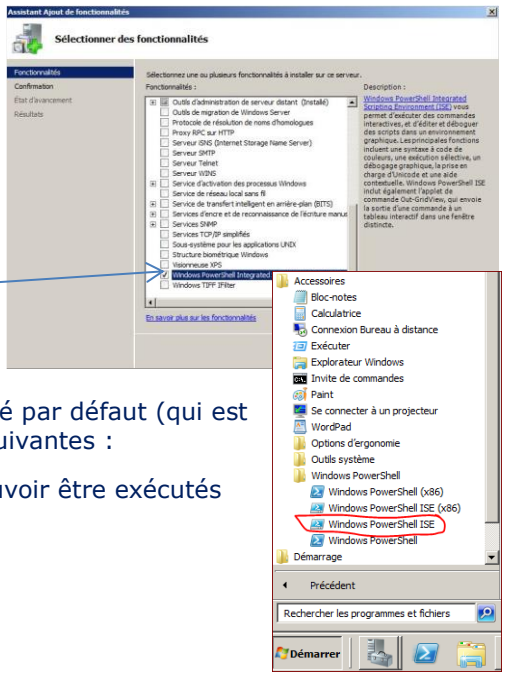
Chap8.2 PowerShell : Scripting

1. PowerShell sous Windows Serveur 2008 (pas 2012)



PowerShell est installé nativement dans cet environnement. Cependant, sous WS2008 (pas 2012) une étape de configuration est nécessaire avant de pouvoir utiliser Windows PowerShell ISE.

- ✓ Lancez votre machine Windows Serveur, puis connectez-vous en tant qu'administrateur.
- ✓ L'IDE PowerShell ISE n'est pas installé par défaut. Pour l'obtenir, il faut ouvrir le gestionnaire de serveur (icône à droite du menu démarrer) et cliquez sur Fonctionnalités dans l'arborescence de gauche. Sur la droite, un lien permet d'ajouter cette fonctionnalité.

Dans la suite de ce document, vous utiliserez l'IDE PowerShell ISE pour exécuter des commandes simples et aussi pour créer des scripts.

Pour exécuter des scripts, il est nécessaire de modifier la politique de sécurité par défaut (qui est « Restricted » ou exécution de scripts interdite) par l'une des deux suivantes :

- ➢ ***RemoteSigned*** : les scripts non locaux doivent être signés pour pouvoir être exécutés
- ➢ ***Unrestricted :*** tous les scripts peuvent être exécutés

La commande est : « > Set-ExecutionPolicy *PolitiqueChoisie* »

2. Les fondamentaux

2.1. Les variables, constantes et opérateurs

2.1.1. Création et affectation

Il suffit d'affecter via l'opérateur " = ", une valeur à votre variable pour déclarer une variable, PowerShell détermine son type. La syntaxe utilisée est la suivante : $variable = valeur d'un type quelconque

À l'inverse pour lire une variable, il suffit de taper tout simplement le nom de la variable dans la console.

Vous pouvez retrouver le type d'une variable avec **la méthode GetType** : **nomVar. GetType().**

- ➢ Quel est le type d'une variable contenant un entier, une chaîne de caractères ? Notez vos commandes.

$a=12; $a="a"; Int32

- ➢ Il est possible de définir explicitement le type d'une variable. Testez l'instruction suivante, quelle est le résultat ?

[int]$var=12; [int]$nombre = read-host 'Entrez un nombre ' Saisir « cent »

Quelle différence entre $var = 12 et [int]$var = 12 ? La différence est le type de variable qui est spécifié dans la deuxième.

Le fait de déclarer vos variables avec un type associé rendra le script beaucoup plus compréhensible pour les autres mais permet surtout **d'éviter qu'une valeur d'un type différent ne lui soit affectée**.

2.1.2. Les variables prédéfinies

Liste non exhaustive : Pour afficher le contenu d'une variable : « echo $Home » par exemple

Variable	Description
$?	Variable contenant true si la dernière opération a réussi ou false dans le cas contraire.
$_	Variable contenant l'objet courant transmis par l'opérateur pipe « | ».
$Args	Variable contenant un tableau des arguments passés à une fonction ou à un script.
$Home	Variable contenant le chemin (path) du répertoire de base de l'utilisateur.
$Host	Variable contenant des informations sur l'hôte.

Variable	Description
$Profile	Variable contenant le chemin (path) du profil Windows PowerShell.
$PWD	Variable indiquant le chemin complet du répertoire actif.

2.1.3. Les opérateurs

2.1.3.1. Les opérateurs arithmétiques

Signe	Signification
+	Addition
-	Soustraction
*	Multiplication
/	Division
%	Modulo

2.1.3.2. Les opérateurs de comparaison

Opérateur	Signification
-eq	Egal
-ne	Non égal (différent)
-gt	Strictement supérieur
-ge	Supérieur ou égal
-lt	Strictement inférieur
-le	Inférieur ou égal

2.1.3.3. Les opérateurs de comparaison générique

Une expression générique est une expression qui contient un caractère dit « générique ». Par exemple « ***** » pour signifier n'importe quelle suite de caractères, ou un « **?** » pour un unique caractère. Il existe deux opérateurs de comparaison qui vous permettent de comparer une chaîne avec une expression générique.

Opérateur	Signification
-like	Comparaison d'égalité d'expression générique.
-notlike	Comparaison d'inégalité d'expression générique.

2.1.3.4. Les opérateurs de comparaison des expressions régulières

Une expression régulière appelée également « RegEx » est une expression composée de ce que l'on appelle des « métacaractères », qui vont correspondre à des valeurs particulières de caractères (Help about_Regular_Expression).

Opérateur	Signification
-match	Comparaison d'égalité entre une expression et une expression régulière.
-notmatch	Comparaison d'inégalité entre une expression et une expression régulière.

Pour mieux comprendre l'utilisation de ces opérateurs, testez les exemples, notez et **expliquez** le résultat :

- PS > 'Powershell' -match 'power[sol]hell'

```
PS C:\Users\teopa> 'Powershell' -match 'power[sol]hell'
True
```

Car [sol] le comprend comme un s et powershell est égale power[s]hell, c'est une comparaison.

- PS > 'powershell' -match 'powershel[a-k]'

```
PS C:\Users\teopa> 'powershell' -match  'powershel[a-k]'
False
```

Car il manque un « l » et ce « l » n'est pas inclus dans [a-k].

- PS > 'powershell' -match 'powershel[a-z]'

```
PS C:\Users\teopa> 'powershell' -match  'powershel[a-z]'
True
```

Car le "l" est présent dans [a-z] et il manque un « l ».

