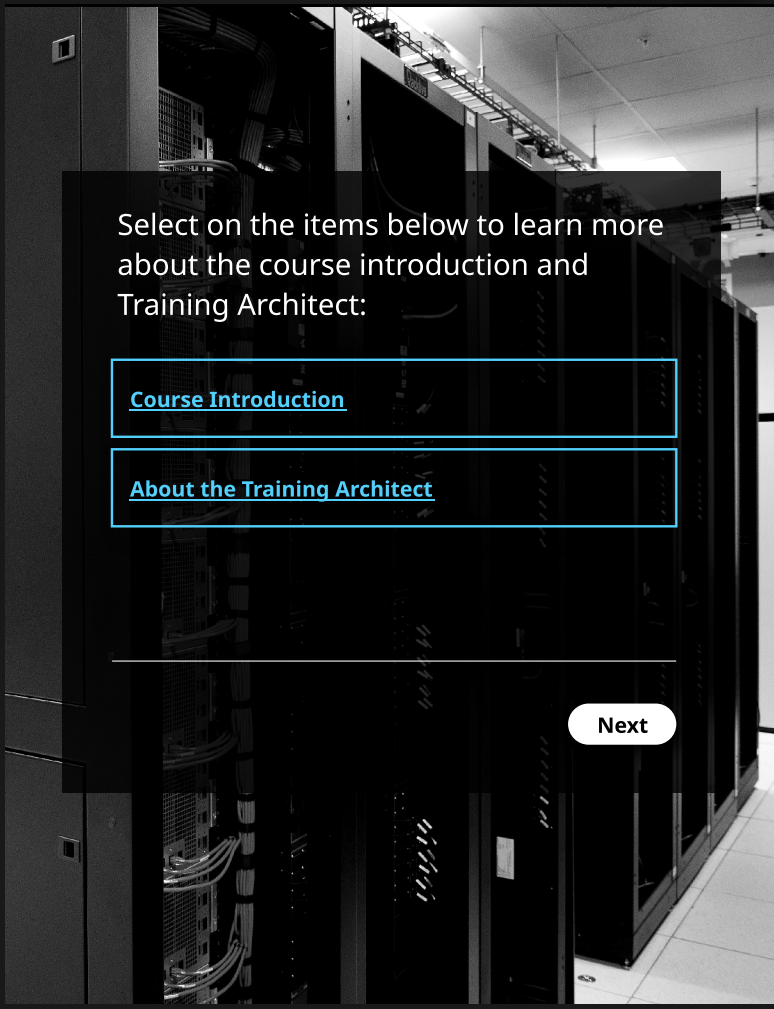
Azure PowerShell Essentials

Course Navigation

Linux Academy

Course Introduction

Welcome to the Course

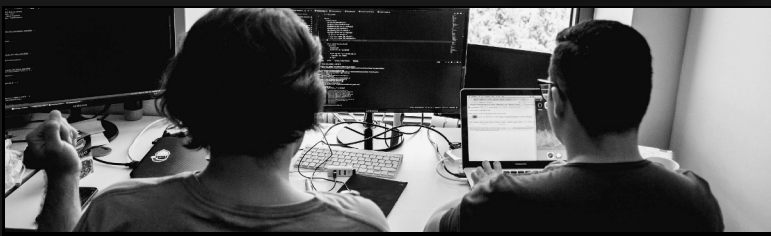


Select on the items below to learn more about the course introduction and Training Architect:

Course Introduction

About the Training Architect

Next



Welcome to the Linux Academy **Azure PowerShell Essentials** course.

In this course, we set the foundation to begin learning PowerShell, specifically with Microsoft Azure. We will start by discussing the basics of PowerShell, including commands and syntax. Next, we'll move into creating and managing Azure resources, subscriptions, and identities with PowerShell. Finally, we'll put everything together and start building our own scripts.

This is a beginner-level course. The only prerequisites are a basic knowledge of networking, client-server communications, and some decent troubleshooting skills.

Learning Azure PowerShell will help automate our administrative tasks, gather useful information from our resources, and take that next step in career advancement.

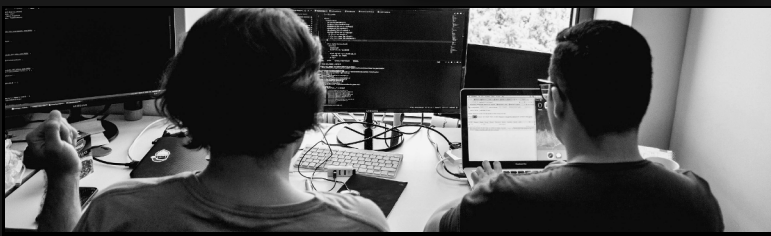
Thanks for taking this course. Let's get started!

