**Workshop Overview:**

This all-day workshop is designed to introduce participants to the basics of Microsoft PowerShell. Through a combination of lectures, demonstrations, and hands-on lab exercises, students will learn how to use PowerShell for automation, configuration management, and general scripting tasks.

**Syllabus and Teaching Plan**

**Session 1: Introduction to PowerShell**

• **Duration:** 1 hour

• **Topics Covered:**

• What is PowerShell?

• History and evolution of PowerShell

• PowerShell vs. Command Prompt vs. Bash

• Installing and updating PowerShell

• **Student Handout Information:**

• PowerShell basics cheat sheet

• PowerShell installation guide

• **Lab Exercise 1:**

• Installing PowerShell on Windows and Linux

• Opening and navigating PowerShell

**Session 2: PowerShell Cmdlets and Syntax**

• **Duration:** 1.5 hours

• **Topics Covered:**

• Understanding cmdlets

• Common cmdlets: Get-Help, Get-Command, Get-Member

• Aliases in PowerShell

• Using and understanding parameters

• **Student Handout Information:**

• List of common cmdlets and their usage

• Cmdlet syntax guide

• **Lab Exercise 2:**

• Exploring Get-Help and Get-Command

• Using Get-Member to inspect objects

• Creating and using aliases

**Session 3: Working with the PowerShell Pipeline**

• **Duration:** 1 hour

• **Topics Covered:**

• Understanding the pipeline

• Using the pipeline to pass data between cmdlets

• Filtering, sorting, and selecting data

• **Student Handout Information:**

• Pipeline usage guide

• Common pipeline cmdlets and examples

• **Lab Exercise 3:**

• Practicing pipeline commands

• Filtering and sorting output data

**Lunch Break**

• **Duration:** 1 hour

**Session 4: Variables, Arrays, and Hash Tables**

• **Duration:** 1.5 hours

• **Topics Covered:**

• Defining and using variables

• Understanding and using arrays

• Working with hash tables

• **Student Handout Information:**

• Variable and array usage guide

• Hash table quick reference

• **Lab Exercise 4:**

• Creating and manipulating variables

• Working with arrays and hash tables in scripts

**Session 5: Scripting Basics**

• **Duration:** 1.5 hours

• **Topics Covered:**

• Writing simple scripts

• Understanding script structure

• Script execution policies

• Handling errors and exceptions

• **Student Handout Information:**

• Script writing guide

• Common scripting pitfalls and solutions

• **Lab Exercise 5:**

• Writing a basic script

• Running and troubleshooting scripts

**Session 6: PowerShell Remoting and Automation**

• **Duration:** 1 hour

• **Topics Covered:**

• Introduction to PowerShell remoting

• Setting up and using remoting

• Automating tasks with scheduled jobs and tasks

• **Student Handout Information:**

• Remoting setup guide

• Automation tips and best practices

• **Lab Exercise 6:**

• Setting up a remote session

• Automating a simple task using a script

**Session 7: Introduction to PowerShell Modules**

• **Duration:** 1 hour

• **Topics Covered:**

• Understanding modules

• Finding and installing modules from the PowerShell Gallery

• Creating and using custom modules

• **Student Handout Information:**

• Module usage guide

• List of useful modules from PowerShell Gallery

• **Lab Exercise 7:**

• Installing a module from the PowerShell Gallery

• Creating a simple custom module

**Student Information:**

• **Prerequisites:** Basic understanding of command-line interfaces

• **Materials Provided:** Laptop with PowerShell installed, workshop handouts, lab exercise guide

• **What to Bring:** Personal laptop (if preferred), notebook, pen

**Student Handout Information:**

1. **PowerShell Basics Cheat Sheet:** Overview of key concepts, commands, and syntax.

2. **Cmdlet Syntax Guide:** Detailed guide on cmdlet structure and usage.

3. **Pipeline Usage Guide:** Examples and explanations of how to use the pipeline effectively.

4. **Variable and Array Usage Guide:** Instructions and examples for using variables, arrays, and hash tables.

5. **Script Writing Guide:** Best practices for writing and troubleshooting PowerShell scripts.

6. **Remoting Setup Guide:** Step-by-step instructions for setting up and using PowerShell remoting.

7. **Module Usage Guide:** Information on finding, installing, and creating PowerShell modules.

**Lab Exercises:**

1. **Installing PowerShell:** Step-by-step installation on various platforms.

2. **Exploring Cmdlets:** Using Get-Help, Get-Command, and Get-Member.

3. **Using the Pipeline:** Filtering and sorting data in the pipeline.