2.1.3.5. Les opérateurs de type

Opérateur	Signification
-is	Test si l'objet est du même type.
-isnot	Test si l'objet n'est pas du même type.

Pour mieux comprendre l'utilisation de ces opérateurs, testez les exemples, notez et **expliquez** le résultat :

- PS > 'Bonjour' -is [string]

```
PS C:\Users\teopa> 'Bonjour' -is [string]
True
```

Car le "l" est présent dans [a-z] et il manque un « l ».

- PS > 20 -is [int]

```
PS C:\Users\teopa> 20 -is [int]
True
```

20 est un entier donc c'est vrai

- PS > 'B' -is [int]

```
PS C:\Users\teopa> 'B' -is [int]
False
```

Le 'B' n'est pas un entier donc c'est faux.

2.1.3.6. Les opérateurs logiques

Opérateur	Signification
-and	Et logique
-or	Ou logique
-not	Non logique
!	Non logique
-xor	OU exclusif

Pour mieux comprendre l'utilisation de ces opérateurs, testez les exemples, notez et **expliquez** le résultat :

- PS > (5 -eq 5) -and (8 -eq 9)

```
PS C:\Users\teopa> (5 -eq 5) -and (8 -eq 9)
False
```

Compare chaque élément donc 5 et 5 donne True mais toute la commande donne False

- PS > (5 -eq 5) -or (8 -eq 9)

```
PS C:\Users\teopa> (5 -eq 5) -or (8 -eq 9)
True
```

Il faut que soit l'une soit l'autre soit vrais ce qui est le cas pour la 1ere

- PS > -not (8 -eq 9)

```
PS C:\Users\teopa> -not (8 -eq 9)
True
```

Cette commande signifie 8 et 9 ne sont pas équivalent ce qui est vrai.

- PS > !(8 -eq 9)

```
PS C:\Users\teopa> !(8 -eq 9)
True
```

C'est la même signification du « -not »

2.1.3.7. Les opérateurs d'affectation

Notation classique	Notation raccourcie
$i=$i+8	$i+=8
$i=$i-8	$i-=8

Notation classique	Notation raccourcie
$i=$i*8	$i*=8
$i=$i/8	$i /=8
$i=$i%8	$i%=8
$i=$i+1	$i++
$i=$i-1	$i--

2.1.3.8. Les opérateurs de redirection

Opérateur	Signification
>	Redirige le flux vers un fichier, si le fichier est déjà créé, le contenu du fichier précédent est remplacé.
>>	Redirige le flux dans un fichier, si le fichier est déjà créé, le flux est ajouté à la fin du fichier.
2>&1	Redirige les messages d'erreurs vers la sortie standard.
2>	Redirige l'erreur standard vers un fichier, si le fichier est déjà créé, le contenu du fichier précédent est remplacé.
2>>	Redirige l'erreur standard vers un fichier, si le fichier est déjà créé, le flux est ajouté à la fin du fichier.

- PS > Get-Process > c:\temp\process.txt

Ca redirige le flux vers le fichier /pocess.text dans le temp

- PS > Get-ChildItem c:\temp\RepInexistant 2> c:\err.txt

Ca envoi les erreurs vers le fichier err.txt

2.1.3.9. Opérateurs de fractionnement et de concaténation

Opérateur	Signification
-split	Fractionne une chaîne en sous-chaînes.
-join	Concatène plusieurs chaînes en une seule.

Pour mieux comprendre l'utilisation de ces opérateurs, testez les exemples, notez et **expliquez** le résultat :

- PS > -split "PowerShell c'est facile"

```
PS C:\Users\Administrateur> -split "Powershell c'est facile"
Powershell
c'est
facile
```

- PS > 'Nom:Prenom:Adresse:Date' -split ':'

```
PS C:\Users\Administrateur> 'Nom:Prenom:Adresse:Data' -split ':'
Nom
Prenom
Adresse
Data
```

- PS > $tableau = 'Lundi', 'Mardi', 'Mercredi', 'jeudi', 'Vendredi', 'Samedi', 'Dimanche'
- PS > -join $tableau

```
PS C:\Users\Administrateur> $tableau = 'Lundi','Mardi','Mercredi', 'Jeudi', 'Vendredi',

PS C:\Users\Administrateur> -join $tableau
LundiMardiMercrediJeudiVendrediSamediDimanche
```

- PS > $tableau -join ', puis '

```
PS C:\Users\Administrateur> $tableau -join ', puis '
Lundi, puis Mardi, puis Mercredi, puis Jeudi, puis Vendredi, puis Samedi, puis Dimanch
e
```

2.2. Alias et profil : personnaliser son environnement

Les alias sont ce que l'on pourrait appeler « surnoms d'une commandelette », ils sont souvent utiles lorsque l'on utilise des commandes un peu longues à taper. Ainsi, l'alias «commande» pourrait par exemple être attribué à la commande « commandevraimenttroplongue ».

Pour ceux qui sont déjà habitués au Shell Unix ou au CMD.exe et qui ont leurs petites habitudes, PowerShell a pensé à eux et leur facilite la tâche grâce à des alias de commandelette mode « Unix » / « CMD » de façon à ne pas les déstabiliser. Par exemple les utilisateurs Unix peuvent utiliser quelques commandes comme : ls, more, pwd, etc.

Ces commandes sont des alias de commandelettes préenregistrées dans PowerShell. Par exemple, ls est un alias de la commande Get-ChildItem qui liste les fichiers et les répertoires.

2.2.1. Lister les alias

Pour rechercher tous les alias de votre session, aussi bien ceux déjà prédéfinis que ceux que vous avez créés, tapez tout simplement : **Get-Alias**

- Quels sont les alias de la commande « Set-Location » ? → Get-Alias -Definition Set-Location

```
PS C:\Users\Administrateur> Get-Alias -Definition Set-Location

CommandType     Name                                               Version    Source
-----------     ----                                               -------    ------
Alias           cd -> Set-Location
Alias           chdir -> Set-Location
Alias           sl -> Set-Location
```

2.2.2. Créer un alias

- Testez et commentez la commande : New-Alias -Name grep -Value Select-String

2.2.3. Modifier un alias

- Testez et commentez la commande : Set-Alias -Name wh -Value Write-Host. Quelle différence avec la commande précédente → Get-Help (!)

Set-alias modifie un alias alors que new alias créer un nouveau un alias

Les créations et modifications d'alias faites en cours de session **sont perdues une fois la session fermée**. Pour retrouver vos alias personnalisés à chaque session, vous devrez les déclarer dans un fichier script particulier, appelé **profil**, qui est chargé automatiquement au démarrage de chaque session PowerShell.

