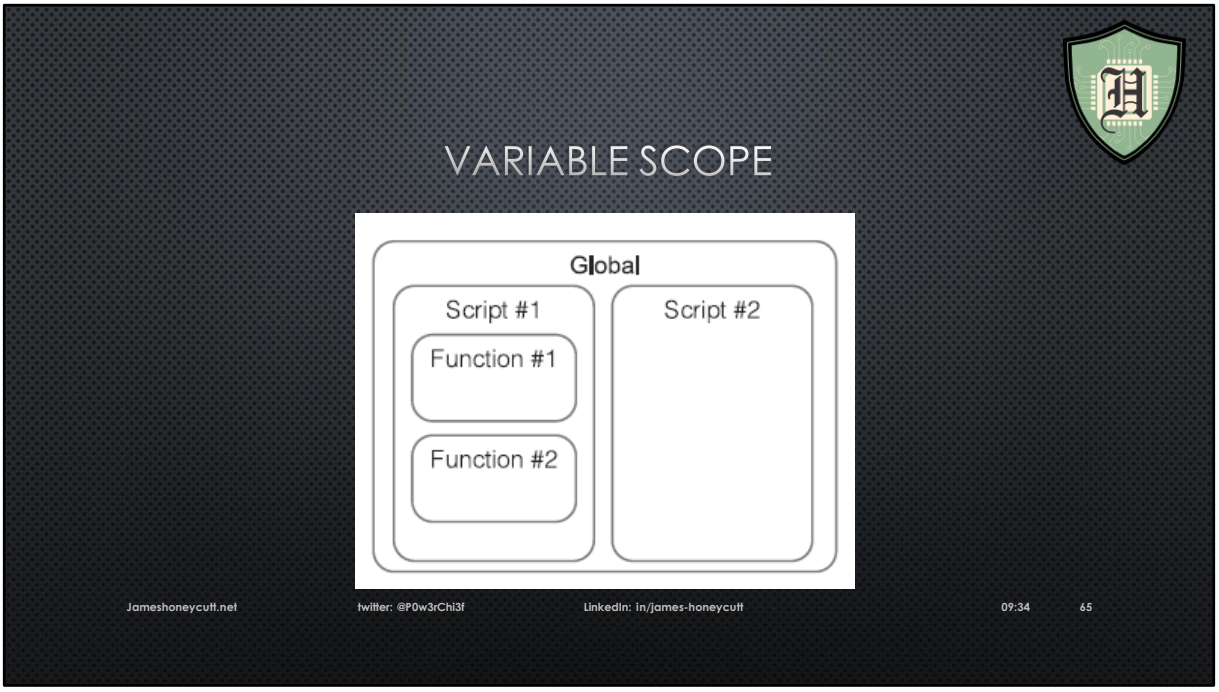


Figure 21.5 All commands within a script run within that script's single pipeline.

Ise "C:\Users\honey\Google Drive\Presentations\PowerShell\PowerShell Crash Course\Scripting-Demo.ps1"
Ise "C:\Users\honey\Google Drive\Presentations\PowerShell\PowerShell Crash Course\Scope.ps1"



Ise “C:\Users\honey\Google Drive\Presentations\PowerShell\PowerShell Crash Course\Scope.ps1”

function x {$x + (.\scope.ps1)}

PARAMETERS

- Turn variables into parameters
- [CmdletBinding()]
- [Parameter(Mandatory=$True,HelpMessage="Enter a computer name to query")
- [Mandatory], [String], [Int], [alias('hostname')]
- [ValidateSet(2,3)]
- help about_functions_advanced_parameters

Ise "C:\Users\honey\Google Drive\Presentations\PowerShell\PowerShell Crash Course\Get-DiskInventory.ps1"

Code "C:\Scripts\Publish\PowerShell\Files\Backup-Files.ps1"

Regex

Used with -match and -cmatch (case sensitive)

Get-WinEvent -Path "C:\Users\honey\Documents\PowerShellCTF\Win7-security.evtx" | where { $_.id -eq 4624 } | select -ExpandProperty message | Select-String -Pattern "Logon\D+:\s+3" | measure

Operators: -as, -is, -replace, -join, -split, -in, -contains

as - operator produces a new object in an attempt to convert an existing object into a different type

1000 / 3 -as [int]

Is - It's designed to return True or False if an object is of a particular type or not

123.45 -is [int]

"SERVER-R2" -is [string]

$True -is [bool]

(Get-Date) -is [datetime]

Replace - operator is designed to locate all occurrences of one string within another and replace those occurrences with a third string (linux sed command)

"192.168.34.12" -replace "34","15"

join and -split operators are designed to convert arrays to delimited lists, and vice versa (linux Cut, and Awk commands)

```
$array = "one","two","three","four","five"
$array -join "|"
$string = $array -join "|"
```

split - It takes a delimited string and makes an array from it

```
$array = (gc computers.tdf) -split "`t"
```

Contains – operator is used to test whether a given object exists within a collection

```
$collection = 'abc','def','ghi','jkl'
$collection -contains 'abc'
```

Like - operator is designed for wildcard string comparisons

```
'this' -contains '*his*'
```

String Manipulation

```
"Hello" | gm
```

IndexOf() tells you the location of a given character within the string:

```
"SERVER-R2".IndexOf("-")
```

ToLower() and ToUpper() convert the case of a string

```
$computername = "SERVER17"
$computername.tolower()
```

Trim() removes whitespace from both ends of a string;

```
$username = " Don "
$username.Trim()
```

TrimStart() and TrimEnd() remove whitespace from the beginning or end of a string, respectively

Date manipulation

```
get-date | gm
$90daysago = $today.adddays(-90)
```

NEVER THE END

- PowerShell's simplified scripting language
- iScope
- Functions, and the ability to build multiple tools into a single script file
- Error handling
- Writing help
- Debugging

- Custom formatting views
- Custom type extensions
- Script and manifest modules
- Using databases
- Workflows
- Pipeline troubleshooting

- Complex object hierarchies
- Globalization and localization
- Proxy functions
- Constrained remoting and delegated administration
- Using .NET
- Splatting

REFERENCES

- https://docs.microsoft.com/en-us/powershell/scripting/learn/windows-powershell-glossary?view=powershell-5.1
- https://docs.microsoft.com/en-us/dotnet/framework/additional-apis/index
- https://docs.microsoft.com/en-us/powershell/scripting/samples/creating-a-custom-input-box?view=powershell-5.1

REFERENCES (CONT.)

- https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.core/about/about_commonparameters?view=powershell-6
- https://devblogs.microsoft.com/scripting/using-scheduled-tasks-and-scheduled-jobs-in-powershell/
- https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.core/about/about_language_modes?view=powershell-5.1

POWERSHELL v5 SECURITY REFERENCES

- https://blogs.msdn.microsoft.com/daviddasneves/2017/05/25/powershell-security-at-enterprise-customers/
- https://www.blackhillsinfosec.com/powershell-logging-blue-team/
- https://www.stigviewer.com/stig/windows_10/2017-02-21/finding/V-68819
- https://www.stigviewer.com/stig/windows_server_20122012_r2_member_server/2018-10-30/finding/V-80475

QUESTIONS

- Upcoming Talks/Training
  - Crash Course in PowerShell (Dec 14)
    - ISSA & BSidesCharm (Potentially)
  - SEC505 Windows Security and PowerShell Automation (Feb 06)
    - SANS
  - PowerShell Pipeline Deep Dive (April)
    - BSidesCharm (Potentially)
- LinkedIn
  - in/James-Honeycutt
- Twitter
  - @P0w3rChi3f
- Website
  - JamesHoneycutt.net