Découverte de PowerShell- Tutoriel

Microsoft France Division DPE

Table des matières

Présentation

Windows PowerShell est un interpréteur de commande et un langage de script conçu initialement pour la gestion et l'administration des systèmes d'exploitation, et sert maintenant aussi pour l'administration de certains produits serveurs tels que Microsoft Exchange Server 2007 ou Microsoft SQL Server 2008. Sa version 2.0 est disponible en standard dans Windows 7 et Windows Server 2008 R2, ou sous forme de mises à jour à télécharger pour Windows Vista et Windows Server 2008 (retrouvez le lien vers les téléchargements à la fin du tutoriel dans les ressources).

Son utilisation passe par l'appel de *cmdlets*, ou applets de commande, qui permettent d'interagir avec le système, de la manipulation de fichiers à l'édition de la base de registre en passant par le suivi des processus ou des services de l'ordinateur.

De plus, contrairement à d'autres « shells », PowerShell vous donne réellement accès à des objets et permet ainsi des scénarios plus poussés puisque ces objets pourront à leur tour proposer leurs propriétés et méthodes.

PowerShell est basé sur le Framework .Net : vous pourrez appeler les API .Net directement depuis vos scripts. Ainsi, si vous êtes déjà développeur, vous pourrez capitaliser sur vos connaissances. Pour finir, sachez que PowerShell n'est pas sensible à la casse, autrement dit il n'y a pas de différence entre les miniscules et les majuscules.

Objectifs

Dans ce tutoriel, vous apprendrez à :

- Utiliser l'environnement intégré de script PowerShell (ISE)
- Créer vos propres scripts
- Lancer des commandes sur une machine distante

Prérequis

Pour compléter ce tutoriel vous devez avoir :

- Microsoft Windows 7

Exercice 1: Utilisation de Windows PowerShell ISE

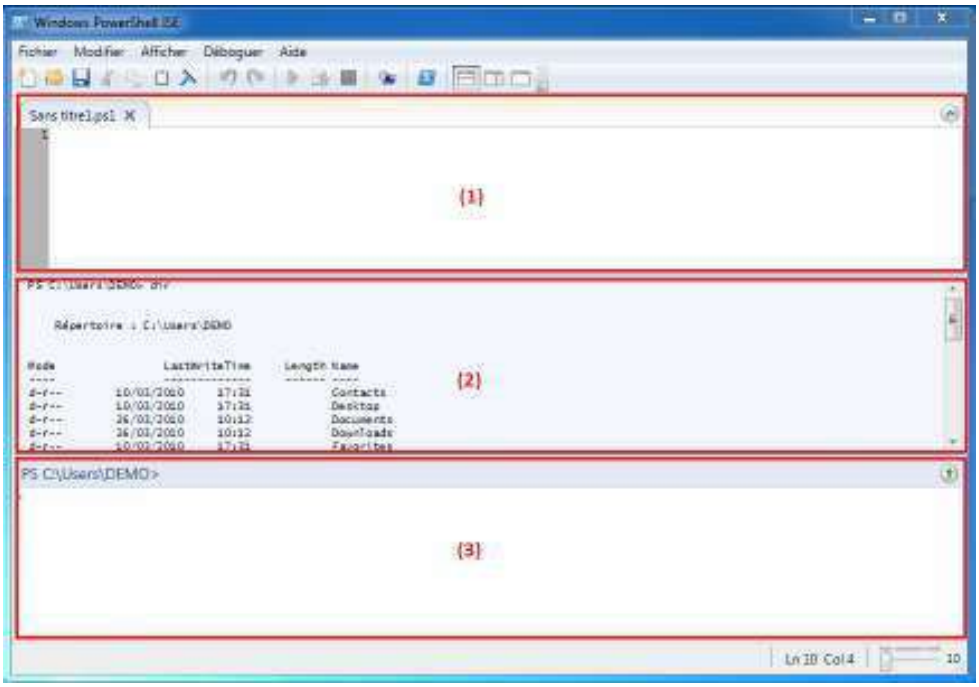
Une des nouveautés de PowerShell 2.0 est l'apparition d'un environnement de développement, ou plus exactement d'écriture de script, ISE signifiant « **I**ntegrated **S**cripting **E**nvironnement ». L'interface classique en ligne de commande continue d'exister, mais l'écriture et l'utilisation de scripts sont grandement simplifiées avec l'ISE.

Dans cet exercice vous allez découvrir cet environnement et y lancer quelques commandes.