2.2.4. Le profil : personnaliser PS en modifiant son profil

La notion de profil existe depuis longtemps dans Windows avec, entre autres, le fameux «profil Windows » (qui peut être local ou itinérant).

Quatre profils supplémentaires sont utilisés par PowerShell. Il faut tout d'abord distinguer deux sortes de profils :

- Les **profils utilisateurs** (au nombre de deux) qui s'appliquent à l'utilisateur courant.
- Les **profils machines** (deux également) qui s'appliquent à tous les utilisateurs d'une machine.

Nous nous intéresserons ici seulement aux profils utilisateurs.

2.2.5. Les profils utilisateurs

Il existe deux profils utilisateurs portant chacun un nom distinct :

- %UserProfile%\Mes documents\WindowsPowerShell\profile.ps1
- %UserProfile%\Mes documents\WindowsPowerShell**Microsoft.PowerShell_profile.ps1**

Le premier profil est un profil commun à plusieurs environnements (Ms Exchange, plate-forme d'entreprise de messagerie Microsoft, Ms System Center Operation Manager 2007, solution de supervision des systèmes, …) alors que le second est propre à l'environnement PowerShell installé par défaut.

2.2.6. Création du profil

Par défaut, aucun profil n'est créé. La méthode la plus simple pour créer son profil consiste à s'appuyer sur la **variable prédéfinie $profile**. Cette variable contient le chemin complet vers votre profil utilisateur de l'environnement par défaut Microsoft.PowerShell, et ce même si vous ne l'avez pas encore créé.

- Que contient la variable $profile ? Notez son contenu :

```
PS C:\Users\Administrateur> $profile
C:\Users\Administrateur\Documents\WindowsPowerShell\Microsoft.PowerShellISE_profile.ps
1
```

- Utilisez la commande New-Item pour créer votre profil (« Get-Help New-Item –Examples » : afficher des exemples et -Detailed : aide complète vous donnera la commande à utiliser). Notez cette commande :

```
$
```

Modifiez votre profil pour ajouter les deux alias "grep et wh créés au chapitre précédent : **> notepad $profile**

- Fermez votre fenêtre PS ISE et relancez-là pour vérifier que les nouveaux alias sont bien conservés

→ Vous aurez besoin de changer la stratégie d'exécution (cf. l'aide) si ça n'a pas été effectué avant.

2.3. Les tableaux, redirection, pipeline

Editable aussi avec l'ISE si vous l'utilisez

2.3.1. Initialiser un tableau à une dimension

Pour à la fois créer un tableau et l'initialiser, il suffit de lui affecter plusieurs valeurs séparées par une virgule.

- Par exemple : $tab = 1,5,9,10,6 que contient $tab ?, $tab[0] ?, $tab[5] ?

```
PS C:\Users\Administrateur> $tab = 1,5,9,10,6

PS C:\Users\Administrateur> $tab[0]
1

PS C:\Users\Administrateur> $tab[5]

PS C:\Users\Administrateur> $tab[5]

PS C:\Users\Administrateur> $tab
1
5
9
10
6
```

- Testez et commentez : $tab=1..20

```
PS C:\Users\Administrateur> $tab=1..20

PS C:\Users\Administrateur> $tab
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
```

- Testez et commentez : $tab+=50

```
PS C:\Users\Administrateur> $tab+=50

PS C:\Users\Administrateur> $tab
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
50
```

- Testez et commentez : $tab[5]=555

```
PS C:\Users\Administrateur> $tab[5]=555

PS C:\Users\Administrateur> $tab
1
2
3
4
5
555
7
8
9
10
11
12
13
14
15
16
17
18
19
20
50
```

- Testez et commentez : $tab=$tab[0..9 + 15..21]

```
PS C:\Users\Administrateur>
PS C:\Users\Administrateur> $tab=$tab[0..9 + 15..21]

PS C:\Users\Administrateur> $tab
1
2
3
4
5
555
7
8
9
10
16
17
18
19
20
50
```

2.3.2. Redirection et pipeline

il est possible de connecter des commandes, de telle sorte que la sortie de l'une devienne l'entrée de l'autre. C'est ce qu'on appelle le pipeline.

- Testez et commentez : Get-Command | Out-File -FilePath fichier.txt

- Testez et commentez : Get-Command | Out-null

- Testez et commentez : "Get-ChildItem C:\Windows | ForEach-Object {$_.Get_extension().toLower()} | Sort-Object | Get-Unique| Out-File -FilePath extensions.txt". **Pour comprendre cette commande**, vous devez tester chaque partie unitairement. La variable $_ représentant l'objet courant passé par le pipe

2.3.3. Filtre Where-object

La commande Where-Object (alias : Where) est très utilisée dans les pipelines. Elle permet de sélectionner les objets retournés par la commande précédente de façon à ne garder que ceux qui nous intéressent.

- Testez et commentez : « Get-Service | Where-Object {$_.Status -eq 'Stopped'} ». La variable $_ représente (toujours) l'objet courant passé par le pipe

- Testez et commentez : Get-Process | Where-Object {$_.TotalProcessorTime.totalmilliseconds -gt 300}

- Trouvez la commande qui permet d'afficher les fichiers dont la taille est supérieure à 500 octets dans le répertoire c:\Windows. Pour vous aider, placez un fichier dans une variable **($maVar = Get-ChildItem fichier**), recherchez ses propriétés par la commande « **$mavar | Get-Member -MemberType Property** ». Lorsque vous avez trouvé la bonne propriété, inspirez-vous des commandes précédentes…

2.4. Les structures itératives et alternatives

Enregistrez les scripts suivants avec l'extension ps1.

2.4.1. Les structures de boucle : While, For et Foreach

- Testez et commentez :

```
$nombre = 0
$tab = 20..30
While($nombre -lt $tab.Length)
{
  Write-Host $tab[$nombre]
  $nombre++
}
```

```
PS C:\Users\Administrateur>
$nombre = 0
$tab =20..30
While($nombre -lt $tab.Length)
{
  Write-Host $tab[$nombre]
  $nombre++
}
20
21
22
23
24
25
26
27
28
29
30
```

- Testez et commentez :

```
Do
{
  Write-host 'Entrez une valeur entre 0 et 10'
  [int]$var = read-host
}
While( ($var -lt 0 ) -or ($var -gt 10))
```

```
PS C:\Users\Administrateur>
Do
{
  Write-host 'Entrez une valeur entre 0 et 10'
  [int]$var = read-host
}
While(($var -lt 0 ) -or ($var -gt 10))
Entrez une valeur entre 0 et 10
11
Entrez une valeur entre 0 et 10
54
Entrez une valeur entre 0 et 10
1
```

- Testez et commentez :

```
$tab = 55..65
For($i=0 ;$i -le $tab.Length ;$i++)
{
  Write-Host $tab[$i]
}
```

Renvoie les valeurs du tableau tab donc entre 55 et 65.

```
PS C:\Users\Administrateur>
$tab = 55..65
For($i=0; $i -le $tab.Length; $i++)
{
  Write-Host $tab[$i]
}
55
56
57
58
59
60
61
62
63
64
65
```

Foreach-Object est une cmdlet et non une instruction de boucle. Cette cmdlet également disponible sous l'appellation Foreach en raison d'un alias, permet de parcourir les valeurs contenues dans une collection.

