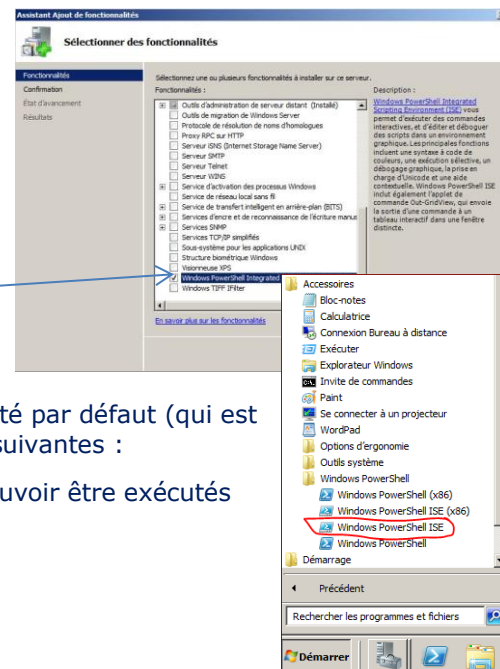


# Chap8.2 PowerShell : Scripting

## 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 cliquer 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";

PS C:\Users\Administrateur> $a=12;
PS C:\Users\Administrateur> $a. GetType()

IsPublic IsSerial Name                                     BaseType
-----
True     True     Int32                                     System.ValueType

PS C:\Users\Administrateur> |
```

- 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 »
```

```

PS C:\Users\Administrateur> [int]$var=12

PS C:\Users\Administrateur> [int]$nombre = read-host '100'
100 : 100

PS C:\Users\Administrateur> [int]$nombre = read-host 'entrez un nombre'
entrez un nombre : 100

PS C:\Users\Administrateur> [int]$nombre = read-host 'entrez un nombre'
entrez un nombre : cent
Impossible de convertir la valeur «cent» en type «System.Int32». Erreur: «Le format
de la chaîne d'entrée est incorrect.»
Au caractère Ligne:1 : 1
+ [int]$nombre = read-host 'entrez un nombre'
+ ~~~~~
+ CategoryInfo          : MetadataError: (:) [], ArgumentTransformationMetadataE
xception
+ FullyQualifiedErrorId : RuntimeException

PS C:\Users\Administrateur>

```

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

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.
\$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\kern6> 'Powershell' -match 'power[sol]hell'
True
PS C:\Users\kern6>
```

car sol le comprend comme un s et powershell est égal à power(s)hell , c'est une comparaison.

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

```
PS C:\Users\kern6> 'powershell' -match 'powershel[a-k]'
False
PS C:\Users\kern6> |
```

Car il manque un L, il n'est pas incluse dans [a-k].

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

```
PS C:\Users\kern6> 'powershell' -match 'powershel[a-z]'
True
PS C:\Users\kern6> |
```

Car le L est présent dans [a-z].

### 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\Administrateur> 'bonjour' -is [string]
True
PS C:\Users\Administrateur>
```

➤ PS > 20 -is [int]

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

➤ PS > 'B' -is [int]

```
PS C:\Users\Administrateur> 'B' -is [int]
false
PS C:\Users\Administrateur> █
```

### 2.1.3.6. Les opérateurs logiques

Opérateur	Signification
-and	Et logique
-or	Ou logique

Opérateur	Signification
-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\Administrateur> (5 -eq 5) -and (8 -eq 9)
False
```

➤ PS > (5 -eq 5) -or (8 -eq 9)

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

➤ PS > -not (8 -eq 9)

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

➤ PS > !(8 -eq 9)

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

### 2.1.3.7. Les opérateurs d'affectation

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

```
PS C:\Users\Administrateur> Get-Process > c:/temp/process.txt
out-file : Impossible de trouver une partie du chemin d'accès 'C:\temp\process.txt'.
Au caractère Ligne:1 : 1
+ Get-Process > c:/temp/process.txt
+ ~~~~~
+ CategoryInfo          : OpenError: (:) [Out-File], DirectoryNotFoundException
+ FullyQualifiedErrorId : FileOpenFailure,Microsoft.PowerShell.Commands.OutFileCommand
```

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

Aucun résultat n'est affiché dans la console.

### 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 C:\Users\Administrateur>
```