Corey Knapp

Next

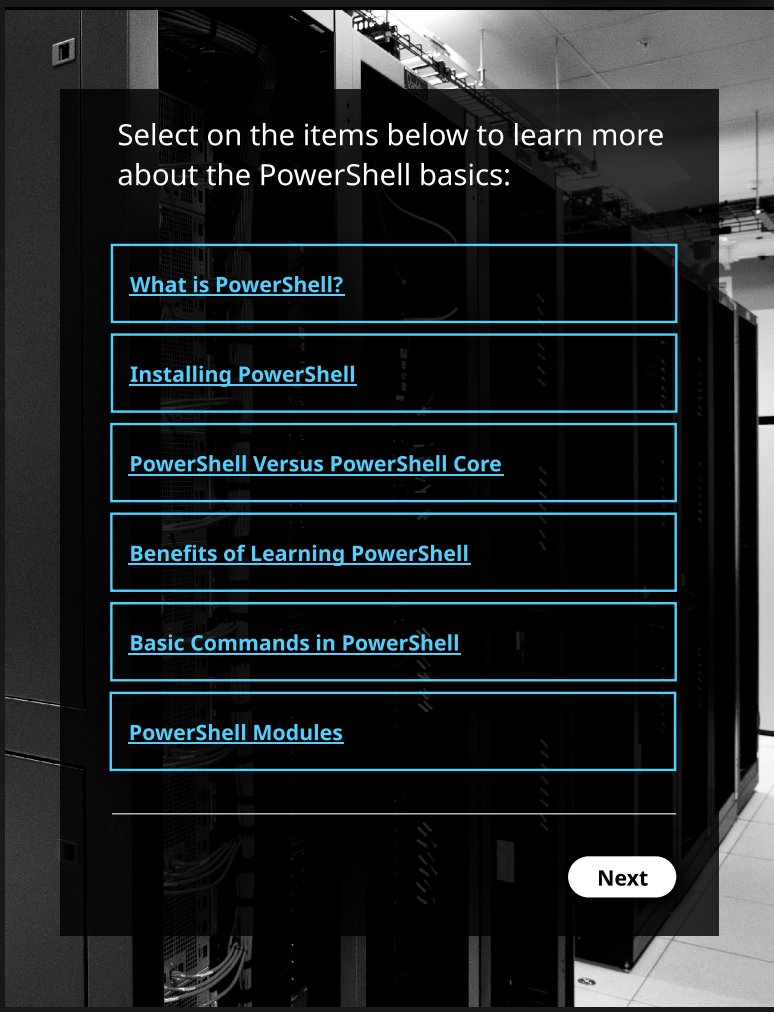
Course Introduction

About the Training Architect



Personal Life: Born in California, raised in New Jersey, but now living in Chapel Hill, North Carolina, Corey Knapp has been all over the United States. He loves the winter but hates the cold, and loves the summer but hates the heat. Even in his own mind, Corey is an enigma. In his free time, he loves spending time with his teenage children, Anna and Logan, and his wife of 18 years, Michele. Corey has a passion for learning and is ready to spread that eagerness to others.

Professional Life: Corey Knapp has been an IT professional for over 22 years, specializing in Microsoft Azure solutions working his way up from Help Desk to Azure Architect over the past two decades. His primary focus during that time has been on everything from Active Directory to MS Exchange, from Office 365 to Unified Communications, and many other infrastructure and data center roles. This current role as an Azure Training Architect has been the single most satisfying role of his career, and he looks forward to sharing his knowledge with you!



Select on the items below to learn more about the PowerShell basics:

- What is PowerShell?
- Installing PowerShell
- PowerShell Versus PowerShell Core
- Benefits of Learning PowerShell
- Basic Commands in PowerShell
- PowerShell Modules

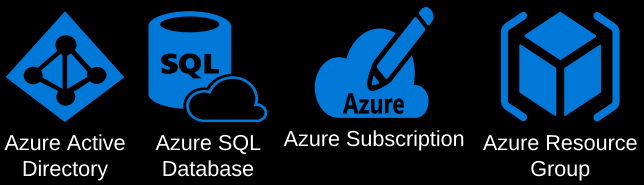
Next

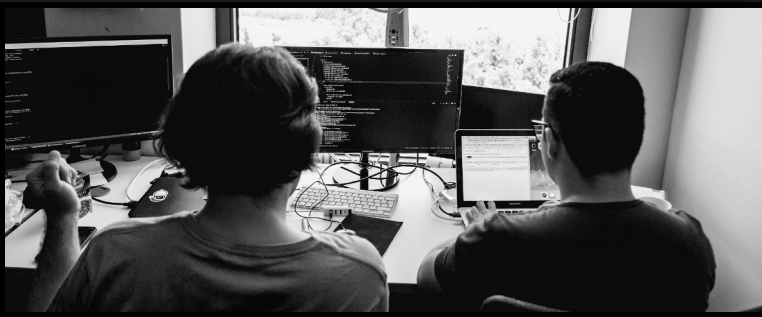
Overview

A What Is PowerShell and What Can We Do With It?

- PowerShell is built on the .NET framework and is both a scripting language and an interactive command environment.
- PowerShell allows for easier administration of systems and the automation of repetitive tasks.

A What Can PowerShell Manage?



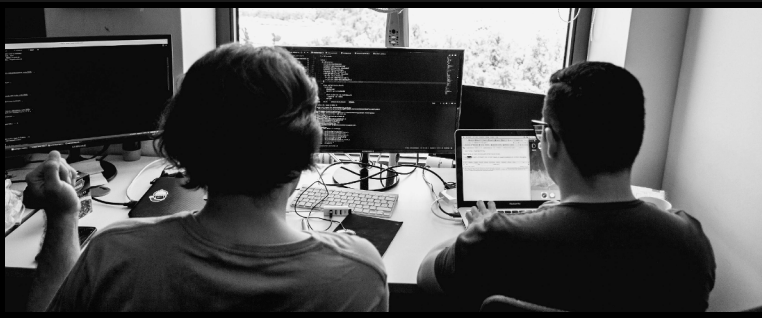


PowerShell

Windows PowerShell comes installed by default in every version of Windows since Win7 SP1 and Server 2008 R2 SP1.

PowerShell Core

Installing PowerShell Core
Download the MSI package from the Microsoft GitHub release page. Under the Assets section, the MSI file looks like `PowerShell-<version>-win-<os-arch>.msi`.



PowerShell

Windows PowerShell comes installed by default in every version of Windows since Win7 SP1 and Server 2008 R2 SP1.

PowerShell Core

Installing PowerShell Core
Download the MSI package from the Microsoft GitHub release page. Under the Assets section, the MSI file looks like `PowerShell-<version>-win-<os-arch>.msi`.

Back
Next

Back to Main

Should We Use PowerShell or PowerShell Core?

1. **PowerShell**
   - PS 5.1 is the last full release.
   - Only supports Microsoft OS.
   - .NET Framework dependency.

2. **PowerShell Core**
   - PSC 6.1 is the latest release.
   - Unsupported modules.
   - Open source (MacOS, Linux).

3. **The Bottom Line**
   - PSC is the next evolution of PS.
   - Version 7.0 will replace both.
   - True cross-platform support.
   - PS for now, PSC in near future.

Why Should We Use PowerShell?

PowerShell is a powerful scripting tool that can expedite our admin tasks and increase our daily productivity. Here are some of the ways we can benefit from PowerShell.

- **Trusted by System Administrators**
- **Most GUIs are PowerShell Front Ends**
- **Easy Automation**
- **Scalable Management**
- **Ability to Manipulate Server and Workstation Components**
- **Prepare for Microsoft Certifications**

In a Nutshell
PowerShell is a powerful, easy-to-use tool that can create text files, create backups, and parse both files and data. It's also native to Windows, allowing interoperability between Windows, Azure, and now both Linux and MacOS.

Course Navigation

Introduction to PowerShell
Section 2

PowerShell Basics
- What is PowerShell?
- Installing PowerShell
- PowerShell Versus PowerShell Core
- Benefits of Learning PowerShell
- Basic Commands in PowerShell
- PowerShell Modules

PowerShell Concepts
- Command and Syntax Basics
- The Three Most Important Commands in PowerShell
- Using Variables
- Understanding PowerShell Pipelines

PowerShell in Azure
- Installing the Azure PowerShell Module
- Signing into Azure
- The Most Important Azure PowerShell Commands
- Working with Outputs
- Using Nested Properties
- Filtering Results

Managing Azure with PowerShell
Section 3

Basic PowerShell Commands

There are a few commands every PowerShell user should know. These commands help make everyday tasks easier and are some of the key building blocks in learning PowerShell.

