Powershell Basics

Table Of Conents

Basic Commands

Get-Command

Displays the commands that PS is aware of.

```
# Gets information about the supplied command.
Get-Command [Command]

# Returns commands that use the verb.
Get-Command -Verb [Verb]

# Returns commands that use the noun.
Get-Command -Noun [Noun]

# Returns commands for the verb and noun combo.
Get-Command -Verb [Verb] -Noun [Noun]
```

Get-Content

Gets content from a resource.

```
# Retrieves the content of the file.
Get-Content -Path [Path to file]
```

Get-Help

Shows the documentation for a given command.

```
# Displays limited information about the command.
Get-Help [Name of command]

# Displays the full documentation for a command.
Get-Help [Name of command] -Full

# Wildcard search for command help
Get-Help -Name [Name of command]*
```

Get-Member

Returns a list of all methods and properties of an object.

```
# Returns a list of all methods and properties of an object.
Get-Member -InputObject [Object]
```

Get-Service

Returns a list of all services running on the OS.

```
# Returns all running services.
Get-Service

# Returns a service by name.
Get-Service [Name]
```

Get-Variable

Returns a list of all variables, both user defined and built in.

```
# Returns a full list of all predefined and user variables
Get-Variable

# Returns the name and value for a variable.
Get-Variable -Name [Variable]
```

Resolve-DnsName

Resolve DNS information for a given IP address.

```
# Resolves DNS information for a given IP
Resolve-DnsName -Name
```

Test-Connection

Pings a host and returns information.

```
# Pings Host.
Test-Connection [Host]

# Pings with hostname.
Test-Connection -ComputerName [Hostname]

# Change number of ICMP packets sent. Default 1.
Test-Connection [Host] -Count [Count]

# Only return bool for result.
Test-Connection [Host] -Quiet
```

Update-Help

Compares local documentation for commands against an online repository and downloads new docs if needed. May Require Admin.

```
# Updates documentation for powershell commands.
Update-Help
```

Config

Set-ExecutionPolicy

Sets the execution policy for scripts.

```
Set-ExecutionPolicy -ExecutionPolicy [Policy]
```

Set-StrictMode

Turns on many errors that are off by default.

```
Set-StrictMode -Version Latest
```

Errors

Try, Catch, Finally

Flow control and error handling.
Note that finally is optional. If it is set then it will run, even after an error was caught.
Can only find terminating errors.

Use $_.Exception.Message in the catch block return just the message.

```
try {
    # Something Dangerous
} catch {
    # Error Occured
} finally {
    # Final code
}
```

$Error

Stores an array of historical errors. Newest errors are added to the front of the array.

Flow Control

```
# not
-not (condition)

# if-else-if
if(statement) {
    # Something
} elseif {
    # Something
} else {
    # Something
}

# switch
switch (statement) {
    caseOne {
        # Something
    }
    caseTwo {
        # Something
    }
    default {
        # Something
    }
}

# foreach
foreach($something in $array) {
    # Something

    ** Modifying an array element inside here does not modify the original array.
}

# for-each
$array | ForEach-Object - Process {
    # Something

    ** Modifying an array element inside here does not modify the original array.
}

# for loop
for($i = 0; $i -lt 10; $i++) {
    # Something
}

#while
while (condition) {
    # Something
}

#do while
do {
    # Something
} while (condition)

# do until
do {
    # Something
} until (condition)
```

Powershell Functions

```
function Write-Name {

    param (

        # Mandatory and allow piping
        [Parameter(Mandatory, ValueFromPipeline)]
        [string] $Name,

        # Optional named parameter w/ validation
        [Parameter()]
        [ValidateSet(33, 44, 55)]
        [int] $Age

    )

    # Runs once before process block
    begin {
        Write-Host "Beginning"
    }

    # Runs once for every value piped in. Required if you are piping in an array
of values.
    process {
        Write-Host "My name is $Name and I am $Age years old."
    }

    # Runs once after the process block is complete.
    end {
        Write-Host "Ending"
    }

}

$Names = ('Geoff', 'Kelly', 'Ellie')
$Names | Write-Name -Age 33
```

Modules

Module Save Locations

```
System - C:\Windows\System32\WindowsPowershell\1.0\Modules
All Users - C:\Program Files\WindowsPowershell\Modules
Current User - C:\Users\<Current User>\Documents\WindowsPowerShell\Modules
```

Add a Module folder

```
$CurrentPath = [Environment]::GetEnvironmentVariable("PSModulePath", "Machine")
[Environment]::SetEnvironmentVariable("PSModulePath", $CurrentValue + ";C:\
<DIRECTORY PATH>", "Machine")
```

Get-Module

Shows the documentation for a given command.

```
# Returns Session only Module list.
Get-Module [Module Name]

# Lists system wide modules
Get-Module -ListAvailable
```

Import-Module

Imports a module manually, if auto-import fails.

```
# Imports a module
Import-Module -Name [Module name]

# Unload, then re-install a module
Import-Module -Name [Module name] -Force
```

Remove-Module

Removes a module from the current session. It does not uninstall it.

```
Remove-Module -Name [Module name]
```

Find-Module

Searches the Powershell Gallery for modules with the name or partial name.

```
Find-Module -Name [Module name]
```

Install-Module

Installs a module from the Powershell Gallery.

```
Install-Module -Name [Module name]
```

Uninstall-Module

Uninstalls a module from the system.

```
Uninstall-Module -Name [Module name]
```

