Your **PowerShell** Cheat Sheet

www.scriptrunner.com

ScriptRunner®
Your #1 for PowerShell Automation

CONFIGURING POWERSHELL

Set-ExecutionPolicy Unrestricted	Unrestricted allow all PowerShell scripts
Set-ExecutionPolicy RemoteSigned / AllSigned	Only allow signed PowerShell scripts
Enable-PSRemoting -SkipNetworkProfileCheck	Enable PowerShell remote access for this machine - even if there are public networks
(Get-Host).PrivateData.ErrorBackgroundcolor ='White'	Change background colour for error messages (increases contrast of red characters)

USING MODULES

Get-Module	List activated modules
Get-Module -ListAvailable	List all installed modules
Import-Module	Enable local module for current session
Find-Module	Search modules in PowerShell Gallery
Install-Module	Download and install modules from PowerShell Gallery
Update-Module	Update module

USING .NET FRAMEWORK CLASSES

Access to static members
```
[System.Environment]::MachineName
[System.Console]::Beep(800, 500)
```

Instantiation and access to instance members
```
$b = New-Object System.Directoryservices.DirectoryEntry
('WinNT://MyServer/ScriptRunner')
$b.FullName
$b.Description = 'PowerShell Automation'
$b.SetInfo()
```

Load and use additional assembly
```
[System.Reflection.Assembly]::LoadWithPartialName('Microsoft.VisualBasic')
$input = [Microsoft.VisualBasic.Interaction]::InputBox('Please enter your
Name!','Title')
```

OBJECT-ORIENTED ACCESS TO PIPELINE OBJECTS

Number of objects in pipeline
```
(Get-Service | where { $_.status -eq 'Running' }).Count
```

Print particular properties of pipeline objects
```
(Get-Date).DayOfWeek
(Get-Process).Name
(Get-Process | sort ws -desc)[0].Name
```

Method call in all pipeline objects
```
(Get-Process iexplore | sort ws -desc).Kill()
```

STRINGS AND EXPRESSIONS

Embedding of a variable in a string
```
"The command is $Command!"
```

{} must be used here to delimit it from the colon
```
"${Command}: executed successfully"
```

The subexpression must be parenthesized in $( )
```
"$($Result.Count) objects in result set"
```

Use of the format operator
```
Get-Process | % { '{0,-40} uses {1:0,000.00}MB' -f $_.Name, ($_.ws/1MB)
}
```

Execute a string as a command
```
$Command = 'Get-Service a*'
$Command += "| where status -eq 'Running'"
$Result = Invoke-Expression $Command
$Command | Format-List
$Result | Format-List
```

POWERSHELL SCRIPTING LANGUAGE

Condition
```
if ((Get-Date).Year -le 2014) { 'Old' } else { 'New' }
```

Loops
```
for($i = 1; $i -le 10; $i++) { $i }
while($i -le 10) { $i; $i++ }
do { $i; $i++ } while ($i -le 10)
foreach ($p in (Get-Process iexplore)) { $p.Kill() }
```

Subroutines with mandatory parameters and optional parameters
```
function Get-DLL([Parameter(Mandatory=$true)][string]$root,
[string]$filter = '*')
{
return Get-ChildItem $root -Filter "$filter.dll"
}
Get-DLL c:\Windows\System32
```

Comment
```
# This is a comment
```

POWERSHELL DATA TYPES

[byte] [int] [long] [single] [double]	Numeric types
[byte] $x = Get-Random -Minimum 1 -Maximum 49	Generate random number between 1 and 49 and store in variable $x
[char] [string]	Character types
[bool] [DateTime]	Boolean and date types

[Datetime] $d = Get-Date	Store current date in variable $d
[Array] [Hashtable]	Object sets
[Array] $services = Get-Service a*	Store list of services starting with "a" in variable $services
[XML] [WMI] [ADSI]	More complex data structures
[psobject].Assembly.GetType ('System.Management.Automation.TypeAccelerators')::Get	A complete list of TypeAccelerators is accessible