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

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

➤ PS > \$tableau = 'Lundi', 'Mardi', 'Mercredi', 'jeudi', 'Vendredi', 'Samedi', 'Dimanche'

➤ PS > -join \$tableau

```
PS C:\Users\Administrateur> $tableau= 'lundi', 'mardi', 'mercredi', 'jeudi', 'vendredi', 'samedi', 'd
PS C:\Users\Administrateur> -join $tableau
lundimardimercredijeudivendredisamedidimanche
PS C:\Users\Administrateur>
```

➤ PS > \$tableau -join ', puis '

## 2.2. Alias et profil : personnaliser son environnement

Les alias sont ce que l'on pourrait appeler « surnoms d'une commande », 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 « commandevraimenttrotroplongue ».

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 commande 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 commandes 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
PS C:\Users\Administrateur>
```

### 2.2.2. Créer un alias

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

```
PS C:\Users\Administrateur> New-Alias -Name grep -Value Select-String
PS C:\Users\Administrateur>
```

### 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 (!)`

```
PS C:\Users\Administrateur> Set-Alias -Name wh -Value Write-Host
PS C:\Users\Administrateur>
```

-`Set-Alias -Name wh -Value Write-Host` : Crée un alias wh pour Write-Host.

-`Get-Help` : Fournit de la documentation ou des informations détaillées sur les cmdlets et les fonctions, mais ne crée pas d'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.PowerShell_profile.ps1
PS C:\Users\Administrateur>
```

- 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 :

```
PS C:\Users\Administrateur> Get-Help New-Item -Examples

NOM
New-Item

ALIAS
ni
```

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

- Fermez votre fenêtre PS ISE et relancez-la 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
```

➤ Testez et commentez : `$tab=1..20`

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

➤ Testez et commentez : `$tab+=50`

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

➤ Testez et commentez : `$tab[5]=555`

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

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

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

### 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`

```
PS C:\Users\Administrateur> Get-Command | Out-File -FilePath fichier.txt
PS C:\Users\Administrateur> |
```

La sortie de cette commande est dans le fichier fichier.txt

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

```
PS C:\Users\Administrateur> Get-Command | Out-null
PS C:\Users\Administrateur>
```

Cette commande supprime immédiatement tout les entrées entrants.

➤ 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



### Recense les extensions des fichiers du répertoire Windows

extensions	13/11/2024 06:49
fichier	06/11/2024 08:49

### 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

```
PS C:\Users\Administrateur> Get-service | Where-Object {$_.Status -eq 'Stopped'}

Status Name                DisplayName
-----
Stopped AJRouter              Service de routeur AllJoyn
Stopped ALG             Service de la passerelle de la couc...
Stopped AppIDSvc        Identité de l'application
Stopped Appinfo         Informations d'application
Stopped AppMgmt         Gestion d'applications
Stopped AppReadiness    Préparation des applications
Stopped AppVClient      Microsoft App-V Client
Stopped AppXSvc         Service de déploiement AppX (AppXSVC)
Stopped AudioEndpointBu... Générateur de points de terminaison...
Stopped Audiosrv        Audio Windows
Stopped AxInstSV        Programme d'installation ActiveX (A...
Stopped BITS            Service de transfert intelligent en...
Stopped bthserv         Service de prise en charge Bluetooth
Stopped CaptureService_... CaptureService_40284
Stopped CertPropSvc     Propagation du certificat
Stopped COMSysApp       Application système COM+
Stopped ConsentUxUserSv... Service utilisateur ConsentUX_40284
Stopped CredentialEnrol... CredentialEnrollmentManagerUserSvc_...
Stopped CscService      Fichiers hors connexion
Stopped defragsvc       Optimiser les lecteurs
Stopped DeviceAssociati... DeviceAssociationBroker_40284
Stopped DeviceAssociati... Service d'association de périphérique
Stopped DeviceInstall   Service d'installation de périphérique
Stopped DevicePickerUse... DevicePicker_40284
Stopped DevicesFlowUser... Flux d'appareils_40284
Stopped DevQueryBroker   Service Broker de découverte en arr...
Stopped diagnosticshub... Service Collecteur standard du conc...
Stopped DmEnrollmentSvc Service d'inscription de la gestion...
Stopped dmwappushservice Service de routage de messages Push...
Stopped DoSvc           Optimisation de livraison
Stopped dot3svc         Configuration automatique de réseau...
```

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