4. **Variables and Arrays:** Creating and manipulating variables, arrays, and hash tables.

5. **Writing Scripts:** Developing and running basic PowerShell scripts.

6. **Remoting and Automation:** Setting up remoting and automating tasks.

7. **Working with Modules:** Installing and creating PowerShell modules.

**Conclusion and Q&A**

• **Duration:** 30 minutes

• **Activities:**

• Recap of key topics covered

• Q&A session

• Feedback collection

This workshop plan is designed to provide a comprehensive introduction to PowerShell, combining theoretical knowledge with practical exercises to ensure students gain a solid foundation in PowerShell.

### Lab Guide 1: Installing PowerShell

#### Objective:

Learn how to install PowerShell on various platforms.

#### Instructions:

\*\*Windows:\*\*

1. Open a web browser and navigate to the [PowerShell GitHub releases page](https://github.com/PowerShell/PowerShell/releases).

2. Download the latest `.msi` package for Windows.

3. Run the installer and follow the prompts:

- Accept the license agreement.

- Choose the installation directory.

- Select additional options as needed (e.g., adding PowerShell to the PATH environment variable).

4. Once the installation is complete, open PowerShell by searching for "PowerShell" in the Start menu and selecting "Windows PowerShell" or "PowerShell".

\*\*Linux:\*\*

1. Open a terminal.

2. Depending on your Linux distribution, follow the appropriate commands to install PowerShell:

\*\*Debian/Ubuntu:\*\*

```bash

sudo apt update

sudo apt install -y wget apt-transport-https software-properties-common

wget -q https://packages.microsoft.com/config/ubuntu/20.04/packages-microsoft-prod.deb

sudo dpkg -i packages-microsoft-prod.deb

sudo apt update

sudo apt install -y powershell

```

Start PowerShell by typing `pwsh`.

\*\*CentOS/RHEL:\*\*

```bash

sudo yum install -y wget

sudo wget -q https://packages.microsoft.com/config/rhel/7/packages-microsoft-prod.rpm

sudo rpm -Uvh packages-microsoft-prod.rpm

sudo yum install -y powershell

```

Start PowerShell by typing `pwsh`.

\*\*macOS:\*\*

1. Open a terminal.

2. Install Homebrew if it is not already installed:

```bash

/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"

```

3. Use Homebrew to install PowerShell:

```bash

brew update

brew install --cask powershell

```

Start PowerShell by typing `pwsh`.

#### Verification:

1. Open PowerShell by typing `powershell` (Windows) or `pwsh` (Linux/macOS).

2. Verify the installation by checking the version:

```powershell

$PSVersionTable.PSVersion

```

---

### Lab Guide 2: Exploring Cmdlets

#### Objective:

Learn how to use Get-Help, Get-Command, and Get-Member cmdlets.

#### Instructions:

1. \*\*Using Get-Help:\*\*

- Open PowerShell.

- Display help for a cmdlet:

```powershell

Get-Help Get-Process

```

- Display detailed help:

```powershell

Get-Help Get-Process -Detailed

```

- Update the help content:

```powershell

Update-Help

```

2. \*\*Using Get-Command:\*\*

- List all available cmdlets:

```powershell

Get-Command

```

- Find a specific cmdlet:

```powershell

Get-Command \*service\*

```

3. \*\*Using Get-Member:\*\*

- Retrieve properties and methods of an object:

```powershell

Get-Process | Get-Member

```

#### Exercises:

1. Use `Get-Help` to find information about the `Get-Service` cmdlet.

2. Use `Get-Command` to find cmdlets related to files.

3. Use `Get-Member` to explore properties and methods of the `Get-Service` output.

---

### Lab Guide 3: Using the Pipeline

#### Objective:

Learn how to filter and sort data using the PowerShell pipeline.

#### Instructions:

1. \*\*Filtering Data:\*\*

- Open PowerShell.

- Use `Where-Object` to filter processes:

```powershell

Get-Process | Where-Object { $\_.CPU -gt 10 }

```

2. \*\*Sorting Data:\*\*

- Sort processes by CPU usage:

```powershell

Get-Process | Sort-Object -Property CPU -Descending

```

3. \*\*Selecting Data:\*\*

- Select specific properties of processes:

```powershell

Get-Process | Select-Object -Property Name, CPU

```

#### Exercises:

1. List all services that are currently running.

2. Sort the list of running processes by memory usage.

3. Select the name and status of all services and display them.

---

### Lab Guide 4: Variables and Arrays

#### Objective:

Learn how to create and manipulate variables, arrays, and hash tables.