SECRET MANAGEMENT MODULE

Install-Module -Name Microsoft.PowerShell.SecretManagement	Install the Secret Management module
Install-Module -Name Microsoft.PowerShell.SecretStore	Install the Secret Store module
Register-MySecretVault -Name SecretStore -ModuleName Microsoft.PowerShell.SecretStore -DefaultVault	Register a local secret vault
Get-SecretVault	Show current secret vaults
Get-SecretStoreConfiguration	Show secret store configuration
Get-SecretInfo	Show list of existing secrets
Get-Secret -Name MySecret	Show details of a secret
Set-Secret	Create a new secret
Set-Secret -Name 'NewCred001' -Secret (Get-Credential 'user@mycompany.com')	Create a new PSCredential secret
Set-SecretStoreConfiguration	Set secret store configuration

EXCHANGE ONLINE

Connect-ExchangeOnline	Establish connection
Get-ExoMailbox -Resultsize Unlimited Get-ExoMailbox | Get-ExoMailboxStatistics Get-Recipient Get-DistributionGroup Get-MailboxPermission Get-TransportRule	Retrieve specific Exchange Online elements
Get-Mailbox –ResultSize Unlimited | Where {$_.GrantSendOnBehalfTo -ne $null} | Select UserprincipalName, GrantSendOnBehalfTo	Retrieve mailboxes with "send on behalf" configured
Set-Mailbox Set-MailboxPermission Set-TransportRule Set-MailboxAutoReplyConfiguration	Configure specific Ex-change Online elements
New-Mailbox New-DistributionGroup New-TransportRule	Create new Exchange Online elements
New-Mailbox -Shared -Name 'Sales Dept' -DisplayName 'Sales Department'	Create a shared Exchange Online mailbox
Remove-Mailbox	Delete Exchange Online mailbox

*requires ExchangeOnlineManagement module

INPUT AND OUTPUT COMMANDLETS

Format-Table (ft)	Table output
Format-List (fl)	Detailed list
Format-Wide (fw)	Multi-column list
Out-Host (oh)	Output to consoles with colour options and paging option
Out-GridView (ogv)	Table with filtering and sorting options
Out-File	Save to file
Out-Printer (lp)	Send to printer
Out-Clipboard	Send to clipboard
Out-Speech	Speech output (requieres module "PSCX")
Out-Null	Objects in pipeline are not passed on
Read-Host	Read from console
Import-CSV Export-CSV	Import/ export CSV file
Import-CLIXML Export-CLIXML	Import/ export XML file

User defined table output
```
Get-Process | ft @{Label='Nr'; Expression={$_.ID}; Width=5},
@{Label='Name'; Expression={$_.Processname}; Width=30},
@{Label='Memory MB'; Expression={$_.WorkingSet64 / 1MB};
Width=7; Format='{0:00000.0}'}
```

POWERSHELL 7

The PowerShell 7 GitHub Repository	https://github.com/PowerShell/PowerShell
iex "& { $(irm https://aka.ms/install-powershell.ps1) } -UseMSI"	Installs the latest PowerShell 7 version on a Windows machine
ForEach-Object -Parallel -ThrottleLimit 10	Parallel execution of pipeline output
Import-Module AzureAD -UseWindowsPowerShell	Runs cmdlets of the imported module in a Windows PowerShell process.
$x = $null $x ?? 100 Output: 100	The null-coalescing operator ?? returns the value of its left-hand operand if it isn't null. Otherwise, it evaluates the right-hand operand and returns its result.
Get-ChildItem -Path 'application.log" || New-Item -Path 'application.log'	Pipeline chain operator "||" executes the right-hand pipeline if the left-hand pipeline failed.
Get-ChildItem -Path 'C:\temp' && Copy-Item 'test.txt' -Path 'C:\temp'	Pipeline chain operator "&&" executes the right-hand pipeline if the left-hand pipeline succeeded.
$IsWindows ? 'yes' : 'no'	Ternary operator "?" evaluates the condition Expression to execute if the condition is true, followed by ":"
$ErrorView = 'ConsiseView'	Improves the readability of interactive and script errors

SPLATTING

```
$params = @{

  ParameterName1 = 'Value1'

  ParameterName2 = 'Value2'

  ParameterName3 = 'Value3'

}

Get-Something @params
```

Splatting is a technique to pass a collection of parameter values to a command using a single variable instead of sending them as separate arguments.