Tâche 1 – Lancement de PowerShell ISE

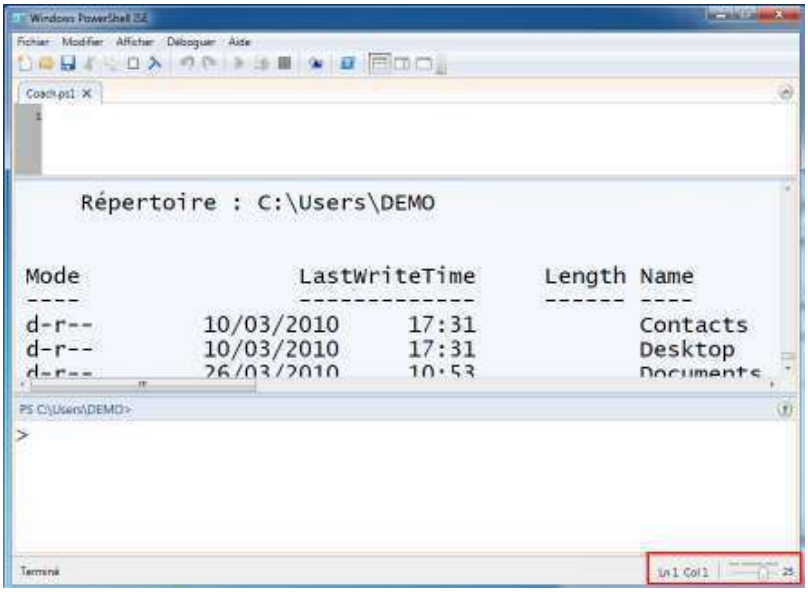
1. Rendez-vous dans le menu démarrer / « tous les programmes » / « Accessoires » puis « Windows PowerShell ». Si votre Windows est en 64-bit, vous remarquez la présence de versions 32-bit (x86).

2. Lancez « Windows PowerShell ISE ». Vous obtenez alors un éditeur graphique décomposé en 3 zones principales :



(1) : Un éditeur de texte pour créer ses propres fichiers de scripts (extension « ps1 »)
(2) : La zone d'affichage du résultat de l'exécution des scripts et commandes
(3) : Le volet de commandes pour saisir directement comme dans le prompt classique

3. En bas à droite de la fenêtre se trouvent les informations de position dans le texte (valable pour les 3 zones) mais surtout un curseur permettant de jouer sur le redimensionnement des fenêtres en mode vectoriel.



4. Le menu principal vous propose les options suivantes :
   - Fichier : ouverture et sauvegarde de scripts, gestion d'onglets supplémentaires (en local ou à distance) et le démarrage de PowerShell en ligne de commande
   - Modifier : édition de texte (copier, coller), recherche
   - Afficher : organisation des différentes fenêtres et de la barre d'outils et fonctions de zoom
   - Déboguer: débogage de vos scripts avec la gestion des points d'arrêts et du mode pas à pas. Nous verrons en détail ces options dans l'exercice sur l'écriture de scripts.
   - Aide : accès à l'aide PowerShell

Tâche 2 – Exécution de commandes

Vous allez maintenant lancer quelques commandes PowerShell.

1. PowerShell possède un nombre important de commandes, et la première étape est d'apprendre à les découvrir. Pour cela, tapez la commande suivante :

PowerShell

```
Get-Command
```

La liste de toutes les commandes, alias et fonctions disponibles s'affiche alors.

```
CommandType     Name                                               Definition
-----------     ----                                               ----------
Alias           %                                                  ForEach-Object
Alias           ?                                                  Where-Object
Function        A:                                                 Set-Location A:
Alias           ac                                                 Add-Content
Cmdlet          Add-Computer                                       Add-Computer [-Dom
Cmdlet          Add-Content                                        Add-Content [-Path
Cmdlet          Add-History                                        Add-History [[-Inp
Cmdlet          Add-Member                                         Add-Member [-Membe
Cmdlet          Add-PSSnapin                                       Add-PSSnapin [-Nam
Cmdlet          Add-Type                                           Add-Type [-TypeDef
Alias           asnp                                               Add-PSSnapIn
Function        B:                                                 Set-Location B:
```

2. Pour filtrer le résultat et rechercher une commande en particulier, il vous suffit de rajouter le filtre à la suite, le symbole « * » faisant office de « Wildcard » (n'importe quel caractère).

PowerShell

```
Get-Command *Process
```

Vous obtenez ainsi uniquement les cmdlets dont le nom finit par « Process ».

```
PS C:\Users\DEMO> Get-Command *process

CommandType     Name                                               Definition
-----------     ----                                               ----------
Cmdlet          Debug-Process                                      Debug-Process [-Name]
Cmdlet          Get-Process                                        Get-Process [[-Name]
Cmdlet          Start-Process                                      Start-Process [-FileP
Cmdlet          Stop-Process                                       Stop-Process [-Id] <I
Cmdlet          Wait-Process                                       Wait-Process [-Name]
```

3. L'aide est aussi évidemment une fonctionnalité très importante, et il ne faut pas hésiter à en user et abuser. Son utilisation est très simple puisqu'il vous suffit de passer le nom de la commande pour obtenir sa description, la liste de ses paramètres attendus... Vous disposez aussi de plusieurs options (-examples, -full, -detailed, …) pour obtenir encore plus d'informations ou des exemples concrets. Pour en savoir plus sur la précédente commande utilisée, entrez le texte :

