POWERSHELL & SCRIPTS

Table des matières

Ce document est un condensé du cours Powershell du Certa

http://www.reseaucerta.org/content/fiches-d'exploration-du-langage-powershell

1. Commandes pour obtenir de l'aide

- Afficher de l'aide sur une commande : Get-Help *Commande* *(ex : Get-Help Get-ChildItem)*
- Afficher les exemples : Get-Help *Commande* –Examples
- Afficher les alias : Get-Alias
- Afficher la liste des méthodes et des propriétés des objets : *Commande* | Get-member

2. Commandes pour gérer les fichiers et les dossiers

Se déplacer dans les dossiers :	Set-Location *chemin* *(ex : Set-Location c:\temps)*
Afficher le chemin du dossier courant :	Get-Location
Afficher le contenu d'un dossier :	Get-ChildItem
Créer un dossier :	New-Item *nomDossier* -ItemType directory
Créer un fichier avec du texte	New-Item *nomFichier.txt* -ItemType file –Value "*texte*"
Supprimer un fichier ou un dossier :	Remove-Item *nomFichier.txt*
Déplacer un fichier :	Move-Item *nomFichier.txt* -Destination *chemin\nomFichier.txt*
Déplacer un dossier :	Move-Item *nomDossier* -Destination *chemin\nomDossier*
Renommer un fichier ou dossier :	Rename-Item *nomFichier.txt* -NewName *nomFichier2.txt*
Copier un fichier :	Copy-Item *nomFichier.txt* -Destination *nomFichier2.txt*
Copier un dossier avec ses fichiers :	Copy-Item *nomDossier* -Destination *nomDossier1* –Recurse

Tester l'existence d'un fichier ou dossier : Test-Path *chemin/nomFichier.txt*

3. Alias

Il est possible de créer ses propres alias avec la commande Set-Alias.

Exemple, la commande suivante permet de créer l'alias gd vers la commande Get-Date :

```
Set-Alias -Name gd -Value Get-Date
```

```
PS C:\Users\Administrateur.WIN-080FDGC98Q0> Set-Alias -Name gd -Value Get-Date
PS C:\Users\Administrateur.WIN-080FDGC98Q0> gd

samedi 9 décembre 2017 10:01:03
```

4. Initiation aux variables, aux propriétés et aux méthodes des objets

Le nom d'une variable commence toujours par $, il peut inclure tout caractère alphanumérique ou le trait de soulignement.

Windows PowerShell permet de créer des variables qui sont pour l'essentiel des objets nommés. La sortie de toute commande Windows PowerShell valide peut être stockée dans une variable.

Exemple : $loc = Get-Location

Il est possible d'utiliser Get-Member pour afficher des informations sur le contenu de variables.

Exemple : $loc | Get-Member (idem Get-Location | Get-Member)

Le nom de la variable suivi du point permet d'accéder aux propriétés de l'objet référencé par la variable,

Exemple pour la propriété Path de la variable $loc. : $loc.Path

→ Remarque : l'usage de la touche tabulation [tab] permet de compléter le nom de la propriété.

De même, l'exécution d'une méthode (action) d'un objet :

Exemple : $fichier.Delete()

→ Remarque : Pour les méthodes, ne pas oublier les parenthèses avec ou sans paramètre.

5. Accéder aux ressources du système d'exploitation Windows

Les classes WMI (Windows Management Instrumentation) décrivent les ressources qui peuvent être gérées. Il existe des centaines de classes WMI, certaines d'entre elles contenant des dizaines de propriétés.

La commande principale est Get-WmiObject, elle permet de lire ces ressources. List affiche l'ensemble des propriétés.

→ Get-WmiObject -List

La liste étant importante, il est possible de ne rechercher que certaines classes avec la commande Select-String.