➢ Testez et commentez :

```
Foreach ($element in Get-Process)
{
  Write-Host "$($element.Name)
      démarré le :
$($element.StartTime)"
}
```

```
PS C:\Users\Administrateur>
Foreach ($element in Get-Process)
{
 Write-Host "$($element.Name)
      démarré le : $($element.StartTime)"
}
AggregatorHost
      démarré le : 11/06/2024 08:22:55
csrss
      démarré le : 11/06/2024 08:22:35
csrss
      démarré le : 11/06/2024 08:22:37
ctfmon
      démarré le : 11/06/2024 08:26:03
dllhost
      démarré le : 11/06/2024 08:28:07
dwm
      démarré le : 11/06/2024 08:22:41
explorer
      démarré le : 11/06/2024 08:26:07
fontdrvhost
      démarré le : 11/06/2024 08:22:40
fontdrvhost
      démarré le : 11/06/2024 08:22:40
Idle
      démarré le :
lsass
      démarré le : 11/06/2024 08:22:38
MicrosoftEdgeUpdate
      démarré le : 11/06/2024 08:23:03
msdtc
      démarré le : 11/06/2024 08:25:07
MsMpEng
      démarré le : 11/06/2024 08:22:53
```

➢ Testez et commentez :

Get-Process | Foreach{$_.Name} | Sort -unique	Get-Process : Récupère une liste de tous les processus en cours d'exécution sur le système. ForEach-Object {$_.Name} : Boucle sur chaque processus et extrait la propriété Name (le nom du processus). Sort-Object -Unique : Trie la liste des noms de processus et supprime les doublons, de manière que chaque nom de processus apparaisse une seule fois.

Foreach-object permet une segmentation entre les tâches à effectuer **avant le premier objet** (**paramètre begin**), les tâches à effectuer **pour chaque objet** (**paramètre process**) et les tâches à effectuer **après le dernier objet** (**paramètre end**).

➢ Testez et commentez, à quoi sert le caractère d'échappement « `n » ?

Get-Process | Foreach-Object -begin {
Write-Host "Début de liste des processus`n"} `
-process {$_.Name } -End {
Write-Host "`nfin de liste des processus `n"}
	Attention le caractère ` (AltGr7) seul signifie que la commande n'est pas terminée et se poursuit à la ligne suivante.
	'N' sert à insérer un saut de ligne.
	Il doit être placé à l'intérieur d'une chaîne de caractères pour être interprété.
	Une mauvaise utilisation peut entraîner des erreurs de syntaxe et des résultats inattendus.

2.4.2. Les structures conditionnelles : If, Else, ElseIf, Switch

➢ Testez :

```
$var2 = Read-Host
If($var2 -eq 'A')
{
  Write-Host "Le caractère saisi par l'utilisateur est un 'A' "
}
Else
{
  Write-Host "Le caractère saisi par l'utilisateur est différent de 'A' "
}
```

➢ Ecrire un programme qui demande la saisie de deux nombres et affiche si le premier nombre est plus petit ou plus grand que le second.

```
$nombre1 = Read-Host "Entrez le premier nombre"

$nombre2 = Read-Host "Entrez le second nombre"

if ($nombre1 -lt $nombre2) {
   Write-Output "Le premier nombre ($nombre1) est plus petit que le second nombre ($nombre2)."
}
elseif ($nombre1 -gt $nombre2) {
   Write-Output "Le premier nombre ($nombre1) est plus grand que le second nombre ($nombre2)."
}
else {
   Write-Output "Le premier nombre ($nombre1) est égal au second nombre ($nombre2)."
}
```

➢ Il est possible de rajouter une commande **ElseIf (condition) {…}.** Réécrivez le programme précédent en prenant en compte l'égalité.

```
if ($nombre1 -lt $nombre2) {
   Write-Output "Le premier nombre ($nombre1) est plus petit que le second nombre ($nombre2)."
}
elseif ($nombre1 -gt $nombre2) {
   Write-Output "Le premier nombre ($nombre1) est plus grand que le second nombre ($nombre2)."
}
elseif ($nombre1 -eq $nombre2) {
   Write-Output "Le premier nombre ($nombre1) est égal au second nombre ($nombre2)."
```

➢ Testez et commentez :

$chaine = Read-Host 'Entrez une chaine'
Switch -regex ($chaine)
{
'^[aeiouy]' {Write-Host 'commence par une voyelle'}
'^[^aeiouy]' {Write-Host 'ne commence pas par une voyelle'}
}
	Renvoi si l chaine commande par une voyelle

2.5. Gestion des chaînes de caractères

➢ Testez et commentez les commandes suivantes qui utilisent les **méthodes de la classe string** :

$message = "Bonjour"	

```
$message = $message + " Monde"

$longueur = $message.Length
Write-Host "Longueur : $longueur"

$sousCh = $message.Substring(0, 6)
Write-Host "Sous-chaîne : $sousCh"

$msgSansEspaces = $message.Replace(" ", "")
Write-Host "Sans espaces : $msgSansEspaces"

$msgMinus = $message.ToLower()
Write-Host "En minuscules : $msgMinus"

$message="Bonjour-Tout-Le-Monde"

$mots = $message.Split("-")
Write-Host $mots[0] # "Bonjour"
Write-Host $mots[1] # "Tout"
Write-Host $mots[2] # "Le"
Write-Host $mots[3] # "Monde"
```

```
$mot
$message
```

2.6. Gestion des fichiers texte

PowerShell dispose de commandes permettant d'interagir avec des fichiers texte.

- Créez un nouveau fichier texte dans le même répertoire que celui de vos scripts PowerShell. Nommez-le whatever.txt.
- Copiez-collez le texte ci-dessous dans votre fichier.

```
Whatever you do
Whatever you say
Yeah I know it's alright
```

- Dans l'IDE PowerShell ISE, créez un nouveau script. Sauvegardez-le au même endroit que le fichier texte sous le nom fichiers.ps1.

