ACTIVE DIRECTORY

L'ADMINISTRER AVEC POWERSHELL

1ERE ÉDITION

IT-CONNECT
FLORIAN BURNEL

ISBN numérique : 979-10-262-8465-9

Librinova

Courriel : contact@librinova.com

Internet : www.librinova.com

Table des matières

A propos de cet eBook

Ce cours au format livre numérique a pour objectif de vous apprendre à administrer votre annuaire Active Directory à l'aide de PowerShell, ce dernier étant devenu incontournable pour l'administration système dans un environnement Microsoft.

Grâce aux différents sujets abordés dans ce cours, que ce soit la gestion des utilisateurs, des ordinateurs ou des groupes, vous serez en mesure d'automatiser certaines tâches du quotidien à l'aide de scripts PowerShell. En complément, vous serez en mesure d'analyser et de maintenir votre annuaire Active Directory en lui-même grâce à différentes commandes qui servent à récolter des informations.

Ce cours s'adresse aux administrateurs système, aux étudiants en informatique, et plus généralement aux personnes qui souhaitent prendre en main l'Active Directory avec PowerShell.

Des scripts associés à ce livre numérique sont disponibles à l'adresse suivante :

https://github.com/florianburnel/PowerShell/tree/master/EBOOK-ActiveDirectory-PowerShell

A propos de l'auteur : Florian Burnel

Véritable passionné par l'informatique et les nouvelles technologies, j'ai près de 10 ans d'expérience dans l'administration système et réseau. Généraliste, je maîtrise particulièrement les technologies Microsoft, y compris l'Active Directory et PowerShell. Le langage PowerShell m'accompagne au quotidien depuis des années, et je ne manque jamais l'opportunité de dégainer une console pour exécuter une commande ou créer un script pour automatiser la moindre tâche !

J'ai écrit et publié de nombreux articles au sujet de l'Active Directory et de PowerShell. Ces articles sont disponibles, comme tous les autres, sur le site IT-Connect, en libre accès.

J'en profite pour remercier Thibault Baheux pour sa relecture technique.

Bonne lecture !

Module 1 - Introduction et prérequis

1. Prérequis Active Directory et PowerShell

Avant de commencer à suivre ce cours, il est préférable de prendre connaissance des prérequis ci-dessous, à la fois pour l'Active Directory et pour PowerShell.

- **Active Directory**

Dans ce cours, nous allons manipuler les objets de l'Active Directory les plus courants, à savoir les utilisateurs, les ordinateurs et les groupes. Par ailleurs, nous allons évoquer à plusieurs reprises les attributs constituant ces objets, notamment le sAMAccountName correspondant à l'identifiant de connexion (login).

Au niveau de l'annuaire Active Directory en lui-même, nous allons récupérer des informations sur le niveau fonctionnel, les rôles FSMO, la réplication, etc.

Je vous invite à lire mon cours sur les bases théoriques de l'Active Directory avant de commencer ce cours si les notions évoquées ci-dessus sont abstraites pour vous.

- **PowerShell**

PowerShell nous accompagnera dans chacun des chapitres de ce cours, nous allons utiliser notamment les notions suivantes :

- Les boucles, notamment ForEach
- Le pipeline
- Manipuler un fichier CSV
- Chaîne de caractères sécurisée

Pour suivre ce cours, vous devez disposer de Windows PowerShell 5.1, une version native sur les systèmes d'exploitation Microsoft les plus récents.

Nous découvrirons ensemble les commandes du module Active Directory pour PowerShell, mais si vous avez des bases, la compréhension sera plus facile.

2. Présentation du Lab'

Avant de commencer le cours, je vais décrire le Lab utilisé afin que vous puissiez en avoir connaissance d'une part, mais aussi que vous puissiez mettre en place quelque chose d'équivalent.

Le Lab à mettre en place est relativement simple puisqu'il repose sur une seule machine virtuelle (4 Go de RAM). En effet, il est nécessaire de disposer d'un serveur virtuel qui sera contrôleur de domaine suite au déploiement du rôle Active Directory Domain Services (ADDS).

Voici quelques informations au sujet de ce serveur virtuel :

Caractéristiques	Valeurs
Nom du système	SRV-ADDS-01
Système d'exploitation	Windows Server 2016
Nom de domaine Active Directory	it-connect.local
Nom de domaine NETBIOS	IT-CONNECT

Ce cours s'applique aussi bien à Windows Server 2012, Windows Server 2012 R2, Windows Server 2016 qu'à Windows Server 2019.

Si vous le souhaitez, vous pouvez opérer depuis un poste sous Windows 10, intégré à ce même domaine et sur lequel vous serez connecté avec une session ayant des droits « Admin du domaine ».

Prêt ? Alors allons-y !

3. Créer un domaine Active Directory via PowerShell

Pour mettre en place les services de domaine Active Directory, il est bien entendu possible de le faire par l'interface graphique, mais aussi en ligne de commande par PowerShell. Ceci sera utile notamment sur un serveur en mode core, mais également pour **automatiser la création d'un domaine**, que ce soit pour votre lab perso ou pour une mise en œuvre.

Pour rappel, un serveur en mode core est une machine installée sous Windows Server sans interface graphique. En local, la gestion d'une machine installée en mode core s'effectue uniquement à partir de la console.

Des outils d'automatisation, comme PowerShell DSC pourrons vous permettre de déployer ADDS ou un contrôleur de domaine rapidement en s'appuyant sur PowerShell. Pour cet exemple, on utilisera simplement des commandes PowerShell qu'il est tout à fait possible de mettre dans un script.

Pour préparer notre lab' dans le cadre de ce cours, voyons ensemble **comment déployer Active Directory avec PowerShell**.

Alternative - créer un domaine Active Directory via l'interface graphique :

https://www.it-connect.fr/creer-un-domaine-ad-avec-windows-server-2016/

I. Nom d'hôte

Commençons par renommer le serveur, dans cet exemple « SRV-ADDS-01 » puis redémarrons le serveur. Ce qui nous donne ces deux commandes :

```
Rename-Computer -NewName SRV-ADDS-01 -Force
Restart-Computer
```

Lorsque le serveur aura redémarré, il aura son joli petit nom !

II. Configuration réseau

Maintenant, on va définir la configuration réseau du serveur, toujours via PowerShell. Dans cet exemple, j'attribue l'adresse IP 192.168.1.10 en IP, avec un masque 255.255.255.0 (/24) et une passerelle par défaut 192.168.1.1.

```
New-NetIPAddress -IPAddress "192.168.1.10" -PrefixLength "24" -InterfaceIndex (Get-
NetAdapter).ifIndex -DefaultGateway "192.168.1.1"
```

Pour définir le serveur DNS, il faut utiliser un autre cmdlet, ce qui nous donne :

```
Set-DnsClientServerAddress -InterfaceIndex (Get-NetAdapter).ifIndex -
ServerAddresses ("127.0.0.1")
```

Pour terminer la configuration réseau, on va renommer notre carte réseau. Le nom actuel est "Ethernet0" que l'on va renommer en "LAN".

```
Rename-NetAdapter -Name Ethernet0 -NewName LAN
```

Si tout va bien, votre serveur a un nom définitif et une configuration réseau adéquate. Nous pouvons passer à la phase la plus intéressante : la mise en place du domaine Active Directory.

III. Installation des rôles

Pour cette installation, nous aurons besoin de trois fonctionnalités : les services de domaine Active Directory (AD-Domain-Services), le DNS (DNS) et les outils d'administration graphique (RSAT-AD-Tools), mais ceci est facultatif.

```
$FeatureList = @("RSAT-AD-Tools","AD-Domain-Services","DNS")

Foreach($Feature in $FeatureList){

   if(((Get-WindowsFeature -Name $Feature).InstallState) -eq "Available"){

     Write-Output "Feature $Feature will be installed now !"

     Try{

        Add-WindowsFeature-Name $Feature -IncludeManagementTools -
IncludeAllSubFeature

        Write-Output "$Feature : Installation is a success !"

     }Catch{

        Write-Output "$Feature : Error during installation !"
     }
   } # if(((Get-WindowsFeature -Name $Feature).InstallState) -eq "Available")
} # Foreach($Feature in $FeatureList)
```

Vous pourrez réutiliser le bloc de code ci-dessus, il va installer les trois fonctionnalités une à une, et vous indiquer le résultat à chaque fois. Si vous ne rencontrez pas d'erreur, vous pouvez passer à la suite.

IV. Créer le domaine Active Directory

Ce qui va suivre correspond à la création du domaine, ce qui est l'équivalent du - célèbre - processus *dcpromo*. En fait, nous allons définir notre nom de domaine DNS dans la variable *$DomainNameDNS* et le nom Netbios de ce domaine dans *$DomainNameNetbios*.

Pour le reste, on va stocker tous les paramètres de configuration dans la variable *$ForestConfiguration*. Les paramètres par défaut peuvent être conservés, sauf si vous avez des besoins spécifiques. Le répertoire SYSVOL sera à son chemin habituel, tout comme la base de données ntfs.dit.

Lorsque vous êtes prêt, vous pouvez lancer la création du domaine avec la commande *Install-ADDSForest* qui créera un nouveau domaine dans une nouvelle forêt.

```
$DomainNameDNS = "it-connect.local"
$DomainNameNetbios = "IT-CONNECT"

$ForestConfiguration = @{
'-DatabasePath'= 'C:\Windows\NTDS';
'-DomainMode' = 'Default';
'-DomainName' = $DomainNameDNS;
'-DomainNetbiosName' = $DomainNameNetbios;
'-ForestMode' = 'Default';
'-InstallDns' = $true;
'-LogPath' = 'C:\Windows\NTDS';
'-NoRebootOnCompletion' = $false;
'-SysvolPath' = 'C:\Windows\SYSVOL';
'-Force' = $true;
'-CreateDnsDelegation' = $false }

Import-Module ADDSDeployment
Install-ADDSForest @ForestConfiguration
```

Si tout va bien, je vous invite à redémarrer votre serveur pour finaliser le processus d'installation. Une fois redémarré, votre domaine Active Directory doit être opérationnel. Différents cmdlets comme Get-ADForest et Get-ADDomain vous permettront d'obtenir des infos sur votre domaine et vérifier qu'il correspond à vos souhaits !

Finalement, c'est plutôt simple et pratique de pouvoir déployer un domaine directement en ligne de commande ! Maintenant que notre serveur contrôleur de domaine est prêt, passons à la préparation du poste client avant de rentrer dans le vif du sujet.

Référez-vous au script « EBOOK-ADDS-Module-1-Chapitre-3-01.ps1 » sur GitHub

4. Installer le module Active Directory sur Windows 10 et Windows Server

Dans ce premier chapitre du cours « Administrer l'Active Directory avec PowerShell » réellement dédié à la pratique, nous allons voir comment installer le module PowerShell « Active Directory » sous Windows 10 et Windows Server 2019 (et antérieur). L'intérêt est de pouvoir l'utiliser depuis un poste client qui serait utilisé par un administrateur de l'annuaire au sein de votre entreprise, ou depuis un serveur dédié à l'administration de l'infrastructure.

Ce module est installable sur n'importe quel serveur et n'importe quel poste de travail Windows.

Pour la petite histoire, le module Active Directory de Windows PowerShell est apparu pour la première fois avec Windows Server 2008 R2. Lorsque l'on déploie le rôle de « contrôleur de domaine » sur un serveur, ce module est automatiquement installé sur l'hôte local.

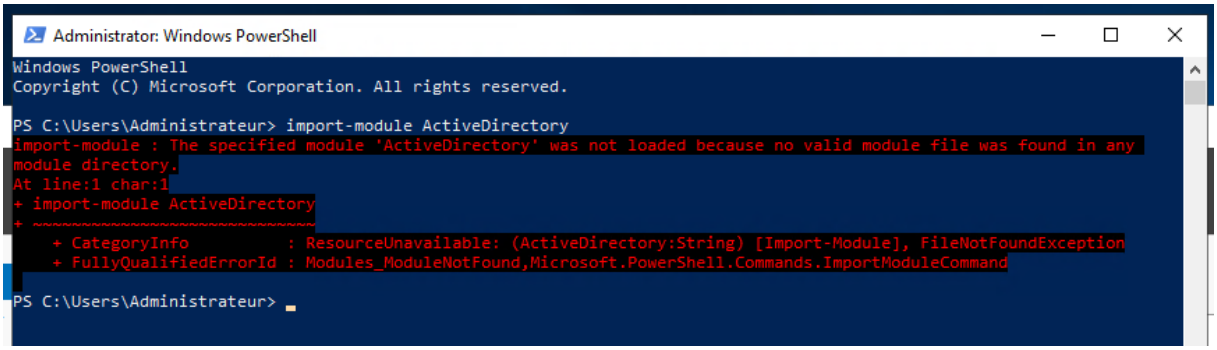
Pour fonctionner, ce module interagit avec un web service intégré au rôle ADDS (Active Directory Domain Services) intégré également avec Windows Server 2008 R2. Si vous avez seulement des contrôleurs de domaine en Windows Server 2003 ou Windows Server 2008 sur votre infrastructure, vous devez installer le service « Active Directory Management Gateway Service » (ADMGS).

V. Installer le module Active Directory sur Windows Server

À partir de Windows Server 2012 R2, le module Active Directory de PowerShell est installable depuis le Gestionnaire de serveur en suivant l'assistant d'ajout d'un rôle/d'une fonctionnalité.

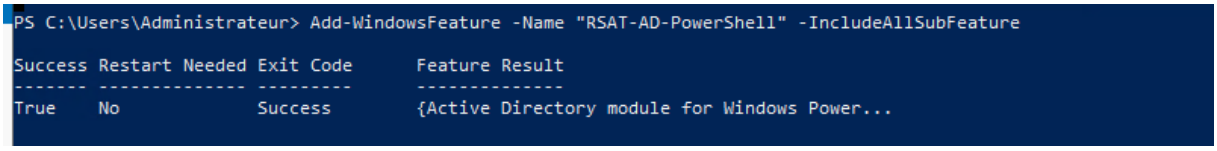
Néanmoins, puisque l'on est là pour faire du PowerShell, nous allons réaliser l'installation directement via la console. Sur un serveur où le module n'est pas installé, si j'essaie de l'importer, j'obtiens bien une erreur :

```
Import-Module ActiveDirectory
```



Pour installer le module sous Windows Server, il suffit d'utiliser la commande « Add-WindowsFeature » et de spécifier la fonctionnalité correspondante :

```
Add-WindowsFeature -Name "RSAT-AD-PowerShell" –IncludeAllSubFeature
```



Il ne reste plus qu'à importer le module et à l'utiliser ! Passons maintenant au cas Windows 10.

VI. Installer le module Active Directory sur Windows 10

Sur Windows 10, l'installation dépend de la version de votre système. Bien que la tendance soit plutôt d'être sur les dernières versions de cet OS si l'on suit les upgrades au fur et à mesure, je vais vous indiquer les deux méthodes.

- **Pour Windows 10 jusqu'à la version 1803**

Note : cette méthode s'applique aussi à Windows 7 et Windows 8.1

Les outils d'administration à distance « RSAT » doivent être téléchargés et installés sur le poste avant de pouvoir ajouter le module.

Pour Windows 10, les outils RSAT sont téléchargeables à cette adresse :
https://www.microsoft.com/en-us/download/details.aspx?id=45520

Une fois les outils RSAT téléchargés et installés sur l'hôte, il devient possible d'activer la fonctionnalité « **Module Active Directory pour PowerShell** » avec l'assistant qui permet d'activer et désactiver des fonctionnalités Windows.

Sinon, et c'est ce que nous allons faire, après avoir installé les outils RSAT, l'installation du module PowerShell est réalisable via PowerShell avec cette commande :

```
Enable-WindowsOptionalFeature -Online -FeatureName RSATClient-Roles-AD-Powershell
```

- **Pour Windows 10 à partir de la version 1809**

Depuis cette version, les outils RSAT sont devenus une fonctionnalité à la demande de Windows 10, ce qui change la donne quant à l'installation du module. L'installation s'effectue via la section « **Fonctionnalités facultatives** » des paramètres de Windows, ou alors en PowerShell : bonne nouvelle !