`Start-Process`	`Get-Help`	`Get-Command`
`Get-Item`	`Get-Content`	`Start-Service` and `Stop-Service`
`Stop-Process`	`Set-ExecutionPolicy`	`Copy-Item` and `Remove-Item`

Back
Next

Back to Main

A

What Is a PowerShell Module?

A module is a package containing PowerShell commands such as cmdlets, providers, functions, workflows, variables, and aliases.

To find installed modules:

```
Get-Module -ListAvailable
```

To find already-installed modules:

```
Get-Module
```

To find all commands in a module:

```
Get-Command -Module <module-name>
```

To get help for commands in a module:

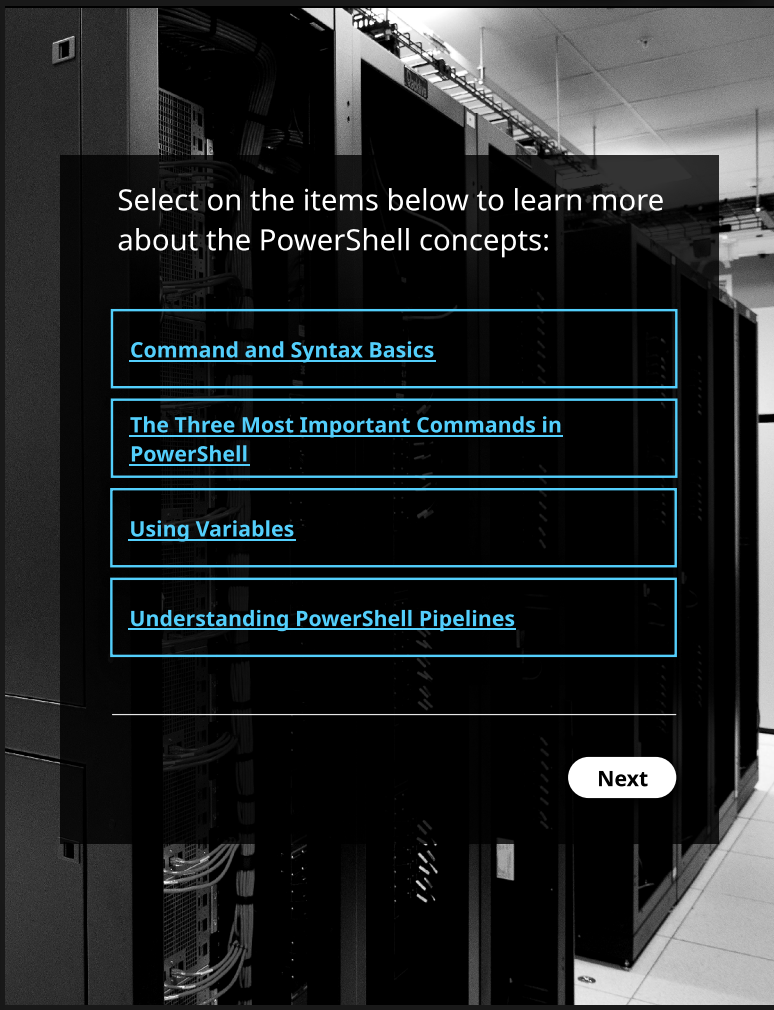
```
Get-Help <command-name>
```

To download and install the help files for commands:

```
Update-Help -Module <module-name>
```

Back
Next

Back to Main



Select on the items below to learn more about the PowerShell concepts:

- Command and Syntax Basics
- The Three Most Important Commands in PowerShell
- Using Variables
- Understanding PowerShell Pipelines

Next

A Syntax Basics

```
Get-Service
```

All PowerShell cmdlets follow a simple verb-noun syntax.

Do something - To something

PowerShell cmdlet:

```
Get-Help
```

PowerShell cmdlet with parameter:

```
Get-Help -name "*network*"
```

Important Commands in PowerShell

A
`Get-Help`
Use this cmdlet to get information on how to use a cmdlet. Also type **`help`** or **`man`**.

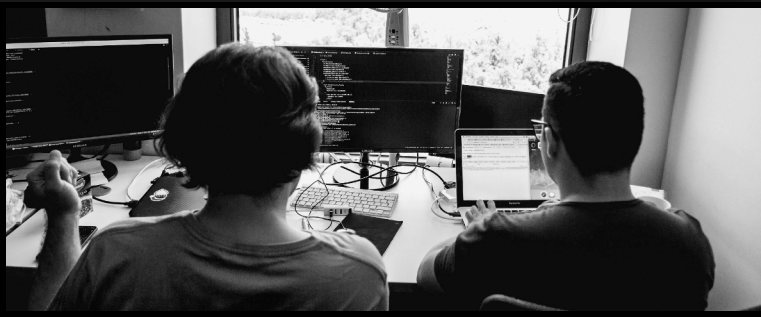
B
`Get-Command`
Use this cmdlet to get only the commands imported into PowerShell.

C
`Get-Service`
This cmdlet gets objects that represent the stopped and started services on a computer.

These three commands are the building blocks of learning to use PowerShell on a daily basis. Admins can use these three commands to get information on all other commands in PowerShell, list which commands are available, and gather basic information on services currently running in their environment.

Introduction to PowerShell

Using Variables



PowerShell works with objects and lets us create named objects known as variables.

Variable names can include the underscore character and any alphanumeric character. We specify variables using the $ character followed by the variable name.

```
PS> $loc
```
Creates a blank variable with the name **loc**.

```
PS> $loc = Get-Location
```
Sends the output of **Get Location** to **$loc**.

```
PS> $loc | Get-Member -MemberType Property
```
Uses **Get-Member** to display information about the contents of the variable.

Back

Next

Back to Main

Why Use Pipelines?

A pipeline is a series of commands connected by pipeline operators (|). Each operator sends the results of the preceding command to the next command.

Command-1 | Command-2 | Command-3

In this example, objects from Command-1 are sent to Command-2. Command-2 then processes the objects and sends them to Command-3.

Ideal for filtering objects
Get a specific piece of information.

Transform or format information
Change information in some way.

Objects, not text
When running cmdlets in PowerShell, text output is displayed because it is necessary to represent objects as text in a console window.