PowerShell

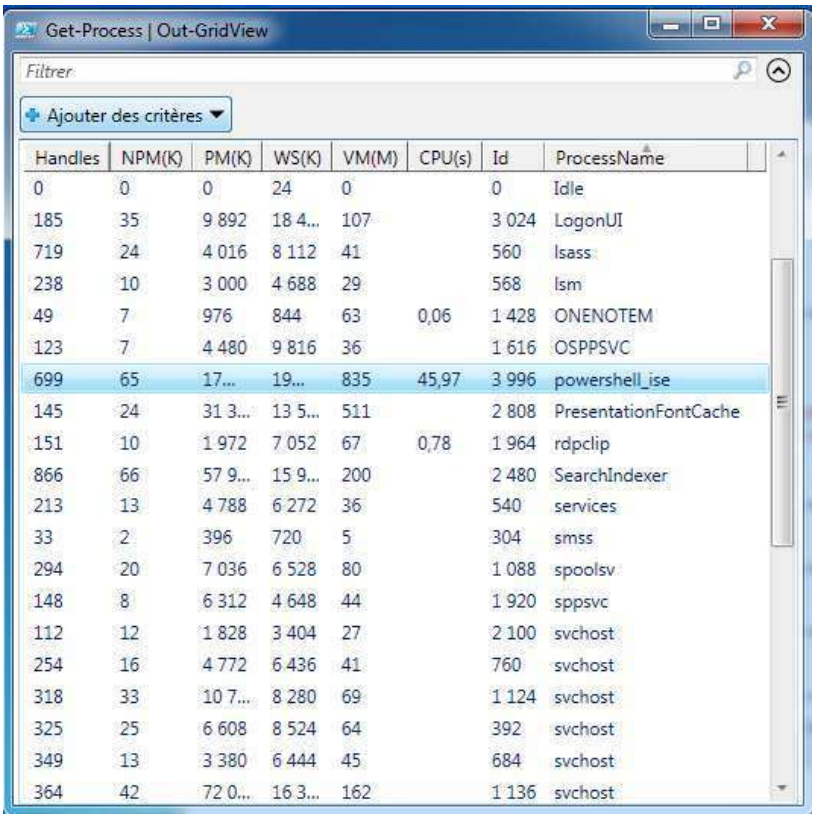
```
Get-Help Get-Process
```

```
PS C:\Users\DEMO> Get-Help Get-Process

NOM
    Get-Process

RÉSUMÉ
    Obtient les processus qui s'exécutent sur l'ordinateur local ou un ordinateur distant.


SYNTAXE
    Get-Process [[-Name] <string[]>] [-ComputerName <string[]>] [-FileVersionInfo] [-Module] [<CommonParameters>]

    Get-Process -Id <Int32[]> [-ComputerName <string[]>] [-FileVersionInfo] [-Module] [<CommonParameters>]

    Get-Process -InputObject <Process[]> [-ComputerName <string[]>] [-FileVersionInfo] [-Module] [<CommonParameters>]


DESCRIPTION
    L'applet de commande Get-Process obtient les processus présents sur un ordinateur local ou distant.
```

4. N'hésitez pas non plus à utiliser la touche tabulation pour bénéficier de l'auto-complétion. Tapez par exemple « Get-c » puis pressez plusieurs fois la touche « Tab » pour voir apparaître les différentes possibilités.

5. Une petite nouveauté de PowerShell 2.0 concerne l'ajout de la commande « Out-GridView ». Celle-ci permet de générer une grille de données non plus en mode texte mais sous forme d'application riche, avec en plus

la possibilité de trier et filtrer les éléments :



Handles	NPM(K)	PM(K)	WS(K)	VM(M)	CPU(s)	Id	ProcessName
0	0	0	24	0		0	Idle
185	35	9 892	18 4...	107		3 024	LogonUI
719	24	4 016	8 112	41		560	lsass
238	10	3 000	4 688	29		568	lsm
49	7	976	844	63	0,06	1 428	ONENOTEM
123	7	4 480	9 816	36		1 616	OSPPSVC
699	65	17...	19...	835	45,97	3 996	powershell_ise
145	24	31 3...	13 5...	511		2 808	PresentationFontCache
151	10	1 972	7 052	67	0,78	1 964	rdpclip
866	66	57 9...	15 9...	200		2 480	SearchIndexer
213	13	4 788	6 272	36		540	services
33	2	396	720	5		304	smss
294	20	7 036	6 528	80		1 088	spoolsv
148	8	6 312	4 648	44		1 920	sppsvc
112	12	1 828	3 404	27		2 100	svchost
254	16	4 772	6 436	41		760	svchost
318	33	10 7...	8 280	69		1 124	svchost
325	25	6 608	8 524	64		392	svchost
349	13	3 380	6 444	45		684	svchost
364	42	72 0...	16 3...	162		1 136	svchost

Cliquez sur « Ajouter des critères », choisissez « ProcessName » et « svchost » pour n'afficher que ces processus.

6. Depuis le volet de commandes, exécutez la cmdlet suivante :

PowerShell

```
Get-ChildItem
```

La liste des fichiers et dossiers du répertoire courant s'affiche dans le volet. Cette commande permet de récupérer les éléments fils d'une structure, par exemple le système de fichier.

7. En effet, même si le parcours dans les répertoires est une option commune, le système de fournisseurs offre d'autres possibilités. Pour obtenir la liste des fournisseurs disponibles :

PowerShell

```
Get-PSProvider
```

```
Name                 Capabilities                                    Drives
----                 ------------                                    ------
WSMan                Credentials                                     {WSMan}
Alias                ShouldProcess                                   {Alias}
Environment          ShouldProcess                                   {Env}
FileSystem           Filter, ShouldProcess                           {C, A, D}
Function             ShouldProcess                                   {Function}
Registry             ShouldProcess, Transactions                     {HKLM, HKCU}
Variable             ShouldProcess                                   {Variable}
Certificate          ShouldProcess                                   {cert}
```

8. Ainsi, vous pouvez consulter les clés de registre grâce à la même commande que pour le système de fichiers, mais en utilisant le lecteur adéquat.

