TechMentor @ Microsoft HQ 2025 - PowerShell Hands-On Labs

Lab 1: Introduction to PowerShell

Install PowerShell

1. Install PowerShell 7 (if you can)

   Answer:

   Follow the instructions to install PowerShell 7 on your system. You can find the installation guide for your operating system in the documentation: Install PowerShell on Windows, Linux, and macOS

   If you are running Windows, Windows PowerShell 5.1 is already installed by default, but you can install PowerShell 7 alongside it. On windows the preferred way to install PowerShell 7 is using WinGet.

2. Get help for a command before help is installed:

   ```
   Get-Help Get-Command -Full
   ```

   Answer:

   Before help is installed, you will see a basic description of the command and its syntax. There are no descriptions or examples. The REMARKS section contains the following message:

   ```
   Get-Help cannot find the Help files for this cmdlet on this computer. It is
   displaying only partial help.
       -- To download and install Help files for the module that includes this
   cmdlet, use Update-Help.
   ```

Update help

1. Run `Update-Help` to install the latest help on your computer.

   ```
   Update-Help -Force
   ```

   Answer:

   This requires that the shell is running with elevated privileges (use the Run as Administrator option). On Windows, you an get error message about failing to download help for specific modules. This is a known issue for those modules.

```
Update-Help: Failed to update Help for the module(s)
'ConfigDefenderPerformance, Dism, Get-NetView, Kds,
Microsoft.PowerShell.ThreadJob, NetQos, Pester, PKI, Whea, WindowsUpdate' with
UI culture(s) {en-US} : One or more errors occurred. (Response status code
does not indicate success: 404 (The requested content does not exist.).).
English-US help content is available and can be installed using: Update-Help -
UICulture en-US.
```

2. Get help after help is installed, run:

```
Get-Help Get-Command -Full
```

Answer:

Notice the difference in the amount of information. You know you have the installed the help when you see the full description, syntax, parameters, examples, and additional information in the output.

Learning resources

- What is PowerShell?
- What is a command shell?
- Installing PowerShell on Windows
- PowerShell Releases
- Install PowerShell on Windows, Linux, and macOS
- PowerShell/PowerShell Releases
- Update-Help
- about_Updatable_Help

PowerShell 7 runs side-by-side with Windows PowerShell 5.1

- Migrating from Windows PowerShell 5.1 to PowerShell 7
- Differences between Windows PowerShell 5.1 and PowerShell 7.x
- Release history of modules and cmdlets
- about_Windows_PowerShell_Compatibility

Lab 2 - Command discovery and problem solving

Command discovery

1. Can you find any cmdlets capable of converting other cmdlets' output into HTML?

Answer:

```
Get-Command -Noun HTML
Get-Help HTML
```

2. How many cmdlets are available for working with processes? (Hint: remember that cmdlets all use a singular noun.)

   Answer:

```
Get-Command -Noun process | Select-Object -Property Name

Name
----
Debug-Process
Get-Process
Start-Process
Stop-Process
Wait-Process
```

3. Is there a way to shutdown a remote computer?

   Answer:

```
Stop-Computer  -ComputerName Server1
```

Problem solving

1. How would you find the Background Intelligent Transfer Service?

   Answer:

```
Get-Service -Name bits
Get-Service -DisplayName Background*
```

2. What's the current status of the Background Intelligent Transfer Service?

   Answer:

```
Get-Service -Name bits

Status   Name               DisplayName
------   ----               -----------
Running  bits               Background Intelligent Transfer Service
```

3. Is the Background Intelligent Transfer Service set to start automatically?

   Answer:

   ```
   Get-Service -Name bits | Select-Object -Property Start*Type
   ```

   In Windows PowerShell 5.1, you see output similar to:

   ```
   StartType
   ---------
   Automatic
   ```

   In PowerShell 7, you see output similar to:

   ```
            StartupType StartType
            ----------- ---------
   AutomaticDelayedStart Automatic
   ```

4. How would you stop the Background Intelligent Transfer Service?

   Answer:

   There are several ways to stop the service:

   ```
   Stop-Service -Name bits
   Get-Service -Name bits | Stop-Service
   (Get-Service -Name bits).Stop()
   ```

Learning resources

- Discover PowerShell
- Get-Help
- about_Command_Syntax
- Get-Command
- Get-Member
- Get-Verb
- Stop-Computer
- Get-Service
- Stop-Service
- Select-Object
- How to use the PowerShell documentation

Lab 3 - Modules & Pipeline

Modules

1. What modules are available on your system?

   Answer:

   ```
   Get-Module -ListAvailable
   ```

2. What is the latest version of the Pester module available in the PowerShell Gallery?

   Answer:

   There are several ways to find the latest version of the Pester module:

   - Search for the module in the PowerShell Gallery
   - Use the `Find-Module` cmdlet from the PowerShellGet module

     ```
     Find-Module -Name Pester

     Version       Name     Repository    Description
     -------       ----     ----------    -----------
     5.7.1         Pester   PSGallery     Pester provides a framework for
     running …

     Find-Module -Name Pester -AllowPrerelease

     Version       Name     Repository    Description
     -------       ----     ----------    -----------
     6.0.0-alpha5  Pester   PSGallery     Pester provides a framework for
     running …
     ```

   - Use the `Find-PSResource` cmdlet from the Microsoft.PowerShell.PSResourceGet module

     ```
     Find-PSResource -Name Pester

     Name   Version Prerelease Repository Description
     ----   ------- ---------- ---------- -----------
     Pester 5.7.1              PSGallery  Pester provides a framework for
     running BDD style Tests to execute and validate P…

     Find-PSResource -Name Pester -Prerelease

     Name   Version Prerelease Repository Description
     ```

```
----   ------- ---------- ---------- -----------
Pester 6.0.0   alpha5     PSGallery  Pester provides a framework for
running BDD style Tests to execute and validate P…
```

Using the pipeline

1. What's difference between the following commands?

```
Get-ChildItem -File | Format-Table Name, Length | Sort-Object Length
Get-ChildItem -File | Format-Table Name, Length | Get-Member
Get-ChildItem -File | Sort-Object Length | Format-Table Name, Length
```

Answer:

- The first command results in an error because `Format-Table` does not output objects that can be sorted.
- The second command uses `Get-Member` to display information about the objects returned by `Format-Table`. You will see objects of the following types:

```
TypeName: Microsoft.PowerShell.Commands.Internal.Format.FormatStartData
TypeName: Microsoft.PowerShell.Commands.Internal.Format.GroupStartData
TypeName: Microsoft.PowerShell.Commands.Internal.Format.FormatEntryData
TypeName: Microsoft.PowerShell.Commands.Internal.Format.GroupEndData
TypeName: Microsoft.PowerShell.Commands.Internal.Format.FormatEndData
```

- The third command sorts the objects by their length before passing them to `Format-Table`, which works as expected.

2. What properties are available on the objects returned by `Get-Process`? How can you display all of them?

Answer:

The `Get-Member` cmdlet shows you the properties of the objects returned by `Get-Process`. Also, you can use the `Select-Object` cmdlet to display all properties and their values.

```
Get-Process | Get-Member -MemberType Property
Get-Process | Select-Object -Property *
```

3. Run the following command and compare the output:

```
Get-Process pwsh
Get-Process pwsh | Format-Table
```

```
Get-Process pwsh | Format-List
Get-Process pwsh | Select-Object -Property *
```

Answer:

The output of the first two commands is identical. By default, the formatting system in PowerShell outputs process objects in a table format. When you run these command in PowerShell 7, notice that some of the column headers are displayed in italics. This indicates that the column names don't match the actual property names.

The third command formats the output as a list. Notice that you see different properties in the output of the third command compared to the first two. The last command uses `Select-Object` to display all properties of the process object, which includes properties that are not shown by default in the table or list formats.

Learning resources

- about_Modules
- about_PSModulePath
- Get-Module
- Install-Module (PowerShellGet)
- Find-Module (PowerShellGet)
- Install-PSResource
- Find-PSResource
- about_Pipelines
- Get-ChildItem
- Sort-Object
- Format-Table
- Format-List
- about_Parameter_Binding
- Visualize parameter binding

Lab 4 - Providers, Functions, & Scripts

Create a function

Create a function called `Get-AssetInfo` that retrieves information about the computer system and BIOS. The function should accept a parameter for the computer name, defaulting to the local computer if none is provided. It should return a custom object with properties such as **ComputerName**, **Manufacturer**, **Model**, **BIOSVersion**, and **SerialNumber**.

Answer:

Your code is probably different. This example shows how you implement advanced features like pipeline support and creating custom objects for output. You should always try to create objects for output instead of