- Testez le code source ci-dessous et modifiez-le pour que chaque ligne affichée commence par son numéro et soit écrite en majuscules.

```
# Parcours du fichier ligne par ligne
ForEach ($ligne in Get-Content "whatever.txt")
{
      Write-Host $ligne
}

Résultat attendu :
(1) WHATEVER YOU DO
(2) WHATEVER YOU SAY
(3) YEAH I KNOW IT'S ALRIGHT
```

2.7. Les fonctions

En PowerShell et comme dans de nombreux langages, une fonction est un ensemble d'instructions auquel on va donner un nom. Le principal intérêt des fonctions est que vous pouvez y faire référence à plusieurs reprises, sans avoir à ressaisir l'ensemble des instructions à chaque fois.

Une fonction est constituée des **éléments** suivants :

- ✓ un nom ;
- ✓ un type de portée (facultatif) ;
- ✓ un ou plusieurs arguments (facultatifs) ;
- ✓ un bloc d'instruction.

Syntaxe :

```
Function [<portée> :] <nom de fonction> (<argument>)
{
  param (<liste des paramètres>)
  # bloc d'instructions
}
```

Appel : nomFonction Arg1 Arg2 Arg3 …

La portée ne sera pas étudiée dans ce cours.

- Testez et commentez. Notez un exemple d'appel pour chaque fonction :

# Sous-programme sans valeur de retour 	
Function DireBonjour([string]$nom, [int]$age)
{
Write-Host "$message $nom, tu as $age ans"
}
 	
# Sous-programme avec valeur de retour 	
Function Additionner([int]$nb1, [int]$nb2)
{
$somme = $nb1 + $nb2
return $somme
}
	Appel :

- Que se passe-t'il si vous remplacez les doubles quotes par des simples quotes dans la fonction DireBonjour() ?

- Testez la fonction suivante et notez un exemple d'appel de cette fonction :

Function Set-Popup
{
$WshShell = New-Object -ComObject wscript.Shell
$WshShell.Popup($args[0], 0, 'Popup PowerShell')
}	

3. PowerShell et Active Directory: gestion des objets d'un CD

Les exercices suivants doivent s'effectuer sur un serveur de domaine 2008 R2.

Sous Windows Server 2008 R2, un module spécifique nommé "Active Directory pour Windows PowerShell" contient des commandes supplémentaires dédiées à la gestion d'un AD avec PowerShell.

- Pour accéder à ces commandes depuis une console PowerShell ou un script, il faut taper ou ajouter la ligne ci-dessous.

```
Import-Module ActiveDirectory
```

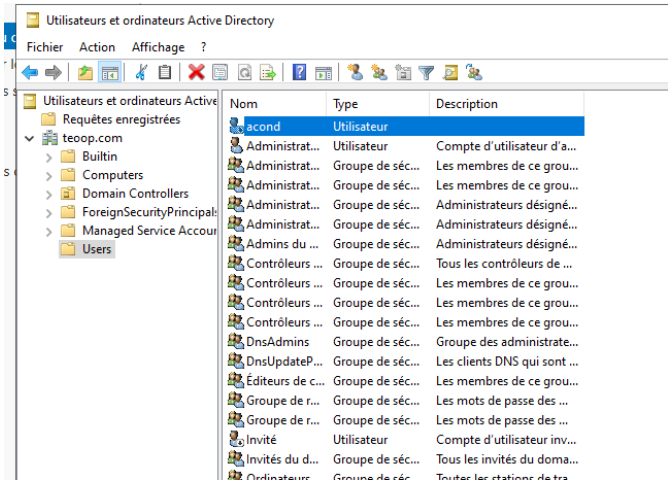
3.1. Ajout d'un utilisateur dans un annuaire

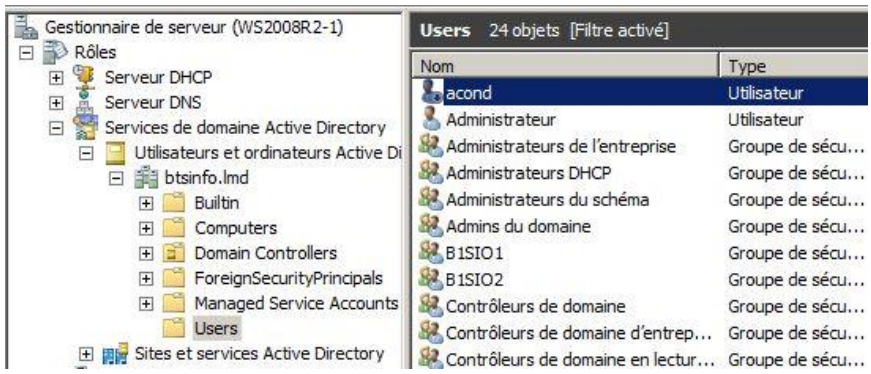
Afin d'illustrer les possibilités de PowerShell, nous allons prendre l'exemple de l'ajout d'un utilisateur dans un annuaire Active Directory. La commande correspondante est **New-ADUser**.

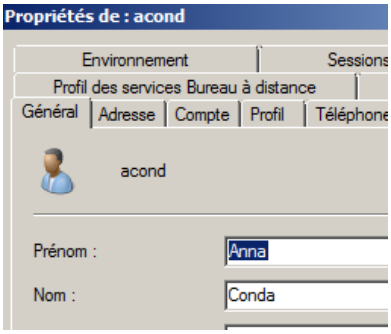
- Testez la commande suivante :

```
> New-ADUser -Name acond -GivenName Anna -SurName Conda
```

- Ouvrez le « Gestionnaire de serveur ». Sous le rôle « Services de domaine AD/Utilisateur et Ordinateur/btsinfo.lmd/Users » : un nouvel utilisateur nommé acond a été ajouté dans l'annuaire.

Quel est son login, son prénom et son nom ?	 Login : acond Son prénom : Anna Son Nom : Conda





3.2. Application : ajout d'une liste d'utilisateurs dans un AD

Afin d'appliquer ce que vous venez de découvrir, vous allez créer un script PowerShell nommé **ajoutAD.ps1**. Son rôle est d'ajouter dans l'AD les utilisateurs définis dans un fichier texte nommé **utils.txt** où chaque ligne contient les informations sur un utilisateur : nom, prénom et login. Vous devrez utiliser, entre autres la commande Split() en indiquant le séparateur de champs utilisé dans le fichier.