La commande est légèrement différente, car cette fois-ci c'est le cmdlet « **Add-WindowsCapability** » que nous allons utiliser. Voici la commande à exécuter en tant qu'administrateur sur votre PC :

```
Add-WindowsCapability –Online –Name "Rsat.ActiveDirectory.DS-LDS.Tools~~~~0.0.1.0"
```

```
PS C:\windows\system32> Add-WindowsCapability -online -Name "Rsat.ActiveDirectory.DS-LDS.Tools~~~~0.0.1.0"

Path          :
Online        : True
RestartNeeded : False
```

Suite à l'installation du module PowerShell, vous pouvez l'importer. Bien entendu, la commande reste la même que sous Windows Server :

```
Import-Module ActiveDirectory
```

Enfin, vous pouvez lister les commandes du module ActiveDirectory de PowerShell grâce à cette commande :

```
Get-Command -Module ActiveDirectory
```

Maintenant que l'installation est réalisée, je peux vous certifier que nous allons l'exploiter ce module ! Il est temps de rentrer dans le vif du sujet…

Module 2 - Gérer les utilisateurs AD avec PowerShell

1. Récupérer des informations sur les utilisateurs avec PowerShell

Suite à l'installation du module PowerShell « Active Directory » que l'on a vue dans le chapitre précédent, nous allons maintenant rentrer dans le vif du sujet et manipuler l'annuaire Active Directory.

Pour commencer, nous allons jouer avec les utilisateurs et dans ce chapitre nous allons voir comment récupérer des informations sur les utilisateurs.

Le **cmdlet Get-ADUser est disponible depuis PowerShell 2.0 dans le module Active Directory** et il sera notre principal allié pour récupérer des informations sur les utilisateurs de l'annuaire. Autrement dit, ce cmdlet est disponible depuis Windows Server 2008 R2. Il offre une excellente alternative aux autres outils que l'on pouvait utiliser jusqu'ici, notamment des scripts VBS ou *dsquery*.

Nous pouvons récupérer différentes informations grâce à la lecture des attributs :

- L'identifiant d'un utilisateur, son nom, son prénom
- L'adresse e-mail associée au compte utilisateur
- La date de dernière connexion au domaine Active Directory
- La date de dernière modification d'un compte utilisateur
- La date de création de l'objet
- Etc.

Tout dépend aussi des données renseignées dans les comptes de vos utilisateurs.

Avant de commencer, si besoin vous pourrez retrouver l'aide de Get-ADUser, mais avant cela pensez à mettre à jour l'aide de PowerShell :

```
Update-Help
```

Ensuite, vous pouvez consulter l'aide dans la console PowerShell avec :

```
Get-Help Get-ADUser
```

Pour consulter l'aide directement en ligne, exécutez la commande suivante :

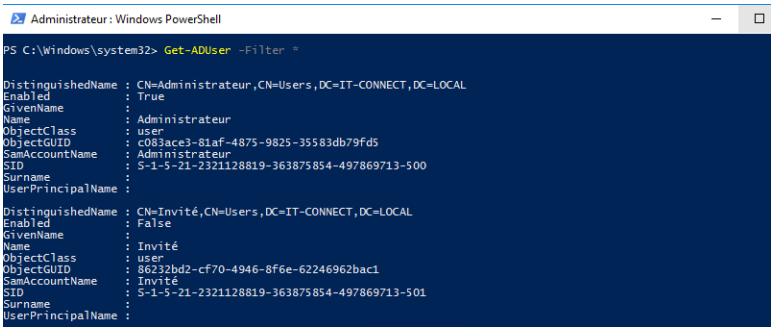
```
Get-Help Get-ADUser -Online
```

I. Syntaxe de Get-ADUser

Il est à noter que la commande **Get-ADUser** utilise un paramètre *-Filter* pour créer une requête et rechercher nos utilisateurs. Nous verrons qu'il y a deux façons de filtrer. Pour ne pas filtrer et donc récupérer la liste de tous les utilisateurs du domaine, il suffit d'exécuter cette commande :

```
Get-ADUser -Filter *
```

Attention : cette commande peut s'avérer très gourmande et durer un certain temps si votre annuaire Active Directory est peuplé de nombreux comptes. Méfiance donc avant de l'exécuter !



Pour la suite, si vous voulez tester au fur et à mesure, voici la syntaxe pour filtrer sur un seul utilisateur en se basant sur son login (attribut *saMAccountName*). Exemple avec le compte « Administrateur » :

```
Get-ADUser -Filter "SamAccountName -eq 'Administrateur'"
```

Une alternative serait d'utiliser le paramètre -Identity pour réaliser le filtre, en lui spécifiant le login de l'utilisateur :

```
Get-ADUser -Identity Administrateur
```

Enfin, il y a une autre alternative :

```
Get-ADUser Administrateur
```

Pour ma part, je préfère **préciser un contrôleur de domaine spécifique lorsque je requête l'annuaire Active Directory.** Au moins, je sais où part ma requête. Pour spécifier un serveur, nous utiliserons cette syntaxe (nom du serveur à adapter) :

```
Get-ADUser -Filter "SamAccountName -eq 'Administrateur'" -Server SRV-ADDS-01.it-connect.local
```

Ce paramètre existe sur l'ensemble des commandes liées à l'AD donc je vous recommande de prendre l'habitude de l'utiliser. En déclarant une variable $DC au début de votre script par exemple, et il suffit ensuite de l'appeler dans chaque commande ce qui facilite le changement de DC si nécessaire.

Comme nous avons pu le voir sur la première copie d'écran, **par défaut il y a 10 propriétés basiques retournées dans la console, mais tout n'est pas là : il y en a plus de 100 pour un objet utilisateur.**

Vous allez me dire, ce filtre par défaut est plutôt intéressant pour éviter de charger des données inutilement… Néanmoins, nous pouvons ajouter un ou plusieurs attributs supplémentaires à afficher.

Le paramètre -Properties doit être ajouté à la commande. Si l'on indique « * » on peut charger toutes les propriétés, alors si en plus on utilise « * » pour le filtre, je vous laisse imaginer la lourdeur de la requête.

Pour spécifier un attribut supplémentaire à charger, il suffit d'indiquer son nom. Voici un exemple pour afficher l'attribut « mail » :

```
Get-ADUser -Filter "SamAccountName -eq 'Administrateur'" -Server SRV-ADDS-01.it-connect.local -Properties mail
```

S'il y a plusieurs attributs à ajouter, il suffit de séparer les noms par une virgule. Par exemple :

```
Get-ADUser -Filter "SamAccountName -eq 'Administrateur'" -Server SRV-ADDS-01.it-connect.local -Properties mail,lastLogon
```

```
PS C:\Windows\system32> Get-ADUser -Filter "SamAccountName -eq 'Administrateur'" -Server SRV-ADDS-01.it-connect.lo
roperties mail,lastLogon


DistinguishedName : CN=Administrateur,CN=Users,DC=IT-CONNECT,DC=LOCAL
Enabled           : True
GivenName         :
lastLogon         : 132332685662976233
mail              : administrateur@it-connect.local
Name              : Administrateur
ObjectClass       : user
ObjectGUID        : c083ace3-81af-4875-9825-35583db79fd5
SamAccountName    : Administrateur
SID               : S-1-5-21-2321128819-363875854-497869713-500
Surname           :
UserPrincipalName :
```

Après avoir vu ces premiers paramètres, indispensables à l'utilisation de la commande Get-ADUser, je vous propose de voir d'autres exemples.

II. Rechercher des utilisateurs dans une OU avec SearchBase

Pour réaliser une action sur un lot d'utilisateurs, il peut s'avérer très utile d'utiliser un filtre basé sur une unité d'organisation (OU).

Le cmdlet Get-ADUser intègre un paramètre nommé *-SearchBase* qui va permettre de répondre à ce besoin. Ce qui donne :

```
Get-ADUser -Filter * -SearchBase "CN=Users,DC=IT-CONNECT,DC=LOCAL" -Server SRV-ADDS-01.it-connect.local
```

Pour l'unité d'organisation, vous devez préciser son « chemin » correspondant à son DN (*DistinguishedName*). Vous pouvez le retrouver dans les propriétés de l'OU via l'onglet « Editeur d'attributs » ou en PowerShell grâce à la commande :

```
Get-ADOrganizationalUnit -Filter * | Format-Table DistinguishedName
```

```
PS C:\Windows\system32> Get-ADOrganizationalUnit -Filter * | Ft DistinguishedName

DistinguishedName
-----------------
OU=Domain Controllers,DC=IT-CONNECT,DC=LOCAL
OU=IT - Domain Servers,DC=IT-CONNECT,DC=LOCAL
OU=IT - Computers,DC=IT-CONNECT,DC=LOCAL
OU=Groupes,DC=IT-CONNECT,DC=LOCAL
OU=Personnel,DC=IT-CONNECT,DC=LOCAL
OU=Archivage,DC=IT-CONNECT,DC=LOCAL
OU=PC,DC=IT-CONNECT,DC=LOCAL
```

Il suffit de copier la valeur correspondante à votre unité d'organisation cible et de l'utilisation dans la commande Get-ADUser.

Pour récupérer la liste des utilisateurs à partir de plusieurs OU, on peut constituer une liste d'unités d'organisation :

```
$OUList = @("OU=Comptabilite,DC=it-connect,DC=local",
            "OU=Secretariat,DC=it-connect,DC=local",
            "OU=Direction,DC=it-connect,DC=local",
            "OU=Commercial,DC=it-connect,DC=local")
```

Ensuite, à l'aide d'une boucle Foreach, nous allons récupérer les utilisateurs de chaque OU et stocker le résultat dans la variable $OUList (en cumulant pour avoir une liste complète). Voici la boucle Foreach basée sur Get-ADUser qui permet de réaliser cette action :

```
$Users = Foreach($OU in $OUList){
    Get-ADUser -Filter * -SearchBase $OU | Select-Object SamAccountName
}

$Users | Export-CSV -Path "C:\Users.csv" -NoTypeInformation
```

Nous finirons par un export des résultats dans un fichier CSV :

```
$Users | Export-CSV -Path "C:\Users.csv" -NoTypeInformation
```

Le bout de code complet :

```
$OUList = @("OU=Comptabilite,DC=it-connect,DC=local",
            "OU=Secretariat,DC=it-connect,DC=local",
            "OU=Direction,DC=it-connect,DC=local",
            "OU=Commercial,DC=it-connect,DC=local")

$Users = Foreach($OU in $OUList){
    Get-ADUser -Filter * -SearchBase $OU | Select-Object SamAccountName
}

$Users | Export-CSV -Path "C:\Users.csv" -NoTypeInformation
```

III. Obtenir la liste des adresses e-mails des utilisateurs

Si l'attribut « mail » est renseigné dans vos comptes utilisateurs alors vous pouvez récupérer facilement une liste des adresses e-mails. Nous allons utiliser la propriété "EmailAddress" ou "mail" : les deux fonctionnent pour récupérer l'adresse e-mail.

Enfin, pour que ce soit plus lisible, nous allons sortir le résultat dans un tableau avec seulement deux colonnes via la commande Format-Table (que l'on peut raccourcir en *ft*).

```
Get-ADUser -Filter * -SearchBase "CN=Users,DC=IT-CONNECT,DC=LOCAL" -Properties
EmailAddress | Format-Table Name,EmailAddress
```

```
PS C:\Windows\system32> Get-ADUser -Filter * -SearchBase "CN=Users,DC=IT-CONNECT,DC=LOCAL" -Properties EmailAddr
mat-Table Name,EmailAddress

Name             EmailAddress
----             ------------
Administrateur   administrateur@it-connect.local
Invité
DefaultAccount
krbtgt
admin
fb.itconnect     fb@it-connect.local
Adm_UserLock
IT-Connect Admin admin@it-connect.local
```

Pour que ce soit plus pertinent, nous pouvons **ajouter un filtre pour récupérer seulement les utilisateurs actifs et où l'attribut EmailAddress n'est pas vide ($null).** Ce qui donne :

```
Get-ADUser -Filter {(EmailAddress -ne $null) -and (Enabled -eq $true)} -SearchBase
"CN=Users,DC=IT-CONNECT,DC=LOCAL" -Properties EmailAddress | Format-Table
Name,EmailAddress
```

IV. Afficher la date de création des comptes

Pour récupérer la date de création d'un ou plusieurs comptes Active Directory, il faut lire la valeur de l'attribut « *whenCreated* », car il contient la date de création de l'objet dans l'Active Directory.

Ainsi, il suffit de l'appeler via le paramètre *-Properties* pour l'afficher. J'aime bien utiliser Select-Object en complément pour affiner le résultat qui s'affiche dans la console.

Voici la commande, avec le filtre à adapter à vos besoins comme à chaque fois :

```
Get-ADUser -Filter * -Properties WhenCreated | Select-Object Name, whenCreated
```

```
Administrateur : Windows PowerShell
PS C:\Windows\system32>
PS C:\Windows\system32> Get-ADUser -Filter * -Properties WhenCreated | Select-Object Name, whenCreated

Name             whenCreated
----             -----------
Administrateur   11/12/2019 18:17:04
Invité           11/12/2019 18:17:04
DefaultAccount   11/12/2019 18:17:04
krbtgt           11/12/2019 18:17:45
admin            12/12/2019 13:52:03
Une Comptable    17/12/2019 20:01:35
fb.itconnect     07/01/2020 22:37:39
comptable 01.    23/01/2020 21:04:24
Adm_UserLock     20/04/2020 18:12:21
IT-Connect Admin 03/05/2020 14:17:03
```

V. Récupérer la liste des utilisateurs créés à une date précise

Toujours en lien avec la notion de date, je vous propose de répondre à la question comment récupérer la liste des utilisateurs créés à une date précise ? Il peut y avoir tout un tas de raison de vouloir faire cette requête, surement au sein d'un script beaucoup plus global, mais cela dépendra de vos besoins. L'intérêt est de vous proposer un bout de code prêt à l'emploi.

Personnellement, je l'ai utilisé pour regarder combien il y avait eu de comptes créés dans l'Active Directory à une date précise, une date qui peut-être le jour même.Pour le script, nous allons appliquer le principe suivant :

1 - Récupérer la date du jour

2 - Stocker dans une variable la date et heure de début de journée, c'est-à-dire à minuit 00:00

3 - Stocker dans une variable la date et heure de fin de journée, c'est-à-dire à 23:59

4 - Rechercher dans l'Active Directory les comptes qui ont une date de création "plus grande" que la date de début de journée et "plus petite" que la date de fin de journée

Si l'on veut récupérer les comptes créés à une date précise, autre que la date du jour, il faudra agir sur la variable $Date avec la méthode AddDays pour calculer la date souhaitée, ou, directement l'indiquer en chaîne de caractères dans la variable $Date en conservant le type [datetime].

Pour récupérer la date du jour, cela donne :

```
[datetime]$Date = Get-Date -Format "MM/dd/yyyy"
```

Par contre, si l'on veut une date précise, on peut l'indiquer comme ça (exemple : 05 mai 2021) :

```
[datetime]$Date = "05/05/2021"
```

Nous pouvons aussi calculer la date en soustrayant un nombre de jours, par exemple pour avoir la date de la veille :

```
$Date = ($Date).AddDays(-1)
```

Maintenant, nous allons créer deux variables : $DateOn et $DateOff pour avoir la date et l'heure de début de journée et celle de fin de journée. Nous allons utiliser les méthodes AddHours, AddMinutes et AddSeconds pour définir l'heure.

Pour définir l'heure de début à 00:00:00, cela donne :

```
$DateOn = $Date.AddHours(00).AddMinutes(00).AddSeconds(00)
```

Ensuite, pour l'heure de fin à 23:59:59 :

```
$DateOff = $Date.AddHours(23).AddMinutes(59).AddSeconds(59)
```

Maintenant que nous avons notre plage de recherche, il ne reste plus qu'à envoyer la requête sur l'annuaire Active Directory. Comme nous l'avons vu, chaque utilisateur dispose d'un attribut natif nommé "WhenCreated" et qui contient la date de création de l'objet.