Create Custom Module

1. Create a folder with the module name. The folder must be the same name as the module.
2. Create custom .psm1 file.
3. Cretae a custom module manifest:

```
New-ModuleManifest -Path '<PATH TO MANIFEST>.psd1' -Author '<AUTHOR>' -RootModule
<PSM1 FILE NAME> -Description '<DESCRIPTION>'
```

Operators

```
-eq # Equal
-ne # Not Equal
-gt # Greater than
-ge # Greater than or equal
-lt # Less than
-le # Less than or equal
```

Operator	JS
-eq	===
-ne	!==
-gt	>
-ge	>=
-lt	<
-le	<=
-contains	array.contains('value')

Parsing Files

CSV

Reading CSV Files

```
Import-Csv -Path [File path]
```

Querying CSV Files

```
Import-Csv -Path [File path] | Where-Object {$_.'[Header]' -eq '[Value]'}
```

Renaming Header

```
Import-Csv -Path [File path] -Header '[Column One]','[Column Two]','[Column
Three]'
```

Creating CSV Files

You can add the -NoTypeInformation to remove the line at the top of the CSV file saying what powershell type this file came from.

Add the -Append flag to prevent this from overwriting the file with each new line.

```
[Object] | Export-Csv -Path [Output file path].csv
```

Excel Sheets

Requires the third party module ImportExcel to be installed from the gallery.

Creating an Excel Sheet

```
[Object] | Export-Excel [File path].xlsx
```

Creating a worksheet

```
[Object] | Export-Excel [File path].xlsx -WorksheetName '[Name]'
```

Querying Worksheet Available

Returns a list of the worksheets. Use a for loop to iterate through them.

```
Get-ExcelSheetInfo -Path [File path]
```

Importing Excel Sheets

By defualt only the first worksheet will be imported.

```
Import-Excel -Path [File path] -WorksheetName '[Name]'
```

Dynamic Fields

Fields whos value is calculated on the fly, then added to the object/excel sheet.

Usually done in a for loop so that each value is set for each row.

```
[Object] | Select-Object -Porperty @{Name = '[Key Name'; Expression = {[Calculated
Expression]}}

# Create a timestamp
[OBJECT] | Select-Object -Property @{Name = 'Timestamp'; Expression = {Get-Date -
Format 'MM-dd-yy hh:mm:ss'}}}
```

JSON

Reading JSON

Raw returns a plain string from the JSON file.

```
Get-Content -Path [File path].json -Raw | ConvertFromJson
```

Creating JSON

Use the -Compress flag to obfuscate the json output.

```
[Object] | ConvertToJson -Compress
```

JSON and Web Requests

Use the latter to ignore the request status code and just show the result.

```
# Returns a response object
Invoke-RestMethod -Uri [Uri]

# Returns the Uri result
(Invoke-RestMethod -Uri [Uri]).result
```

Remote Execution

Invoke-Command

Sends a command to a remote server.

If no Session variable is set then this command establishes connection, executes the command then closes the session. Think one off. Not very fast.

Computers must be on the same AD domain and the executing computer must have priviledges on the host.

Notice that local variables don't always work on the host.

** The 'using' method can cause problems when testing with Pester. You may have to use the ArgumentList.

```

# Single line command block
Invoke-Command -ComputerName [Hostname] -ScriptBlock {[Script block to execute]}

# Execute a PS1 file
Invoke-Command -ComputerName [Hostname] -FilePath [File path].ps1

# Passes local variables to remote code
Invoke-Command -ComputerName [Hostname] -FilePath [File path].ps1 -ArgumentList
[Arguments]
```

New-PSSession

Creates a new session with a host.

```
New-PSSession -ComputerName [Hostname]
```

Get-PSSession

Returns a list of past sessions. Can be saved as a variable.

```
# Lists all available sessions
Get-PSSession

## Return a session by ID
Get-PSSession -Id [Session ID]
```

Disconnect-PSSession

Disconnects the session but allows you to connect back later. You pipe in the session from Get-Session.

```
# Disconnect all sessions
Get-PSSession | Disconnect-PSSession

# Disconnect session by id
Get-PSSession -Id [Session ID] | Disconnect-PSSession
```

Connect-PSSession

Connects to previouslly disconnected hosts, that you have already connected to.

```
Connect-PSSession -ComputerName [Hostname]
```

Remove-PSSession

Removes a previouslly connected session. You will not be allowed to connect again without creating a new session.

```
# Remove all sessions
Get-PSSession | Remove-PSSession

# Remove session by ID
Get-PSSession -Id [Session ID] | Remove-PSSession
```

Types

```
# Non-value assignment
$uniniatedVariable = $null

# Bools
$isTrue = $true
$isFalse = $false

# Arrays
$array = ('one', 'two', 'three')
$array += 'four'
$array[0] # First
$array[-1] # Last
$array[1..2] # Second through third
$array = $array.Remove('four')

# Array Lists
$arrayList = [System.Collections.ArrayList]('one','two','three')
$arrayList.Add('four')
$arrayList.Remove('four')

# Hash Tables
$hashTable = @{name='name', age=33}
$hashTable.Keys # Returns a list keys
$hasTable.Values # Returns a list of values
$hashTable['Key'] = 'Value' # Creates or Updates
$hashTable.Remove('Key')
$hashTable.ContainsKey('Key') # Bool
$hashTable.ContainsValue('Value') # Bool

# Custom Object
$customObject = [PSCustomObject]@{'key' = 'value';}
```