Why Use Pipelines?

A pipeline is a series of commands connected by pipeline operators (|). Each operator sends the results of the preceding command to the next command.

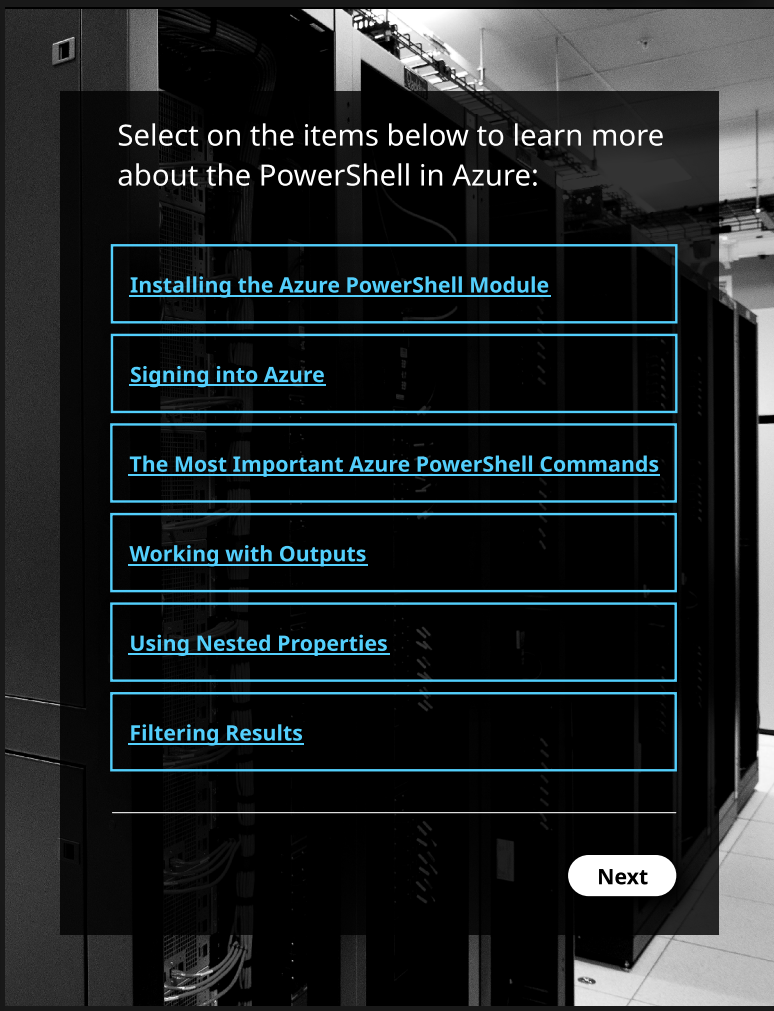
Command-1 | Command-2 | Command-3

In this example, objects from Command-1 are sent to Command-2. Command-2 then processes the objects and sends them to Command-3.

Ideal for filtering objects
Get a specific piece of information.

Transform or format information
Change information in some way.

Objects, not text
When running cmdlets in PowerShell, text output is displayed because it is necessary to represent objects as text in a console window.



Select on the items below to learn more about the PowerShell in Azure:

- Installing the Azure PowerShell Module
- Signing into Azure
- The Most Important Azure PowerShell Commands
- Working with Outputs
- Using Nested Properties
- Filtering Results

Next

Prerequisites

1. Update to PowerShell 5.1, if needed.
2. Install .NET Framework 4.7.2 or later.

Installing the Module

Installing for the active user:

```
Install-Module -Name Az -AllowClobber -Scope CurrentUser
```

Installing for all users (requires admin privileges):

```
Install-Module -Name -AllowClobber -Scope AllUsers
```

Installing Offline

Azure PowerShell for Windows is installed using the MSI file available from GitHub.

Connecting to Azure

A

Sign In

Sign in interactively
`Connect-AzAccount`

B

Authenticate

Enter credentials
The system prompts us to sign in interactively using a browser.

C

Authorize

Permissions are required
The credentials we use must have permissions granted for the resource group and resources.

Useful Azure PowerShell Commands

These commands provide a starting point for key PowerShell commands necessary for Azure administrators. It is not all encompassing, but is a great jumping-off point.

`Get-AzVM`
`New-AzVM`
	`Get-AzSubscription`	`Start-AzVM`
		`Stop-AzVM`
		`Remove-AzVM`

`Get-AzResource`
`Move-AzResource`
	`Get-AzResourceGroup`
	`New-AzResourceGroup`
	`Set-AzResourceGroup`
		`Get-AzStorage Account`
		`New-AzStorage Account`

`Login-AzAccount` `Logout-AzAccount`	`Get-AZVirtualNetwork` `Get-AzLoadBalancer`	`New-AzVirtual Network` `New-AzPublicIP Address`

By default, each Azure PowerShell cmdlet formats output to be **easy to read.** PowerShell allows you to **convert or format cmdlet output** by piping to one of the following cmdlets:

A. **`Format-Table`**
Default output in PowerShell. Doesn't display all the information of the requested resource.

B. **`Format-List`**
Displays two columns: one for property names and one for property values.

C. **`Format-Wide`**
Displays one property name per query. Displayed property controlled by argument.

The **`ConvertTo-*`** family of cmdlets allows for converting results of Azure PowerShell cmdlets to **machine-readable formats,** including CSV, JSON, XML, and HTML formats.

In PowerShell, **object properties** tell us about the object. Each time a **`Get-`** command is run, you receive a single object or set of objects. Properties are **attributes about the objects themselves.**

A **Not all properties are displayed**
Based on the object type, not all of an object's properties are displayed. Why?

B **Using `Get-Member`**
The **`Get-Member`** cmdlet shows us a PowerShell object's defined properties.

C **`-Property` for the win!**
Using the **`-Property`** parameter with a wildcard value displays **all** of the properties.

`Get-Member` and Microsoft documentation may give you an idea of what properties are helpful to you, but **interrogating a sample object directly** using **`-Property *`** will help you validate that a property has the **information you need.**

One of the best things about using an **object-based pipeline** is that you can **filter objects** out of the pipeline at **any stage.** How?

A **`Where-Object` cmdlet**
`Where-Object` determines which objects to pass along the pipeline by evaluating a script block that may include a reference to an object being filtered.

B **Using the `$_` operator**
`$_` represents the current pipeline object. From here, you can choose the property you want to filter on.

C **Comparison operators**
Using these operators in conjunction with **`$_`**, you can narrow your list down even further.