Donc, pour lister les utilisateurs nous allons utiliser Get-ADUser et réaliser un filtre sur l'attribut WhenCreated, ce qui donne :

```
Get-ADUser -Filter 'whenCreated -ge $DateOn -and whenCreated -lt $DateOff' -
Properties whenCreated
```

Au lieu de lister les utilisateurs, nous pouvons aussi les compter :

```
(Get-ADUser -Filter 'whenCreated -ge $DateOn -and whenCreated -lt $DateOff' -
Properties whenCreated | Measure-Object).count
```

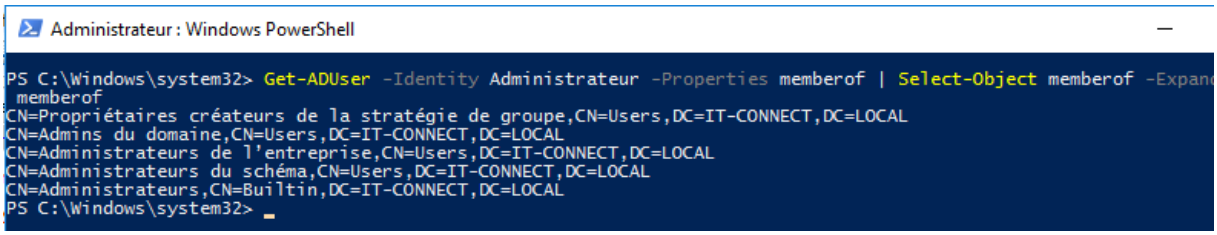
Ce qui au final nous donne le bout de code PowerShell suivant :

```
[datetime]$Date = Get-Date -Format "MM/dd/yyyy"
$Date = ($Date).AddDays(-1)
$DateOn = $Date.AddHours(00).AddMinutes(00).AddSeconds(00)
$DateOff = $Date.AddHours(23).AddMinutes(59).AddSeconds(59)

Get-ADUser -Filter 'whenCreated -ge $DateOn -and whenCreated -lt $DateOff' -
Properties whenCreated
```

VI. Quels sont les groupes dont l'utilisateur est membre ?

Une autre chose qui est utile, c'est de **récupérer la liste des groupes dont un utilisateur est membre**. Là encore, on va pouvoir s'appuyer sur un attribut : *memberof*. Pour afficher la liste des groupes dont un utilisateur spécifique est membre, voici la commande à utiliser :

```
Get-ADUser -Identity Administrateur -Properties memberof | Select-Object memberof -
ExpandProperty memberof
```



Nous pouvons vérifier ensuite si l'utilisateur est membre du groupe « Admins du domaine » (ou un autre groupe) et si c'est le cas, réaliser un ensemble d'actions sur ce compte. Voici un exemple :

```
$MemberOf = Get-ADUser -Identity Administrateur -Properties memberof | Select-
Object SamAccountName,memberof

if($MemberOf.memberof -match "Admins du domaine"){

  Write-Host "attention il s'agit d'un compte Administrateur !"

}
```

VII. Coupler l'utilisation du Filter et du Like

Comme vu précédemment, **le paramètre -Filter sert à appliquer un filtre sur la requête exécutée par la commande Get-ADUser afin d'affiner la recherche sur un ou plusieurs attributs**. Cela est intéressant pour rechercher un compte précis notamment, même si l'on peut préférer l'utilisation du paramètre -Identity.

Si l'on associe **le paramètre -Filter à l'utilisation de l'opérateur Like nous pouvons appliquer un filtre plus large.** Par exemple, récupérer la liste de tous les utilisateurs qui ont un prénom ou un nom spécifique, ou un caractère spécifique dans le nom, etc.

Pour rechercher les utilisateurs qui ont la chaîne de caractères "connect" dans l'attribut Name, on utilisera :

```
Get-ADUser -Filter "Name -Like '*connect*'" -Properties Name,SamAccountName |
Select Name,SamAccountName
```

À noter l'importance des astérisques avant et après le terme à rechercher pour signifier qu'il peut y avoir X caractères avant et X caractères après. Nous n'avons pas utilisé l'opérateur -match, car il n'est pas compatible avec le paramètre -Filter, donc avec l'opérateur -like cela permet de contourner cette contrainte.

```
PS C:\Windows\system32> Get-ADUser -Filter "Name -Like '*connect*'" -Properties * | select Name,SamAccountName

Name            SamAccountName
----            --------------
fb.itconnect    fb.itconnect
IT-Connect Admin itconnect_admin
```

Lorsque l'on définit un filtre, il est tout à fait possible de se baser sur plusieurs conditions, séparées par un opérateur comme -and (et) ou -or (ou) afin d'affiner la requête.

Si l'on veut rechercher la chaîne "itconnect" à la fois dans l'attribut Name et dans l'attribut saMAccountName, cela donnera la commande suivante :

```
Get-ADUser -Filter "(Name -Like '*connect*') -or (SamAccountName -Like
'*connect*')" -Properties * | Select Name,SamAccountName
```

VIII. Récupérer la liste des comptes utilisateurs désactivés

Lorsque vous souhaitez faire un état des lieux de votre Active Directory, vous pouvez vous poser la question suivante : **combien ai-je de comptes utilisateurs désactivés au sein de mon annuaire ?** Là encore, PowerShell et le cmdlet Get-ADUser sont là pour nous permettre de répondre à cette question.

L'attribut « Enabled » est un booléen présent au sein de la fiche de chaque utilisateur, donc si sa valeur est égale à « false » cela signifie que le compte est désactivé : il ne reste plus qu'à filtrer en ce sens.

Ce qui donne :

```
Get-ADUser -Filter { Enabled -eq $False } | Format-Table SamAccountName , Name
```

Dans cet exemple, nous allons retourner un tableau avec deux attributs. Sur l'image ci-dessous, on peut voir que cela fonctionne aussi avec "False" plutôt que $False, mais il vaut mieux préférer l'écriture $False qui est plus correcte (valeur booléenne).

```
PS C:\Windows\system32> Get-ADUser -Filter { Enabled -eq "False" } | Format-Table SamAccountName , Name

SamAccountName   Name
--------------   ----
Invité           Invité
DefaultAccount   DefaultAccount
krbtgt           krbtgt
une-comptable-01 Une Comptable
```

IX. Récupérer la liste des comptes où le mot de passe est expiré

En termes de reporting sur l'état de vos comptes et en complément de la liste des comptes désactivés, il est intéressant **d'obtenir la liste des comptes où le mot de passe est expiré**. Uniquement pour les comptes actifs pour que ce soit plus pertinent.

Nous devons charger la propriété « PasswordExpired » et réaliser un filtre sur cette propriété. J'en profite pour vous indiquer que pour faire un filtre, il y a à chaque fois la possibilité d'inclure la condition dans -Filter ou alors de réaliser un filtre via un Where (Where-Object), voire même de mixer les deux…

Voici la commande associée à la requête que l'on recherche :

```
Get-ADUser -Filter { Enabled -eq "True" } -Properties PasswordExpired | Where
{$_.PasswordExpired -eq $true }
```

Note : le caractère $_ permet d'utiliser une variable automatique qui reprend l'occurrence courante renvoyée par la commande avant le « | », c'est-à-dire Get-ADUser.

Pour finir ce chapitre, nous allons réaliser deux requêtes d'un style différent. Je vous laisse lire la suite sans plus attendre.

X. Compter le nombre d'utilisateurs dans l'Active Directory

Avoir la liste des comptes désactivés, des comptes où le mot de passe est expiré, etc… C'est bien, mais il peut être intéressant de tout simplement savoir combien il y en a. Pour obtenir cette information, nous pouvons faire appel à **la propriété « count » qui va simplement nous retourner le nombre de résultats, sans nous les afficher.**

Imaginons que nous prenons cette requête toute simple pour afficher les comptes actifs dans l'AD :

```
Get-ADUser -Filter { Enabled -eq $True }
```

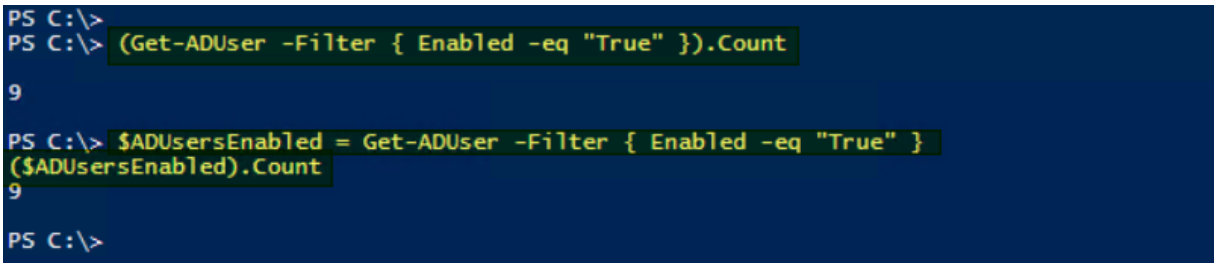
Si l'on souhaite savoir combien il y a de comptes actifs dans l'AD, il suffit d'exécuter cette commande :

```
(Get-ADUser -Filter { Enabled -eq $True }).Count
```

Note : comme je le disais précédemment, lorsque l'on filtre sur un attribut booléen comme *Enabled*, on peut soit indiquer « *false* » ou « *true* » entre guillemets, soit indiquer *$false* ou *$true*.

Si l'on veut récupérer la liste des utilisateurs, et ensuite dans un second temps, compter le nombre d'objets retournés par la commande sans multiplier les requêtes dans l'annuaire, il faut passer par une variable « intermédiaire ». Ce qui donne :

```
$ADUsersEnabled = Get-ADUser -Filter { Enabled -eq $True }
($ADUsersEnabled).Count
```



XI. Exporter les utilisateurs dans un fichier CSV

Pour finir ce chapitre, nous allons voir **comment exporter dans un fichier CSV les résultats d'une requête Get-ADUser.** Pour cela, le résultat de la commande sera envoyé via le pipeline dans une commande Select-Object pour sélectionner les propriétés à inclure au fichier CSV puis grâce à un second pipeline, nous allons réaliser l'export grâce au cmdlet Export-Csv.

Ce cmdlet sera accompagné de trois paramètres : **-Path** pour indiquer le nom du fichier CSV et le dossier dans lequel le créer ; **-Encoding** pour le type d'encodage des données (utile pour les accents) ; **-NoTypeInformation** pour ne pas inclure une ligne sur la provenance des données intégrées : cela est préférable pour avoir une première qui correspond à nos en-têtes de colonnes du fichier CSV. Si nous n'indiquons pas le paramètre -NoTypeInformation la première ligne du CSV serait :

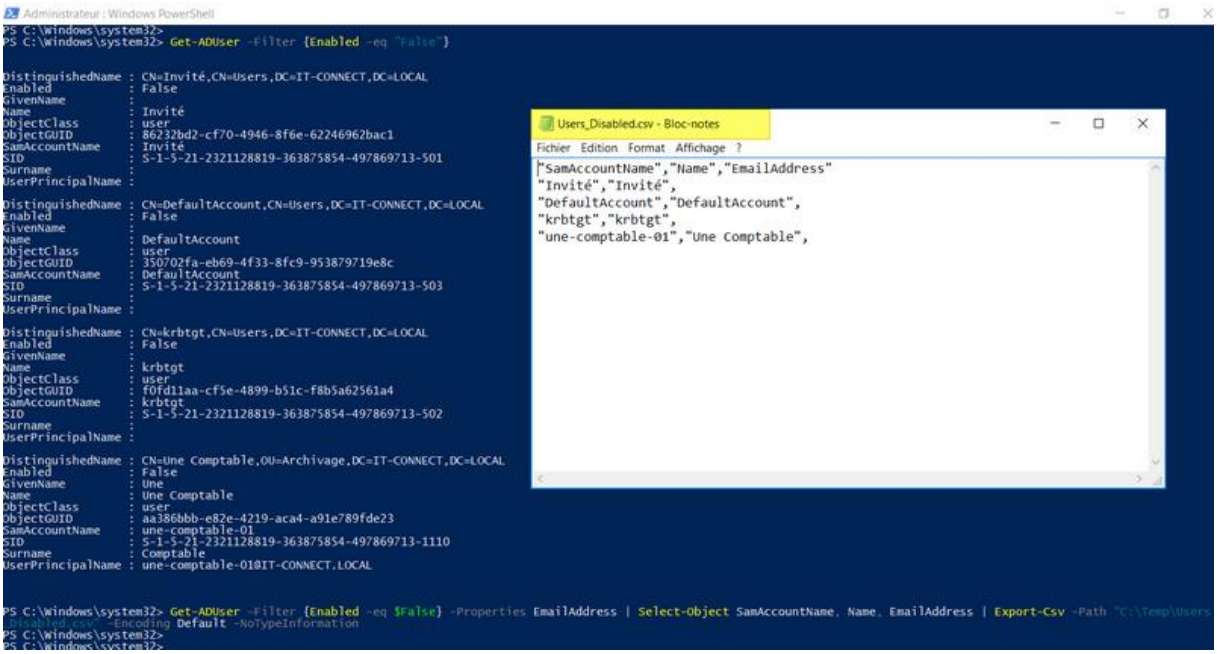
#TYPE Selected.Microsoft.ActiveDirectory.Management.ADUser

Le paramètre -Delimiter peut être ajouté si l'on désire modifier le type de délimiteur : celui par défaut est la virgule.

Voici un exemple pour exporter dans un fichier CSV « *Users_Disabled.csv* » les comptes désactivés en intégrant trois propriétés : SamAccountName, Name, EmailAddress.

```
Get-ADUser -Filter {Enabled -eq $False} -Properties EmailAddress | `
           Select-Object SamAccountName, Name, EmailAddress | `
           Export-Csv -Path "C:\Temp\Users_Disabled.csv" -Encoding Default `
           -NoTypeInformation
```

Suite à l'exécution de cette commande, nous obtenons le fichier suivant :



Enfin, sachez que **les données dans le fichier CSV peuvent être triées sur la base d'un attribut**, pour cela il faut insérer une commande supplémentaire : Sort-Object, suivie de l'attribut à utiliser pour réaliser le tri alphabétique. Voici un exemple :

```
Get-ADUser -Filter {Enabled -eq $False} -Properties EmailAddress | `
           Select-Object SamAccountName, Name, EmailAddress | `
           Sort-Object Name | `
           Export-Csv -Path "C:\Temp\Users_Disabled.csv" -Encoding Default `
           -NoTypeInformation
```

Grâce aux différents exemples d'utilisation de la commande *Get-ADUser*, vous êtes désormais en mesure de réaliser des requêtes dans l'Active Directory pour récupérer des informations sur vos users.

La notion de filtre via le paramètre *-Filter* est très importante et c'est grâce à ce filtrage que l'on va pouvoir récupérer ceux que l'on souhaite. Ensuite, l'utilisation de Select-Object ou Format-Table sert à affiner la sortie et avoir un résultat plus synthétique.

Enfin, pensez à utiliser le paramètre -Server pour préciser le contrôleur de domaine que vous souhaitez interroger, je considère cela comme étant une bonne pratique.

2. Créer un utilisateur dans l'AD avec PowerShell

Lorsque l'on mêle PowerShell et Active Directory, il y a bien une chose que l'on va chercher à faire, c'est créer un nouvel utilisateur dans l'Active Directory en ligne de commande. Surement parce que c'est l'une des tâches basiques que l'on faisait en interface graphique pendant des années, et qu'il est possible d'automatiser aujourd'hui avec PowerShell. C'est d'autant plus vrai lorsqu'il s'agit de créer un lot d'utilisateurs à partir d'un fichier de données, au format CSV par exemple. Nous y reviendrons, pour le moment, nous allons réaliser la création d'un seul utilisateur afin de bien comprendre l'utilisation du cmdlet PowerShell « New-ADUser ».

Mon objectif c'est de vous faire oublier la console « Utilisateurs et ordinateurs Active Directory » pour la création d'un compte utilisateur.

I. Premiers pas avec New-ADUser

La commande New-ADUser dispose de nombreux paramètres afin de personnaliser et configurer le compte à sa création. L'exécution de la commande ci-dessous parle d'elle-même…

```
Get-Command New-ADUser -Syntax
```

```
PS C:\> Get-Command New-ADUser -Syntax

New-ADUser [-Name] <string> [-WhatIf] [-Confirm] [-AccountExpirationDate <datetime>] [-AccountNotDelega
>] [-AuthenticationPolicySilo <ADAuthenticationPolicySilo>] [-AuthType <ADAuthType>] [-CannotChangePass
Supported <bool>] [-Country <string>] [-Credential <pscredential>] [-Department <string>] [-Description
bled <bool>] [-Fax <string>] [-GivenName <string>] [-HomeDirectory <string>] [-HomeDrive <string>] [-Ho
LogonWorkstations <string>] [-Manager <ADUser>] [-MobilePhone <string>] [-Office <string>] [-OfficePhon
asswordNotRequired <bool>] [-Path <string>] [-POBox <string>] [-PostalCode <string>] [-PrincipalsAllowe
vicePrincipalNames <string[]>] [-SmartcardLogonRequired <bool>] [-State <string>] [-StreetAddress <stri
ers>]
```

Aussi surprenant que cela puisse paraître, notamment après avoir vu le résultat de la commande précédente, il est possible de créer très facilement un compte. Par exemple, la commande ci-dessous va créer le compte « f.burnel » :

```
New-ADUser f.burnel
```

Sans aucun paramètre, sans aucune option : le compte est bien créé, il suffit de le vérifier avec cette commande :

```
Get-ADUser -Identity f.burnel
```

```
PS C:\> Get-ADUser -Identity f.burnel


DistinguishedName : CN=f.burnel,CN=Users,DC=IT-CONNECT,DC=LOCAL
Enabled           : False
GivenName         :
Name              : f.burnel
ObjectClass       : user
ObjectGUID        : a4c6530a-c305-487e-8565-f27eb0b47e44
SamAccountName    : f.burnel
SID               : S-1-5-21-2321128819-363875854-497869713-1125
Surname           :
UserPrincipalName :
```

Néanmoins, il ne faut pas s'emballer trop rapidement… Le compte est créé, mais il est désactivé, les champs « Nom », « Prénom », « Adresse de messagerie », ou encore le « Nom complet » sont vides. L'attribut UserPrincipalName de l'utilisateur n'est pas défini également.

