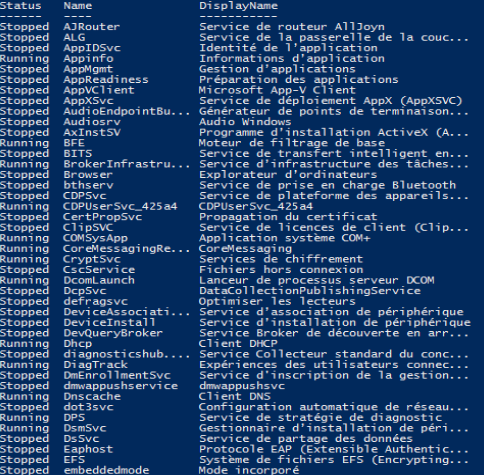
Cours Powershell pour les débutants

I. Structure des commandes

1. Applets de commande «cmdlet»

Sur Powershell, les applets de commandes utilisent une paire composés de: verbe-nom (ex: get-service)



Cette commande permet de voir nos services actifs et inactifs

2. Les variables

Les variables commencent par le signe $ et contient la valeur qu’on lui a donnée et peu contenir des lettres, des chiffres et des tirets

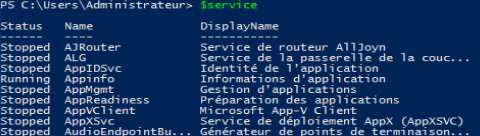
Pour utiliser une variable il faudra simplement faire: $Nom\_Variable

Pour créer une variable il faudra indiquer le nom de la variable et sa valeur (ex: $var = «valeur»)

Dans notre exemple nous allons créer une variable get-service



La variable $service nous permettras d’effectuer un get-service



3. Variables d’environnement

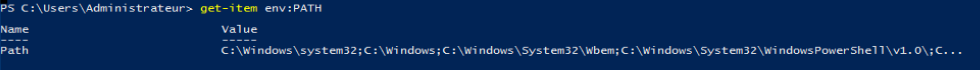
Les variables d’environnement contiennent les informations liées au système d’exploitation



Cette commande permet d’afficher la valeur de toutes les variables d’environnement

Si nous recherchons une variable d’environnement en particulier, nous utiliserons la cmdlet: « Get-Item: »

Exemple: pour la variable PATH



4. Pipeline

Le pipeline symbolisé par « | » permet de chaîner les commandes



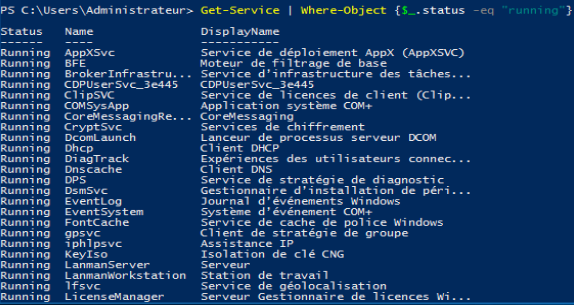
Exemple: Get-Service | Sort-Object Status permet d’effectuer un tri en affichant les services stopped en priorité

Il est aussi possible de filtrer les sorties:



Dans notre exemple **Get-Service | Where-Object {$\_.status -eg «running» }** permet de lister que les service actifs

On remplacera «running» par «stopped» si l’on souhaite avoir seulement les services stopped

Pour avoir un service en particulier, on remplaceras «status» par «name» et «running» par «NomDuService»

II. Commandes à maitriser

|  |  |  |
| --- | --- | --- |
| **Cmdlet** | **Description** | **Alias** |
| Get-Command | Information de base sur les commandes | Gcm |
| Get-Help | Aide de base | Help, man |
| Get-Member | Informations sur les méthodes et propriétés des objects | Gm |
| Get-PSDrive | Information sur les lecteurs powershell | Gdr |
| Get-Module | Liste les modules actuellement chargés | Gmo |
| Get-PSSnapin | Liste les snapins actuellement chargés | gsnp |

1. Get-Help

Get-Help permet l’obtention d’information, que ce soit sur une commande cmdlet, fonction ou script