PowerShell

```
cd HKLM:
cd .\System\CurrentControlSet\Services
Get-ChildItem
```

```
    Hive: HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\services


SKC  VC Name                           Property
---  -- ----                           --------
  2   0 .NET CLR Data                  {}
  2   0 .NET CLR Networking            {}
  2   0 .NET Data Provider for Oracle  {}
  2   0 .NET Data Provider for SqlS... {}
  1   0 .NETFramework                  {}
  0   6 1394ohci                       {Start, Type, ErrorControl, ImagePath...}
  2   8 ACPI                           {Start, Type, ErrorControl, ImagePath...}
  0   6 AcpiPmi                        {Start, Type, ErrorControl, ImagePath...}
  1   6 adp94xx                        {Start, Type, ErrorControl, ImagePath...}
```

Remarque : vous pouvez ici aussi utiliser la complétion automatique avec la touche Tabulation.

Exercice 2 : Ecriture de scripts

Tâche 1 : Autoriser l'exécution de scripts

Un script est un fichier texte portant l'extension « ps1 » comprenant des cmdlets et fonctions. Ce fichier peut être chargé et exécuté et permettre ainsi d'automatiser des opérations répétitives.

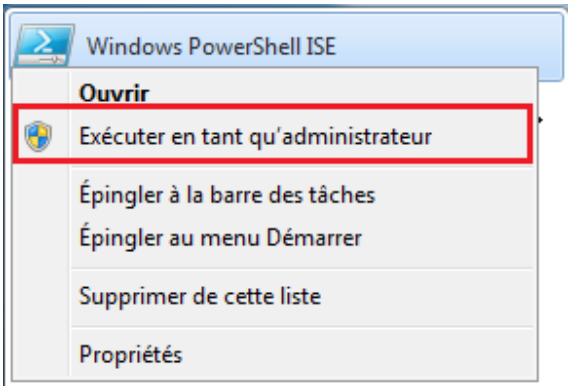
Avant de pouvoir écrire vos propres scripts, vous allez devoir autoriser leur exécution sur votre poste. En effet, PowerShell dispose de 4 modes d'exécutions :

- **Restricted** : aucun script ne peut être exécuté, seules les cmdlets sont autorisées
- **AllSigned** : les scripts peuvent être exécutés mais ils doivent être obligatoirement signés avec un certificat
- **RemoteSigned** : les scripts créés en local n'ont pas besoin de signature
- **Unrestricted** : les scripts n'ont pas besoin d'être signés, qu'ils proviennent du poste local ou d'Internet.

Vous trouverez les informations nécessaires sur la signature de script via la commande « get-help about_signing ».

Le niveau de sécurité est par défaut à « Restricted », il va falloir donc commencer par autoriser l'exécution de nos scripts locaux.

1. Relancez Windows PowerShell en mode administrateur : clic droit sur le raccourci et choisir « Exécuter en tant qu'administrateur ».



2. Vérifiez la stratégie d'exécution courante en tapant dans le volet de commandes :

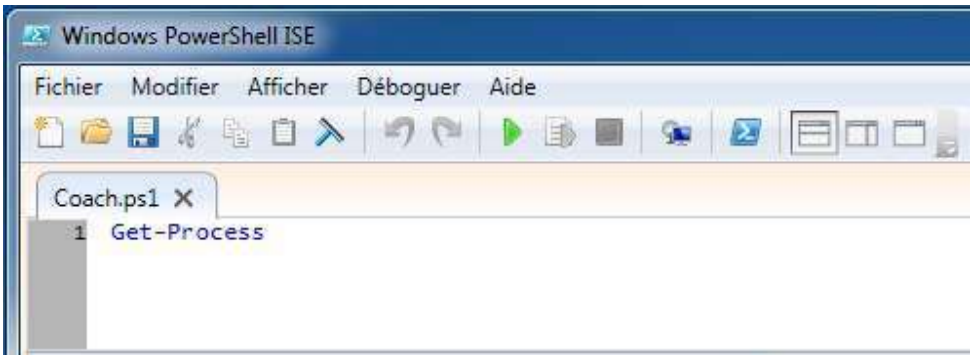
PowerShell

```
Get-ExecutionPolicy
```

Vous devriez obtenir ceci :

```
PS C:\Users\DEMO> Get-ExecutionPolicy
Restricted
```

3. Entrez la commande « Get-Process » dans la zone de script et sauvegardez le fichier en tant que « Coach.ps1 ». L'onglet porte le nom du fichier.



4. Pour exécuter le script, cliquez sur la flèche verte dans la barre d'outils ou bien appuyez sur « F5 ». Une erreur explique que l'exécution de script est désactivée.