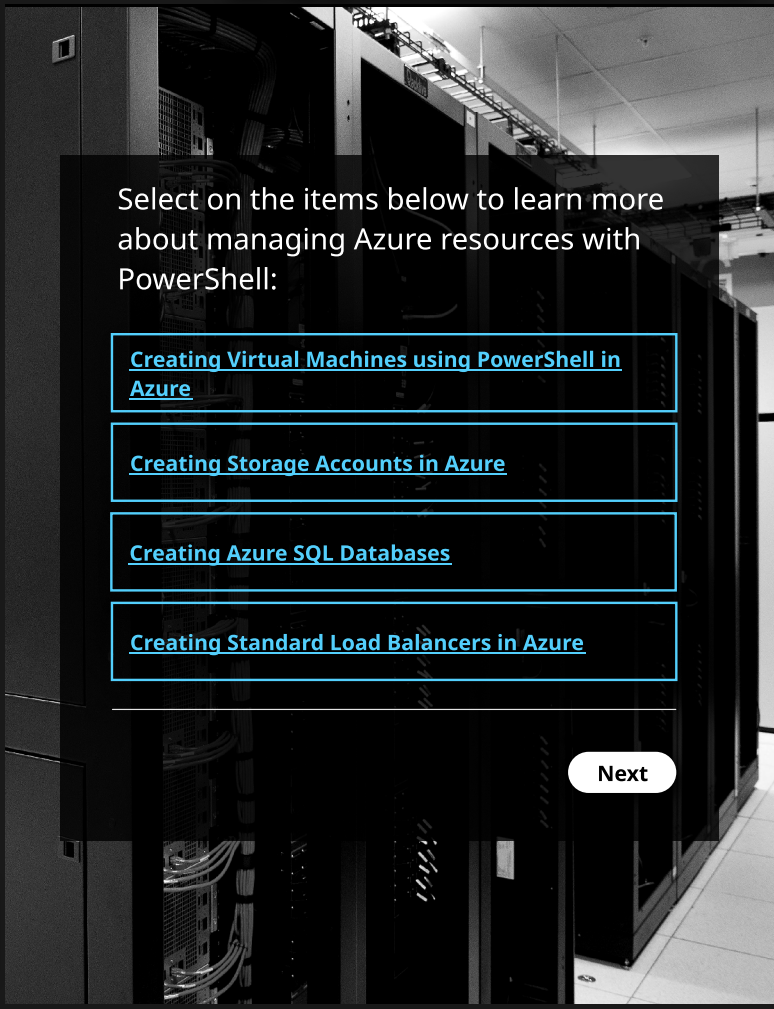
```
Get-Service | Where-Object {$_.Status -eq "Running"}
```

In this example, we are searching the **`Status`** property of the **`Service`** list and filtering to see if the **`Status`** property is equal to **`Running`**.

Back

Managing Azure with PowerShell

Managing Azure Resources with PowerShell



Select on the items below to learn more about managing Azure resources with PowerShell:

- Creating Virtual Machines using PowerShell in Azure
- Creating Storage Accounts in Azure
- Creating Azure SQL Databases
- Creating Standard Load Balancers in Azure

Next

Managing Azure with PowerShell

Creating Virtual Machines using PowerShell in Azure

What We're Going to Do

In this exercise, we will create an Azure Virtual Machine, connect via RDP to the virtual machine, and install a web server. We will do this using only PowerShell.

Create a Virtual Machine

Azure PowerShell command:

```
New-AzVM -ResouceGroupName
"myResourceGroup" -Name "myVM
-VirtualNetworkName "myVnet"
-SubnetName "mySubnet"
-SecurityGroupName
"mySecurityGroup"
-PublicIpAddressName
"myPublicIpAddress"
-OpenPorts 80,3389
```

Connect to VM

Azure PowerShell command:

```
Get-AzPublicIpAddress
-ResourceGroupName
"myResourceGroup" | Select
"IpAddress"
```

Install the Web Server

Azure PowerShell command:

```
Install-WindowsFeature -name
Web-Server
-IncludeManagementTools
```

What We're Going to Do

In this exercise, we will create a general-purpose v2 standard Azure Storage account using only PowerShell.

Connect to Azure

Azure PowerShell command:

```
Connect-AzAccount
```

Set the Variables

Azure PowerShell command:

```
$resourceGroup = "teststorage"
New-AzResourceGroup -Name
$resrouceGroup -Location
eastus
```

Create the Storage Account

Azure PowerShell command:

```
New-AzStorageAccount
-ResourceGroupName
$resourceGroup -Name
<account-name> -Location
eastus -SkuName Standard_LRS
-Kind StorageV2
```

This lesson shows how to create a single database using PowerShell and the Azure Portal.

```
# Set variables for your server and database

$subscriptionID = '<SunscriptionID>'
$resourceGroupName = "DBResourceGrp-$(Get-Random)"
$location = "East US"
$adminLogin = "azureuser"
$password = "Password123!"
$serverName = "mysqlserver-$(Get-Random)"
$databaseName = "mySampleDatabase"

# Create an IP Address range that you want to allow
# to access your server (Leaving at 0.0.0.0
# prevents outide-of-Azure connections)

$startIp = "0.0.0.0"
$endIp = "0.0.0.0"

# Connect to Azure

Connect-AzAccount

# Set subscription ID

Set-AzContext -SubscriptionId $subscriptionId
```

Next

Managing Azure with PowerShell

Creating Standard Load Balancers in Azure

This lesson shows how to create a Standard Load Balancer using Azure PowerShell.

```
# Create a Resource Group

$rgName='MyResourceGroupSLB'
$location='eastus'
New-AzResourceGroup -Name $rgName -Location $location

# Create a Public IP Address

$publicIp = New-AzPublicIpAddress `
 -ResourceGroupName $rgName `
 -Name 'myPublicIP' `
 -Location $location `
 -AllocationMethod static `
 -SKU Standard

# Create the Front-End IP

$feip = New-AzLoadBalancerFrontendIpConfig -Name
'myFrontEndPool' -PublicIpAddress $publicIp

# Configure the Back-End Address Pool

$bepool = New-AzLoadBalancerBackendAddressPoolConfig
-Name 'myBackEndPool'

# Create a Health Probe

$probe = New-AzLoadBalancerProbeConfig `
 -Name 'myHealthProbe' `
 -Protocol Http -Port 80 `
 -RequestPath / -IntervalInSeconds 360 -ProbeCount 5
```