#### Instructions:

1. \*\*Creating Variables:\*\*

- Open PowerShell.

- Create a variable:

```powershell

$greeting = "Hello, PowerShell!"

```

2. \*\*Creating Arrays:\*\*

- Create an array:

```powershell

$numbers = 1, 2, 3, 4, 5

```

3. \*\*Creating Hash Tables:\*\*

- Create a hash table:

```powershell

$person = @{

Name = "John Doe"

Age = 30

Occupation = "Developer"

}

```

4. \*\*Manipulating Variables:\*\*

- Access array elements:

```powershell

$numbers[0]

```

- Add elements to an array:

```powershell

$numbers += 6

```

- Access hash table values:

```powershell

$person["Name"]

```

#### Exercises:

1. Create a variable to store your name.

2. Create an array of the first five prime numbers.

3. Create a hash table to store information about your favorite book.

---

### Lab Guide 5: Writing Scripts

#### Objective:

Learn how to write and run basic PowerShell scripts.

#### Instructions:

1. \*\*Writing a Script:\*\*

- Open a text editor (e.g., Notepad).

- Write a simple script:

```powershell

# My first script

$greeting = "Hello, PowerShell!"

Write-Output $greeting

```

2. \*\*Saving a Script:\*\*

- Save the file with a `.ps1` extension, e.g., `MyScript.ps1`.

3. \*\*Running a Script:\*\*

- Open PowerShell.

- Navigate to the directory where the script is saved.

- Run the script:

```powershell

.\MyScript.ps1

```

4. \*\*Handling Errors:\*\*

- Add error handling to your script:

```powershell

try {

Get-Process -Name "NonExistentProcess"

} catch {

Write-Output "Process not found."

}

```

#### Exercises:

1. Write a script that prints "Hello, [Your Name]!".

2. Modify the script to handle cases where a non-existent process is queried.

3. Save and run your script.

---

### Lab Guide 6: Remoting and Automation

#### Objective:

Learn how to set up PowerShell remoting and automate tasks.

#### Instructions:

1. \*\*Setting Up Remoting:\*\*

- Open PowerShell with administrative privileges.

- Enable remoting:

```powershell

Enable-PSRemoting -Force

```

2. \*\*Using Remoting:\*\*

- Start a remote session:

```powershell

Enter-PSSession -ComputerName RemoteComputerName

```

- Run a command on a remote computer:

```powershell

Invoke-Command -ComputerName RemoteComputerName -ScriptBlock { Get-Process }

```

3. \*\*Automating Tasks:\*\*

- Create a scheduled task:

```powershell

$action = New-ScheduledTaskAction -Execute 'PowerShell.exe' -Argument '-File C:\Scripts\MyScript.ps1'

$trigger = New-ScheduledTaskTrigger -Daily -At 7am

Register-ScheduledTask -Action $action -Trigger $trigger -TaskName "MyDailyTask"

```

#### Exercises:

1. Enable remoting on your machine.

2. Start a remote session to another computer (use a virtual machine if needed).

3. Create a scheduled task to run a script at a specific time.

---

### Lab Guide 7: Working with Modules

#### Objective:

Learn how to install and create PowerShell modules.

#### Instructions:

1. \*\*Installing Modules:\*\*

- Open PowerShell.

- Find a module in the PowerShell Gallery:

```powershell

Find-Module -Name PSReadLine

```

- Install the module:

```powershell

Install-Module -Name PSReadLine

```

2. \*\*Using Modules:\*\*

- Import a module:

```powershell

Import-Module -Name PSReadLine

```

3. \*\*Creating Modules:\*\*

- Create a module file:

```powershell

function Get-Greeting {

param ($Name)

"Hello, $Name!"

}

Export-ModuleMember -Function Get-Greeting

```

- Save the file with a `.psm1` extension, e.g., `MyModule.psm1`.

4. \*\*Using Custom Modules:\*\*

- Import your custom module:

```powershell

Import-Module -Name C:\Path\To\MyModule.psm1

```

- Use the function from your module:

```powershell

Get-Greeting -Name "PowerShell"

```

#### Exercises:

1. Install a module from the PowerShell Gallery.

2. Create a custom module with a function that prints a greeting.

3. Import and use your custom module.

### 1. PowerShell Basics Cheat Sheet: Overview of Key Concepts, Commands, and Syntax

\*\*PowerShell Basics Cheat Sheet\*\*

\*\*Basic Commands:\*\*

- `Get-Help <cmdlet>`: Displays help about cmdlets.

- `Get-Command`: Lists all available cmdlets.