Exemple, recherche des classes sur le poste local, comportant la chaine "networkadapter":

```
PS C:\WINDOWS\system32> Get-WmiObject -List | Select-String "networkadapter"

\\SSD-W10\ROOT\cimv2:CIM_NetworkAdapter
\\SSD-W10\ROOT\cimv2:Win32_NetworkAdapter
\\SSD-W10\ROOT\cimv2:Win32_NetworkAdapterSetting
\\SSD-W10\ROOT\cimv2:Win32_NetworkAdapterConfiguration
\\SSD-W10\root\cimv2:Win32_PerfFormattedData_EthernetPerfProvider_HyperVLegacyNetworkAdapter
\\SSD-W10\root\cimv2:Win32_PerfRawData_EthernetPerfProvider_HyperVLegacyNetworkAdapter
\\SSD-W10\root\cimv2:Win32_PerfFormattedData_NvspNicDropReasonsStats_HyperVVirtualNetworkAdapterDropReasons
\\SSD-W10\root\cimv2:Win32_PerfRawData_NvspNicDropReasonsStats_HyperVVirtualNetworkAdapterDropReasons
```

La commande Get-Member permet de consulter la liste des propriétés et méthodes de la classe spécifiée.

```
PS C:\WINDOWS\system32> Get-WmiObject win32_networkadapterconfiguration | Get-Member


   TypeName : System.Management.ManagementObject#root\cimv2\Win32_NetworkAdapterConfiguration

Name                      MemberType    Definition
----                      ----------    ----------
PSComputerName            AliasProperty PSComputerName = __SERVER
DisableIPSec              Method        System.Management.ManagementBaseObject DisableIPSec()
EnableDHCP                Method        System.Management.ManagementBaseObject EnableDHCP()
EnableIPSec               Method        System.Management.ManagementBaseObject EnableIPSec(System.String[] IPSecP...
```

Il est possible de **réduire les propriétés à lister** avec la commande Select-Object et le paramètre –Property.

Exemple, lister seulement le lecteur et l'espace disque libre avec la classe WMI win32_logicalDisk :

```
Get-WmiObject Win32_LogicalDisk | Select-Object -Property DeviceID, FreeSpace
```

Exemple pour lister l'adresse MAC, l'adresse IP et les données DHCP avec la classe WMI Win32_NetworkAdapterConfiguration :

```
Get-WmiObject Win32_NetworkAdapterConfiguration -Filter IPEnabled=True | Select-Object -Property MACAddress, IPAddress, DHCP*
```

Il est toujours possible d'affecter le résultat de la commande Get-WmiObject à une variable, et de consulter les propriétés et les méthodes de l'objet à l'aide de la commande Get-Member.

Si le résultat de la commande est un ensemble d'objets, la variable affectée est un tableau d'objet, l'accès au premier élément se fait alors de la manière suivante $var[0], au second élément : $var[1], etc..

6. Scripts

Utiliser Powershell ISE pour écrire vos scripts.

Dans PowerShell, il existe quatre paramètres de stratégie d'exécution des scripts qui sont :

- Restricted : paramètre par défaut, n'autorise pas l'exécution des scripts,
- AllSigned : n'exécute que les scripts de confiance, donc signés,
- RemoteSigned : exécute les scripts locaux sans obligation de confiance et les scripts de confiance issus d'Internet,
- Unrestricted : autorise l'exécution de tous les scripts.

Démarrer Windows PowerShell en tant qu'administrateur, puis utiliser la commande suivante pour définir la stratégie :

- Commande pour connaître la stratégie en cours : *Get-ExecutionPolicy*
- Commande pour modifier la stratégie : Set-ExecutionPolicy RemoteSigned