De plus, son mot de passe n'est pas défini : il devra le modifier à la première ouverture de session. Au niveau de l'annuaire AD, il est créé dans l'OU par défaut, à savoir « Users ». Pour le rendre utilisable, il faudrait commencer par l'activer et définir un mot de passe.

Tout ça pour dire que, même si l'utilisateur est créé, cela n'est pas vraiment pertinent… Il va falloir faire un effort et ajouter des paramètres à la suite de la commande New-ADUser.

Maintenant, créons un utilisateur avec les données suivantes :

- Nom complet : Maude Zarella
- Nom d'affichage Windows : ZARELLA Maude
- Prénom : Maude
- Nom : Zarella
- Identifiant : maude.zarella
- User Principal Name (UPN) : maude.zarella@it-connect.local
- Adresse e-mail : maude.zarella@it-connect.local
- Unité d'organisation cible : OU=Personnel,DC=IT-CONNECT,DC=LOCAL
- Compte actif
- Mot de passe définit sur « P@ssw0rd » et l'utilisateur devra le changer à la première connexion

Concrètement pour chacune des données ci-dessus, il est possible de mettre en face un attribut dans l'Active Directory, et par extension un paramètre du cmdlet New-ADUser.

La commande magique pour créer un compte à partir de ces données est la suivante :

```
New-ADUser -Name "Maude Zarella" `
           -DisplayName "ZARELLA Maude" `
           -GivenName "Maude" `
           -Surname "Zarella" `
           -SamAccountName "maude.zarella" `
           -UserPrincipalName "maude.zarella@it-connect.local" `
           -EmailAddress "maude.zarella@it-connect.local" `
           -Path "OU=Personnel,DC=IT-CONNECT,DC=LOCAL" `
           -AccountPassword (Read-Host -AsSecureString "Mot de passe ?") `
           -ChangePasswordAtLogon $true `
           -Enabled $true
```

Note : dans la commande ci-dessus, les paramètres ne sont pas mis sur la même ligne pour que la commande soit plus facile à lire ! Pour se permettre de sauter des lignes, il ne faut pas oublier le caractère « ` » à la fin de chaque ligne (sauf la dernière). Néanmoins, il est tout à fait possible de tout écrire sur une seule ligne.

Les paramètres ont un nom relativement parlant, sous réserve d'avoir quelques notions basiques d'anglais. Certains paramètres, comme le prénom (*GivenName*), le nom (*Surname*) ou encore l'adresse e-mail (*EmailAddress*) accepte une chaîne de caractères comme valeur.

Ce n'est pas le cas de tous les paramètres (donc les attributs), notamment le paramètre *-Enabled* qui est un booléen qui sert à activer ou désactiver un compte, ou *-ChangePasswordAtLogon* qui sert à indiquer si l'utilisateur doit changer ou non son mot de passe à la première ouverture de session.

Enfin, le mot de passe (*AccountPassword*) quant à lui doit être une chaîne de caractères sécurisée, ce qui nécessite d'intégrer la valeur d'une certaine manière : les guillemets ne suffiront pas. Dans cet exemple, je récupère la chaîne sécurisée via Read-Host c'est-à-dire qu'une pop-up va s'afficher à l'écran afin d'inviter l'administrateur à saisir le mot de passe à attribuer à cet utilisateur.

Pour écrire notre mot de passe en clair directement dans le script, la syntaxe du paramètre -AccountPassword est la suivante :

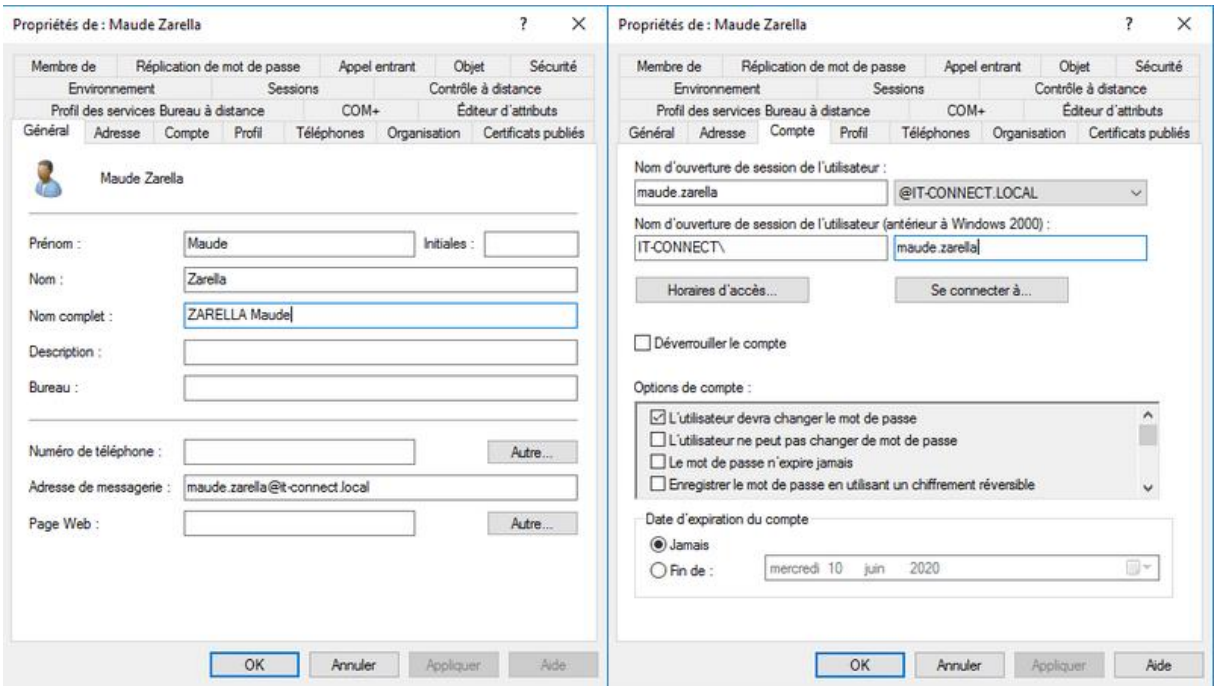
```
-AccountPassword (ConvertTo-SecureString "P@ssw0rd" -AsPlainText -Force)
```

Ce n'est pas top, mais pour automatiser la création d'utilisateurs en lot, on ne peut pas se permettre de saisir chaque mot de passe manuellement. Pour un compte, comme ici, disons que c'est acceptable. Retenez l'essentiel : l'utilisateur va modifier le mot de passe à la première ouverte de session.

Note : le paramètre -Server qui permet d'indiquer un contrôleur de domaine à cibler est bien sûr disponible pour New-ADUser.

Si l'on regarde le compte créé dans la console « Utilisateurs et ordinateurs Active Directory », nous pouvons voir que c'est conforme. Exceptionnellement, je vous autorise à y jeter un coup d'œil. En PowerShell, nous pouvons bien entendu récupérer des informations au sujet de cet utilisateur :

```
Get-ADUser -Identity "maude.zarella"
```



Au passage, j'en profite pour vous féliciter : vous venez de créer votre premier utilisateur personnalisé dans l'Active Directory !

II. Créer un utilisateur AD : découverte d'une autre syntaxe

Nous venons de voir comment utiliser le cmdlet New-ADUser en spécifiant les paramètres les uns après les autres. Il y a une autre façon de faire en s'appuyant sur une hashtable qui va contenir toutes nos valeurs et ensuite nous allons passer ces valeurs directement à la commande New-ADUser.

Si l'on reprend l'exemple de l'utilisateur précédent, cela donne l'écriture suivante :

```
$NewUserParams = @{
    Name = "Maude Zarella"
    DisplayName = "ZARELLA Maude"
    GivenName = "Maude"
    Surname = "Zarella"
    SamAccountName = "maude.zarella"
    UserPrincipalName = "maude.zarella@it-connect.local"
    EmailAddress = "maude.zarella@it-connect.local"
    Path = "OU=Personnel,DC=IT-CONNECT,DC=LOCAL"
    AccountPassword = (Read-Host -AsSecureString "Mot de passe ?")
    ChangePasswordAtLogon = $true
    Enabled = $true
}

New-ADUser @NewUserParams
```

Dans la variable $NewUserParams, les paramètres sont déclarés et les valeurs associées. Ensuite, on appelle cette variable directement dans New-ADUser en spécifiant @NewUserParams (on remplace « $ » par « @ » devant le nom).

III. New-ADUser : les paramètres additionnels

Lorsque vous allez commencer à vouloir affiner la création de votre utilisateur, il se peut que vous ne trouviez pas le paramètre associé à l'attribut dans lequel vous souhaitez injecter des données. La bonne nouvelle, c'est qu'il y a une solution à cette problématique puisque **le paramètre -OtherAttributes sert à écrire dans n'importe quel attribut de l'Active Directory**.

Par exemple, il est intéressant de modifier l'attribut « ProxyAddresses » pour définir l'adresse e-mail principale de l'utilisateur si vous utilisez la synchronisation via Azure AD Connect sur Office 365. La commande New-ADUser n'intègre pas de paramètre pour cet attribut, il faut utiliser -OtherAttributes. Si l'on reprend l'exemple précédent, cela donne :

```
New-ADUser -Name "Maude Zarella" `
           -DisplayName "Maude Zarella" `
           -GivenName "Maude" `
           -Surname "Zarella" `
           -SamAccountName "maude.zarella" `
           -UserPrincipalName "maude.zarella@it-connect.local" `
           -EmailAddress "maude.zarella@it-connect.local" `
           -Path "OU=Personnel,DC=IT-CONNECT,DC=LOCAL" `
           -AccountPassword(Read-Host -AsSecureString "Mot de passe ?") `
           -ChangePasswordAtLogon $true `
           -Enabled $true `
           -OtherAttributes @{'proxyaddresses'="SMTP:maude.zarella@it-
connect.local"}
```

La syntaxe est un peu particulière, mais c'est logique, il faut indiquer à la fois le nom de l'attribut AD et la valeur que l'on souhaite injecter.

Note : même si un attribut dispose d'un paramètre correspondant, il peut être défini via le paramètre OtherAttributes.

Bien entendu, plusieurs attributs peuvent être configurés de cette façon, voici la syntaxe à adopter :

```
-OtherAttributes @{'surname'="Zarella";'givenName'="Maude"}
```

Enfin, nous avons vu comment imposer le changement du mot de passe à la première ouverture de session (*ChangePasswordAtLogon*), mais il existe aussi des paramètres pour gérer les autres options courantes liées aux mots de passe :

- **-CannotChangePassword** = L'utilisateur ne peut pas changer de mot de passe
- **-PasswordNeverExpires** = Le mot de passe n'expire jamais

Un petit bonus, pour définir une date d'expiration du compte, le paramètre -AccountExpirationDate doit être ajouté à la commande New-ADUser en spécifiant une date. Pour ce paramètre, attention tout de même au format qui doit correspondre à une date. Voici un exemple pour forcer la chaîne de caractères au format DateTime :

```
$AccountExpirationDate = [Datetime]::ParseExact("27/04/2021", 'dd/MM/yyyy', $null)
```

En cas de besoin, la documentation officielle de Microsoft est une bonne aide pour vous renseigner sur les différents paramètres :

https://docs.microsoft.com/en-us/powershell/module/addsadministration/new-aduser?view=win10-ps

Suite à ces premiers pas avec la commande New-ADUser, il est temps de passer à la vitesse supérieure et de voir comment créer des comptes en lot, à partir d'un fichier CSV. Direction le prochain chapitre.

3. Créer des utilisateurs dans l'AD à partir d'un CSV

La commande New-ADUser étant désormais maîtrisée puisque nous l'avons bien manipulée au sein du chapitre précédent, il est temps d'en tirer profit et de coupler l'utilisation de cette commande avec un fichier CSV.

L'objectif de ce chapitre, c'est de créer un ensemble d'utilisateurs dans l'Active Directory à partir de données récupérées dans un fichier CSV. **Ainsi, que l'on ait 1 utilisateur, 10 ou 100, le script PowerShell va s'occuper de créer les utilisateurs à notre place, de quoi gagner un temps précieux**.

Au-delà du plaisir de réaliser cette action en ligne de commande, il y a un véritable intérêt : **créer les utilisateurs de la même façon, sans se tromper, sans oublier de remplir un champ.** Le script, contrairement à l'humain, ne fera pas ce genre d'oubli. En tout cas, si une erreur survient c'est que le code doit être amélioré, probablement pour gérer un cas particulier.

Je souhaite vous montrer une méthode simple sans vous noyer dans les lignes de code, il conviendra de faire évoluer ce que l'on voit ensemble avec vos besoins propres et au fur et à mesure de votre montée en compétence en PowerShell. Cela reste du scripting alors il y a mille et une façons de faire…

I. Le fichier CSV avec les utilisateurs

Dans la pratique, en entreprise, le fichier CSV récupéré sera probablement issu d'une base de données, d'un ERP par exemple, et l'on vous demandera de créer les comptes à partir de cet export. Les colonnes contenues dans ce fichier seront variées, et potentiellement il y aura des colonnes inutiles pour la création des comptes dans l'AD, mais ce n'est pas gênant.

Prenons pour l'exemple le fichier CSV suivant :

Prenom;Nom;Fonction
Gérard;Mensoif;Directeur
Sophie;Fonfek;Secrétaire
John;Doeuf;Comptable
Juda;Nanas;Secrétaire
Cécile;Ourkessa;Secrétaire

Ce fichier contient simplement trois valeurs : le prénom, le nom et la fonction. À partir de ça, nous allons devoir créer des comptes utilisateurs dans l'AD. L'idée c'est d'injecter ces valeurs dans l'AD dans les bons champs et de créer des identifiants sous la forme : p.nom (première lettre du prénom, suivie du nom, le tout séparé par un point). L'adresse e-mail est à générer également avec le domaine de l'entreprise, en reprenant le login.

II. Importer le fichier CSV dans un script

La première étape, c'est déjà de récupérer dans une variable le contenu de notre fichier CSV afin d'obtenir la collection de données à traiter. Pour réaliser cette opération, le cmdlet *Import-CSV* doit être utilisé :

```
$CSVFile = "C:\Scripts\AD_USERS\Utilisateurs.csv"
$CSVData = Import-CSV -Path $CSVFile -Delimiter ";" -Encoding UTF8
```

Si l'on affiche le contenu de la variable $CSVData, on peut voir que nos données sont bien chargées :

```
PS C:\Scripts> $CSVData

Prenom Nom      Fonction
------ ---      --------
Gérard Mensoif  Directeur
Sophie Fonfek   Secrétaire
John   Doeuf    Comptable
Juda   Nanas    Secrétaire
Cécile Ourkessa Secrétaire
```

Les données étant chargées dans une variable, nous allons pouvoir réaliser des actions sur ces données :

- Construire l'identifiant sous la forme p.nom, car pour le moment nous avons seulement le prénom et le nom de façon indépendante
- Créer l'utilisateur dans l'Active Directory

Pour réaliser un traitement sur chaque « ligne » (objet) de notre collection de données, nous allons utiliser une boucle Foreach en PowerShell.

III. Créer les utilisateurs AD en masse

Pour la création des utilisateurs en masse dans l'Active Directory, nous allons commencer par déclarer la boucle Foreach avant de venir l'alimenter :

```
Foreach($Utilisateur in $CSVData){

}
```

Commençons par stocker le prénom et le nom dans deux variables différentes. Il est intéressant dans l'objet AD de chaque utilisateur d'injecter ces deux valeurs.

```
Foreach($Utilisateur in $CSVData){

    $UtilisateurPrenom = $Utilisateur.Prenom
    $UtilisateurNom = $Utilisateur.Nom
}
```

Ici, vous remarquerez que pour récupérer la valeur d'une colonne spécifique du fichier CSV, on indique $Utilisateur qui contient l'objet en cours (la ligne en cours de traitement) suivi du nom de la colonne. Pour récupérer le prénom, cela donne : *$Utilisateur.Prenom*

Maintenant, nous allons construire le login en respectant la convention de nommage : p.nom. Pour récupérer la première lettre du prénom, il suffit d'appliquer la méthode Substring() sur la variable $UtilisateurPrenom.