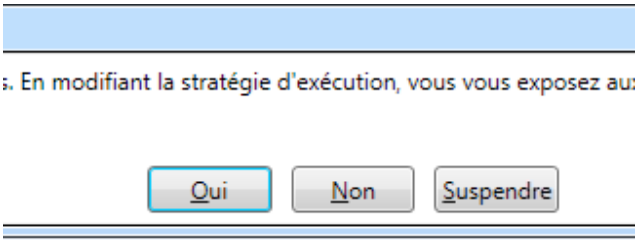
```
PS C:\Users\DEMO> C:\Users\DEMO\Documents\Coach.ps1
Impossible de charger le fichier C:\Users\DEMO\Documents\Coach.ps1, car l'exécution de scri
pts est désactivée sur ce système. Pour plus d'informations, consultez « get-help about_sig
ning ».
At line:0 char:0
```

5. Dans la zone de commandes, modifiez la stratégie d'exécution en « RemoteSigned »

PowerShell

```
Set-ExecutionPolicy RemoteSigned
```

Une popup vous demandera confirmation, validez en cliquant sur « Oui »



Si la modification provoque une erreur, cela provient du fait que vous n'avez pas les droits suffisants pour éditer la base de registre. Vérifiez alors que vous avez bien lancé PowerShell en tant qu'administrateur et que vous avez les droits suffisants.

Tâche 2 : Ecriture d'un script

Maintenant que les scripts réalisés sur le poste local peuvent être exécutés, vous allez pouvoir écrire un premier script PowerShell plus avancé.

1. Vous pouvez relancer votre script qui cette fois-ci fonctionnera parfaitement.
2. Le script n'utilise pour l'instant qu'une commande, or les scripts sont plutôt utilisés pour définir des fonctions réutilisables plus complexes. Effacez le contenu du fichier et remplacez le par le code suivant :

PowerShell

```
function DisplayProcessInfo($processName)
{
    Write-Host "Affichage des processus contenant : $processName"
    $p = Get-Process | Where-Object { $_.Name -like "*$processName*" }
    $p | Format-List Id, Name, Description, Product, VM, StartTime
}
```

Vous avez maintenant votre première fonction « DisplayProcessInfo ». Elle prend en paramètre une chaîne de caractère (variable « $processName ») et affiche tous les processus dont le nom contient cette chaîne.

Vous pouvez remarquer que cette variable est à la fois utilisée pour afficher l'action en cours (ligne « Write-Host ») et pour filtrer la récupération des processus grâce à la clause « Where ». Le terme « $_ » situé entre les crochets correspond à l'élément de la liste récupérée. On accède ainsi à ses propriétés, ici « Name », pour effectuer notre test. Côté opérateurs, outre le « -like » vous aurez les classiques « -eq » (égal) « -ne » (différent) ,« -gt » (plus grand), « -lt » (plus petit)… Pour plus d'informations sur le sujet, n'oubliez pas l'aide intégrée :

PowerShell

```
Get-Help about_operators
```

Enfin, l'affichage des processus est formaté sous forme de liste avec uniquement une certaines des propriétés. Vous pouvez remplacer toutes les propriétés suivant « Format-List » par « * » si vous désirez toutes les afficher.

3. Sauvegardez le fichier, chargez-le depuis le volet de commandes et appelez la fonction.

PowerShell

```
./Coach.ps1
DisplayProcessInfo("power")
```

La liste des processus dont le nom contient « power » s'affiche avec les informations demandées :

```
PS C:\users\DEMO\Documents> DisplayProcessInfo("power")
Affichage des processus contenant : power


Id          : 2392
Name        : powershell_ise
Description : Windows PowerShell ISE
Product     : Microsoft® Windows® Operating System
VM          : 830586880
StartTime   : 26/03/2010 15:10:43

Id          : 3336
Name        : powershell_ise
Description : Windows PowerShell ISE
Product     : Microsoft® Windows® Operating System
VM          : 859557888
StartTime   : 26/03/2010 15:27:53
```

4. Afin de simplifier l'appel, modifiez le fichier pour inclure l'appel à la fonction juste après celle-ci. Vous lancerez automatiquement son exécution lors du chargement du fichier de script.

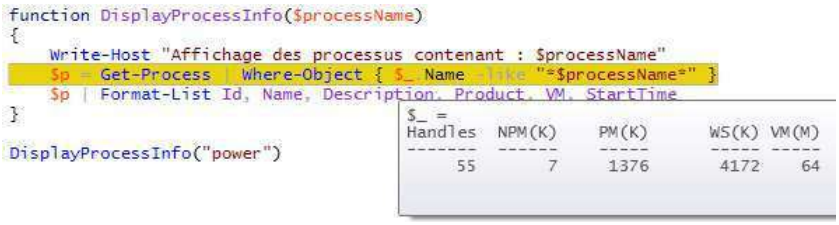
Tâche 3 : Le mode debug

Un autre intérêt des scripts c'est de pouvoir déboguer vos scripts : vous pourrez ainsi facilement suivre l'exécution de votre code et les valeurs de vos variables. Il vous sera plus facile de trouver et comprendre le comportement de votre script et détecter les erreurs potentielles.

1. Effectuez un clic droit sur la ligne 4 et choisissez « Activer/désactiver le point d'arrêt ». Vous pouvez aussi utiliser la touche « F9 », les habitués de Visual Studio retrouveront aussitôt leurs marques.

2. Lancer l'exécution du script en pressant « F5 » ou depuis le menu ou la barre d'outils. Cette fois-ci l'exécution s'arrêtera sur votre point d'arrêt. Vous pouvez dès lors faire du pas à pas avec F10.