Script WMI

à partir de noms de postes contenus dans un fichier texte

Une première version sans pipeline

```
# Premier script avec WMI script02.ps1
# Recherche des adresses IP et des adresses MAC des ordinateurs dont les noms
# sont contenus dans un fichier texte listeposte.txt (1 nom par ligne)
# Le résultat est placé dans un fichier texte nommé adresseposte.txt
# les postes non trouvés sont placés dans le fichier erreur.txt

$fichierPoste="c:\temp\listeposte.txt"
$fichierAdresse="c:\temp\adresseposte.txt"
$fichierErreur="c:\temp\erreur.txt"
$listePoste=Get-Content $fichierPoste
foreach($nom in $listeposte) {
    $filtre="address='"+$nom+"'"
    $test=Get-WmiObject Win32_PingStatus -Filter $filtre
        if ($test.StatusCode -eq 0){
            $colCarte=Get-WmiObject Win32_NetworkAdapterConfiguration -Filter IPEnabled=True -ComputerName $nom
            foreach($carte in $colCarte) {
                $ligne=$nom+"/"+$carte.MACAddress+"/"+$carte.IPAddress
                Add-Content $fichierAdresse $ligne
            }
        }
        else{
            Add-Content $fichierErreur "pas trouvé le poste : $nom"
        }
}
```

Cette classe WMI permet de faire un test de connexion vers le poste avec un filtre construit à partir du nom. La connexion est OK si StatusCode est égal à 0.

Cette classe WMI retourne une collection d'objet (configuration adaptateur réseau) du

Pour chaque configuration retournée, écriture

Remarque : Ne sont pas traités les différents tests d'existence des fichiers.

Une autre version avec un pipeline pour récupérer la configuration des cartes réseaux :

```
# Idem premier script avec WMI script02.ps1
# mais le résultat d'un objet WMI est transmis à un pipeline

$fichierPoste="c:\temp\listeposte.txt"
$fichierAdresse="c:\temp\adresseposte.txt"
$fichierErreur="c:\temp\erreur.txt"
foreach ($nom in Get-Content $fichierPoste){
    $filtre="address='"+$nom+"'"
    $test=Get-WmiObject Win32_PingStatus -Filter $filtre
    if ($test.StatusCode -eq 0) {
        Get-WmiObject Win32_NetworkAdapterConfiguration -Filter IPEnabled=True -ComputerName $nom |
        ForEach-Object {Add-Content $fichierAdresse ($nom+"/"+$_.MACAddress+"/"+$_.IPAddress)}
    }
    else{
        Add-Content $fichierErreur "pas trouvé le poste : $nom"
    }
}
```

Chaque objet (configuration adaptateur réseau) est transmis au pipeline

Chaque objet transmis est référencé par $_ pour obtenir les propriétés MAC et IP

Script de création de comptes utilisateurs à partir d'un fichier texte au format CSV

```
# Premier script ADSI, script03.ps1
# Création de comptes utilisateur à partir d'un fichier texte
# au format CSV : (prenom,nom,description,tel)

$domaine=",dc=labo,dc=ig"
$unite="test"
$ActiverCompte=512                      # Pour activer le compte
$pwd="toto"                             # Mot de passe première connexion
$fichierUser="c:\temp\users.txt"
$cible="LDAP://ou="+$unite+$domaine

$ldapUnit=[ADSI] $cible
Import-Csv $fichierUser | ForEach-Object {
    $user=$ldapUnit.create("user","cn="+$_.prenom+" "+$_.nom)
    $user.put("sAMAccountName",$_.nom)
    $user.put("sn",$_.nom)
    $user.put("givenName",$_.prenom)
    $user.put("DisplayName",$_.nom+" "+$_.prenom)
    $user.put("userPrincipalName",$_.nom)
    $user.put("description", $_.description)
    $user.put("telephoneNumber",$_.tel)
    $user.SetInfo()
    $user.SetPassword($pwd)
    $user.put("userAccountControl", $ActiverCompte)      # Active le compte
    $user.put("pwdLastSet", 0)  # Changement du mot de passe 1ère connexion
    $user.SetInfo()
}
```

Valeurs de userAccountControl, voir site :

Cette commande permet de lire un fichier CSV, chaque objet (ligne) est transmis au pipeline.

Chaque élément du compte utilisateur est accessible par la désignation définie dans le fichier (1ère ligne) et la variable $_ qui représente la ligne en cours.