Sur le même principe, nous pouvons générer l'adresse e-mail avec le domaine de messagerie @it-connect.fr. Ce qui donne :

```
Foreach($Utilisateur in $CSVData){

    $UtilisateurPrenom = $Utilisateur.Prenom
    $UtilisateurNom = $Utilisateur.Nom
    $UtilisateurLogin = ($UtilisateurPrenom).Substring(0,1) + "." + $UtilisateurNom
    $UtilisateurEmail = "$UtilisateurLogin@it-connect.fr"
}
```

Dans le cas où le CSV n'est pas bien formaté, par exemple avec des prénoms ou des noms avec des majuscules mal placées, ou tout en majuscules, je vous recommande d'utiliser la méthode ToLower(). Elle va permettre de basculer toute la chaîne en minuscules (sur une seule ligne) :

```
$UtilisateurLogin = ($UtilisateurPrenom).Substring(0,1).ToLower() + "." +
$UtilisateurNom.ToLower()
```

Ce que l'on a oublié de gérer pour le moment, c'est le mot de passe. Disons que l'on va définir le mot de passe « IT-Connect@2020 » à tous les utilisateurs, par défaut. Nous demanderons à ce qu'il soit changé à la première connexion. Nous pourrions tout à fait avoir une variable qui va stocker ce mot de passe, nous allons l'ajouter dans la boucle, ainsi qu'une variable pour stocker la fonction.

```
Foreach($Utilisateur in $CSVData){

    $UtilisateurPrenom = $Utilisateur.Prenom
    $UtilisateurNom = $Utilisateur.Nom
    $UtilisateurLogin = ($UtilisateurPrenom).Substring(0,1).ToLower() + "." +
$UtilisateurNom.ToLower()
    $UtilisateurEmail = "$UtilisateurLogin@it-connect.fr"
    $UtilisateurMotDePasse = "IT-Connect@2020"
    $UtilisateurFonction = $Utilisateur.Fonction

}
```

Maintenant que nous avons formaté les données que nous souhaitons injecter, il ne reste plus qu'à intégrer à la boucle la commande New-ADUser.

Je vais en profiter pour intégrer une condition pour créer l'utilisateur dans l'AD seulement s'il n'existe pas déjà, sinon la commande New-ADUser va retourner une erreur, donc autant le vérifier en amont.

La condition à inclure dans la boucle est la suivante :

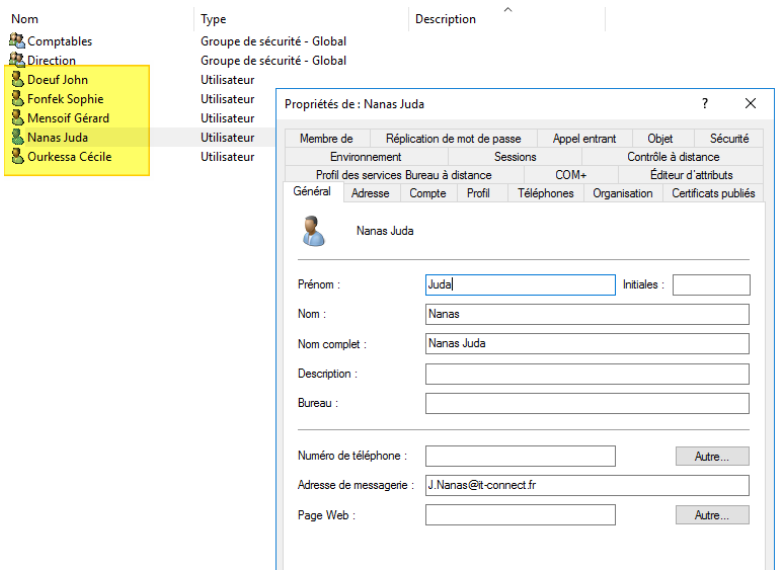
```
# Vérifier la présence de l'utilisateur dans l'AD
if (Get-ADUser -Filter {SamAccountName -eq $UtilisateurLogin})
{
    Write-Warning "L'identifiant $UtilisateurLogin existe déjà dans l'AD"
}
else
{
      # New-ADUser...
}
```

Au final, le code complet est le suivant :

```
$CSVFile = "C:\Scripts\AD_USERS\Utilisateurs.csv"
$CSVData = Import-CSV -Path $CSVFile -Delimiter ";" -Encoding UTF8

Foreach($Utilisateur in $CSVData){

    $UtilisateurPrenom = $Utilisateur.Prenom
    $UtilisateurNom = $Utilisateur.Nom
    $UtilisateurLogin = ($UtilisateurPrenom).Substring(0,1).ToLower() + "." +
$UtilisateurNom.ToLower()
    $UtilisateurEmail = "$UtilisateurLogin@it-connect.fr"
    $UtilisateurMotDePasse = "IT-Connect@2020"
    $UtilisateurFonction = $Utilisateur.Fonction

    # Vérifier la présence de l'utilisateur dans l'AD
    if (Get-ADUser -Filter {SamAccountName -eq $UtilisateurLogin})
    {
        Write-Warning "L'identifiant $UtilisateurLogin existe déjà dans l'AD"
    }
    else
    {

        New-ADUser -Name "$UtilisateurNom $UtilisateurPrenom" `
                   -DisplayName "$UtilisateurNom $UtilisateurPrenom" `
                   -GivenName $UtilisateurPrenom `
                   -Surname $UtilisateurNom `
                   -SamAccountName $UtilisateurLogin `
                   -UserPrincipalName "$UtilisateurLogin@it-connect.local" `
                   -EmailAddress $UtilisateurEmail `
                   -Title $UtilisateurFonction `
                   -Path "OU=Personnel,DC=IT-CONNECT,DC=LOCAL" `
                   -AccountPassword(ConvertTo-SecureString $UtilisateurMotDePasse
-AsPlainText -Force) `
                   -ChangePasswordAtLogon $true `
                   -Enabled $true

        Write-Output "Création de l'utilisateur : $UtilisateurLogin
($UtilisateurNom $UtilisateurPrenom)"
    }
}
```

Pour la commande New-ADUser, j'ai repris ce que nous avons pu voir dans le chapitre précédent, la différence ici c'est que nous utilisons des variables à la place des valeurs fixes. Les comptes seront tous créés dans la même unité d'organisation au niveau de l'Active Directory, à savoir « OU=Personnel,DC=IT-CONNECT,DC=LOCAL ». Pour rendre cela dynamique, nous pouvons ajouter une colonne dans le CSV et ajouter l'OU cible pour chaque compte, ou ajouter une condition pour affecter une OU selon la fonction de l'utilisateur.

L'exécution de ce script va créer les utilisateurs dans l'annuaire Active Directory :



Nous pouvons également collecter les données de ces nouveaux utilisateurs via PowerShell :

```
Get-ADUser  -Filter * -SearchBase "OU=Personnel,DC=IT-CONNECT,DC=LOCAL" `
            -Properties Name,GivenName,Surname,EmailAddress,Title  |`
            Select-Object Name,GivenName,Surname,EmailAddress,Title | Ft
```

On obtient en retour un joli tableau et on peut vérifier que les comptes sont bien conformes :

```
Name             GivenName Surname  EmailAddress              Title
----             --------- -------  ------------              -----
Mensoif Gérard   Gérard    Mensoif  G.Mensoif@it-connect.fr   Directeur
Fonfek Sophie    Sophie    Fonfek   S.Fonfek@it-connect.fr    Secrétaire
Doeuf John       John      Doeuf    J.Doeuf@it-connect.fr     Comptable
Nanas Juda       Juda      Nanas    J.Nanas@it-connect.fr     Secrétaire
Ourkessa Cécile  Cécile    Ourkessa C.Ourkessa@it-connect.fr Secrétaire
```

Félicitations ! Vous venez de créer des comptes utilisateurs dans l'Active Directory à partir d'un fichier CSV. J'ai pris le temps d'expliquer les étapes une à une, afin d'avoir une certaine progression dans la construction du script, pour que ce soit clair et plus facile à suivre.

💡 *Référez-vous au script « EBOOK-ADDS-Module-2-Chapitre-3-01.ps1 » sur GitHub*

4. Modifier le mot de passe d'un compte AD avec PowerShell

La gestion des utilisateurs implique également de gérer le cycle de vie de ces comptes utilisateurs, et il y a fort à parier qu'un jour on vous demande de réinitialiser le mot de passe d'un compte utilisateur, voire même de plusieurs utilisateurs. Par ailleurs, dans un processus de désactivation des comptes utilisateurs, il peut être intéressant de systématiquement réinitialiser le mot de passe du compte en question.

Dans ce chapitre, nous allons justement voir comment modifier le mot de passe d'un utilisateur Active Directory avec PowerShell.

I. Réinitialiser un mot de passe d'un utilisateur Active Directory

Le cmdlet « **Set-ADAccountPassword** » intégré au module Active Directory de PowerShell sera notre allié pour réaliser cette action.

L'utilisation de cette commande nécessite de spécifier plusieurs paramètres :

- -Identity pour spécifier l'utilisateur cible
- -NewPassword pour indiquer le nouveau mot de passe (via une chaîne SecureString)
- -Reset pour indiquer que nous souhaitons réinitialiser le mot de passe

Pour modifier le mot de passe de l'utilisateur « j.nanas », cela donne :

```
Set-ADAccountPassword -Identity j.nanas -Reset -NewPassword (ConvertTo-SecureString -AsPlainText "GhYjKb!M05n" -Force)
```

En fait, le fait d'indiquer -Reset permet de réinitialiser le mot de passe, c'est-à-dire qu'on le change sans connaître le mot de passe actuel, car les droits le permettent.

Pour respecter le processus classique où l'on se met dans la peau d'un utilisateur, il faudrait spécifier à la fois le mot de passe actuel et le nouveau mot de passe, dans ce cas il serait nécessaire d'utiliser le paramètre -OldPassword à la place de -Reset. Néanmoins, dans un script il faudra toujours utiliser la commande via le -Reset, mais c'est intéressant de connaître les deux options.

Par ailleurs, si vous souhaitez simplement modifier un mot de passe et avoir un prompt qui vous demande de le saisir deux fois :

```
Set-ADAccountPassword -Identity j.nanas -Reset
```

```
PS C:\> Set-ADAccountPassword -Identity j.nanas -Reset
Entrez le mot de passe souhaité pour « CN=Nanas Juda,OU=Personnel,DC=IT-CONNECT,DC=LOCAL »
Mot de passe : **************
Répétez le mot de passe : **************
PS C:\> _
```

II. Modifier des mots de passe en lot

Pour réinitialiser des mots de passe pour un lot d'utilisateurs, on peut se baser sur un fichier CSV ou alors cibler une unité d'organisation spécifique dans l'Active Directory. Pour récupérer une liste d'utilisateurs d'une OU spécifique, vous pouvez vous reporter au chapitre précédent où nous avons vu comment récupérer des informations sur les utilisateurs dans l'Active Directory.

Pour cet exemple, prenons un fichier CSV avec deux colonnes : SamAccountName et Password, qui correspondent respectivement à l'identifiant de connexion de l'utilisateur et au mot de passe que l'on veut lui attribuer.

Le fichier CSV sera alors :

```
SamAccountName;Password
j.doeuf;BOnJOuR*50
s.fonfek;BOn$O!R*14
```

Note : On peut nommer les colonnes comme on le souhaite, le principal c'est que les noms soient significatifs.

En quelques lignes de code et une boucle ForEach, on peut parcourir notre fichier CSV et effectuer des modifications de mots de passe dans l'Active Directory :

```
# Controleur de domaine cible
$Server = "SRV-ADDS-01"

# Chemin vers le fichier CSV
$fichier = "C:\Scripting\CSV\Users_Password.csv"

# Importer le fichier CSV
$users = Import-Csv -Path $fichier -Delimiter ";" -Encoding UTF8

# Pour chaque ligne du CSV... (Pour chaque utilisateur)
foreach($user in $users){

    # Récupère le SamAccountName de l'utilisateur
    $UserSAM = $user.SamAccountName

    # Récupérer le mot de passe de l'utilisateur
    $UserPwd = $user.Password

    Set-ADAccountPassword -Server $Server -Identity $UserSAM -Reset -NewPassword
(ConvertTo-SecureString -AsPlainText $UserPwd -Force)
}
```

Référez-vous au script « EBOOK-ADDS-Module-2-Chapitre-4-01.ps1 » sur GitHub

Ressource associée :

https://www.it-connect.fr/powershell-comment-modifier-des-mots-de-passe-dans-lactive-directory/

III. Demander à l'utilisateur de changer son mot de passe

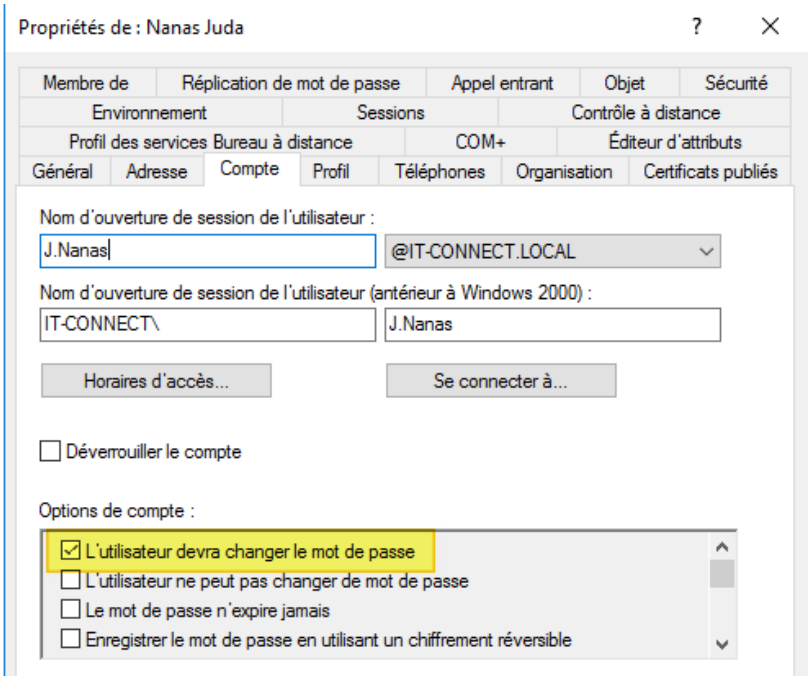
Réinitialiser le mot de passe c'est une chose, mais vous ne devez pas oublier que vous devez lui demander de le personnaliser à la prochaine ouverture de session.

Cela nécessite une action supplémentaire afin d'activer l'option « L'utilisateur devra changer le mot de passe » dans les propriétés de l'objet de cet utilisateur.

En l'occurrence, pour demander à l'utilisateur « j.nanas » de modifier son mot de passe à la prochaine ouverture de session (que ce soit une ouverture de session complète ou simplement ouvrir une session verrouillée), il faudra modifier le compte utilisateur avec cette commande :

```
Set-ADUser -Identity j.nanas -ChangePasswordAtLogon $true
```

Nous passons simplement le paramètre -ChangePasswordAtLogon sur $true (vrai) afin d'activer l'option « L'utilisateur devra changer le mot de passe ». Pour que ce soit plus visuel, voici à quoi je fais référence :



Cette dernière commande me donne l'occasion de faire la transition avec le prochain chapitre puisque nous allons voir comment modifier les comptes utilisateurs dans l'Active Directory, avec PowerShell.

5. Modifier des comptes AD en lot avec PowerShell

Dans ce chapitre, nous allons mettre en application ce que nous avons pu voir précédemment, à savoir la récupération d'informations sur des utilisateurs Active Directory, ainsi que la boucle Foreach que l'on a utilisée pour créer des utilisateurs en lot à partir d'un CSV.

Pour **modifier des utilisateurs Active Directory en masse**, nous allons réutiliser la même boucle. Ensuite, pour sélectionner les utilisateurs à modifier, soit on peut avoir un fichier CSV en entrée, ou alors on récupère une sélection d'utilisateurs directement dans l'Active Directory avec Get-ADUser.

La commande PowerShell à utiliser pour modifier un compte utilisateur est Set-ADUser. Pour modifier le mot de passe, nous avons pu voir dans le chapitre précédent qu'il fallait utiliser Set-ADAccountPassword.

