PowerShell CheatSheet

# 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.

# 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.

# 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.

# 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"]

```

# 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.

# 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

```

# 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"

```