```
PS C:\Users\Administrateur> Get-Process | Where-Object {$_.TotalProcessorTime.totalmilli
```

Handles	NPM(K)	PM(K)	WS(K)	CPU(s)	Id	SI	ProcessName
435	19	1876	6160	0,41	396	0	csrss
285	14	1820	5904	0,44	468	1	csrss
764	40	28484	64792	5,66	972	1	dwm
1820	87	30624	110788	5,47	3684	1	explorer
906	21	4036	13424	0,80	624	0	lsass
605	70	144688	121424	12,16	2240	0	MsMpEng
772	56	152160	190008	74,31	5076	1	powershell_ise
0	7	3992	71068	0,34	100	0	Registry
518	24	9308	36032	0,44	4388	1	RuntimeBroker
1300	79	90864	164332	3,67	4280	1	SearchApp
573	38	104972	117000	2,61	4564	1	ServerManager
526	11	4228	8948	1,48	604	0	services
487	16	4864	26120	0,33	3880	1	sihost
247	11	4132	14496	2,75	2572	0	sppsvc
572	28	13516	54272	0,53	1036	1	StartMenuExperienceHost
972	19	6636	22692	0,69	732	0	svchost
872	16	4916	11288	1,13	848	0	svchost
277	13	3256	11292	0,45	1016	0	svchost
375	15	14932	19564	3,22	1132	0	svchost
372	17	4804	15040	0,45	1464	0	svchost
414	32	7820	17336	0,72	1504	0	svchost
159	9	4960	12844	0,84	1552	0	svchost
466	20	11884	27512	0,69	2028	0	svchost
382	15	9084	19056	3,28	2164	0	svchost
287	16	12308	13700	2,53	4620	0	svchost
1879	0	40	124	5,17	4	0	System
528	23	10060	42256	0,31	1316	1	TextInputHost

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

do

- ➤ Testez et commentez :

<pre>Do {     Write-host 'Entrez une valeur entre 0 et 10'     [int]\$var = read-host } While( (\$var -lt 0 ) -or (\$var -gt 10))</pre>	<pre>PS C:\Users\Administrateur&gt; Do {     Write-host 5     [int]\$var = read-host } While((\$var -Lt 0) -or (\$var -gt 10)) 5</pre>
---	--

➤ Testez et commentez :

<pre>\$tab = 55..65 For(\$i=0 ;\$i -le \$tab.Length ;\$i++) {     Write-Host \$tab[\$i] }</pre>	<pre>PS C:\Users\Administrateur&gt; For(\$i=0 ;\$i -Le \$tab.Length ;\$i++) {     Write-Host \$tab[\$i] } 55,65  PS C:\Users\Administrateur&gt;  </pre>
---	---

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 :

<pre>Foreach (\$element in Get-Process) {     Write-Host "\$(\$element.Name)     démarré le : \$(\$element.StartTime)" }</pre>	<pre>PS C:\Users\Administrateur&gt; Foreach (\$element in Get-Process) {     Write-Host "\$(\$element.Name)     démarré le : \$(\$element.StartTime)" }  AggregatorHost     démarré le : 11/13/2024 07:23:09 ApplicationFrameHost     démarré le : 11/13/2024 07:33:43 csrss     démarré le : 11/13/2024 07:22:51 csrss     démarré le : 11/13/2024 07:22:54 ctfmon     démarré le : 11/13/2024 07:23:35 dfsrs     démarré le : 11/13/2024 09:39:29</pre>
--	---

➤ Testez et commentez :

<pre>Get-Process   Foreach{\$_Name}   Sort -unique</pre>	<pre>PS C:\Users\Administrateur&gt; Get-Process   Foreach{\$_Name}   Sort -unique AggregatorHost ApplicationFrameHost csrss ctfmon dfsr dfssvc DismHost dllhost dwm explorer fontdrvhost GenValObj Idle lsass MicrosoftEdgeUpdate msdtc MsMpEng NisSrv powershell_ise Registry RuntimeBroker SearchApp ServerManager services ShellExperienceHost sihost</pre>
--	--

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 » ?

<pre>Get-Process   Foreach-Object -begin { Write-Host "Début de liste des processus`n"} ` -process {\$_Name} -End { Write-Host "`nfin de liste des processus `n"}</pre>	<p>Attention le caractère ` (AltGr7) seul signifie que la commande n'est pas terminée et se poursuit à la ligne suivante.</p>
---	---

## 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' "
}

PS C:\Users\Administrateur>
$var2 = Read-Host
If($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'"
}

Le caractère saisi par l'utilisateur est différent de 'A'
PS C:\Users\Administrateur>
```

➤ 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.

