# Découverte de PowerShell – partie 2

## Présentation

Cette activité permet de découvrir la création de script en abordant les comptes utilisateurs locaux sur un système Windows.

Les scripts sont présentés avec l’environnement d'écriture de scripts intégré Windows PowerShell ISE (*Integrated Scripting Environment*) installé par défaut avec Windows 7. Il est possible d’utiliser un autre éditeur pour PowerShell, comme PowerGUI.

Pour réaliser l’activité, vous devez disposer des scripts Powershell disponibles dans le dossier scriptPowershell.

Un corrigé est présenté en fin du document.

### Utilisation de Windows PowerShell ISE

Lancer Windows PowerShell ISE sous Windows 7:

Démarrer\Tous les programmes\Accessoires\Windows PowerShell\Windows PowerShell ISE

Présentation des différentes fenêtres :

L’éditeur présente trois volets, un premier volet pour l'éditeur de scripts, un deuxième volet de sortie pour afficher le résultat d’exécution et un volet qui correspond à la console de saisie interactive des commandes PowerShell (PS). Suite à l’exécution d’un script, les variables de celui-ci sont accessibles dans la console de saisies des commandes.



Efface le volet de sortie

Exécute le script (F5)

Edition du script

Volet de sortie, résultat de l’exécution

Console de saisie des commandes PS

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

### Exercice 1. Accès aux comptes locaux du système

Le premier script étudié permet d’afficher la date de dernière connexion d’un compte local du système.

### A faire :

- Ouvrir le script lastlogin.ps1 dans Windows PowerShell ISE (Fichier\Ouvrir).

- Exécuter le script, saisir un compte inexistant, vérifier le résultat dans le volet de sortie.

- Exécuter le script, saisir un compte local existant, Administrateur par exemple, vérifier la date de dernière connexion dans le volet de sortie.

Si le compte existe, affiche la dernière connexion

Saisie du nom du compte, placé dans la variable $nom

Explications de la ligne 4 : L’accès à la base locale de comptes utilisateurs d’un système Windows est réalisé avec l’instruction : *[ADSI]"WinNT://."*

Remarque : Ici, les majuscules et les minuscules doivent être respectées.

Le point représente le nom du système sur lequel est lancée l’instruction, il peut être remplacé par le nom de l’ordinateur cible. Il faut bien sûr avoir des droits d’administration sur l’ordinateur distant.

Pour filtrer les éléments de la base de comptes, il est possible de spécifier le nom de l’élément recherché, ici il est contenu dans la variable $nom : *[ADSI]"WinNT://./$nom"*

### A faire (seulement si le compte a été trouvé) :

### - Dans la console de saisie des commandes (Volet 3), afficher les propriétés de la variable $compte, en utilisant la commande suivante : *$compte | Get-Member*

### - Dans le volet de sortie (volet 2), relever le nom des propriétés qui permettent d’afficher le nom complet et la description d’un compte.

### - Modifier le script lastlogin.ps1 pour qu’il affiche ces informations en plus de la date de dernière connexion. Tester (si rien ne s’affiche, c’est que la propriété n’est pas renseignée).

### Exercice 2. Ajout d’un compte local du système

Le second script étudié permet d’ajouter un compte local à partir de la saisie du nom et de la description du compte.

### A faire :

- Ouvrir le script ajoutCompte.ps1 dans Windows PowerShell ISE (Fichier\Ouvrir).

- Exécuter le script, saisir un nom et une description pour le compte local à ajouter, exemple : test01, description : Test d’ajout d’un compte.

- Vérifier la création du compte : Sur le bureau, clic droit sur ordinateur, menu Gérer, Utilisateurs et groupes locaux, dossier Utilisateurs :



Remarque : Par défaut, le nom complet est identique au nom.

Le script :

Connexion à la base locale

Saisie du nom et de la description

Teste si le compte existe déjà

Renseigne la description

Valide la création et la modification des informations

### Explications :

Ligne 11 : La variable $local possède une méthode create() qui permet de créer un objet de type "user" (premier paramètre), dont le nom du compte est spécifié par le deuxième paramètre, ici la variable $nom. Le résultat est une variable nommée $utilisateur.

Ligne 12 : La variable $utilisateur possède une méthode InvokeSet() qui permet de renseigner une information du compte, spécifiée par le premier paramètre, dont la valeur est contenue dans le deuxième, ici $description.

### A faire :

### - Modifier le script ajoutCompte.ps1 pour que le nom complet soit également saisi et renseigné au moment de l’ajout du compte.

- Faire un test avec un nouveau compte (test02) et un nom complet

### Exercice 3. Parcours d’un fichier texte contenant les informations des comptes utilisateurs

Ce script permet d’afficher tous les noms des comptes utilisateurs contenus dans un fichier texte.