Benefits:
- Better readability
- Improved reusability
- Conditionally adding parameter

```
$params = @{

  Path = 'C:\ProgramData\
  ScriptRunner\Service'

}
if(<condition>) {

  $params.Add('Recurse', $true)

}

Get-ChildItem @params
```

Setting up a list of parameters for the Get-ChildItem cmdlet. If a specific condition is met, it adds the -Recurse parameter to retrieve items from the specified directory and all its subdirectories; otherwise, it retrieves items only from the specified directory.

COMPARISON OPERATORS

Compare case in-sensitive	Compare case sensitive	Meaning
-lt -ilt	-clt	Less than
-le -ile	-cle	Less or equal
-gt -igt	-cgt	Greater than
-ge -ige	-cge	Greater or equal
-eq -ieq	-ceq	Equal
-ne -ine	-cne	Not equal
-like -ilike	-clike	Similarity between strings, use of wildcards (* and ?) possible
-notlike -inotlike	-cnotlike	No similarity between strings, use of wildcards (* and ?) possible
-match -imatch	-cmatch	Compare with regular expression
-notmatch -inotmatch	-cnotmatch	Does not match regular expression
-is	-	Type comparison, e.g. (Get-Date) -is [DateTime]
-in -contains	-	Is included in set
-notin -notcontains	-	Is not included in set

For logical conjunction, -and, -or as well as -not (alias !) are used
Example: ((1MB + $a + $b) -gt 2000KB) -and !($a -le 2KB)
KB, MB, GB, TB, and PB are valid units for memory sizes.

CONFIGURING AND USING NETWORKS

Get-NetAdapter	List network cards (also virtual ones)
Get-NetAdapterBinding	Properties of a network connection
Set-NetIPInterface	Enable or disable DHCP
New-NetIPAddress Remove-NetIPAddress	Set or remove static IP address
Set-DnsClientServerAddress	Set or remove DNS server
Remove-NetRoute	Remove gateway from network connection
Resolve-DnsName	Resolve DNS name
Enable-NetFirewallRule Disable-NetFirewallRule	Enable or disable a Windows Firewall rule
Test-Connection	Perform a ping
Send-MailMessage	Send email
Invoke-WebRequest	HTTP request
New-WebServiceProxy	Create a proxy for SOAP-based service
Export-ODataEndpoint-Proxy	Create a proxy for OData-based service

ACCESS TO WMI

List of all WMI classes from a namespace of a computer
```
Get-CimClass -Namespace root/cimv2 -Computer MyServer
```

List all instances of a WMI class on a computer
```
Get-CimInstance Win32_LogicalDisk -Namespace root/cimv2 -Computer
MyServer
```

WQL query on a computer
```
Get-CimInstance -Query "Select * from Win32_Networkadapter where
adaptertype like '%802%'" -Computer MyServer
```

Access to an instance and change to the instance
```
$c = Get-CimInstance Win32_LogicalDisk -Namespace root/cimv2 -Filter
"DeviceID='C:'" -Computer MyServer
Set-CimInstance $c
```

Alternatively with old WMI commandlets
```
$c = [WMI] "\\MyServer\root\cimv2:Win32_LogicalDisk.DeviceID='C:'"
$c.Put()
```

Calling a WMI method
```
Invoke-CimMethod -Path "\\MyServer\root\cimv2:Win32_Computersystem.
Name=MyServer" -Name 'Rename' -ArgumentList 'MyNewServer'
```

PROCESSES, SERVICES, EVENTS, PERFORMANCE

Get-Process	Running processes
Start-Process Stop-Process	Start/terminate process
Wait-Process	Wait for process to terminate
Get-Service	Windows system services
Start-Service Stop-Service Suspend-Service Resume-Service	Change service state
Get-WinEvent	Event log entries
New-WinEvent	Create entry in event log
Limit-EventLog	Set size for event log
Get-Counter	Retrieve important performance indicators
Get-Counter -ListSet *	List all performance indicators
Get-Counter -Counter '\Processor(_Total)\% ProcessorTime'	Retrieve particular performance indicator