Next

Back to Main

The variables have been set. Let's begin building the SQL Server.

```
# Create a resource group

$resourceGroup = New-AzResourceGroup -Name
    $resourceGroupName -Location $location

# Create a server with a system wide unique name

New-AzSqlServer -ResourceGroupName
    $resourceGroupName `
    -ServerName $servername `
    -Location $location `
    -SQLAdministratorCredentials $(New-Object -TypeName
    System.Management.Automation.PSCredential `
    -ArgumentList $adminLogin, $(ConvertTo-SecureString
    -String $password -AsPlainText -Force))

# Create a server firewall rule that allows access from
# the specified range

NewAzSQLServerFirewallRule
    -ResourceGroupName $resourceGroupName `
    -ServerName $serverName `
    -FirewallRuleName "Allowed IPs" -StartIpAddress
        $startIp -EndIpAddress $endIp
```

Next

This part of the lesson focuses on creating the load balancer and NAT rules.

```
# Create a Load Balancer Rule

$rule = New-AzLoadBalancerRuleConfig `
  -Name 'myLoadBalancerRuleWeb' -Protocol Tcp `
  -Probe $probe -FrontendPort 80 -BackendPort 80 `
  -FrontendIpConfiguration $feip `
  -BackendAddressPool $bePool

# Create the NAT Rules

$natrule1 = New-AzLoadBalancerInboundNatRuleConfig `
  -Name 'myLoadBalancerRDP1' `
  -FrontendIpConfiguration $feip `
  -Protocol tcp -FrontendPort 4221 `
  -BackendPort 3389
$natrule2 = New-AzLoadBalancerInboundNatRuleConfig `
  -Name 'myLoadBalancerRDP2' `
  -FrontendIpConfiguration $feip `
  -Protocol tcp `
  -FrontendPort 4222 `
  -BackendPort 3389
$natrule3 = New-AzLoadBalancerInboundNatRuleConfig `
  -Name 'myLoadBalancerRDP3' `
  -FrontendIpConfiguration $feip `
  -Protocol tcp `
  -FrontendPort 4223 `
  -BackendPort 3389
```

Next

Back to Main

Now that the server has been built, we can create the database.

```
# Create General Purpose Gen5 database with 2 vCore

New-AzSqlDatabase -ResourceGroupName
    $resourceGroupName `
    -ServerName $serverName
    -DatabaseName $databaseName `
    -Edition GeneralPurpose `
    -VCore 2 `
    -ComputeGeneration Gen5 `
    -MinimumCapacity 2 `
    -SampleName "AdventureWorksLT"
```

Next

Back to Main

Managing Azure with PowerShell

Creating Standard Load Balancers in Azure

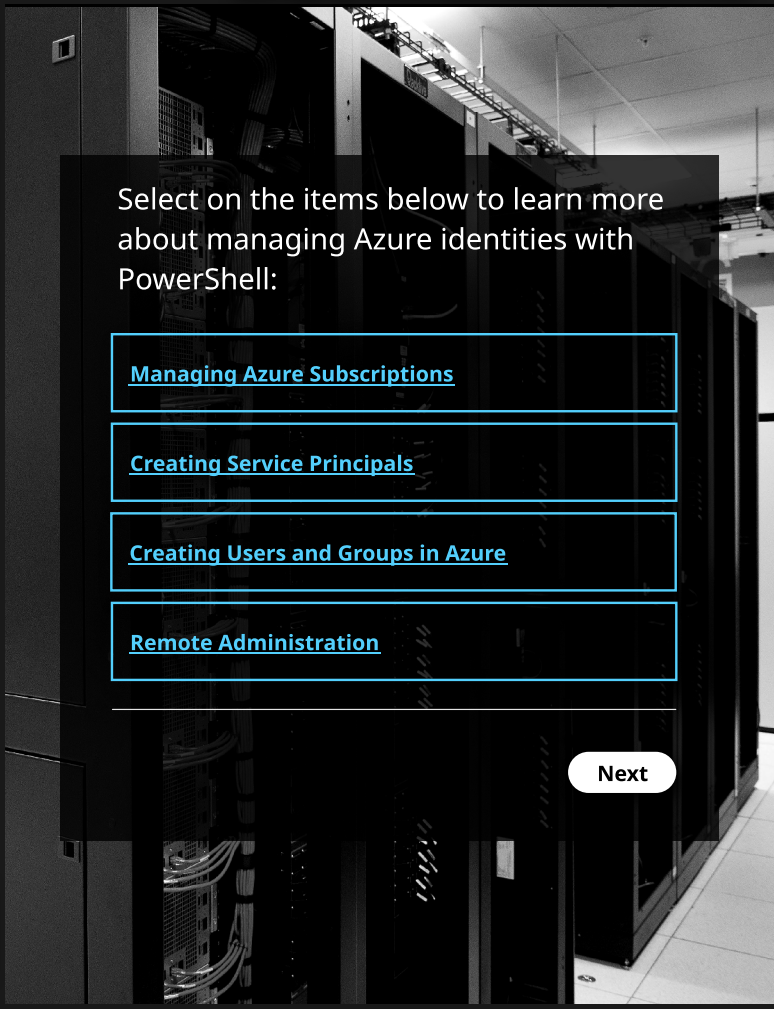
With the prerequisites out of the way, we can now install the Standard Load Balancer.

```
# Create the Standard Load Balancer

New-AzLoadBalancer `
  -ResourceGroupName $rgName `
  -Name 'MyLoadBalancer' `
  -SKU Standard `
  -Location $location `
  -FrontendIpConfiguration $feip `
  -BackendAddressPool $bepool `
  -Probe $probe `
  -LoadBalancingRule $rule `
  -InboundNatRule $natrule1,$natrule2,$natrule3


# With the load balancer created, we can now connect it
to our Azure Virtual Machines.
```

Next

Managing Azure with PowerShell

Managing Azure Identities with PowerShell



Select on the items below to learn more about managing Azure identities with PowerShell:

- Managing Azure Subscriptions
- Creating Service Principals
- Creating Users and Groups in Azure
- Remote Administration

Next

Managing Azure with PowerShell

Managing Azure Subscriptions

Most Azure users will only ever have a single subscription. However, there are cases where a user might have multiple subscriptions, such as being part of more than one organization or being in an organization that has divided access to certain resources across groupings.