Les informations sont sous la forme : nomCompte/nomComplet/Description

### A faire :

- Ouvrir le script lireFichier.ps1 dans Windows PowerShell ISE (Fichier\Ouvrir).

- Copier le fichier listeCompte.txt dans le dossier c:\testPowerShell

- Exécuter le script et vérifier la liste des noms affichés dans le volet de sortie (volet 2).

Chemin du fichier listeCompte.txt

Tableau ($colLignes) qui contient toutes les lignes du fichier

Ce tableau peut être parcouru comme une collection de lignes.

Affiche le premier élément d’un compte : nomCompte

### Explications :

Ligne 10 : Cette instruction peut s’interpréter de cette manière : Pour chaque ligne ($ligne) contenue dans l’ensemble des lignes ($colLignes). La variable $ligne va successivement prendre la valeur de chaque ligne du tableau $colLignes.

Ligne 11 : $ligne est une chaîne de caractères. La variable $ligne possède une méthode Split() qui permet de retourner un tableau construit à partir de la chaîne de caractères contenue dans $ligne. Les éléments du tableau correspondent aux chaînes de caractères délimitées par le séparateur "/".

### A faire :

### - Modifier le script lireFichier.ps1 pour que le nom complet et la description soient également affichés en dessous du nom du compte. Tester.

- A l’aide des deux derniers scripts, ajoutCompte.ps1 et lireFicher.ps1, écrire un script qui permet d’ajouter dans la base locale du système, tous les comptes contenus dans le fichier listeCompte.txt.

- Tester l’ajout de ces comptes pour obtenir :



- Faire un deuxième test avec le même fichier, que se passe-t-il pour les noms déjà existants ?

### Exercice 4. Suppressions de comptes utilisateurs

Ce script permet de supprimer tous les comptes utilisateurs dont les noms sont contenus dans un fichier texte.

Les informations sont toujours sous la forme : nomCompte/nomComplet/Description

### A faire :

- Ouvrir le script suppressionCompteFichier.ps1 dans Windows PowerShell ISE (Fichier\Ouvrir).

- Exécuter le script et vérifier la liste des noms affichés dans le volet de sortie (volet 2).

- Vérifier la suppression des comptes dans Utilisateurs et groupes locaux, dossier Utilisateurs.

Le script :

Suppression du compte s’il existe

### Explications :

Ligne 17 : Comme pour la création, la variable $local possède une méthode delete() qui permet de supprimer un objet de type "user" (premier paramètre), dont le nom du compte est spécifié par le deuxième paramètre, ici la variable $nom.

### A faire :

- Réaliser les modifications pour que la suppression concerne également les comptes test01 et test02 créés au paragraphe 2.

- Exécuter le script et vérifier si tous les comptes créés ont été supprimés.

### Le pipeline et la variable $\_

Le pipeline, symbolisé par le caractère « | » ([AltGr] + [6]) permet de « chainer » plusieurs commandes. Autrement dit, la sortie d'une commande est liée à l'entrée de la suivante.

Exemple :

Get-Children | Get-Member

La sortie de la commande Get-Children n’est pas affichée mais envoyée à la commande Get-Member.

Autre exemple, plus complexe :

Get-ChildItem -recurse | Where {$\_.extension -eq ".txt"}

La première commande renvoie le contenu du dossier courant et de tous ses sous-dossiers. Chaque résultat, un par un, est passé à la commande where qui permet de filtrer. Ici, on ne veut que les fichiers dont l’extension est .txt. Le $\_ correspond donc à chaque résultat envoyé par la première commande. Enfin, on peut filtrer l’affichage, par exemple si on ne veut que les noms de fichiers :

Get-ChildItem -recurse | Where {$\_.extension -eq ".txt"} | Select fullname

### Les structures de contrôle

#### Les conditions if else elseif

Une structure conditionnelle permet, via une évaluation de condition, d’orienter l’exécution vers un bloc d’instructions ou vers un autre. La syntaxe générale d’une structure conditionnelle est la suivante :

If (condition)

{

 #bloc d’instructions

}

Prenons un exemple, imaginons que nous souhaitions déterminer si une valeur entrée par l’utilisateur est la lettre A. Pour cela, nous allons utiliser une structure conditionnelle avec une condition sur la valeur de la variable testée. En utilisant un opérateur de comparaison, la structure est la suivante :

$var = Read-Host "Entrez un caractère"

If ($var -eq ’A’)

{

 "Le caractère saisi par l’utilisateur est un ‘A’"

}

On peut aussi mettre un else :

$var = Read-Host "Entrez un caractère"

If ($var -eq ’A’)

{

 "Le caractère saisi par l’utilisateur est un ‘A’"

}

Else

{

 "Le caractère saisi par l’utilisateur n’est pas un ‘A’ !"