Le script PowerShell doit parcourir ce fichier et ajouter chaque utilisateur à l'annuaire.

➔ Vérifier dans l'AD la création effective des utilisateurs.

➢ Voici le contenu de ce fichier. Notez le contenu de votre script sur la partie droite :

AU APPAVOU:Mickael:mauap
BONON:Mélanie:mbono
LEROY:Mélodie:mlero
PEREIRA:Thomas:tpere
MORRIS:Prisca:pmorr
ALLOUCHE:Josepha:jallo
HIDA:Mohamed:mhida
IBANEZ:Gregory:giban
KLEIN:Laurène:lklei
NADOUA:Jean Baptiste:jnado
SANCHEZ:Héloise:hsanc
	Set-Location "C:\Users\Administrateur"
	foreach ($ligne in Get-Content "utils.txt")
	{
	$util=$ligne.Split(",")
	$name=$util[2]
	$givenName=$util[1]
	$surName=$util[0]
	Write-Host "Création de " $name " / " $givenName " / " $surName
	New-ADuser -name $name -GivenName $givenName -Surname $surName
	}

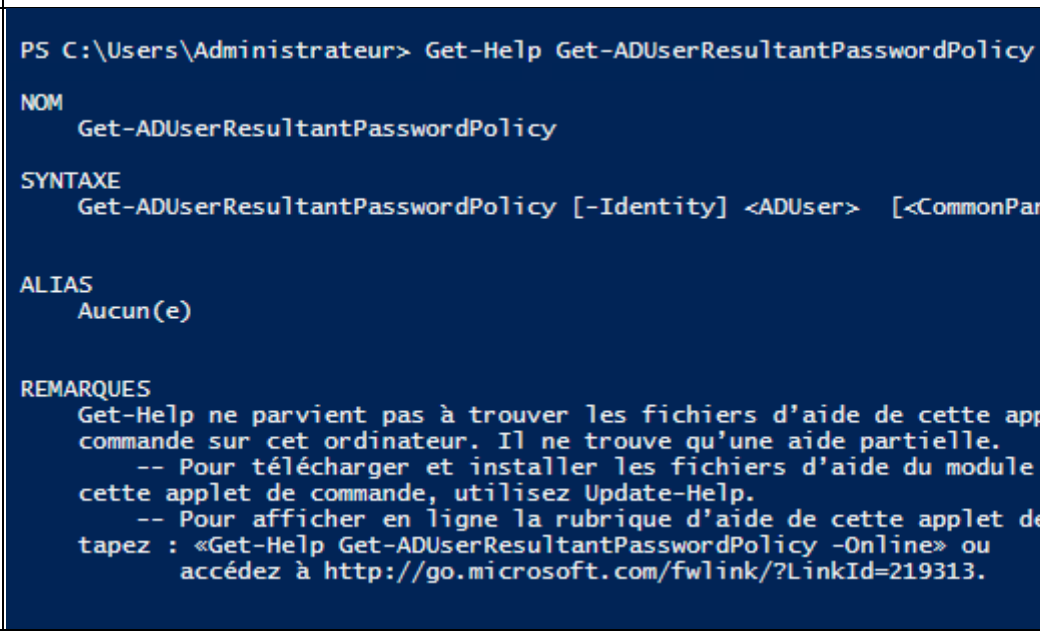
3.3. Complément sur la gestion des utilisateurs

3.3.1. Commandes du module « ActiveDirectory » pour la gestion des utilisateurs

Pour la gestion des utilisateurs, nous disposons d'un jeu de quelques commandes que nous pouvons obtenir de la manière suivante :

➢ PS > Get-Command -Module ActiveDirectory -Name *user*
➢ Complétez la description de chaque commande. Recherchez le détail de ces commandes avec la commande Get-Help –Examples et Get-Help -Detailed

Commande	Description
Get-ADUser	PS C:\Users\Administrateur> Get-Help Get-ADUser NOM Get-ADUser SYNTAXE Get-ADUser [<CommonParameters>] Get-ADUser [-Identity] <ADUser> [<CommonParameters>] Get-ADUser [<CommonParameters>] ALIAS Aucun(e) REMARQUES Get-Help ne parvient pas à trouver les fichiers d'aide de cette applet de commande sur cet ordinateur. Il ne trouve qu'une aide partielle. -- Pour télécharger et installer les fichiers d'aide du module comportant cette applet de commande, utilisez Update-Help. -- Pour afficher en ligne la rubrique d'aide de cette applet de commande, tapez : «Get-Help Get-ADUser -Online» ou accédez à http://go.microsoft.com/fwlink/?LinkId=301397.
Set-ADUser	PS C:\Users\Administrateur> Get-Help Set-ADUser NOM Set-ADUser SYNTAXE Set-ADUser [-Identity] <ADUser> [<CommonParameters>] Set-ADUser [<CommonParameters>] ALIAS Aucun(e) REMARQUES Get-Help ne parvient pas à trouver les fichiers d'aide de cette applet de commande sur cet ordinateur. Il ne trouve qu'une aide partielle. -- Pour télécharger et installer les fichiers d'aide du module comportant cette applet de commande, utilisez Update-Help. -- Pour afficher en ligne la rubrique d'aide de cette applet de commande, tapez : «Get-Help Set-ADUser -Online» ou accédez à http://go.microsoft.com/fwlink/?LinkId=301403.
New-ADUser	PS C:\Users\Administrateur> Get-Help New-ADUser NOM New-ADUser SYNTAXE New-ADUser [-Name] <string> [<CommonParameters>] ALIAS Aucun(e) REMARQUES Get-Help ne parvient pas à trouver les fichiers d'aide de cette applet de commande sur cet ordinateur. Il ne trouve qu'une aide partielle. -- Pour télécharger et installer les fichiers d'aide du module comportant cette applet de commande, utilisez Update-Help. -- Pour afficher en ligne la rubrique d'aide de cette applet de commande, tapez : «Get-Help New-ADUser -Online» ou accédez à http://go.microsoft.com/fwlink/?LinkId=301400.