- `Get-Alias`: Lists all aliases for cmdlets.

- `Get-Process`: Displays all running processes.

- `Get-Service`: Lists all services on the system.

\*\*Common Aliases:\*\*

- `ls`: Alias for `Get-ChildItem`.

- `cd`: Alias for `Set-Location`.

- `dir`: Alias for `Get-ChildItem`.

- `gc`: Alias for `Get-Content`.

\*\*Navigation:\*\*

- `Get-Location`: Displays the current directory.

- `Set-Location <path>`: Changes the current directory.

\*\*File Operations:\*\*

- `Get-ChildItem`: Lists items in a directory.

- `Copy-Item <source> <destination>`: Copies an item.

- `Move-Item <source> <destination>`: Moves an item.

- `Remove-Item <path>`: Deletes an item.

- `New-Item -Path <path> -ItemType <type>`: Creates a new item.

\*\*Variables:\*\*

- `$variable = <value>`: Declares a variable.

- `$variable`: Displays the value of a variable.

\*\*Useful Commands:\*\*

- `Start-Process <program>`: Starts a new process.

- `Stop-Process -Name <process\_name>`: Stops a process by name.

- `Get-EventLog -LogName <log\_name>`: Retrieves event log entries.

\*\*Control Flow:\*\*

- `if (<condition>) { <statement> }`: Conditional statement.

- `foreach ($item in $collection) { <statement> }`: Iterates over a collection.

\*\*For more details:\*\* [Microsoft PowerShell Documentation](https://docs.microsoft.com/en-us/powershell/)

---

### 2. Cmdlet Syntax Guide: Detailed Guide on Cmdlet Structure and Usage

\*\*Cmdlet Syntax Guide\*\*

\*\*Cmdlet Structure:\*\*

- `<Verb>-<Noun> -Parameter <Value>`

\*\*Common Cmdlet Verbs:\*\*

- `Get`: Retrieves data.

- `Set`: Sets or changes data.

- `New`: Creates a new item.

- `Remove`: Deletes an item.

- `Start`: Starts an operation.

- `Stop`: Stops an operation.

\*\*Parameters:\*\*

- Mandatory parameters are required.

- Optional parameters are not required but can be used to modify behavior.

\*\*Examples:\*\*

- `Get-Process -Name "notepad"`: Retrieves the process named "notepad".

- `Set-Service -Name "Spooler" -StartupType Automatic`: Sets the Spooler service to start automatically.

- `New-Item -Path "C:\Temp\NewFolder" -ItemType Directory`: Creates a new directory.

\*\*Using Get-Help:\*\*

- `Get-Help <cmdlet> -Detailed`: Provides detailed information about a cmdlet.

- `Get-Help <cmdlet> -Examples`: Provides usage examples of a cmdlet.

\*\*For more details:\*\* [Cmdlet Overview - PowerShell](https://docs.microsoft.com/en-us/powershell/scripting/developer/cmdlet/cmdlet-overview)

---

### 3. Pipeline Usage Guide: Examples and Explanations of How to Use the Pipeline Effectively

\*\*Pipeline Usage Guide\*\*

\*\*Basics of the Pipeline:\*\*

- The pipeline (`|`) passes the output of one cmdlet as input to another cmdlet.

\*\*Examples:\*\*

- `Get-Process | Sort-Object -Property CPU`: Sorts processes by CPU usage.

- `Get-Service | Where-Object { $\_.Status -eq 'Running' }`: Filters running services.

\*\*Pipeline Operators:\*\*

- `Where-Object`: Filters objects based on a condition.

- `Sort-Object`: Sorts objects by property.

- `Select-Object`: Selects specific properties of objects.

- `ForEach-Object`: Performs an operation on each item in the pipeline.

\*\*Advanced Usage:\*\*

- `Get-ChildItem | Where-Object { $\_.Length -gt 1MB } | Select-Object Name, Length`: Lists files larger than 1MB and displays their name and size.

\*\*For more details:\*\* [Working with the Pipeline - PowerShell](https://docs.microsoft.com/en-us/powershell/scripting/learn/deep-dives/everything-about-pipeline)

---

### 4. Variable and Array Usage Guide: Instructions and Examples for Using Variables, Arrays, and Hash Tables

\*\*Variable and Array Usage Guide\*\*

\*\*Variables:\*\*

- `$variable = <value>`: Declares a variable.

- `$variable`: Accesses the value of a variable.

\*\*Arrays:\*\*

- `$array = @(1, 2, 3, 4, 5)`: Creates an array.

- `$array[0]`: Accesses the first element of the array.