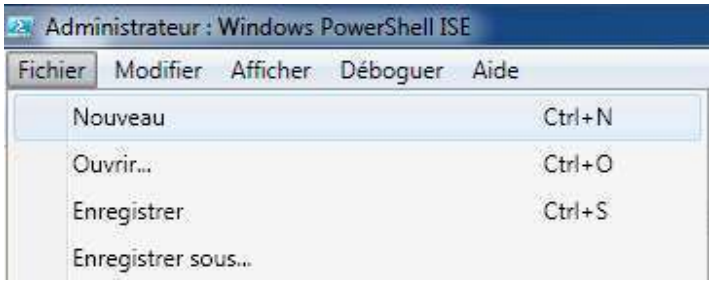
3. Les variables sont visualisables en maintenant le curseur de la souris dessus. Regardez par exemple les variables « $processName » et « $_.Name ».



Tâche 4 : Aller plus loin dans l'écriture de scripts

Afin d'explorer quelques autres aspects de l'écriture de scripts, vous allez maintenant réaliser une petite application permettant de démarrer ou arrêter un service d'après son nom.

1. Créez un nouveau fichier



2. Saisissez le code suivant :

PowerShell

```
$servName = Read-Host "Veuillez saisir le nom du service : "
Write-Host "Voici les services trouvés : "
$services = Get-Service "*$servName*"
if ($services.Count -gt 0)
{
    $count = 0;
    # Affichage des services trouvés
    foreach($serv in $services)
    {
        $count++
        Write-Host -foregroundcolor Red [$count] $serv.DisplayName - $serv.Name
        Write-Host -foregroundcolor Red "   " Status : $serv.Status
    }

    # Récupération de l'élément
    $servNum = Read-Host "Veuillez saisir le numéro du service : "
    $selectedService = $services[$servNum-1]
    if ($selectedService)
    {
        Write-Host -foregroundcolor Green Service sélectionné : $selectedService.Name
        # Selon l'état du service, proposer l'arrêt ou le démarrage
```

```
        switch ($selectedService.Status)
        {
            "Running" {
                $stopIt = Read-Host "Souhaitez-vous arrêter le service ? [O/N] : "
                if ($stopIt -like "o")
                {
                    Write-Host Arrêt du service
                    Stop-Service $selectedService.Name
                }
                else {
                    Write-Host Commande annulée
                }
            }
            "Stopped" {
                $startIt = Read-Host "Souhaitez-vous démarrer le service ? [O/N] : "
                if ($startIt -like "o")
                {
                    Write-Host Démarrage du service
                    Start-Service $selectedService.Name
                }
                else {
                    Write-Host Commande annulée
                }
            }
            default
            {
                Write-Host Le service ne peut pas être démarré ou arrêté
            }
        }
    }
    else
    {
        Write-Host Le numéro ne correspond pas à un des services listés
    }
}
else
{
    Write-Host Aucun service trouvé
}
```

3. Quelques explications :
    - Vous allez effectuer la saisie grâce à la commande « Read-Host » que vous allez stocker dans des variables réutilisées par la suite
    - La commande « Write-Host » peut être personnalisée pour afficher le texte dans d'autres couleurs, très pratique lorsque les scripts affichent beaucoup de lignes pour améliorer la lisibilité
    - Les cmdlets de gestion des services permettent de récupérer l'ensemble des services, de les démarrer ou de les arrêter (« Get-Service », « Start-Service », « Stop-Service »).

4. Mettez à profit les notions de deboguage de l'exercice précédent pour tester votre script et son déroulement.

Exercice 3 : Exécution de commandes distantes

Une des grosses nouveautés de cette deuxième version de PowerShell tient en sa capacité à exécuter des commandes à distance. Il devient alors très pratique d'administrer ses machines sans avoir à s'y connecter. Nous allons couvrir de sa configuration à son utilisation.

Tâche 1 – Configuration de l'accès distant

Pour pouvoir utiliser PowerShell à distance, il vous faut configurer le service « Windows Remote Management » (WinRM) sur l'ordinateur cible. Vous trouverez les informations complètes d'installation et de configuration de WinRM sur TechNet à la page suivante : http://msdn.microsoft.com/en-us/library/aa384372(VS.85).aspx.

Dans cette première étape, nous allons montrer la configuration simplifiée d'une machine pour être administrable à distance. Nous partons des postulats suivants :

- Les deux machines sont dans le même domaine
- Windows Remote Management est correctement installé sur l'ordinateur et le service est lancé. Dans le doute, lancez la commande « Start-Service WinRM »

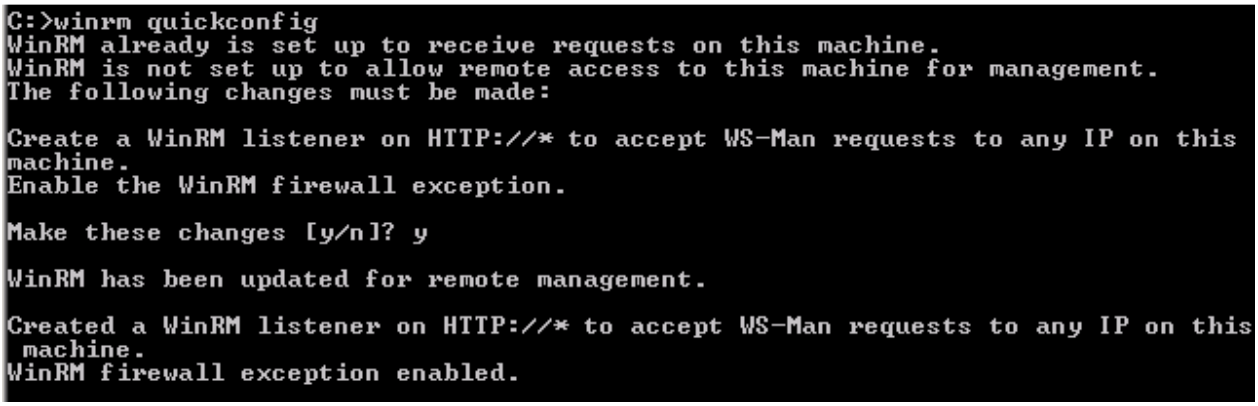
1. Windows Remote Management propose en standard plusieurs scripts d'administration, dont une configuration simplifiée que vous allez utiliser ici. Tout d'abord, vérifiez que vous êtes connecté avec un compte administrateur local.