Commande	Description
Remove-ADUser	
Get-ADUserResultantPasswordPolicy	

3.3.2. Obtenir la liste des utilisateurs

La forme la plus simple pour effectuer une recherche d'utilisateurs à travers tout l'annuaire Active Directory est la suivante :

- PS > Get-ADUser -Filter *

```
PS C:\Users\Administrateur> Get-ADUser -FIlter *


DistinguishedName : CN=Administrateur,CN=Users,DC=teoop,DC=com
Enabled           : True
GivenName         :
Name              : Administrateur
ObjectClass       : user
ObjectGUID        : bf578c00-50d6-4b30-b34d-2ddc7780d262
SamAccountName    : Administrateur
SID               : S-1-5-21-611347392-1980578312-1773222509-500
Surname           :
UserPrincipalName :

DistinguishedName : CN=Invité,CN=Users,DC=teoop,DC=com
Enabled           : False
GivenName         :
Name              : Invité
ObjectClass       : user
ObjectGUID        : 0a44340b-bcf2-47d4-85bf-bcc9fec2e5fe
SamAccountName    : Invité
SID               : S-1-5-21-611347392-1980578312-1773222509-501
Surname           :
UserPrincipalName :

DistinguishedName : CN=krbtgt,CN=Users,DC=teoop,DC=com
Enabled           : False
GivenName         :
Name              : krbtgt
```

Pour obtenir une liste facilement interprétable d'utilisateurs, il est utile de formater le résultat sous forme de tableau, comme ci-dessous :

- PS > Get-ADUser -Filter * | Format-Table GivenName,Surname, Name, Sam*

3.3.3. Affecter un mot de passe à la création

Nous pouvons affecter un mot de passe à la création d'un compte, avec la commande **New-ADUser**. Par contre, pour modifier le mot de passe d'un compte existant, nous devons utiliser **Set-ADAccountPassword.**

Comme la valeur attendue pour le paramètre -AccountPassword est de type chaîne sécurisée (SecureString), il faut effectuer quelques petites manipulations supplémentaires :

- Testez les lignes suivantes

```
PS > $passwd = 'Passw0rd123*!'
PS > $passwd = ConvertTo-SecureString $passwd -AsPlainText -Force
PS > New-ADUser -SamAccountName DucableJR -Name 'JR Ducable' -AccountPassword $passwd
```

```
PS C:\Users\Administrateur> $passwd = 'Passw0rd123'

PS C:\Users\Administrateur> $passwd = ConvertTo-SecureString $passwd -AsPlainText -Force

PS C:\Users\Administrateur> New-ADUser -SamAccountName DucableJr -Name 'Jr Ducable' -Acc
```

3.3.4. Affecter un mot de passe à un compte existant

- Si le compte existe déjà, alors dans ce cas il faudra faire comme ceci :

```
PS > Set-ADAccountPassword -Identity DucableJR -NewPassword $passwd -Reset
```

```
PS C:\Users\Administrateur> Set-ADAccountPassword -Identity DucableJr -Newpassword $pass
```

La commande **`Set-ADUser`** n'autorise pas le changement de mot de passe. Pour ce faire, il faut utiliser la commande **`Set-ADAccountPassword`**.

3.3.5. Activer un compte à la création

Pour activer un compte à la création, il est nécessaire de lui affecter un mot de passe. Ce mot de passe doit bien entendu être en adéquation avec la politique de sécurité de votre domaine.

- Pour créer un compte qui soit actif, suivez l'exemple ci-après :

```
PS > $passwd = 'Passw0rd123*!'
PS > $passwd = ConvertTo-SecureString $passwd -AsPlainText -Force
```

Vous pouvez créer une fonction qui reçoit comme argument un mot de passe et le transforme en une chaîne sécurisée plutôt que de retaper ces 2 lignes à chaque fois….

```
PS > New-ADUser -SamAccountName Posichon -Name 'Paul Posichon' `
        -GivenName Paul -Surname Posichon                    `
        -DisplayName 'Paul Posichon' -description 'Utilisateur terrible !' `
        -AccountPassword $passwd -Enabled $true
```

```
PS C:\Users\Administrateur> New-ADUser -SamAccountName Posichon -Name 'Paul Posichon' -GivenName Pau
```

3.3.6. Activer un compte existant

- Pour activer un compte existant, il faut procéder en deux étapes. La première va consister à lui affecter un mot de passe, et la seconde à activer le compte.

```
PS > Set-ADAccountPassword -Identity BracameE -NewPassword $passwd
PS > Set-ADUser -Identity BracameE -Enabled $true
```

3.3.7. Lire un ou plusieurs attributs

Pour lire un attribut ou propriété d'un utilisateur, il faut tout d'abord se connecter à l'objet d'annuaire correspondant. Pour ce faire, nous utilisons la commande Get-ADUser avec le paramètre -Identity.

- Pour obtenir les propriétés affichées par défaut d'un objet Utilisateur :

```
PS > Get-ADUser -Identity Posichon
```

```
PS C:\Users\Administrateur> Get-ADUser -Identity Posichon


DistinguishedName : CN=Paul Posichon,CN=Users,DC=teoop,DC=com
Enabled           : True
GivenName         : Paul
Name              : Paul Posichon
ObjectClass       : user
ObjectGUID        : 56509227-9968-4fef-ad60-0f930b408712
SamAccountName    : Posichon
SID               : S-1-5-21-611347392-1980578312-1773222509-1105
Surname           : Posichon
UserPrincipalName :
```

- Pour obtenir toutes les propriétés qu'un objet Utilisateur possède :

```
PS > Get-ADUser -Identity Posichon -Properties *
```

```
PS C:\Users\Administrateur> Get-ADUser -Identity Posichon -Properties *


AccountExpirationDate                 :
accountExpires                        : 9223372036854775807
AccountLockoutTime                    :
AccountNotDelegated                   : False
AllowReversiblePasswordEncryption     : False
AuthenticationPolicy                  : {}
AuthenticationPolicySilo              : {}
BadLogonCount                         : 0
badPasswordTime                       : 0
badPwdCount                           : 0
CannotChangePassword                  : False
CanonicalName                         : teoop.com/Users/Paul Posichon
Certificates                          : {}
City                                  :
CN                                    : Paul Posichon
codePage                              : 0
Company                               :
CompoundIdentitySupported             : {}
Country                               :
countryCode                           : 0
Created                               : 13/11/2024 10:03:59
createTimeStamp                       : 13/11/2024 10:03:59
Deleted                               :
Department                            :
Description                           : Utilisateur terrible !
DisplayName                           : Paul Posichon
DistinguishedName                     : CN=Paul Posichon,CN=Users,DC=teoop,DC=com
Division                              :
DoesNotRequirePreAuth                 : False
dSCorePropagationData                 : {01/01/1601 01:00:00}
EmailAddress                          :
EmployeeID                            :
EmployeeNumber                        :
Enabled                               : True
Fax                                   :
GivenName                             : Paul
HomeDirectory                         :
HomedirRequired                       : False
```

➢ Utilisez ce paramètre pour afficher des propriétés qui ne sont pas incluses dans le jeu par défaut.