Idem mais avec les tests d'existence de l'unité d'organisation et des comptes utilisateurs

Extrait du script :

```
11  # avec tests d'existence de l'organisation et du compte utilisateur
12  $ldapUnit=[ADSI] $cible
13  if (!$ldapUnit.DistinguishedName){
14      Write-Host "$unite : cette unité d'organisation n'existe pas"
15  }
16  else{
17      $recherche=New-Object DirectoryServices.DirectorySearcher
18      Import-Csv $fichierUser | ForEach-Object {
19          $cn=$_.prenom+" "+$_.nom
20          $recherche.Filter="(cn=$cn)"
21          $user=$recherche.FindOne()
22          if($user -eq $null){
23              $user=$ldapUnit.create("user","cn="+$cn)
```

En fait, on teste l'existence de l'unité ou du domaine.

Avec la version 2.0 : ldapDomaine=[ADSI]"LDAP://labo.ig"

Cet objet permet de faire une recherche dans tout le domaine et pas seulement dans l'unité d'organisation

Remarque : Ligne 23, on peut aussi écrire : "cn=$cn", comme à la ligne 20, ce qui facilite le traitement ses parenthèses () dans la chaîne de caractères.

Une autre façon de faire

Avoir un fichier csv (champ séparé par des virgules ou point virgule)

Remarques sur le CSV

- Attention supprimer les espaces, les accents dans les champs (utiliser la fonction rechercher remplacer)
- Pour passer un champ en majuscule : utiliser la fonction =majuscule sur la cellule, puis copier la cellule et pour la coller utiliser **Coller la valeur**.
- Faire attention à la 1ere ligne (nommage des champs) : doit correspondre a ceux donné dans le script

Le script

A démarrer dans Powershell ISE (et non dans powershell)

Le script a adapter : (modifier les zones surlignées)

```
Import-Module ActiveDirectory
$Users = Import-Csv -Delimiter ";" -Path "c:\scripts\user.csv"
foreach ($User in $Users)
{
        $ou = "OU=sio14-16,DC=SIO,DC=NET"
# l'uo ou doivent être crée les utilisateurs : créer une uo spécifique, l'import dans users ne fonctionne pas)
        $domain = '@sio.net'
        $password =$User.password
# (déclaration de la variable a aller chercher dans le fichier user champ password)
         $detailname = $user.name + " " +$user.givenname
# ((déclaration de la variable a aller chercher dans le fichier user champ name concatené avec le champ givenname)
        $userfirstname = $user.givenname
        $userlastname = $user.name
        $UPN = $lastname + "."+ $domain
        $SAM = $userlastname
```

```
 New-ADUser -name $detailname -sAMAccountname $SAM -userPrincipalName $SAM -displayName $detailname -Enabled $True -AccountPassword (ConvertTo-SecureString $password -AsPlainText -force) -ChangePasswordAtLogon $True -Path $ou
}
```

Script qui liste tous les noms de poste qui ne sont pas des serveurs.

```
1  #script03_03, Liste des noms des postes de travail sans les serveurs
2  $searcher=New-Object System.DirectoryServices.DirectorySearcher
3  $searcher.filter="(&(Objectcategory=computer)(!operatingsystem=*server*))"
4  $searcher.findall() | foreach-object {$_.properties.name}
```

Le résultat de la méthode est transmis au pipeline qui lui ne liste que le nom du poste avec la variable $

Le filtre sélectionne les objets de la catégorie 'computer' avec un 'operatingsystem' différent (!) de '*server*'.

Le résultat de cette recherche peut être placé dans un fichier texte avec Add-content, mais aussi avec Export-Csv :

```
4  $searcher.findall() | select @{name="Nom";Expression={$_.properties.name}},
5                        @{name="Date creation";Expression={$_.properties.whencreated}}| Export-Csv "c:\temp\poste.txt"
```

Remarques : Ne sont sélectionnées que les propriétés noms et date de création du poste. La transmission est réalisée avec 2 pipelines.