2. Lancez alors la commande suivante en prenant soin d'effectuer un « Lancer en tant qu'administrateur » soit dans un prompt classique, soit depuis PowerShell (ISE ou commande).

Prompt / PowerShell

```
winrm quickconfig
```

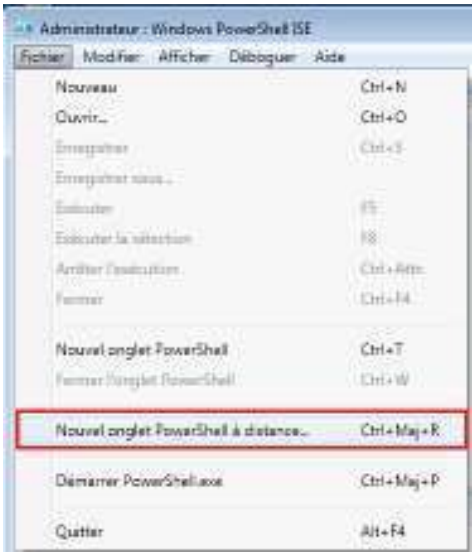
WinRM vous proposera de configurer automatiquement l'ordinateur pour accepter les requêtes, ajouter un « listener » (le moyen de recevoir des commandes) et ajouter les règles d'exception à votre pare-feu. Répondez par l'affirmative pour les changements à effectuer (« y »).



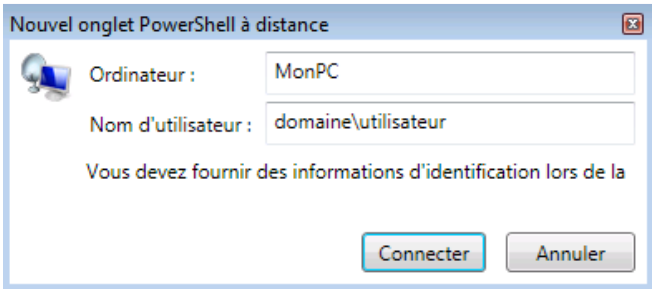
Votre poste supporte maintenant les connexions.

3. Pour le vérifier, connectez-vous depuis l'autre machine et lancez « Windows PowerShell ISE ».

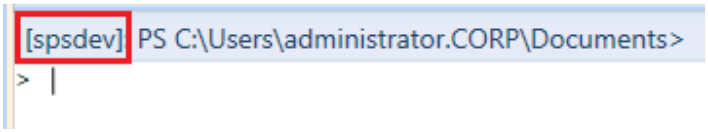
4. Depuis la barre d'outils, lancez un nouvel onglet PowerShell à distance depuis le menu « Fichier » ou l'icône correspondante dans la barre d'outils (le petit écran).



5. Une fenêtre vous demandant les informations de connexion apparaît, saisissez le nom de la machine cible et le nom de l'utilisateur.



6. Une fois la connexion effectuée, le nom de la machine distante est présente dans la zone de commandes avant le fil d'ariane (ici « spsdev ») :



7. Pour s'en assurer, vous pouvez afficher le nom de la machine courante grâce à la commande suivante.

PowerShell

```
hostname
```

```
[spsdev]: PS C:\Users\administrator.CORP\Documents> hostname
SPSDEV
```

Le résultat est évidemment le nom de la machine cible.
Vous pouvez comparer en exécutant la même commande dans un autre onglet PowerShell en local.

Tâche 2 : Gestion des informations de connexions

Vous vous êtes connecté à distance et vous avez pu lancer des cmdlets. Mais avez-vous remarqué comment PowerShell a traduit votre saisie pour effectuer l'ouverture d'une session ? En effet, on retrouve dans le volet de sortie les 2 lignes suivantes :

```
PS C:\Users\administrator> Enter-PSSession 'spsdev' -Credential:'corp\administrator'
if ($?) {$psISE.CurrentPowerShellTab.DisplayName = 'spsdev'}
```

PowerShell a lancé la commande « Enter-PSSession » permettant de créer une session sur une machine dont le nom est passé en argument et a ajouté le paramètre « Credential » pour passer le compte à utiliser. La deuxième ligne modifie le nom de l'onglet pour le rendre plus lisible.

Nous allons voir maintenant comment gérer ces connexions.

1. Ouvrez un nouvel onglet (local).

2. Nous allons utiliser une session pour créer un contexte afin de pouvoir lancer nos commandes à distance. Commencez par consulter les cmdlets de gestion des sessions.