<pre># Demander à l'utilisateur de saisir le premier nombre \$n1 = Read-Host "Veuillez entrer le premier nombre"</pre>
--

```

➤ # Demander à l'utilisateur de saisir le second nombre
➤ $n2 = Read-Host "Veuillez entrer le second nombre"
➤
➤ # Comparer les deux nombres
➤ if ($n1 -lt $n2) {
➤     Write-Host "Le premier nombre ($n1) est plus petit que le second ($n2)."

```

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

```

➤ # Demander à l'utilisateur de saisir le premier nombre
➤ $n1 = Read-Host "Veuillez entrer le premier nombre"
➤
➤ # Demander à l'utilisateur de saisir le second nombre
➤ $n2 = Read-Host "Veuillez entrer le second nombre"
➤
➤ # Comparer les deux nombres
➤ if ($n1 -lt $n2) {
➤     Write-Host "Le premier nombre ($n1) est plus petit que le second ($n2)."

```

- 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'}
}

```

```

PS C:\Users\Administrateur>
$chaine = Read-Host 'Entrez une chaine'
Switch -regex ($chaine)
{
    '[aeiouy]' {Write-Host 'commence par une voyelle'}
    '[^aeiouy]' {Write-Host 'ne commence pas par une voyelle'}
}
Entrez une chaine : eren
commence par une voyelle
    '[^aeiouy]' {Write-Host 'ne commence pas par une voyelle'}
PS C:\Users\Administrateur>

```

## 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"

```

<pre> \$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" </pre>	<pre> PS C:\Users\Administrateur&gt; \$chaîne = Read-Host 'Entrez une chaîne' Switch -regex (\$chaîne) { '^[aeiouv]' {Write-Host 'commence par une voyelle'} '^^[aeiouv]' {Write-Host 'ne commence pas par une voyelle'} } Entrez une chaîne : eren commence par une voyelle} ^[^aeiouv] {Write-Host 'ne commence pas par une voyelle  PS C:\Users\Administrateur&gt; \$message= "Bonjour"  PS C:\Users\Administrateur&gt; \$message= \$message + "Monde"  PS C:\Users\Administrateur&gt; \$message= "Bonjour" \$message= \$message + "Monde"  PS C:\Users\Administrateur&gt; \$longueur = \$message.Length Write-Host "Longueur : \$longueur" Longueur : 12  PS C:\Users\Administrateur&gt; \$sousCh = \$message.Substring(0, 6) Write-Host "Sous-chaîne : \$sousCh" Sous-chaîne : Bonjour  PS C:\Users\Administrateur&gt; \$msgSansEspaces = \$message.Replace(" ", "") Write-Host "Sans espaces : \$msgSansEspaces" Sans espaces : BonjourMonde  PS C:\Users\Administrateur&gt;   </pre>
---	---