}

Et un ou plusieurs elseif :

$var = Read-Host "Entrez un caractère"

If ($var -eq ’A’)

{

 "Le caractère saisi par l’utilisateur est un ‘A’"

}

Elseif ($var -eq ’B’)

{

 "Le caractère saisi par l’utilisateur est un ‘B’"

}

Else

{

 "Le caractère saisi par l’utilisateur n’est pas un ‘A’ ni un ‘B’ !"

}

Il est important de préciser que l'instruction "Else" doit toujours être la dernière si vous désirez inclure une ou plusieurs instructions ElseIf : question de logique en fait.

Les différents opérateurs de comparaison :

|  |  |
| --- | --- |
| -eq | égal à |
| -lt | plus petit que |
| -gt | plus grand que  |
| -ge | plus grand ou égal  |
| -le | plus petit ou égal  |
| -ne | différent  |

Et quelques autres opérateurs utiles :

|  |  |
| --- | --- |
| -not  | Not |
| !  | Not |
| -and  | And |
| -or  | Or |

On peut réaliser des conditions complexes :

If (($var1 -eq 15) -and -not ($var2 -eq 18))

{

 write-host "lkjlkj" }

Quelques variables automatiques :

|  |  |
| --- | --- |
| $null | Représente la valeur INDEFINIE |
| $true | Représente la valeur VRAI |
| $false | Représente la valeur FAUX |

### Les répétitions

PowerShell propose toutes les boucles utilisées dans les langages de programmation :

— While

— Do…While

— Do…Until

— For

— Foreach

La dernière est la plus utilisée dans les scripts. Le Foreach est pratique lorsqu'il y a besoin de manipuler une collection de données. Elle va automatiquement traiter, un par un, tous les éléments de cette collection et il n'y a pas besoin de connaître à l'avance le nombre !

Contrairement à un simple for. La syntaxe est la suivante :

Foreach(<élément> in <collection>)

{

 # bloc d'instructions qui traite un élément

}

Et voici un exemple qui utilise la commande Get-Childitem déjà présentée :

|  |
| --- |
| $collection = get-childitemForeach ($element in $collection){ if(Test-Path -Path $element -PathType Container) { write-host($element.Name + " est un dossier") } else { write-host($element.Name + " est un fichier") }} |

On obtient tous les éléments (dossiers et fichiers) contenus dans le dossier courant. On parcourt cette collection et on teste si c’est un dossier ou un fichier.

### Les paramètres de script

Une script peut recevoir des paramètres en utilisant le bloc param :

param(

[string]$p1,

[int]$p2

)

Write-Host($p1 + ' a ' + $p2 + ' ans')

On l’exécute de la façon suivante :

Un contrôle est réalisé automatiquement par PowerShell des types de paramètres. Ici, une erreur indique que « toto » n’a pas pu être converti en nombre

## Pour en savoir plus

* Site Web de Windows PowerShell : <http://www.microsoft.com/powershell>
* Site Openclassroom :
	+ <https://openclassrooms.com/fr/courses/3664366-creez-votre-premier-script-avec-powershell>
	+ <https://openclassrooms.com/fr/courses/2356306-prenez-en-main-windows-server/5836391-administrez-votre-serveur-a-l-aide-de-powershell>

## Proposition de Corrigé

### Exercice 1. Accès aux comptes locaux du système

### - Modifier le script lastlogin.ps1 pour qu’il affiche ces informations en plus de la date de dernière connexion. Tester (si rien ne s’affiche, c’est que la propriété n’est pas renseignée).

### Le script de la correction lastlogincorrection.ps1 :

****

### Exercice 2. Ajout d’un compte local du système

### - Modifier le script ajoutCompte.ps1 pour que le nom complet soit également saisi et renseigné au moment de l’ajout du compte.

### Le script de la correction ajoutcomptecorrection.ps1 :

****

### Exercice 3. Parcours d’un fichier texte contenant les informations des comptes utilisateurs

### - Modifier le script lireFichier.ps1 pour que le nom complet et la description soient également affichés en dessous du nom du compte. Tester.

### Le script de la correction lireFichierCorrection.ps1 :



- A l’aide des deux derniers scripts, ajoutCompte.ps1 et lireFicher.ps1, écrire un script qui permet d’ajouter dans la base locale du système, tous les comptes contenus dans le fichier listeCompte.txt.

### Le script de la correction ajoutCompteFichierCorrection.ps1 :



- Faire un deuxième test avec le même fichier, que se passe-t-il pour les noms déjà existants ?

Ils sont affichés comme déjà existant.

### Exercice 4. Suppressions de comptes utilisateurs

- Réaliser les modifications pour que la suppression concerne également les comptes test01 et test02 créés au paragraphe 2.

Il suffit de rajouter les comptes test01 et test02 dans le fichier listeCompte.txt. Bien sûr, il ne faut pas modifier le script de suppression.

- Exécuter le script et vérifier si tous les comptes créés ont été supprimés.

Oui, les comptes qui ont été supprimés la première fois sont affichés comme inexistant.