PowerShell

```
Get-Command *session
```

```
Name                                Category  Synopsis
----                                --------  --------
New-PSSession                       Cmdlet    New-PSSession [[-ComputerName]
Get-PSSession                       Cmdlet    Get-PSSession [[-ComputerName]
Remove-PSSession                    Cmdlet    Remove-PSSession [-Id] <Int32[]
Enter-PSSession                     Cmdlet    Enter-PSSession [-ComputerName]
Exit-PSSession                      Cmdlet    Exit-PSSession [-Verbose] [-Deb
Export-PSSession                    Cmdlet    Export-PSSession [-Session] <PS
Import-PSSession                    Cmdlet    Import-PSSession [-Session] <PS
```

3. Créez une nouvelle session sur l'ordinateur distant.

PowerShell

```
New-PSSession –ComputerName MonPC –Credential:Domaine\Utilisateur
```

Un message indique la création de la session.

```
PS C:\Users\administrator> New-PSSession -computername spsdev -Credential:corp\administrator

 Id Name            ComputerName    State    ConfigurationName     Availability
 -- ----            ------------    -----    -----------------     ------------
  2 Session2        spsdev          Opened   Microsoft.PowerShell     Available
```

4. Ensuite, connectez vous à cette session.

PowerShell

```
Enter-PSSession –Name Session2
```

5. Vous passez alors en mode connecté. Affichez le nom de la machine pour le vérifier.

PowerShell

```
hostname
```

6. Listez les services de l'ordinateur distant actuellement lancés.

PowerShell

```
Get-Service | Where { $_.Status -eq 'Running' }
```

```
Status   Name               DisplayName
------   ----               -----------
Running  AppHostSvc         Application Host Helper Service
Running  BFE                Base Filtering Engine
Running  CertPropSvc        Certificate Propagation
Running  CryptSvc           Cryptographic Services
Running  DcomLaunch         DCOM Server Process Launcher
Running  Dhcp               DHCP Client
Running  Dnscache           DNS Client
Running  DPS                Diagnostic Policy Service
```

7. Quittez la session :

PowerShell

```
exit
```

8. Si vous manipulez plusieurs sessions, par exemple pour pouvoir atteindre plusieurs machines, sachez que vous pouvez récupérer à tout moment leur liste :

PowerShell

```
Get-PSSession
```

Ici par exemple 2 sessions sont disponibles.

```
Id Name            ComputerName    State    ConfigurationName     Availability
-- ----            ------------    -----    -----------------     ------------
 2 Session2        spsdev          Opened   Microsoft.PowerShell     Available
 3 Session3        spsdev          Opened   Microsoft.PowerShell     Available
```

9. Vous pouvez réutiliser une session une fois quittée, sauf si vous la supprimez comme suit :

PowerShell

```
Remove-PSSession –Name Session2
```

Tâche 3 – Lancer une commande simple à distance

Jusqu'à présent nous avons ouvert directement une session pour pouvoir lancer plusieurs commandes sans avoir à se reconnecter. Cependant, il n'est pas forcément nécessaire de passer par ces sessions lorsque l'on ne désire lancer qu'une ou deux commandes. C'est ici que « Invoke-Command » prend tout son sens.

> Remarque : remplacez les occurrences de « *MonPC* » par le nom de l'ordinateur cible.

1. Lancez « Windows PowerShell ISE » et ouvrez un onglet local.

2. Regardez l'aide de la commande « Invoke-Command »

PowerShell

```
Get-Help Invoke-Command
```

Il faut fournir le nom de l'ordinateur (« computername ») et les commandes à exécuter (« scriptblock »).

3. Pour obtenir le nom de l'ordinateur distant, lancez la commande suivante

PowerShell

```
Invoke-Command -ComputerName MonPC -ScriptBlock { hostname }
```

4. Si vous avez besoin de lancer plusieurs commandes avec la même session, utilisez « New-PSSession » pour générer l'identité que vous passerez à chaque appel via l'argument « session » comme suit :

PowerShell

```
$session = New-PSSession –computername MonPC
Invoke-Command –session $session –scriptblock { $p = get-process }
Invoke-Command –session $session –scriptblock { $p | Format-List Id, Name, Description }
```

Vous remarquez que la session est réutilisable et que le contexte d'utilisation est préservé : la variable « $p » est conservée entre les deux appels.

Remarque : attention, dans l'exercice précédent, nous passions le nom de la session pour s'y connecter. Ce nom ne peut pas être utilisé ici car c'est l'objet session qui est passé à « Invoke-Command » et non une chaîne de caractères.

5. Plutôt qu'un bloc de script, vous pouvez aussi exécuter un script local sur une machine distante sans avoir à le copier. Pour cela, utilisez le paramètre « FilePath » en lui passant le chemin du fichier ps1 situé sur la machine source.

PowerShell

```
Invoke-Command –computername MonPC –FilePath .\Coach.ps1
```

Remarque : vous pouvez évidemment réutiliser la session comme dans l'exemple précédent.

Ressources

Vous trouverez de nombreuses ressources sur Internet, mais voici déjà un bon point de départ :

- Site officiel Windows PowerShell : http://www.microsoft.com/powershell
- Blog Windows PowerShell: http://blogs.msdn.com/powershell
- Téléchargement du Windows Management Framework dont fait partie Windows PowerShell : http://support.microsoft.com/kb/968929