Get-Help utilisé seulement avec une commande montreras les informations de base

Exemple: Get-Help Get-Service



Il est possible d’ajouter un paramètre a la cmdlet:

|  |  |  |
| --- | --- | --- |
| **Paramètre** | **Description** | **Exemple** |
| -Full | Permet d’afficher la totalité de l’aide d’une cmdlet | Get-Help Get-Service -Full |
| -Category | Permet de lister l’aide d’un éléments  se trouvant dans une catégorie  spécifique (Fournisseur dans notre  cas) | Get-Help -Category -Provider |
| -Online | Permet d’afficher l’aide en ligne sur microsoft docs | Get-Help Get-Service -Online |
| -Examples | Donne le nom, le synopsis et les exemples | Get-Help Get-Service-Examples |
| -Detailed | Permet d’afficher une vue détaillé de l’aide contenant des descriptions de paramètres suivi d’exemples | Get-Help Get-Service-Detailed |

2. Get-Command

Get-Command affiche toute les commandes disponibles

Pour connaître le nombre de commande disponible, faites ces commandes:

(gcm).count

(gcm \* ).count  
(gcm -CommandType cmdlet).count

Le nombre diffère car Powershell exploite différentes catégories de commande:

-Alias

-Function

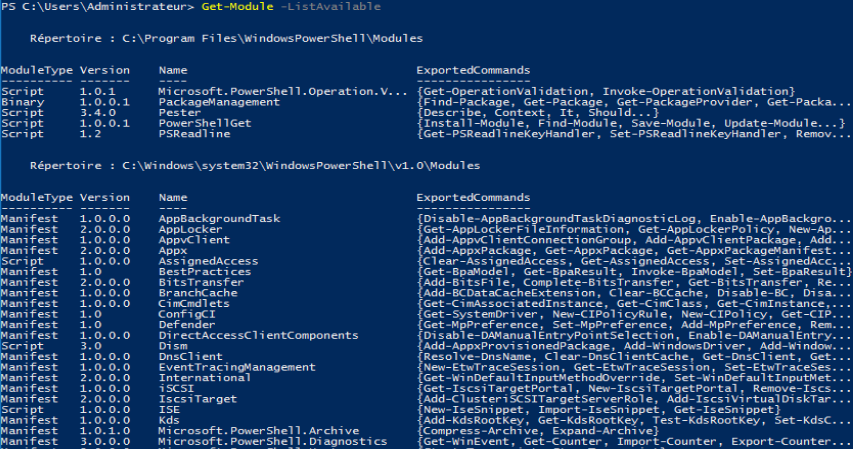
-Cmdlet

-Application

Powershell possède un noyau composé de commande de base (alias, function, cmdlet) qui peuvent être augmenter en chargeant des modules, les modules porteront l’extension \*.psm1, \*.psd1 ou \*.ddl si ils sont compilés

Pour savoir les modules disponible:

Get-Module -ListAvailable



Et pour charger un module:

Import-module NomDuModule (indiquer le nom complet)

Pour décharger un module de la mémoire:

remove-module (cela ne le détruira pas)

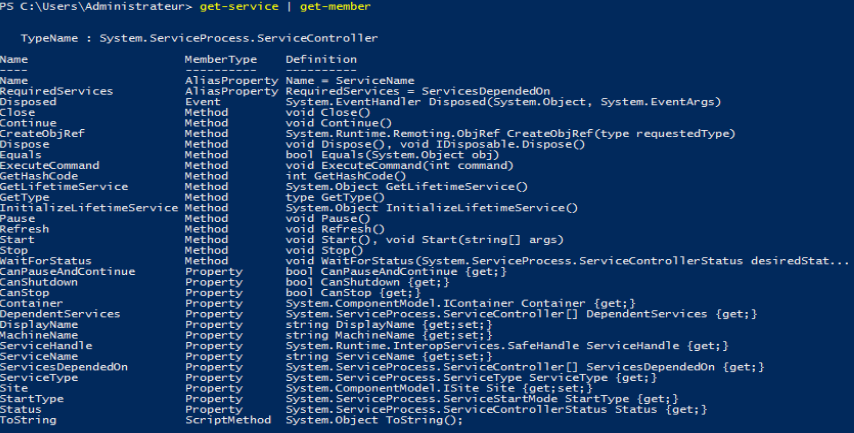
3. Get-Member

Get-Member obtient les propriétés et les méthodes des objets, il permet d’avoir les membres des objets

Exemple:

Get-Service | Get-Member

Contrairement a «Get-service», «Get-Service | Get-Member» affiche le TypeName, MemberType et Definition



III. Script

1. Base

Les script Powershell s’exécutent sous une stratégie d’exécution

Pour afficher la stratégie d’exécution:

Get-ExecutionPolicy

Les principales valeurs seront:

|  |  |
| --- | --- |
| Mode | Détails du mode |
| Restricted | Aucune exécution de script n’est autorisé |
| RemoteSigned | Les scripts locaux sont autorisés mais les distants doivent être signés |
| AllSigned | Tous les scripts doivent être signés pour être utilisés |
| Unrestricted | Tous les scripts peuvent êtres exécutés |

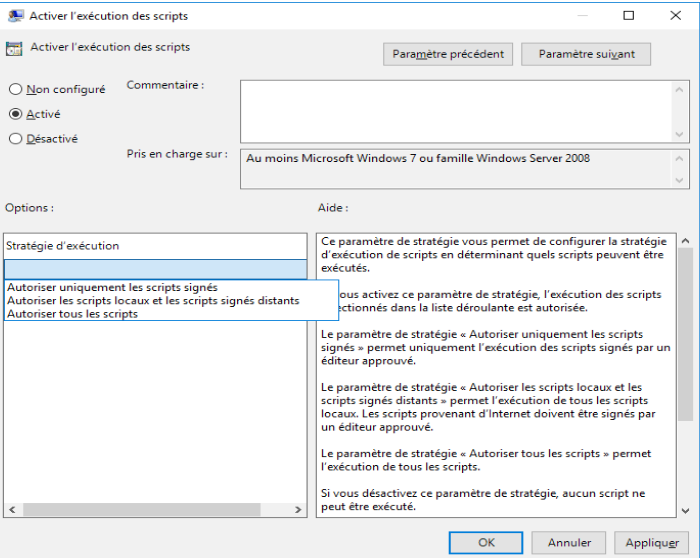
Lors du lancement de l’interpréteur il est possible d’outrepasser la stratégie d’exécution:

Powershell.exe -ExecutionPolicy bypass -file monscript.ps1

2. Edition de GPO

Pour autoriser l’exécution de script, «gpedit.msc» permet d’accéder au gestionnaire de stratégie au groupe

Puis: Configuration Ordinateur > Modèles d’administrateur > Composants Windows > Windows Powershell > Activer l’exécution des scripts

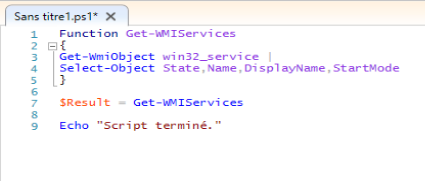


IV. Dot-sourcing

Le dot-sourcing consiste a mettre un point devant le nom d’un script a exécuter (exemple: . .\monscript.ps1)

Lorsque cette syntaxe est utilisé, elle a pour but de garder les variables et fonctions dans le contexte d’exécution, c’est a dire qu’elle agit comme une commande «include» qui permet de charger une bibliothèque de fonctions se trouvant dans un fichier complémentaire

Exemple:



Function Get-WMIServices  
{  
Get-WmiObject win32\_service |  
Select-Object State,Name,DisplayName,StartMode  
}

$Result = Get-WMIServices

Echo "Script terminé."

Après avoir fait ce code, enregistrez le puis exécutez le dans Powershell avec

.\testscript.ps1

. .\testscript.ps1

les deux résultats sont identiques mais la fonction «Get-VMIServices» est chargée dans le contexte de la console. Pour vérifier, appelez la fonction ou la variable $result

Le dot-sourcing permet d’exécuter un script (les variables et les fonctions) qui aurait été détruit à la fin de l’exécution, le rendant réutilisable dans le contexte appelant

Qu’est ce que le scope (portée, étendue)? Le scope détemrine la visibilité des variables et des fonctions,

lorsque vous exécutez un script sur Powershell, cela créer un contexte d’exécution nommée «portée enfant», elle correspond a la session sur laquelle elle a été créée. Elle hérite des variables et des fonctions de son parent. Elles peuvent être lues et exploitées dans la portée enfant mais pas modifié. Les portées parents ne peuvent ni lire, ni modifier les fonctions et variables de leurs enfants.