Let's clarify the difference between tenants, users, and subscriptions in Azure. A **tenant** is the Azure Active Directory entity that encompasses the whole organization. **Users** are those accounts that sign in to Azure to create, manage, and use resources. **Subscriptions** are the agreements with Microsoft to use cloud services, including Azure. **Every resource is associated with a subscription.**

```
# Getting a list of Azure Subscriptions

Get-AzSubscription

# Changing the active Azure Subscription

Set-AzContext -SubscriptionId <ID Number>
```

Next

Managing Azure with PowerShell

Creating Service Principals

Microsoft Azure offers service principals instead of having apps sign in as a fully privileged user.

An Azure service principal is an identity created for use with apps, hosted services, and automated tools to access Azure resources.

There are two types of authentication available for service principals: Password-based and Certificate-based.

```
# Creating a Password-based authentication

New-AzADServicePrincipal -DisplayName
    ServicePrincipalName



Import-Module Az.Resources
$credentials = New-Object Microsoft.Azure.Commands.
    ActiveDirectory.PSADPasswordCredential -Property @{
    StartDate=Get-Date; EndDate=Get-Date -Year 2024;
    Password=<Choose a strong password>}
Get-AzAdServicePrincipal -DisplayName
    ServicePrincipalName
```

Next

Users, Groups, and Object IDs

1. `New-AzureADUser`
   - Requires `DisplayName`, `PasswordProfile`, and UPN.
   - Retain the `ObjectID`.
2. `New-AzureADGroup`
   - Requires `DisplayName`, `MailEnabled`, and `SecurityEnabled`.
   - Again, retain the `ObjectID`.
3. `Add-AzureADGroupMember`

```
Add-AzureADGroupMember -ObjectId
"62438306-7c37-4638-a72d-0ee8d9217680"
-RefObjectId
"0a1068c0-dbb6-4537-9db3-b48f3e31dd76"
```

Connecting to Azure VMs using PowerShell

A

`Enable-AzureVMPSRemoting`

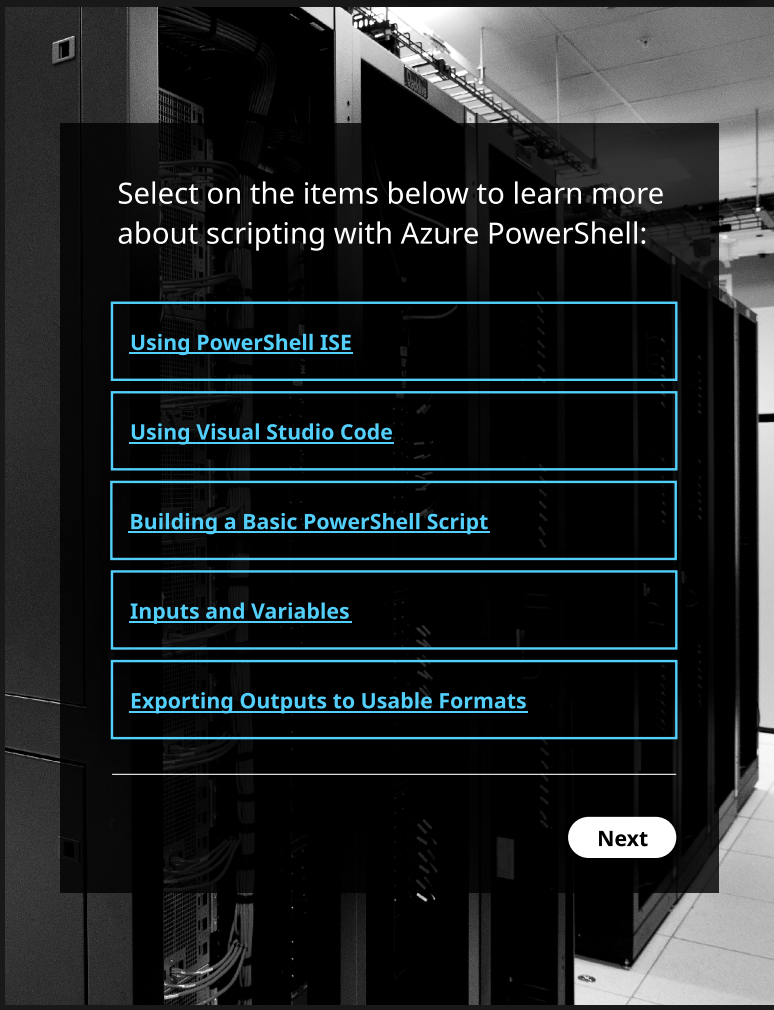
With the `Enable-AzureVMPSRemoting` cmdlet, Azure PowerShell configures the pieces necessary for running commands and code against target VMs in Azure, much like we do on-premises. To do this, it performs the following:

- Based on the operating system, it ensures WinRM (Windows) or SSH (Linux) is setup.
- It ensures network security group rules are in place to allow communication to the target, again based on communications type.
- For Linux VMs, it installs PowerShell Core on the target system.

`Enter-AzVM`	`Invoke-AzVMCommand`

Back

Back to Main



Select on the items below to learn more about scripting with Azure PowerShell:

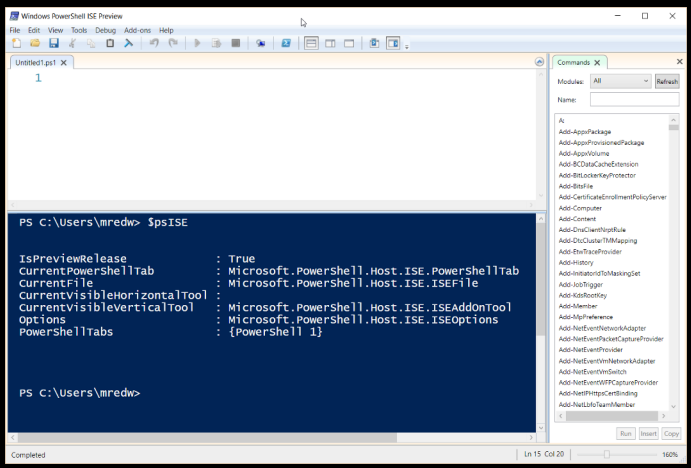
- Using PowerShell ISE
- Using Visual Studio Code
- Building a Basic PowerShell Script
- Inputs and Variables
- Exporting Outputs to Usable Formats

Next

A

PowerShell ISE:
We can use the Windows PowerShell Integrated Scripting Environment (ISE) to create, run, and debug commands and scripts. The Windows PowerShell ISE consists of the menu bar, Windows PowerShell tabs, the toolbar, script tabs, a Script Pane, a Console Pane, a status bar, a text-size slider, and context-sensitive help.

PowerShell ISE Tabs
A PowerShell tab is the environment a PowerShell script runs in. We may have up to eight PowerShell tabs open at the same time.