GET HELP

Get-Command Get-*	All commands with "Get-"
Get-Command -Module *ActiveDirectory* | Format-Table Name, Module	All commands of a module
Get-Alias	Show all aliases
Get-Help Stop-Process -full	Full help content for a command
Get-Help about	List all "About" documents
Get-Help about_WMI	Show help for WMI
Get-Service | Get-Member	Show all properties and methods of the result objects

ACTIVE DIRECTORY

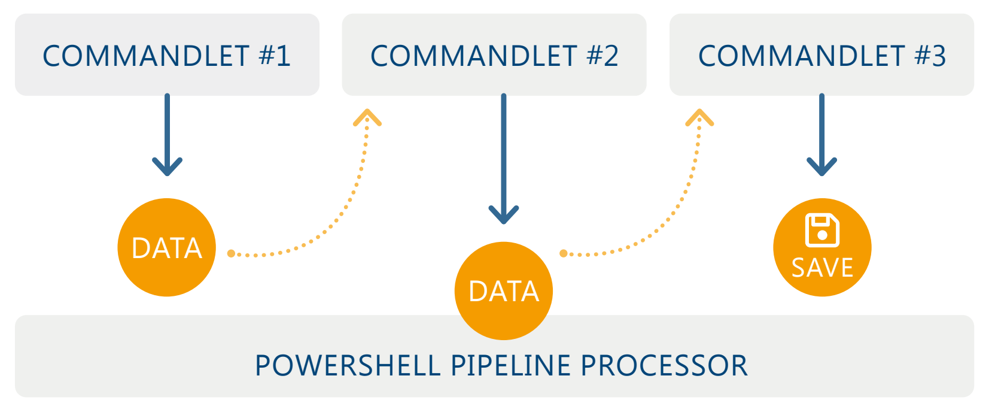
Get-ADObject	Retrieve arbitrary objects from AD
Get-ADUser Get-ADGroup Get-ADOrganizationalUnit Get-ADDomain Get-ADComputer	Retrieve particular AD elements
Set-ADObject Set-ADUser Set-ADGroup Set-ADComputer	Set properties for an object
New-ADUser New-ADGroup New-ADOrganizationalUnit	Create new AD object
Remove-ADObject	Delete AD object
Rename-ADObject	Rename AD object
Move-ADObject	Move AD object
Set-ADAccountPassword	Set password
Get-ADGroupMember	List group members of an AD group
Add-ADGroupMember	Add member to an AD group
Remove-ADGroupMember	Remove member from an AD group

PIPELINING

Any number of commandlets can be joined using the pipe symbol |.
```
Get-Service a* | Where-Object {$_.status -eq 'running'} |
Out-File c:\temp\runningservices.txt
```

Alternatively, you can store intermediate results in variables starting with $.
```
$services = Get-Service a* | Where-Object {$_.status -eq 'running'}
$services | Out-File c:\temp\runningservices.txt
```

The pipeline forwards .NET objects. Forwarding is asynchronous
(except from some "blocking" commandlets like the sort object)



EXAMPLE:
Get-Service a* | Where-Object { $_.status -eq 'running' } | Out-File c:\file-name.txt

Commandlet #1:
Get-Service a*
Object of type: System. ServiceProcess. ServiceController

Commandlet #2 - selection:
Where-Object { $_.status -eq 'running' }

Commandlet #3 - storage in file system:
Out-file c:\filename.txt

IMPORTANT PIPELINING COMMANDLETS

Where-Object (where, ?)	Filter using conditions
Select-Object (select)	Truncate result set from its start/end reduction of object attributes, respectively
Sort-Object (sort)	Sort objects
Group-Object (group)	Group objects
Foreach-Object { $_... } (%)	Loop over all objects
Get-Member (gm)	Print metadata (reflection)
Measure-Object (measure)	Calculation: -min -max -sum -average
Compare-Object (compare, diff)	Compare two sets of objects