## 2.6. Gestion des fichiers texte

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

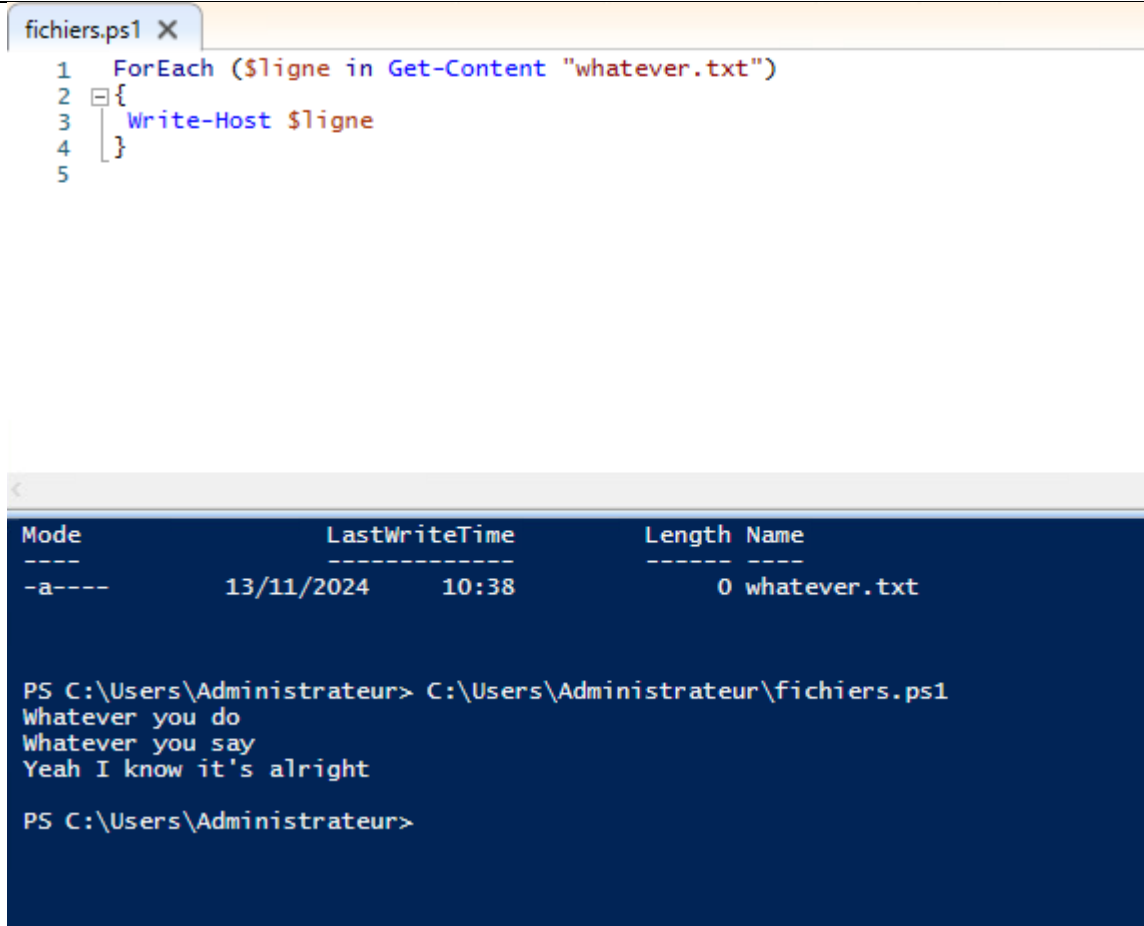
- Créez un nouveau fichier texte dans le même répertoire que celui de vos scripts PowerShell. Nommez-le whatever.txt.
- Copiez-collez le texte ci-dessous dans votre fichier.

```

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.

<pre># Parcours du fichier ligne par ligne ForEach (\$ligne in Get-Content "whatever.tx t") {     Write- Host \$ligne }  Résultat attendu : (1) WHATEVER YOU DO (2) WHATEVER YOU SAY (3) YEAH I KNOW IT'S ALRIGHT</pre>	
---	---

## 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.

<p>Une fonction est constituée des <b>éléments</b> suivants :</p> <ul style="list-style-type: none"> <li>✓ un nom ;</li> <li>✓ un type de portée (facultatif) ;</li> <li>✓ un ou plusieurs arguments (facultatifs) ;</li> <li>✓ un bloc d'instruction.</li> </ul>	<p><b>Syntaxe :</b>            Function [&lt;portée&gt; :] &lt;nom de fonction&gt; (&lt;argument&gt;)            {                param (&lt;liste des paramètres&gt;)                # bloc d'instructions            }  <b>Appel :</b> nomFonction Arg1 Arg2 Arg3 ...            La portée ne sera pas étudiée dans ce cours.</p>
---	---

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

<pre># 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 }</pre>	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 :

<pre>Function Set-Popup {</pre>	
---------------------------------	--