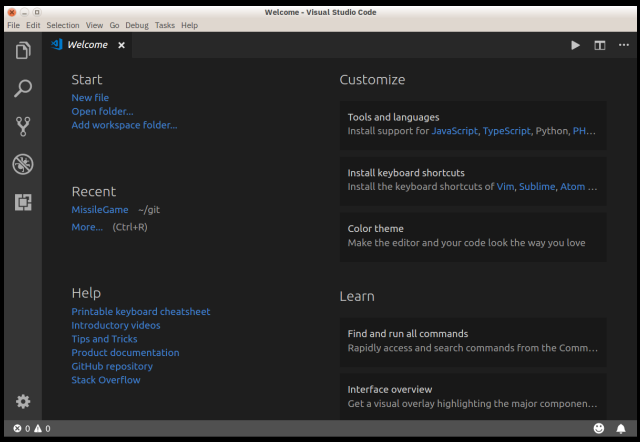


Next

A

Visual Studio Code is a code editor. Like many other code editors, VS Code adopts a common user interface and layout of an explorer on the left, showing all of the files and folders a user has access to. The right side shows the content of any open files.

PowerShell Extension
The official PowerShell extension can be installed from the **Extension** view or from the VS Code command line.



A Our First PowerShell Script

```
Write-Host "Hello from Linux Academy!!!"
```

We save this file with a `.ps1` file extension and run the script by typing the following.

```
.\First.ps1
```

Change the execution policy:

```
Set-ExecutionPolicy RemoteSigned
```

Run the script again:

```
First.ps1
```

A Adding Inputs and Variables to Our Script

```
$date = get-date
Write-Host "Today is " $date
```

We input the cmdlet `Get-Date` to the variable of `$date`. The output from this Write-Host cmdlet will read as follows (but with the local date and time):

```
Today is 11/23/19 3:29:44 PM
```

```
# This cmdlet will show Azure Resources

$Resources = Get-AzResource | ft
$Resources
```

A Adding Inputs and Variables to Our Script

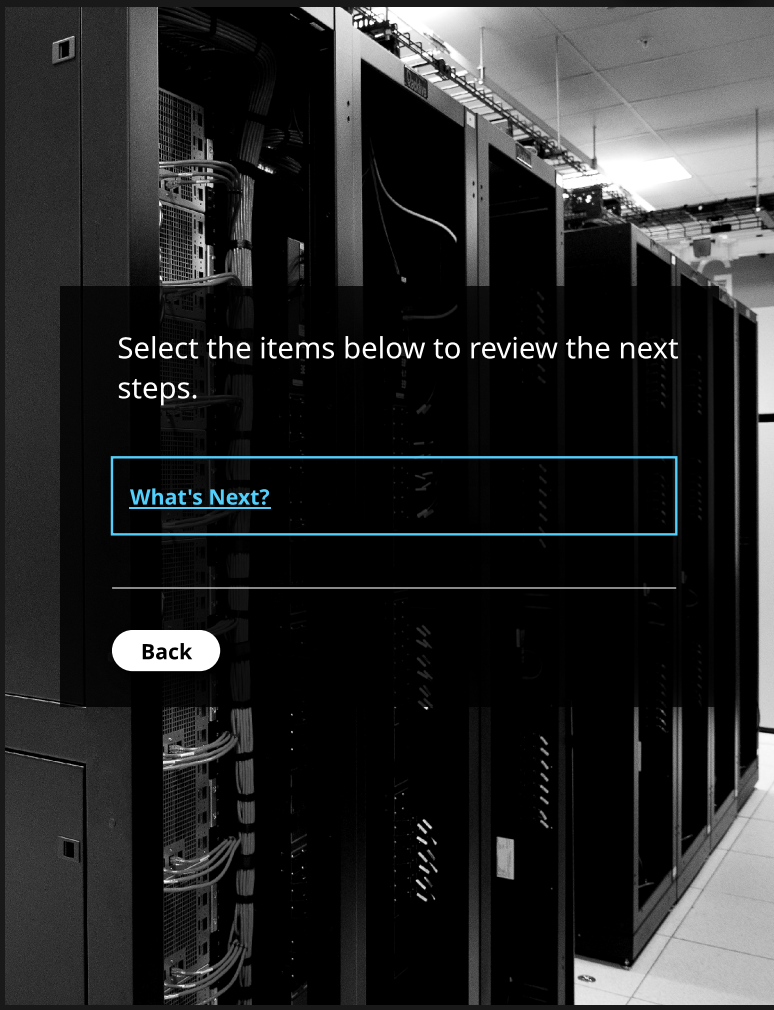
```
Get-AzResource | Export-CSV -Path
C:\Scripts\Test.csv
```

This command gives us a CSV output of the `Get-AzResource` cmdlet which had been set to the `$Resources` variable.

```
# Output in HTML Format

Get-AzResource | ConvertTo-HTML | Out-File
    C:\Scripts\Test.html
```

Create more variables!
Select specific properties for outputs!
Set variables for parameters!

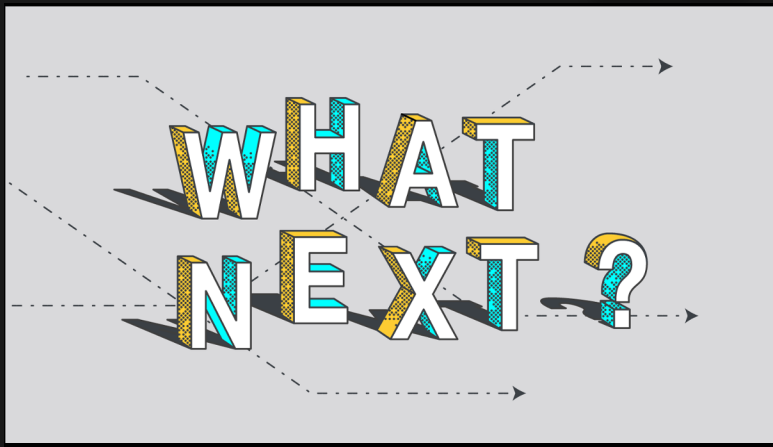
Anything is possible!



Select the items below to review the next steps.

What's Next?

Back



A

Practice:
The only true way to learn PowerShell is to use it every day. Find a need and try it!

B

Imitation:
Find a blog or website with a PowerShell script you like. Try to imitate or reproduce it!

C

Docs Are Our Friends!!!
Refer to the official PowerShell documentation as much or as little as necessary.

Back