7. Accès distant avec PowerShell

1. Introduction

La gestion distante permet d'exécuter des commandes sur un ou plusieurs ordinateurs distants à partir d'un seul ordinateur. PowerShell permet plusieurs moyens de connexion, interactif (1:1) ou un-à-plusieurs (1:n).

2. Démarrer le service WinRM

Sur les ordinateurs distants (serveurs), utiliser la commande *Enable-PSRemoting* pour démarrer le service WinRM.

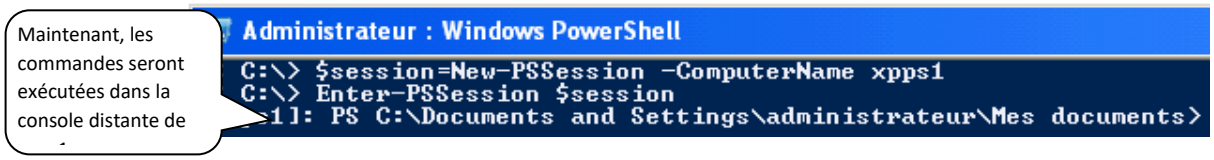
Rmq : *–force* permet de ne pas répondre aux invites. Par défaut, vous êtes invité à confirmer chaque opération.

Dans Windows 7 et Windows server 2008 R2, pour utiliser cette commande, il faut lancer PowerShell en tant que administrateur (clic droit, exécuter en tant que ….)

Ce service utilise Kerberos par défaut pour l'authentification. Quand Kerberos n'est pas disponible (postes non intégrés à un domaine), WinRM utilise un transport HTTPS basé sur un certificat SSL installé sur le poste distant. Pour simplifier, cette fiche est donc réalisée avec un domaine Active Directory.

3. Connexion interactive (1:1)

Sur l'ordinateur client, exécuter l'instruction *$session=New-PSSession –computername NomOrdinateur* et utiliser la commande *Enter-PSSession $session* pour se connecter de manière interactive avec le poste distant.



La commande *Exit-PSSession* permet de quitter la console distante et revenir à la console locale.

Rmq : Il est possible de spécifier un nom d'utilisateur pour les sessions *avec le paramètre –Credential*

4. Accès à distance un-à-plusieurs (1:n)

L'accès à distance de PowerShell permet aussi d'exécuter une ou un ensemble de commandes simultanément sur plusieurs ordinateurs. La technique est très simple, il suffit de spécifier la liste des ordinateurs concernés avec le paramètre *–computername NomOrdi1,NomOrdi2,NomOrdi3.*

Exemple avec un fichier texte : $session=New-PSSession –computername (Get-Content c:\listeOrdi.txt)

Pour exécuter une commande sur l'ensemble des postes, il faut utiliser : *Invoke-Command –Scriptblock {cmd}*

Exemple pour exécuter IPconfig sur chaque poste : Invoke-Command –Scriptblock {ipconfig} –session $session

Cette commande affiche le résultat des deux *ipconfig* lancés à partir de chaque poste distant.

Il est possible d'utiliser *Invoke-Command* sans sessions ouvertes préalablement et de spécifier directement la liste des postes avec *–computername*, dans ce cas les sessions sont ouvertes au début et fermées juste après l'exécution.

Exemple : Invoke-Command –Scriptblock {ipconfig} –computername NomOrdi1,NomOrdi2,NomOrdi

En général, on crée une session uniquement lorsqu'on exécute une série de commandes sur l'ordinateur distant.

Par défaut, le nombre de connexions simultanées est de 32, cette valeur peut être modifiée avec le paramètre *–throttlimit* de la commande *Invoke-Command.*

Autres exemples avec *Invoke-command* :

Exécute le script local sur le poste distant :

Invoke-Command -filepath c:\scripts\test.ps1 -computerName NomOrdi1

Plein d'exemples dans le point zip mis à disposition sur le serveur (Sisr4 -scripts)

Pour aller plus loin

http://www.reseaucerta.org/content/fiches-d'exploration-du-langage-powershell