I. Ajouter une valeur dans un attribut en masse à partir d'un CSV

Dans ce premier exemple, nous allons reprendre notre CSV précédent, à savoir celui qui contient les données suivantes :

Prenom;Nom;Fonction
Gérard;Mensoif;Directeur
Sophie;Fonfek;Secrétaire
John;Doeuf;Comptable
Juda;Nanas;Secrétaire
Cécile;Ourkessa;Secrétaire

Nous allons faire en sorte, pour l'exemple, d'injecter les valeurs de la colonne « Fonction » dans le champ « Description » de nos utilisateurs.

Comme point de départ, nous allons donc charger le fichier CSV dans la variable $CSVData et ajouter une boucle Foreach pour parcourir la collection d'objets. Dans l'idéal, il faudrait avoir les identifiants dans le fichier CSV, ce serait plus simple pour cibler les utilisateurs. Ici, nous n'avons pas les identifiants dans le fichier CSV, mais on sait qu'ils sont sous la forme « p.nom » : on va donc le construire à partir du prénom et du nom (comme vu dans le chapitre précédent).

Voici le point de départ :

```
$CSVFile = "C:\Scripts\AD_USERS\Utilisateurs.csv"
$CSVData = Import-CSV -Path $CSVFile -Delimiter ";" -Encoding UTF8

Foreach($Utilisateur in $CSVData){

    # Construction du login (SamAccountName)
    $UtilisateurLogin = ($Utilisateur.Prenom).Substring(0,1) + "." + $Utilisateur.Nom


}
```

Nous allons intégrer à notre boucle la commande Set-ADUser pour modifier le champ « Description » pour chaque utilisateur. La commande est :

```
Set-ADUser -Identity $UtilisateurLogin -Description $Utilisateur.Fonction
```

Si vous souhaitez utiliser le champ « Description » pour autre chose, je vous recommande d'intégrer la fonction directement dans l'attribut « Title » de l'Active Directory.

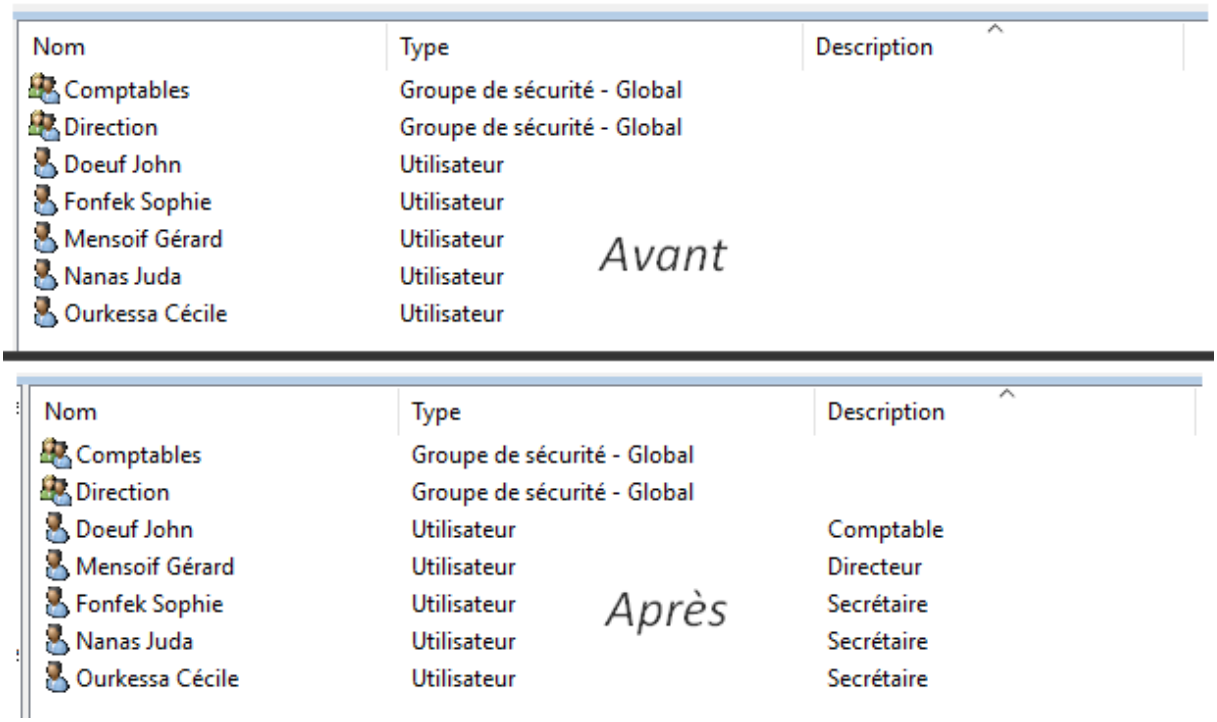
Il suffit d'utiliser le paramètre -Title plutôt que -Description.

```
Set-ADUser -Identity $UtilisateurLogin -Title $Utilisateur.Fonction
```

Dans le script, on va au préalable s'assurer que l'utilisateur existe avant de chercher à le modifier. Au final, voici le code complet de la boucle :

```
Foreach($Utilisateur in $CSVData){

    # Construction du login (SamAccountName)
    $UtilisateurLogin = ($Utilisateur.Prenom).Substring(0,1) + "." + $Utilisateur.Nom

    if (Get-ADUser -Identity $UtilisateurLogin)
    {
       Write-Output "L'identifiant $UtilisateurLogin existe, l'utilisateur va être modifié."
       Set-ADUser -Identity $UtilisateurLogin -Description $Utilisateur.Fonction
    }
}
```

Suite à l'exécution de ce bout de code, les changements sont directement visibles dans l'Active Directory :



II. Modifications en masse via Get-ADUser et Set-ADUser

Comme je le disais, on peut utiliser Get-ADUser pour récupérer une liste d'utilisateurs et ensuite réaliser des modifications sur les objets récupérés : ajouter une valeur dans un champ, vider un champ, écraser la valeur d'un champ, etc.

Ainsi, je vais vous proposer quelques exemples, simples, mais utiles, pour découvrir les différentes façons d'utiliser Set-ADUser. **Chaque exemple commencera par Get-ADUser et le résultat de la requête sera envoyé à Set-ADUSer via le pipeline.**

1 – Vider la valeur du champ « Description » pour tous les utilisateurs dont la description contient le mot « Directeur »

```
Get-ADUser -Filter {Description -like '*Directeur*'} | `
           Set-Aduser -Clear Description
```

2 – Mettre à jour le nom d'affichage (DisplayName) de tous les utilisateurs de l'OU « OU=Personnel,DC=IT-CONNECT,DC=LOCAL » avec une valeur au format « Nom Prénom » en récupérant les valeurs déjà dans l'objet (attributs Surname et GivenName)

```
Get-ADUser -Filter * -SearchBase "OU=Personnel,DC=IT-CONNECT,DC=LOCAL" -Properties
DisplayName | ForEach-Object{ Set-ADUser $_ -DisplayName ($_.Surname + ' ' +
$_.GivenName) }
```

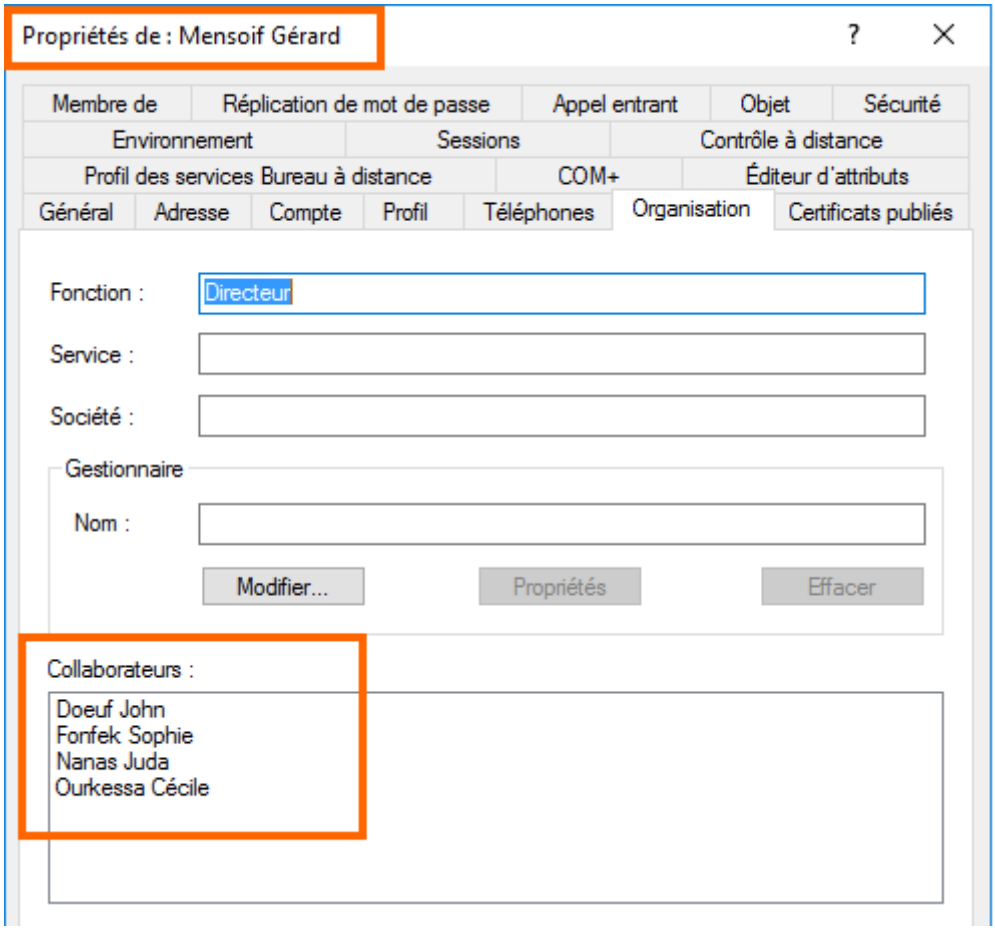
3 – Remplacer/écraser la valeur du champ « ProxyAddresses » des utilisateurs de l'OU « OU=Personnel,DC=IT-CONNECT,DC=LOCAL » pour indiquer l'adresse e-mail principale sous la forme « SMTP:<e-mail> » en récupérant l'adresse e-mail de l'attribut *mail*

```
Get-ADUser -Filter * -SearchBase "OU=Personnel,DC=IT-CONNECT,DC=LOCAL" -Properties
mail | ForEach-Object{ Set-ADUser $_ -Replace @{proxyAddresses="SMTP:$($_.mail)"} }
```

4 – Attribuer le directeur « g.mensoif » comme manager des comptes de l'OU « OU=Personnel,DC=IT-CONNECT,DC=LOCAL » dont la description ne contient pas le mot « directeur »

```
Get-ADUser -Filter {Description -notlike '*Directeur*'} -SearchBase
"OU=Personnel,DC=IT-CONNECT,DC=LOCAL" | ForEach-Object{ Set-ADUser $_ -Manager
"G.Mensoif" }
```

Lorsque l'on spécifie le manager (-Manager), il faut bien spécifier le SamAccountName de l'utilisateur à définir en tant que responsable.



En résumé, la commande Set-ADUser est un excellent moyen de mettre à jour rapidement les valeurs des attributs Active Directory d'un ou plusieurs utilisateurs.

Module 3 - Gérer les groupes AD avec PowerShell

1. Créer un groupe Active Directory avec PowerShell

Après avoir manipulé les utilisateurs Active Directory de différentes façons, il est temps désormais de s'intéresser à la gestion des groupes Active Directory avec PowerShell.

Dans ce chapitre, nous allons apprendre à créer un groupe de sécurité Active Directory, ainsi qu'un groupe de distribution. En préambule, je vous propose de voir comment récupérer des informations sur un groupe puisque ce sera utile pour vérifier si le groupe est bien créé.

I. Récupérer les informations sur un groupe

Le cmdlet PowerShell qui sert à récupérer des informations sur un groupe AD se nomme « Get-ADGroup », sans surprise.

Sa syntaxe est similaire à celle de son cousin « *Get-ADUser* » et reprend le principe des paramètres « *-Identity* » et « *-Filter* » pour filtrer la requête. Par exemple, pour obtenir des informations sur le groupe « Direction », la commande sera la suivante :

```
Get-ADGroup -Identity "Direction"
```

```
PS C:\Windows\system32> Get-ADGroup -Identity "Direction"


DistinguishedName : CN=Direction,OU=Personnel,DC=IT-CONNECT,DC=LOCAL
GroupCategory     : Security
GroupScope        : Global
Name              : Direction
ObjectClass       : group
ObjectGUID        : 31a4bfdc-0487-4251-aade-3011e9c8a4fb
SamAccountName    : Direction
SID               : S-1-5-21-2321128819-363875854-497869713-1117
```

Pour afficher l'ensemble des propriétés d'un groupe, il suffira de faire appel au paramètre « *-Properties* », comme ceci :

```
Get-ADGroup -Identity "Direction" -Properties *
```

Le résultat retourné est, en toute logique, beaucoup plus détaillé puisque nous avons toutes les informations sur ce groupe, y compris **ses membres et les groupes dans lesquels il est lui-même membre.**

Si l'on souhaite rechercher tous les groupes qui contiennent le mot « admin » par exemple, nous devons ajouter un filtre sur le « Name », comme ceci :

```
Get-ADGroup -Filter 'Name -like "*admin*"' | Format-Table
```

```
PS C:\Windows\system32> Get-ADGroup -Filter 'Name -like "*admin*"' | Format-Table

DistinguishedName                                                GroupCategory  GroupScope Name
-----------------                                                -------------  ---------- ----
CN=Administrateurs,CN=Builtin,DC=IT-CONNECT,DC=LOCAL                  Security DomainLocal Administrateurs
CN=Administrateurs Hyper-V,CN=Builtin,DC=IT-CONNECT,DC=LOCAL          Security DomainLocal Administrateurs Hype
CN=Storage Replica Administrators,CN=Builtin,DC=IT-CONNECT,DC=LOCAL   Security DomainLocal Storage Replica Admi
CN=Administrateurs du schéma,CN=Users,DC=IT-CONNECT,DC=LOCAL          Security   Universal Administrateurs du s
CN=Administrateurs de l'entreprise,CN=Users,DC=IT-CONNECT,DC=LOCAL    Security   Universal Administrateurs de l
CN=Admins du domaine,CN=Users,DC=IT-CONNECT,DC=LOCAL                  Security      Global Admins du domaine
CN=Administrateurs clés,CN=Users,DC=IT-CONNECT,DC=LOCAL               Security      Global Administrateurs clés
CN=Administrateurs clés Enterprise,CN=Users,DC=IT-CONNECT,DC=LOCAL    Security   Universal Administrateurs clés
CN=DnsAdmins,CN=Users,DC=IT-CONNECT,DC=LOCAL                          Security DomainLocal DnsAdmins
```

Passons maintenant à la création d'un groupe de sécurité.

II. Créer un groupe AD avec PowerShell

La création d'un groupe Active Directory via PowerShell nécessite d'utiliser le cmdlet « New-ADGroup », et nous allons utiliser plusieurs paramètres :

- **Name** : le nom du groupe
- **Path** : l'unité d'organisation dans laquelle on souhaite créer ce groupe
- **GroupScope** : l'étendue du groupe (domaine local / globale / universelle)
- **Description** : la description du groupe

Pour créer le groupe « *Informatique* » dans l'unité d'organisation « *OU=Personnel,DC=IT-CONNECT,DC=LOCAL* », avec l'étendue « *Globale* » et la description « *Service informatique* », la commande sera :

```
New-ADGroup -Name "Informatique" -Path "OU=Personnel,DC=IT-CONNECT,DC=LOCAL" -
GroupScope Global -Description "Service informatique"
```

Ensuite, nous pouvons vérifier le résultat à l'aide de la commande vu précédemment :

```
Get-ADGroup -Identity "Informatique"
```

```
PS C:\Windows\system32> Get-ADGroup -Identity "Informatique"


DistinguishedName : CN=Informatique,OU=Personnel,DC=IT-CONNECT,DC=LOCAL
GroupCategory     : Security
GroupScope        : Global
Name              : Informatique
ObjectClass       : group
ObjectGUID        : 5feb0d72-c439-40f6-ab63-1e1a9514cf30
SamAccountName    : Informatique
SID               : S-1-5-21-2321128819-363875854-497869713-1139
```

La bonne nouvelle c'est que le résultat est conforme aux attentes.

III. Créer un groupe de distribution

En complément des groupes de sécurité, nous avons généralement recourt aux groupes de distribution pour la partie messagerie. **La création d'un groupe de distribution est possible avec PowerShell et le même cmdlet : New-ADGroup.** Pour rappel, un groupe de distribution sert à créer une liste de d'adresses e-mail en ajoutant des contacts : envoyer un e-mail sur l'adresse associée à ce groupe permettra de contacter tous les contacts associés.