PS > Get-ADUser -Identity Posichon -Properties CanonicalName, PasswordLastSet, whenCreated

```
PS C:\Users\Administrateur> Get-ADUser -Identity Posichon -Properties CanonicalName, PasswordLastS


CanonicalName     : teoop.com/Users/Paul Posichon
DistinguishedName : CN=Paul Posichon,CN=Users,DC=teoop,DC=com
Enabled           : True
GivenName         : Paul
Name              : Paul Posichon
ObjectClass       : user
ObjectGUID        : 56509227-9968-4fef-ad60-0f930b408712
PasswordLastSet   : 13/11/2024 10:03:59
SamAccountName    : Posichon
SID               : S-1-5-21-611347392-1980578312-1773222509-1105
Surname           : Posichon
UserPrincipalName :
whenCreated       : 13/11/2024 10:03:59
```

Nous nous retrouvons avec tous les autres attributs par défaut.

➢ Pour sélectionner les champs, il faut formater l'affichage avec la commande Format-Table.

PS > Get-ADUser -Identity Posichon -Properties * | Format-Table Name, CanonicalName, PasswordLastSet, SID, whenCreated

```
PS C:\Users\Administrateur> Get-ADUser -Identity Posichon -Properties * | Format-Table Name, Canonic

Name          CanonicalName                PasswordLastSet      SID
----          -------------                ---------------      ---
Paul Posichon teoop.com/Users/Paul Posichon 13/11/2024 10:03:59 S-1-5-21-611347392-1980578312-1...
```

3.3.8. Modifier un attribut

La modification d'un attribut s'effectue avec la commande Set-ADUser.

- Par exemple, pour modifier l'attribut SamAccountName : Ser

```
PS > Set-ADUser -Identity Posichon -SamAccountName Cornichon
```

On peut aussi utiliser le paramètre -Replace pour modifier une ou plusieurs propriétés en une seule opération.

- Exemple à tester : Modification de la description, du numéro de téléphone principal et des autres numéros de téléphone

```
PS > Set-ADUser -Identity Posichon -Replace @{
    Description = 'Utilisateur sympathique' ;
    TelephoneNumber = '0110203040';
    OtherTelephone = @('0250403020','0340506070')}
```

3.3.9. Effacer un attribut

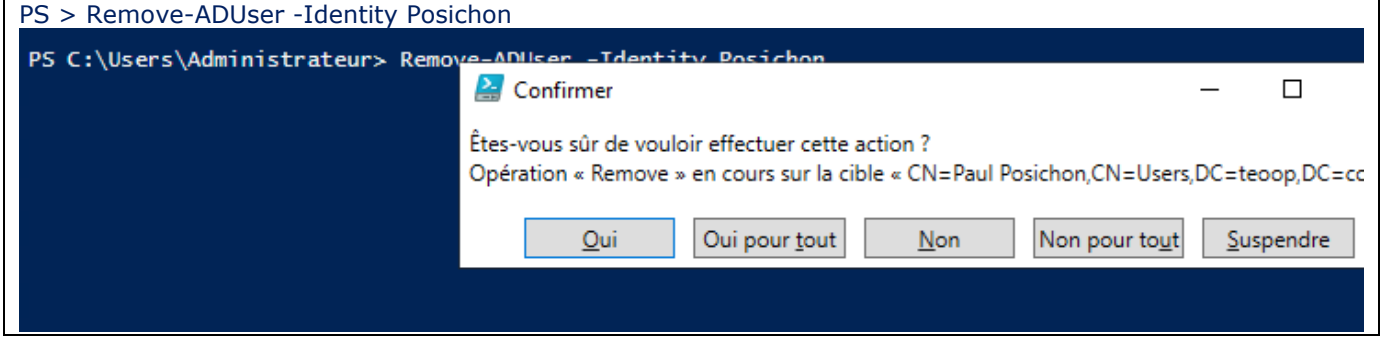
- Supprimons les numéros de téléphone secondaires de l'utilisateur « Posichon », comme ceci :

```
PS > Set-ADUser -Identity Posichon -Clear OtherTelephone
```

3.3.10. Supprimer un utilisateur

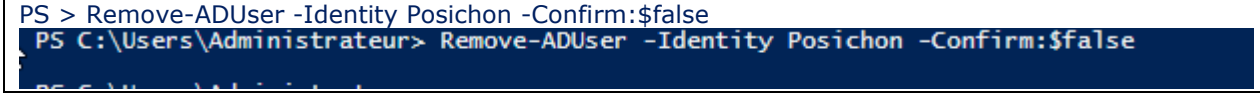
- Il faut utiliser la commande Remove-ADUser.

```
PS > Remove-ADUser -Identity Posichon
```



- Pour outrepasser la confirmation, vous devez faire comme ceci :

```
PS > Remove-ADUser -Identity Posichon -Confirm:$false
```



3.4. Exercices d'application

A partir du même fichier **utils.txt utilisé pour la création des utilisateurs**, écrire les scripts suivants :

3.4.1. Script UtilEnabled.ps1

Ce script lit le fichier utils.txt et active les comptes des utilisateurs (ils sont désactivés par défaut)

- Notez le contenu de votre script :

```
Enabled $true #pour que le compte soit activé
```

3.4.2. Script utilSuppression.ps1

Ce script lit le fichier utils.txt et supprime les utilisateurs

➢ Notez le contenu de votre script :

```
if (Test-Path $fichierUtilisateurs) {
    Get-Content $fichierUtilisateurs | ForEach-Object {
        # Divise la ligne en fonction du séparateur (ici une virgule)
        $infosUtilisateur = $_.Split(',')

        # Assigne chaque champ à une variable
        $nom = $infosUtilisateur[0]
        $prenom = $infosUtilisateur[1]
        $login = $infosUtilisateur[2]

        # Supprime l'utilisateur dans l'AD
        try {
            # Vérifie si l'utilisateur existe avant de tenter la suppression
            $utilisateur = Get-ADUser -Filter { SamAccountName -eq $login }
            if ($utilisateur) {
                Remove-ADUser -Identity $utilisateur -Confirm:$false
                Write-Output "Utilisateur $prenom $nom ($login) supprimé avec succès."
            }
            else {
                Write-Output "Utilisateur $prenom $nom ($login) introuvable dans Active Directory."
            }
        }
        catch {
            Write-Output "Erreur lors de la suppression de l'utilisateur $prenom $nom ($login) : $_"
        }
    }
}
else {
    Write-Output "Le fichier $fichierUtilisateurs est introuvable."
}
```