Les fonctions et variables sont déclarés en local mais nous pouvons choisir la portée via l’option

«-scope» ou avec un des préfixe suivant:

|  |  |
| --- | --- |
| Préfixe | Description |
| $**global**:var | Permet d’accéder a la portée générale ou initiale |
| $**local**:var | Portée actuelle créé lors de l’exécution d’une fonction, d’un script ou d’un bloc d’instructions |
| $**script**:var | Permet d'accéder à une variable du script à partir d'une fonction déclarée au sein même de celui-ci |
| $**private**:var | Usage de variables strictement réservées et visibles uniquement dans la portée actuelle (=non cessible ou pas d'héritage) |
| $**using**:var | Permet de passer une variable "locale" à un script exécuté sur un ordinateur distant |

Pour plus d’information: help about\_scope

V. Les profils

Contrairement au «.profile» d’Unix, cette notion, lors d’une l’ouverture d’une session Powershell permettras de définir des préférences. Les préférences sont un ensemble de scripts .ps1 exécuter au démarrage de Powershell

Les scripts de profils sont exécutés dans cet ordre

Niveau machine

1.%windir%/system32/WindowsPowerShell/v1.0/profile.ps1  
2.% windir %/system32/WindowsPowerShell/v1.0/Microsoft. WindowsPowerShell\_profile.ps1

Niveau utilisateur

3.%userprofile%/Mesdocuments/WindowsPowerShell/profile.ps1  
4.%userprofile%/Mesdocuments/WindowsPowerShell/Microsoft. WindowsPowerShell\_profile.ps1

Pour créer votre profil, tapez la commande:

new-item -path $PROFILE -type file -force

Voici un exemple de script de profil:

#-----------------------------------------------------------------------

# fonction alternative à l'applet "Get-Service" afin d'obtenir

# le type de démarrage (via WMI)

Function Get-WMIServices

{

Get-WmiObject win32\_service | Select-Object State,Name,DisplayName,StartMode

}

#-----------------------------------------------------------------------

# fonction d'affichage graphique d'un message (basée sur l'objet COM)

function Popup

{

param (

$Message='Pas de message',

$Titre='Indiquez un titre',

$Secondes=0

)

$wshShell = New-Object -ComObject wscript.shell

$wshShell.popup($Message, $Secondes, $Titre)

}

#-----------------------------------------------------------------------

# fonction alternative à la commande "Whoami" (basée sur un objet .NET)

function Qui-SuisJe

{

[system.security.principal.windowsidentity]::getcurrent().Name

}

#-----------------------------------------------------------------------

# Change la couleur d'affichage des messages d'erreur ('red' par défaut)

$Host.PrivateData.ErrorForegroundColor='magenta'

#-----------------------------------------------------------------------

# Définition de mes alias favoris 

New-Alias -Name Grep -Value Select-String

Set-Alias -Name Out-Clip -Value C:\Windows\System32\clip.exe

set-Location C:\Scripts

Popup -Message "Bonjour $env:USERNAME" -Titre "Accueil Powershell" -Secondes 3

pour tester votre profil, ouvrez une nouvelle session puis exécuter le en dot-sourcing:

. $PROFILE

VI. Gestion à distance

1. Configuration

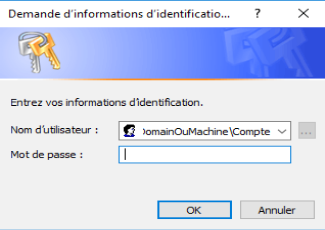
Pour configurer l’accès à distance, entrez cette commande sur le poste sur lequel vous voulez accéder à distance:

Enable-PSRemoting

La commande précédente active l’accès distance via le protocole HTTP, cela requiert l’authentification Kerberos pour la sécurité

Pour avoir une identifiant autre que celui que celui de notre session pour accéder a distance, préparez une pièce d’identité:

$cred = get-credential DomainOuMachine\Compte



2. Accès distant en mode session

Enter-PSSession -ComputerName NomMachineCible -Credential $Cred

Si vous n’avez pas d’identifiant:

Enter-PSSession -ComputerName NomMachineCible Credential $GetCredential)

3. Mode invocation ponctuelle

$command = { get-service }Invoke-Command –ComputerName NomMachineCible –ScriptBlock $command –Credential $(Get-Credential)

Pour envoyer des commandes à plusieurs machines en ordinateurs temps:

Invoke-Command –ComputerName Server1,Server2,Server3 –ScriptBlock $command –Credential $(Get-Credential)

Avec un fichier texte contenant la liste des machines a contacter:

Invoke-Command –ComputerName (Get-Content machines.txt) –ScriptBlock { get-service }

4. Mode Web

Powershell Web Access permet d’accéder en mode commande à des machines depuis un navigateur internet