La différence se situe dans la syntaxe de la commande : pour **créer un groupe de distribution**, il est nécessaire de préciser le paramètre « *-GroupCategory* » suivi de la valeur « *Distribution* ». Les autres paramètres, à savoir : GroupScope, Path, Description, etc… Restent applicables.

```
New-ADGroup "Service_Informatique" -GroupCategory Distribution -GroupScope Global
```

Nous pouvons vérifier la présence du groupe avec la commande habituelle :

```
Get-ADGroup -Identity "Service_Informatique"
```

```
PS C:\Windows\system32> Get-ADGroup -Identity "Service_Informatique"


DistinguishedName : CN=Service_Informatique,CN=Users,DC=IT-CONNECT,DC=LOCAL
GroupCategory     : Distribution
GroupScope        : Global
Name              : Service_Informatique
ObjectClass       : group
ObjectGUID        : 0d591c6d-d39e-45da-96f9-42f9ab078d66
SamAccountName    : Service_Informatique
SID               : S-1-5-21-2321128819-363875854-497869713-1140
```

Vous êtes désormais capable de créer un groupe Active Directory, que ce soit un groupe de sécurité, ou un groupe de distribution, au sein de votre environnement et grâce à PowerShell. Passons au chapitre suivant où nous verrons comment ajouter ou retirer des membres d'un groupe.

2. Ajouter ou retirer des membres d'un groupe AD

Créer un groupe Active Directory, c'est une chose, et nous avons vu comment le faire dans le chapitre précédent, mais être capable de le gérer c'est mieux. J'entends par là, être capable d'ajouter ou supprimer des membres dans un groupe Active Directory grâce à PowerShell. C'est ce que nous allons voir dans ce chapitre.

I. Ajouter un utilisateur dans un groupe AD

Vous pouvez ajouter un utilisateur à un groupe Active Directory en utilisant le cmdlet « Add-AdGroupMember ». Prenons un exemple et reprenons notre groupe créé précédemment, à savoir « Informatique » : nous allons ajouter deux utilisateurs à ce groupe : « l.leuze » et « g.zola ».

Il suffit d'indiquer le nom du groupe et les membres à ajouter à la suite du paramètre « **-Members** ». Quand il y a plusieurs utilisateurs à ajouter, les noms doivent être séparés par une virgule.

Ce qui donne :

```
Add-AdGroupMember -Identity Informatique -Members "l.leuze", "g.zola"
```

Pour récupérer la liste des membres d'un groupe Active Directory, le cmdlet « Get-ADGroupMember » doit être utilisé. Nous reviendrons plus en détail sur ce cmdlet dans le chapitre précédent, mais en attendant, voici comment l'utiliser basiquement :

```
Get-ADGroupMember -Identity Informatique
```

La bonne nouvelle c'est que nos deux utilisateurs sont désormais membre du groupe :

```
PS C:\Windows\system32> Get-ADGroupMember -Identity Informatique


distinguishedName : CN=Leuze Lara,OU=Personnel,DC=IT-CONNECT,DC=LOCAL
name              : Leuze Lara
objectClass       : user
objectGUID        : fa73b6ab-175c-4787-a976-695f801a7dbd
SamAccountName    : L.Leuze
SID               : S-1-5-21-2321128819-363875854-497869713-1141

distinguishedName : CN=Zola Gordon,OU=Personnel,DC=IT-CONNECT,DC=LOCAL
name              : Zola Gordon
objectClass       : user
objectGUID        : cc23ba52-0321-4dd0-a0ca-cfb0530a2ab5
SamAccountName    : G.Zola
SID               : S-1-5-21-2321128819-363875854-497869713-1142
```

II. Ajouter un utilisateur dans un groupe : autres exemples

Maintenant que nous avons vu comment ajouter des utilisateurs dans un groupe Active Directory, je vous propose quelques exemples d'utilisation ci-dessous.

- **Prendre les membres d'un groupe et les ajouter dans un autre groupe**

Imaginons un groupe nommé « GroupeA » qui contient 100 membres, et nous souhaitons récupérer les membres de ce groupe afin de les ajouter à un autre groupe nommé « GroupeB ». PowerShell va nous aider à réaliser cette opération facilement.

Grâce à l'enchaînement de deux pipelines et d'une boucle ForEach-Object, nous pouvons récupérer la liste des membres du premier groupe, ensuite récupérer les utilisateurs associés, puis envoyer tout ça dans une boucle. La boucle quant à elle va ajouter chaque utilisateur dans le groupe « GroupeB ».

```
Get-ADGroupMember "GroupeA" | Get-ADUser | ForEach-Object { Add-ADGroupMember -
Identity "GroupeB" -Members $_ }
```

Nous pouvons faire la même chose en intégrant la notion de récursivité, c'est-à-dire que l'on va prendre tous les membres du « GroupeA », mais aussi tous les utilisateurs qui sont dans les groupes eux-mêmes membres de « GroupeA ». Il suffit d'ajouter le paramètre -Recursive, comme ceci :

```
Get-ADGroupMember "GroupeA" -Recursive | Get-ADUser | ForEach-Object { Add-
ADGroupMember -Identity "GroupeB" -Members $_ }
```

Voyons un autre exemple.

- **Ajouter les utilisateurs d'un CSV dans un groupe AD**

Si l'on prend un fichier CSV avec quelques identifiants de l'AD, comme ceci :

Login;
"l.leuze";
"g.zola";

On peut exécuter une commande pour ajouter tous les utilisateurs de ce fichier CSV dans le groupe « Informatique ». Comme ceci :

```
Import-CSV "C:\Scripts\AD_USERS\Login.csv" -Delimiter ";" | ForEach-Object { Add-
AdGroupMember -Identity "Informatique" -Members $_.login }
```

Voyons comment retirer un utilisateur d'un groupe AD.

III. Retirer un utilisateur d'un groupe AD

Pour retirer un membre d'un groupe Active Directory, finalement, la logique est la même que pour l'ajouter : on précise le nom du groupe, et le nom du ou des membres à retirer du groupe. La différence se situe au niveau du cmdlet que l'on va utiliser : **Remove-ADGroupMember**.

Reprenons nos deux utilisateurs de toute à l'heure, à savoir, « l.leuze » et « g.zola ». Désormais, nous allons les retirer du groupe « Informatique ». Ce qui donne :

```
Remove-AdGroupMember -Identity Informatique -Members l.leuze, g.zola -
Confirm:$false
```

En complément, faut penser à ajouter « *-Confim:$false* » dans cette commande, car sinon une demande de confirmation va s'afficher à l'écran (pas pratique dans un script).

L'exemple vu précédemment basé sur le fichier CSV et également applicable pour retirer des membres d'un groupe, il suffit de remplacer le nom du cmdlet et d'ajouter la suppression de la confirmation.

IV. Mettre à jour la liste des groupes dont un utilisateur est membre

Partons du principe que nous souhaitons qu'un utilisateur soit membre de trois groupes : « Groupe1 », « Groupe2 » et Utilisa. du domaine ». Nous laissons l'utilisateur dans le groupe « Utilisa. du domaine » puisqu'il s'agit de son groupe principal par défaut.

L'idée c'est de mettre à jour l'utilisateur pour qu'il soit membre seulement de ces groupes. Autrement dit, on va épurer la liste des groupes dont il fait partie pour ne laisser que ces trois-là.

On va commencer par définir la liste des groupes dans la variable $UserTargetGroupList. Ensuite, la variable $UserMemberOf va contenir la liste des groupes dont l'utilisateur est membre actuellement.

Enfin, grâce à une boucle Foreach nous allons vérifier pour chaque groupe dont il est membre s'il se situe dans le tableau $UserTargetGroupList. Autrement dit, si le groupe est l'un des trois groupes dont il doit être membre. Si ce n'est pas le cas, on le retire de ce groupe.

Ce bout de code pratique met en pratique différentes commandes que l'on a découvertes précédemment. Voici le résultat :

```
# Liste des groupes dont l'utilisateur doit être membre
$UserTargetGroupList = @("Groupe1","Groupe2","Utilisa. du domaine")

# Récupérer la liste des groupes dont il est membre
$UserMemberOf = Get-ADUser -Filter 'UserPrincipalName -eq $UserUPN' -Properties
memberof | Select-Object -ExpandProperty memberof

Foreach($Group in $UserMemberOf){

    if(($Group.Split(",")[0]).Split("=")[1] -notin $UserTargetGroupList){

        Remove-ADGroupMember -Identity $Group -Members $UserLogin -Confirm:$false
    }

}
```

V. Précisions sur les cmdlets

Pour terminer, il est à noter que pour ajouter un utilisateur dans un nouveau groupe, le cmdlet « Add-ADPrincipalGroupMembership » doit être utilisé. Cela ne remet pas en cause ce que l'on vient de voir, en fait il y a deux façons de faire :

- Soit on ajoute un nouveau membre au groupe via Add-AdGroupMember,
- Soit on ajoute un utilisateur dans le groupe avec « Add-ADPrincipalGroupMembership »

Nous retrouvons le même principe avec les cmdlets « Remove-AdGroupMember » et « Remove-ADPrincipalGroupMembership ».

Dans le prochain chapitre, nous verrons comment obtenir plus de détails sur les membres d'un groupe.

3. Récupérer la liste des membres d'un groupe AD

Dans le précédent chapitre où nous avons vu comment ajouter et retirer des membres d'un groupe Active Directory, j'ai rapidement évoqué le cmdlet « **Get-ADGroupMember** ». Ce dernier sert à récupérer les membres d'un groupe, comme son nom l'indique.

Dans ce chapitre, nous allons revenir sur ce cmdlet afin qu'il soit maîtrisé.

I. Utiliser « Get-ADGroupMember »

Ce cmdlet s'utilise assez simplement puisqu'il suffit de préciser le nom d'un groupe pour lequel on souhaite récupérer les membres. Par exemple, pour récupérer les membres du groupe « Admins du domaine » :

```
Get-ADGroupMember -Identity "Admins du domaine"
```

Si l'on veut un résultat plus lisible, ce qui peut être pratique pour un audit rapide des membres de ce groupe avec notamment l'OU dans laquelle ils se situent :

```
Get-ADGroupMember -Identity "Admins du domaine" | Format-Table Name,
DistinguishedName
```

Il ne reste plus qu'à regarder si ces membres sont légitimes 😉

```
PS C:\Windows\system32> Get-ADGroupMember -Identity "Admins du domaine" | Format-Table Name, DistinguishedName

Name             DistinguishedName
----             -----------------
Administrateur   CN=Administrateur,CN=Users,DC=IT-CONNECT,DC=LOCAL
admin            CN=admin,CN=Users,DC=IT-CONNECT,DC=LOCAL
fb.itconnect     CN=fb.itconnect,CN=Users,DC=IT-CONNECT,DC=LOCAL
Adm_UserLock     CN=Adm_UserLock,CN=Users,DC=IT-CONNECT,DC=LOCAL
IT-Connect Admin CN=IT-Connect Admin,CN=Users,DC=IT-CONNECT,DC=LOCAL
```

Note : pour rappel, le paramètre -Identity accepte différentes valeurs. En effet, nous pouvons préciser un GUID (*objectGUID*), un SID (*objectSID*), un identifiant/login (*sAMAccountName*) ou encore un DN (*DistinguishedName*).

II. Utiliser « Get-ADPrincipalGroupMembership »

Nous avons vu comment récupérer les membres d'un groupe, mais pour récupérer la liste des groupes dont est membre un utilisateur, un groupe ou un ordinateur, nous pouvons utiliser le cmdlet **« Get-ADPrincipalGroupMembership ».**

C'est une approche différente, mais qui peut s'avérer pratique. Nous avons pu voir dans le chapitre précédent que la propriété MemberOf est récupérable aussi directement via le cmdlet Get-ADUser.

Pour récupérer la liste des groupes dont est membre le groupe « Admins du domaine » lui-même, voici la commande :

```
Get-ADPrincipalGroupMembership -Identity "Admins du domaine"
```

```
PS C:\Windows\system32> Get-ADPrincipalGroupMembership -Identity "Admins du domaine"


distinguishedName : CN=Administrateurs,CN=Builtin,DC=IT-CONNECT,DC=LOCAL
GroupCategory     : Security
GroupScope        : DomainLocal
name              : Administrateurs
objectClass       : group
objectGUID        : 1c16a9ae-108e-44d8-af17-bf97948e3556
SamAccountName    : Administrateurs
SID               : S-1-5-32-544

distinguishedName : CN=Groupe de réplication dont le mot de passe RODC est refusé,CN=Users,DC=IT-CONNECT,DC=LOCAL
GroupCategory     : Security
GroupScope        : DomainLocal
name              : Groupe de réplication dont le mot de passe RODC est refusé
objectClass       : group
objectGUID        : e63d7efb-64fd-42a0-8a9b-0b260a1187cd
SamAccountName    : Groupe de réplication dont le mot de passe RODC est refusé
SID               : S-1-5-21-2321128819-363875854-497869713-572
```

III. Compter le nombre de membres dans un groupe

Pour compter le nombre de membres au sein d'un groupe, nous allons utiliser *count*, comme ceci :

```
(Get-ADGroupMember -Identity "Admins du domaine").Count
```

Cette commande retournera tout simplement le nombre de membres, avec un chiffre, sans texte. Cette valeur est récupérable facilement dans une variable.

À la suite de ça, nous pouvons récupérer la liste des groupes qui ont plus de X membres.

Voici comment réaliser cette requête :

```
$GroupList = Get-ADGroup -Filter *
Foreach($Group in $GroupList){

    $GroupMemberCount = (Get-ADGroupMember -Identity $($Group.Name)).Count

    if($GroupMemberCount -gt 5){

        Write-Warning "Le groupe '$($Group.Name)' a $GroupMemberCount membres !"

    }
}
```

La sortie sera sous cette forme :

```
PS C:\Windows\system32> $GroupList = Get-ADGroup -Filter *
Foreach($Group in $GroupList){

    $GroupMemberCount = (Get-ADGroupMember -Identity $($Group.Name)).Count

    if($GroupMemberCount -gt 5){

        Write-Warning "Le groupe '$($Group.Name)' a $GroupMemberCount membres !"

    }
}
AVERTISSEMENT : Le groupe 'Ordinateurs du domaine' a 8 membres !
AVERTISSEMENT : Le groupe 'Utilisateurs du domaine' a 15 membres !
AVERTISSEMENT : Le groupe 'Groupe de réplication dont le mot de passe RODC est refusé' a 8 membres !
```

IV. Récupérer la liste des groupes vides

Lorsqu'il s'agit de faire du nettoyage dans l'annuaire Active Directory, il est pertinent de rechercher les groupes vides. Un groupe vide est un groupe inutile.

Une simple requête sur la base du cmdlet *Get-ADGroup* va permettre de retourner le nom de ces groupes :

```
Get-ADGroup -Filter * -Properties Members | Where-Object { -not $_.members} |
Select-Object Name
```

Vous pouvez compléter le *Select-Object* pour ajouter éventuellement des propriétés supplémentaires, comme le *DistinguishedName*.

V. La notion de rechercher récursive

Pour finir ce chapitre, je vais m'intéresser au paramètre **-Recursive** du cmdlet « **Get-ADGroupMember** ». En fait, il est intéressant **si vous avez des « *nested groups* » : c'est-à-dire des groupes dans des groupes.**

Grâce à lui, plutôt que d'afficher que le « Groupe1 » est membre du « Groupe2 » il va directement vous afficher les membres du « Groupe1 ». Autrement dit, il va afficher les membres directs, c'est-à-dire les membres de niveau 1, mais aussi les membres indirects, c'est-à-dire les membres de niveau 2 correspondants à ceux qui sont dans les groupes imbriqués.

Cela est plus significatif, mais aussi plus gourmand en ressources.

Voici un exemple :

```
Get-ADGroupMember -Identity "Admins du domaine" -Recursive
```

Il y a seulement le paramètre à ajouter à la commande, PowerShell s'occupe du reste 😉

Ce chapitre se termine, vous êtes désormais en mesure de récupérer la liste des membres d'un groupe et la liste des groupes dont un utilisateur est membre (ou un ordinateur, ou un groupe).

4. Modifier et supprimer un groupe AD avec PowerShell

Pour terminer ce module sur la gestion des groupes Active Directory avec PowerShell, nous allons apprendre à modifier et supprimer un groupe AD. Nous avons vu comment gérer les membres d'un groupe, mais nous n'avons pas vu comment renommer un groupe ou lui ajouter une description.