\*\*Hash Tables:\*\*

- `$hashTable = @{Key1="Value1"; Key2="Value2"}`: Creates a hash table.

- `$hashTable["Key1"]`: Accesses the value associated with "Key1".

\*\*Examples:\*\*

- Declaring and using a variable:

```powershell

$name = "John"

Write-Output $name

```

- Creating and accessing an array:

```powershell

$numbers = @(1, 2, 3, 4, 5)

$numbers[2]

```

- Creating and accessing a hash table:

```powershell

$person = @{

Name = "Jane Doe"

Age = 28

Occupation = "Engineer"

}

$person["Name"]

```

\*\*For more details:\*\* [Variables - PowerShell](https://docs.microsoft.com/en-us/powershell/scripting/learn/deep-dives/everything-about-variables)

---

### 5. Script Writing Guide: Best Practices for Writing and Troubleshooting PowerShell Scripts

\*\*Script Writing Guide\*\*

\*\*Best Practices:\*\*

- \*\*Use Comments:\*\* Add comments to describe your code.

```powershell

# This script prints a greeting

Write-Output "Hello, World!"

```

- \*\*Use Meaningful Names:\*\* Name variables and functions meaningfully.

```powershell

$userName = "John"

```

- \*\*Error Handling:\*\* Use try/catch for error handling.

```powershell

try {

Get-Process -Name "NonExistentProcess"

} catch {

Write-Output "Process not found."

}

```

- \*\*Modular Code:\*\* Break down complex scripts into functions.

```powershell

function Get-Greeting {

param ($name)

"Hello, $name!"

}

```

\*\*Running Scripts:\*\*

- Save the script with a `.ps1` extension.

- Run the script:

```powershell

.\MyScript.ps1

```

\*\*Troubleshooting:\*\*

- \*\*Debugging:\*\* Use `Write-Output` or `Write-Host` to print values.

- \*\*Verbose Mode:\*\* Use `-Verbose` parameter to get detailed output.

\*\*For more details:\*\* [Writing Scripts - PowerShell](https://docs.microsoft.com/en-us/powershell/scripting/learn/deep-dives/everything-about-script-writing)

---

### 6. Remoting Setup Guide: Step-by-Step Instructions for Setting Up and Using PowerShell Remoting

\*\*Remoting Setup Guide\*\*

\*\*Enabling Remoting:\*\*

- Open PowerShell with administrative privileges.

- Enable remoting:

```powershell

Enable-PSRemoting -Force

```

\*\*Using Remoting:\*\*

- Start a remote session:

```powershell

Enter-PSSession -ComputerName RemoteComputerName

```

- Run a command on a remote computer:

```powershell

Invoke-Command -ComputerName RemoteComputerName -ScriptBlock { Get-Process }

```

\*\*Configuring Trusted Hosts:\*\*

- Add a computer to trusted hosts:

```powershell

Set-Item WSMan:\localhost\Client\TrustedHosts -Value "RemoteComputerName"

```

\*\*Using Credentials:\*\*

- Provide credentials for remoting:

```powershell

$cred = Get-Credential

Enter-PSSession -ComputerName RemoteComputerName -Credential $cred

```

\*\*For more details:\*\* [PowerShell Remoting - PowerShell](https://docs.microsoft.com/en-us/powershell/scripting/learn/remoting)

---

### 7. Module Usage Guide: Information on Finding, Installing, and Creating PowerShell Modules

\*\*Module Usage Guide\*\*

\*\*Finding Modules:\*\*

- Search for modules in the PowerShell Gallery:

```powershell

Find-Module -Name <module\_name>

```

\*\*Installing Modules:\*\*

- Install a module from the PowerShell Gallery:

```powershell

Install-Module -Name <module\_name>

```

\*\*Using Modules:\*\*

- Import a module:

```powershell

Import-Module -Name <module\_name>

```

- List imported modules:

```powershell

Get-Module

```

\*\*Creating Modules:\*\*

- Create a module file (e.g., `MyModule.psm1`):

```powershell

function Get-Greeting {

param ($Name)

"Hello, $Name!"

}

Export-ModuleMember -Function Get-Greeting

```

- Save the file with a `.psm1` extension.

\*\*Importing Custom Modules:\*\*

- Import the custom module:

```powershell

Import-Module

-Name C:\Path\To\MyModule.psm1

```

- Use the function from the module:

```powershell

Get-Greeting -Name "PowerShell"

```

\*\*For more details:\*\* [Managing Modules - PowerShell](https://docs.microsoft.com/en-us/powershell/scripting/learn/deep-dives/everything-about-modules)