```
$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

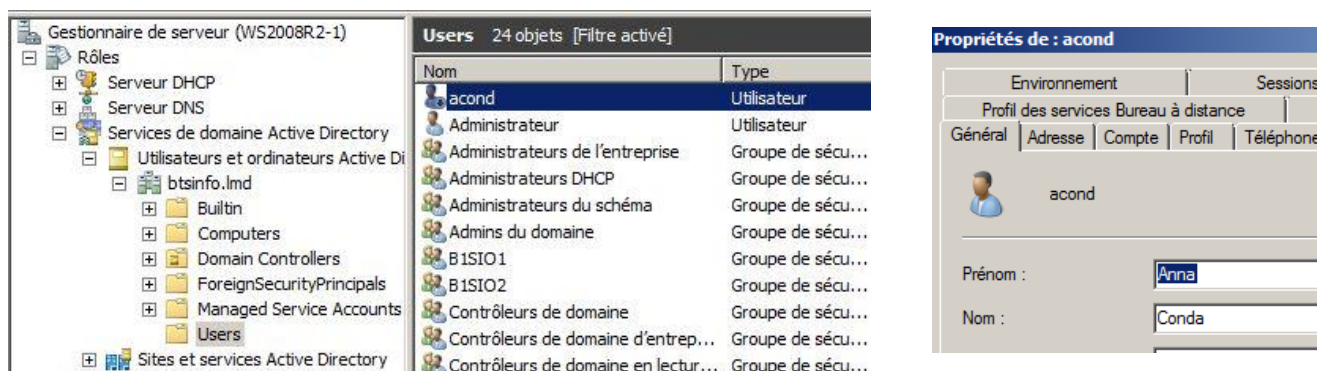
Afin d'illustrer les possibilités de PowerShell, nous allons prendre l'exemple de l'ajout d'un utilisateur dans un annuaire Active Directory. La commande correspondante est **New-ADUser**.

- 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 ? **Son login est acond, Anna et 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:m	
bono	
LEROY:Mélodie:mle	
ro	
PEREIRA:Thomas:t	
pere	
MORRIS:Prisca:pm	
orr	
ALLOUCHE:Josepha	
:jallo	
HIDA:Mohamed:m	
hida	
IBANEZ:Gregory:gi	
ban	
KLEIN:Laurène:lkle	
i	
NADOUA:Jean	
Baptiste:jnado	
SANCHEZ:Héloise:	
hsanc	

The screenshot shows the Windows PowerShell ISE interface. The top pane displays a list of users and their details:

Utilisateur	Groupe de sécurité	Description
Contrôleurs ...	Groupe de séc...	Les membres de ce grou...
Contrôleurs ...	Groupe de séc...	Les membres de ce grou...
DnsAdmins	Groupe de séc...	Groupe des administrate...
DnsUpdateP...	Groupe de séc...	Les clients DNS qui sont ...
Éditeurs de c...	Groupe de séc...	Les membres de ce grou...
giban	Utilisateur	
Groupe de r...	Groupe de séc...	Les mots de passe des ...
Groupe de r...	Groupe de séc...	Les mots de passe des ...
Invité	Utilisateur	Compte d'utilisateur inv...
Invités du d...	Groupe de séc...	Tous les invités du doma...
jallo	Utilisateur	
lklei	Utilisateur	
mauap	Utilisateur	
mbono	Utilisateur	
mhida	Utilisateur	
Ordinateurs ...	Groupe de séc...	Toutes les stations de tra...
pmorr	Utilisateur	
Propriétaires...	Groupe de séc...	Les membres de ce grou...
Protected Us...	Groupe de séc...	Les membres de ce grou...
Serveurs RA...	Groupe de séc...	Les serveurs de ce group...
tpere	Utilisateur	

The bottom pane shows a PowerShell script named `ajoutAD.ps1`:

```

1 Set-Location "C:\Users\Administrateur\Documents"
2 foreach ($ligne in Get-Content "utils.txt")
3 {
4     $util=$ligne.Split(":")
5     $name=$util[2]
6     $givenName=$util[1]
7     $surName=$util[0]
8     Write-Host "Création de " $name " / "$givenName " / " $surName
9     New-ADuser -name $name -GivenName $givenName -SurName $surName
10 }
  