trying to output formatted strings. That way, your function can be used in pipelines and with other cmdlets that expect objects as input.

```
function Get-AssetInfo {

    param(
        [Parameter(ValueFromPipeline, ValueFromPipelineByPropertyName)]
        [string[]]$ComputerName
    )

    begin {
        if ($null -eq $ComputerName) {
            $ComputerName = $env:COMPUTERNAME
        }
    }

    process {
        foreach ($name in $ComputerName) {
            $CompSys = Get-CimInstance Win32_ComputerSystem -ComputerName $name
            $SysBios = Get-CimInstance Win32_BIOS -ComputerName $name
            $SysEncl = Get-CimInstance Win32_SystemEnclosure -ComputerName $name
            $SysDisk = Get-CimInstance Win32_DiskDrive -ComputerName $name
            $SysProc = Get-CimInstance Win32_Processor -ComputerName $name

            [pscustomobject]@{
                ComputerName = $CompSys.Name
                Manufacturer = $CompSys.Manufacturer
                Model = $CompSys.Model
                BIOSVersion = $SysBios.BIOSVersion
                AssetTag = $SysEncl.SMBIOSAssetTag
                SerialNumber = $SysBios.SerialNumber
                TotalRAM = '{0:N2} GB' -f ($CompSys.TotalPhysicalMemory / 1GB)
                DiskSize = $SysDisk.Size | %{ '{0:N2} GB' -f ($_ / 1GB)}
                Processor = ($SysProc.Name -join ', ')
            }
        }
    }

}
```

Learning resources

- about_Providers
- Get-PSDrive
- about_Function_Provider
- about_Registry_Provider
- about_Functions

- about_Functions_Advanced_Parameters
- about_PSCustomObject
- about_Command_Precedence

Lab 5 - Create a profile

1. Create a profile script

   Answer:

   You can use the following command to create an empty profile script in the proper location:

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

   This command creates the file and any parent directories that don't exist. You can then open the profile script in your favorite text editor and add your commands.

2. Add the function you created in Lab 4

3. Configure PSReadLine key bindings for the following keys:

Key chord	Function
Enter	ValidateAndAcceptLine
Alt+a	SelectCommandArgument
F1	ShowCommandHelp
Alt+h	ShowParameterHelp

The final profile script might look like this:

```
<#
    My Profile Script
    Updated: 2025-08-06
#>
#---------------------------------
#region Define helper functions
function Get-AssetInfo {

    param(
        [Parameter(ValueFromPipeline, ValueFromPipelineByPropertyName)]
        [string[]]$ComputerName
    )
```

```
    begin {
        if ($null -eq $ComputerName) {
            $ComputerName = $env:COMPUTERNAME
        }
    }

    process {
        foreach ($name in $ComputerName) {
            $CompSys = Get-CimInstance Win32_ComputerSystem -ComputerName $name
            $SysBios = Get-CimInstance Win32_BIOS -ComputerName $name
            $SysEncl = Get-CimInstance Win32_SystemEnclosure -ComputerName $name
            $SysDisk = Get-CimInstance Win32_DiskDrive -ComputerName $name
            $SysProc = Get-CimInstance Win32_Processor -ComputerName $name

            [pscustomobject]@{
                ComputerName = $CompSys.Name
                Manufacturer = $CompSys.Manufacturer
                Model = $CompSys.Model
                BIOSVersion = $SysBios.BIOSVersion
                AssetTag = $SysEncl.SMBIOSAssetTag
                SerialNumber = $SysBios.SerialNumber
                TotalRAM = '{0:N2} GB' -f ($CompSys.TotalPhysicalMemory / 1GB)
                DiskSize = $SysDisk.Size | %{ '{0:N2} GB' -f ($_ / 1GB)}
                Processor = ($SysProc.Name -join ', ')
            }
        }
    }

}
#endregion
#---------------------------------
#region Configure PSReadLine
Set-PSReadLineKeyHandler -Key Enter -Function ValidateAndAcceptLine
Set-PSReadLineKeyHandler -Key Alt+a -Function SelectCommandArgument
Set-PSReadLineKeyHandler -Key F1 -Function ShowCommandHelp
Set-PSReadLineKeyHandler -Key Alt+h -Function ShowParameterHelp
#endregion
#---------------------------------
```

Learning resources

- about_Profiles
- about_Prompts
- Get-PSReadLineOption
- Set-PSReadLineOption
- Set-PSReadLineKeyHandler
- Using tab-completion in the shell

- Using predictors in PSReadLine
- Using dynamic help
- Customizing your shell environment
- Using PSReadLine key handlers

Lab 6 - PowerShell Remoting

There is no Lab 6 in this workshop.

Learning resources

- about_Remote
- about_Remote_Requirements
- about_Remote_Troubleshooting
- Running Remote Commands
- Using WS-Management (WSMan) Remoting in PowerShell
- PowerShell Remoting Over SSH
- PowerShell Remoting FAQ
- about_Windows_PowerShell_Compatibility
- PowerShell 7 module compatibility in Windows Server 2025

Other useful links

History of PowerShell

- The Monad Manifesto

Learning PowerShell

- PowerShell 101 book by Mike F. Robbins
- What's New in PowerShell-Docs for 2025
- What's New in PowerShell 7.5

Presenter content

- Sean's blog and presentations
- Steven's blog and presentations

Community resources

- PowerShell community support resources
- PowerShell Virtual User Group
  - Slack
  - Discord
- The PowerShell Podcast