```

The output of the script is shown in the console:

```

PS C:\Users\Administrateur\Documents> C:\Users\Administrateur\Documents\ajoutAD.ps1
Création de mauap / Mickael / AU APPAVOU
Création de mbono / MAélanie / BONON
Création de tpere / Thomas / LEROY
Création de pmorr / Prisca / PEREIRA
Création de jallo / Josepha / MORRIS
Création de mhida / Mohamed / ALLOUCHE
Création de giban / Gregory / HIDA
Création de lklei / Laurène / IBANEZ

PS C:\Users\Administrateur\Documents>
  
```

### 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	Permet d'ajouter un util
Set-ADUser	Permet de paramétrer un util
New-ADUser	Permet de créer un util
Remove-ADUser	Supprimer un util
Get-ADUserResultantPasswordPolicy	Permet de connaître la police pour les mdp

### 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 \*

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
```

### 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
```

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

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....

### 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
```

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

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

➤ 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
```

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
```

### 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 :

```
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 :

```
# Définir le chemin du fichier utils.txt
$fichier = "C:\Chemin\vers\le\répertoire\utils.txt"

# Vérifier que le fichier existe
if (-Not (Test-Path $fichier)) {
    Write-Host "Le fichier $fichier n'existe pas."
    exit
}

# Charger chaque ligne du fichier texte
$utilisateurs = Get-Content -Path $fichier

# Parcourir chaque ligne du fichier pour ajouter les utilisateurs
foreach ($ligne in $utilisateurs) {
    # Utiliser la méthode Split() pour séparer les informations par la virgule
    $infos = $ligne.Split(',')

    # Assigner les informations séparées à des variables
    $nom = $infos[0]
    $prenom = $infos[1]
```

```

$login = $infos[2]

# Créer un objet utilisateur avec les informations obtenues
$userPrincipalName = "$login@domaine.com" # Remplacez 'domaine.com' par votre domaine AD
$motDePasse = ConvertTo-SecureString "MotDePasseInitial123!" -AsPlainText -Force # Mot de passe initial
(à personnaliser)

# Création de l'utilisateur dans l'Active Directory
try {
    New-ADUser -SamAccountName $login `
        -UserPrincipalName $userPrincipalName `
        -Name "$prenom $nom" `
        -GivenName $prenom `
        -Surname $nom `
        -DisplayName "$prenom $nom" `
        -Path "OU=Utilisateurs,DC=domaine,DC=com" `
        -AccountPassword $motDePasse `
        -Enabled $false ` # Créer le compte désactivé par défaut
        -PassThru

    Write-Host "Utilisateur $prenom $nom ajouté avec succès."

    # Activer le compte après sa création
    Enable-ADAccount -Identity $login
    Write-Host "Le compte de $prenom $nom a été activé."

}
catch {
    Write-Host "Erreur lors de l'ajout de l'utilisateur $prenom $nom : $_"
}
}

```

### 3.4.2. Script utilSuppression.ps1

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

➤ Notez le contenu de votre script :

```

# Définir le chemin du fichier utils.txt
$fichier = "C:\Chemin\vers\le\répertoire\utils.txt"

# Vérifier que le fichier existe
if (-Not (Test-Path $fichier)) {
    Write-Host "Le fichier $fichier n'existe pas."
    exit
}

# Charger chaque ligne du fichier texte
$utilisateurs = Get-Content -Path $fichier

# Parcourir chaque ligne du fichier pour supprimer les utilisateurs
foreach ($ligne in $utilisateurs) {
    # Utiliser la méthode Split() pour séparer les informations par la virgule
    $infos = $ligne.Split(',')

    # Assigner les informations séparées à des variables
    $nom = $infos[0]

```

```
$prenom = $infos[1]
$login = $infos[2]

# Suppression de l'utilisateur de l'Active Directory
try {
    # Recherche et suppression de l'utilisateur dans AD
    $utilisateurAD = Get-ADUser -Filter {SamAccountName -eq $login}

    # Si l'utilisateur existe, on le supprime
    if ($utilisateurAD) {
        Remove-ADUser -Identity $utilisateurAD -Confirm:$false
        Write-Host "L'utilisateur $prenom $nom ($login) a été supprimé."
    } else {
        Write-Host "Utilisateur $prenom $nom ($login) non trouvé dans Active Directory."
    }
}
catch {
    Write-Host "Erreur lors de la suppression de l'utilisateur $prenom $nom : $_"
}
}
```