I. Modifier un groupe AD avec PowerShell

La modification passe par l'utilisation du cmdlet « **Set-ADGroup** », ce qui va notamment permettre de modifier la description d'un groupe. Pour cela, le paramètre **-Identifiy** doit être utilisé pour spécifier le groupe à cibler et **-Description** pour indiquer la description.

Voici un exemple :

```
Set-ADGroup -Identity "CN=Direction,OU=Personnel,DC=IT-CONNECT,DC=LOCAL" -
Description "Groupe avec les membres de la direction"
```

Ci-dessus, j'ai spécifié le groupe à modifier en indiquant son DistinguishedName (DN), mais on peut aussi directement spécifier son nom, comme ceci :

```
Set-ADGroup -Identity "Direction" -Description "Groupe avec les membres de la
direction"
```

Nous pourrions aussi modifier la description de tous les groupes qui contiennent le terme « VPN » dans le nom, par exemple. Ce qui donnerait :

```
Get-ADGroup -Filter 'Name -like "*VPN*"' | Set-ADGroup -Description "Groupe VPN"
```

II. Comment changer le type de groupe et son étendue ?

Il est également possible de changer le type de groupe, par exemple pour transformer un groupe de sécurité en groupe de distribution. **Le paramètre -GroupCategory sert à réaliser cette opération, il accepte deux valeurs : Distribution / Security.**

Pour que le groupe « Mailing » devienne un groupe de distribution, voici la commande :

```
Set-ADGroup -Identity "Mailing" -GroupCategory Distribution
```

Ensuite, pour modifier l'étendue du groupe, il faut s'appuyer sur le paramètre **-GroupScope**. Dans la limite de ce qui est possible : un groupe d'étendue « Globale » ne peut pas être modifié en groupe d'étendue « Domaine local » (il faut d'abord le mettre en étendue universelle, ce qui va nécessiter d'utiliser deux commandes..

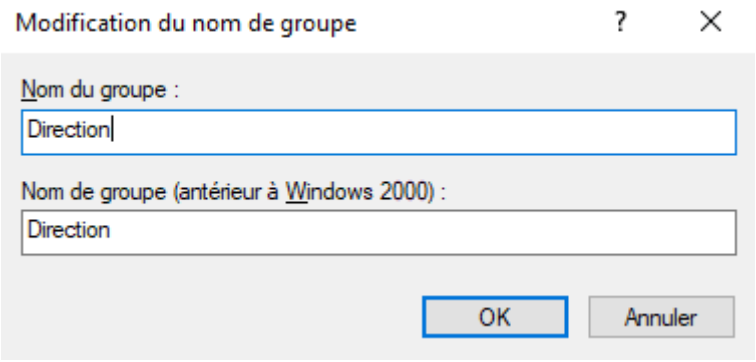
Par exemple, pour modifier un groupe avec l'étendue « Globale » sur l'étendue « Universelle », voici la commande :

```
Set-ADGroup -Identity "Mailing" -GroupScope Universal
```

Ce paramètre accepte trois valeurs : **DomainLocal / Global / Universal**

III. Renommer un groupe AD avec PowerShell

Pour renommer un groupe Active Directory avec PowerShell, l'opération doit être réalisée en deux étapes. En effet, nous allons devoir modifier le nom d'affichage du groupe, son SamAccountName, mais aussi l'objet AD en lui-même. En mode graphique, cela correspond à cette fenêtre :



Commençons par définir deux variables, la première qui contient le nom du groupe à renommer, et la deuxième qui contient le nouveau nom :

```
$Groupe = "CN=Direction,OU=Personnel,DC=IT-CONNECT,DC=LOCAL"
$GroupeNewName = "Direction-2"
```

Nous allons devoir coupler l'utilisation de Set-ADGroup à Get-ADGroup pour récupérer l'objet à modifier. Voici la commande pour modifier le nom d'affichage et le SamAccountName :

```
Get-ADGroup -Identity $Groupe | Set-ADGroup -DisplayName $GroupeNewName -
SamAccountName $GroupeNewName
```

Pour que ce soit cohérent, il est important de renommer également l'objet AD, ce qui nécessite d'utiliser une deuxième commande, cette fois-ci via le cmdlet **Rename-ADObject**.

```
Get-ADGroup -Identity $Groupe | Rename-ADObject -NewName $GroupeNewName
```

Voilà, grâce aux deux commandes ci-dessus, le groupe est désormais renommé ! En résumé :

```
$Groupe = "CN=Direction,OU=Personnel,DC=IT-CONNECT,DC=LOCAL"
$GroupeNewName = "Direction-2"

Get-ADGroup -Identity $Groupe |
    Set-ADGroup -DisplayName $GroupeNewName -SamAccountName $GroupeNewName

Get-ADGroup -Identity $Groupe | Rename-ADObject -NewName $GroupeNewName
```

Les attributs suivants sont modifiés : cn, distinguishedName, displayName, name, sAMAccountName.

IV. Supprimer un groupe AD

Avant de se quitter, sachez que la suppression d'un groupe s'appuie sur le cmdlet « Remove-ADGroup ». Il suffit de préciser le nom du groupe et de retirer la demande de confirmation qui s'affiche par défaut pour éviter les suppressions accidentelles.

Ce qui donne, pour supprimer le groupe « Direction » :

```
Remove-ADGroup -Identity "Direction" -Confirm:$false
```

Commande à manier avec précaution ! Pour faire un test, vous pouvez ajouter le paramètre -WhatIf à la fin de la commande.

Qu'est-ce que le paramètre -WhatIf ? Si une commande est exécutée avec le paramètre -WhatIf, elle ne fera aucune modification sur votre système. Ce qui est intéressant, c'est que grâce à -WhatIf, des informations vont s'afficher dans la console pour indiquer qu'elles sont les actions réalisées pour chaque commande, ainsi que l'objet affecté. Un bon moyen d'anticiper un éventuel bug en complément des tests que l'on peut réaliser progressivement pendant la construction du script.

Si l'on reprend l'exemple précédent :

```
Remove-ADGroup -Identity "Direction" -Confirm:$false -WhatIf
```

Dans la console, on obtiendra la sortie suivante :

WhatIf : Opération « Remove » en cours sur la cible « CN=Direction,OU=Groupes,DC=it-connect,DC=local ».

Puisque l'on est rassuré, on peut exécuter de nouveau la commande sans -WhatIf pour procéder à la suppression.

Ce chapitre est terminé, bien souvent lorsque l'on souhaite modifier un groupe de sécurité ou de distribution, c'est pour gérer sa description ou le modifier. Enfin, l'action la plus courante reste la gestion des membres. La fin de ce chapitre sonne également la fin de ce module dédié à la gestion des groupes Active Directory via PowerShell. On se retrouve pour le prochain module dédié à la collecte d'informations sur l'annuaire Active Directory en lui-même.

Module 4 - Analyser et maintenir l'Active Directory

1. Obtenir des informations sur les contrôleurs de domaine

Au sein du module Active Directory de PowerShell, Microsoft a introduit un cmdlet nommé « Get-ADDomainController » et il s'avère excellent pour obtenir des informations au sujet des contrôleurs de domaine. Dans ce chapitre, nous allons le manipuler afin de récupérer différentes informations au sujet des contrôleurs de domaine.

I. Lister les contrôleurs de domaine avec PowerShell

Pour commencer, nous pouvons exécuter la commande sans filtre, simplement en demandant un affichage en mode tableau :

```
Get-ADDomainController -Filter * | Format-Table
```

Cette commande retourne de nombreuses propriétés. Je vous propose d'en sélectionner quelques-unes : le nom, l'adresse IP, le système d'exploitation, catalogue global (oui ou non) et le nom du site auquel est associé le contrôleur de domaine. Une commande intéressante pour faire un état des lieux rapide de ses contrôleurs de domaine !

```
Get-ADDomainController -Filter *| Select-Object Name, ipv4Address, OperatingSystem, IsGlobalCatalog, site
```

Voici le résultat retourné dans mon cas :

```
PS C:\> Get-ADDomainController -Filter *| Select-Object Name, ipv4Address, OperatingSystem, IsGlobalCatalog, site


Name            : SRV-ADDS-01
ipv4Address     : 192.168.1.10
OperatingSystem : Windows Server 2016 Standard
IsGlobalCatalog : True
site            : Default-First-Site-Name
```

Cette commande très générique peut être affinée pour rechercher un contrôleur de domaine en particulier. Par exemple, pour rechercher le contrôleur de domaine « SRV-ADDS-01 » :

```
Get-ADDomainController -Filter { name -eq "SRV-ADDS-01"}
```

En fait, on pourrait également rechercher tous les contrôleurs de domaine qui ont le nom qui commence par « SRV-ADDS » en utilisant l'opérateur -like à la place de -eq :

```
Get-ADDomainController -Filter { name -like "SRV-ADDS*"}
```

Vous n'êtes pas sans savoir que les contrôleurs de domaine peuvent être répartis entre plusieurs sites Active Directory. Grâce à ce cmdlet, nous pouvons lister les contrôleurs de domaine qui appartiennent à un site spécifique. Par exemple, avec le site par défaut « Default-First-Site-Name » :

```
Get-ADDomainController -Discover -ForceDiscover -Site "Default-First-Site-Name"
```

Le paramètre -Discover sert à réaliser une découverte de l'environnement Active Directory, il peut également être utilisé pour localiser un catalogue global :

```
Get-ADDomainController -Discover -Service "GlobalCatalog"
```

Dans le même esprit, mais directement à l'aide d'un filtre, nous pouvons afficher la liste des contrôleurs de domaine en lecture seule (RODC) grâce à un filtre sur la propriété « isReadOnly » :

```
Get-ADDomainController -Filter { IsReadOnly -eq $true }
```

Pour finir avec cette commande, nous pouvons également exploiter cette commande pour compter le nombre de contrôleurs de domaine que contient ce domaine.

```
(Get-ADDomainController -Filter * | Measure-Object).Count
```

II. Vérifier la disponibilité des contrôleurs de domaine

À l'aide de quelques lignes de code en PowerShell, nous pouvons vérifier la disponibilité des contrôleurs de domaine. L'objectif : s'assurer que tous les contrôleurs de domaine sont joignables.

Nous allons réutiliser la commande Get-ADDomainController que nous allons coupler à l'utilisation de Test-NetConnection. Dans cet exemple, j'utilise le port 389 correspondant au LDAP pour vérifier la connexion, mais il est tout à fait possible d'utiliser le port TCP/636 pour le LDAPS.

Ci-dessous, le bout de code qui permettra de vérifier la disponibilité des contrôleurs de domaine :

```
ForEach($DC in (Get-ADDomainController -Filter *))
{
    $NetConnectionResult = Test-NetConnection -ComputerName $DC.Hostname -Port 389
-InformationLevel Quiet

    if ($NetConnectionResult -ne $True){
        Write-Host "$($DC.Hostname) indisponible !" -ForegroundColor Red
    }else {
        Write-Host "$($DC.Hostname) disponible !" -ForegroundColor Green
    }
}
```

Une ligne sera générée pour chaque contrôleur de domaine :

```
PS C:\>
ForEach($DC in (Get-ADDomainController -Filter *))
{
    $NetConnectionResult = Test-NetConnection -ComputerName $DC.Hostname -Port 389 -Inf

    if ($NetConnectionResult -ne $True){
        Write-Host "$($DC.Hostname) indisponible !" -ForegroundColor Red
    }else {
        Write-Host "$($DC.Hostname) disponible !" -ForegroundColor Green
    }
}

SRV-ADDS-01.IT-CONNECT.LOCAL disponible !

PS C:\>
```

2. Lister les rôles FSMO en PowerShell

Lorsqu'il s'agit de lister les rôles FSMO afin d'obtenir le nom du contrôleur de domaine qui détient chacun des rôles, il y a plusieurs méthodes possibles. En effet, il est possible d'utiliser l'interface graphique à l'aide des outils d'administration, mais aussi d'utiliser l'outil ntdsutil en ligne de commande. De son côté, PowerShell offre une alternative supplémentaire et très pratique : découvrons dans ce chapitre comment obtenir des informations au sujet des rôles FSMO.

Le module Active Directory de PowerShell 4.0 intègre plusieurs cmdlets intéressants, dont deux qui nous permettent de récupérer facilement le maître d'opération pour chacun des rôles FSMO.

Pour rappel, il existe 5 rôles FSMO différents ayant chacun un maître d'opération unique, deux d'entre eux (Maître de schéma - Maître d'attribution des noms de domaine) au niveau de la forêt, les trois autres au niveau du domaine (Émulateur PDC - Maître RID - Maître d'infrastructure). Pour plus d'informations à ce sujet, je vous invite à lire mon cours sur les notions de base de l'Active Directory.

Voici le lien vers le chapitre : https://www.it-connect.fr/chapitres/les-cinq-roles-fsmo/

Passons à la manipulation de PowerShell, commencez par ouvrir une console. Nous devons utiliser deux cmdlets différents, car il y en a un qui va permettre de récupérer des informations au sujet des rôles FSMO uniques au niveau du domaine et un autre pour les deux rôles FSMO uniques au niveau de la forêt.

Pour lister les maîtres d'opérations définis au niveau de la forêt, on utilisera la commande Get-ADForest où l'on indiquera uniquement deux propriétés à afficher :

- **Maître de schéma** = SchemaMaster

- **Maître d'attribution des noms de domaine** = DomainNamingMaster

Ce qui donnera la commande suivante. Il n'est pas nécessaire de préciser le domaine si on exécute la commande directement depuis le contrôleur de domaine :

```
Get-ADForest | Format-Table SchemaMaster,DomainNamingMaster
```

L'information s'affiche tout naturellement à l'écran :

```
PS C:\> Get-ADForest | Format-Table SchemaMaster,DomainNamingMaster

SchemaMaster                 DomainNamingMaster
------------                 ------------------
SRV-ADDS-01.IT-CONNECT.LOCAL SRV-ADDS-01.IT-CONNECT.LOCAL
```

Concernant les 3 derniers rôles FSMO, on passera par l'utilisation de Get-ADDomain à la place de Get-ADForest, le principe de la commande reste le même.

- **Émulateur PDC** = PDCEmulator

- **Maître RID** = RIDMaster

- **Maître d'infrastructure** = InfrastructureMaster

Finalement, il suffit de remplacer Get-ADForest par Get-ADDomain et d'afficher les trois propriétés qui nous intéressent dans un tableau :

```
Get-ADDomain | Format-Table PDCEmulator,RIDMaster,InfrastructureMaster
```

Dans le cas où il y a plusieurs domaines dans la forêt, vous pouvez spécifier le nom du domaine cible de cette façon :

```
Get-ADDomain it-connect.local | Format-Table PDCEmulator,RIDMaster,
InfrastructureMaster
```

De mon côté, cela donne le résultat suivant :

```
PS C:\> Get-ADDomain | Format-Table PDCEmulator,RIDMaster,InfrastructureMaster

PDCEmulator                  RIDMaster                    InfrastructureMaster
-----------                  ---------                    --------------------
SRV-ADDS-01.IT-CONNECT.LOCAL SRV-ADDS-01.IT-CONNECT.LOCAL SRV-ADDS-01.IT-CONNECT.LOCAL


PS C:\> Get-ADDomain it-connect.local | Format-Table PDCEmulator,RIDMaster,InfrastructureMaster

PDCEmulator                  RIDMaster                    InfrastructureMaster
-----------                  ---------                    --------------------
SRV-ADDS-01.IT-CONNECT.LOCAL SRV-ADDS-01.IT-CONNECT.LOCAL SRV-ADDS-01.IT-CONNECT.LOCAL
```

Pour obtenir directement le nom du serveur qui détient le rôle, pour le stocker dans une variable par exemple, vous pouvez utiliser cette méthode :

```
(Get-ADForest it-connect.local).SchemaMaster
(Get-ADForest it-connect.local).DomainNamingMaster
(Get-ADDomain it-connect.local).PDCEmulator
(Get-ADDomain it-connect.local).RIDMaster
(Get-ADDomain it-connect.local).InfrastructureMaster
```

Chaque commande ci-dessus va vous retourner le nom du contrôleur de domaine qui détient le rôle FSMO concerné.

L'idée est simple : on peut imaginer un script où l'on définit la valeur que doit avoir chaque maître d'opération, et, qu'on la compare à la valeur récupérée en production avec les commandes ci-dessus. Grâce à cette méthode, ce sera facile de voir s'il y a eu une modification du « maître » pour l'un des rôles FSMO. Dans ce cas, il faudra générer une alerte par e-mail ou dans votre système de supervision.

Ce script dont je faisais référence à la page précédente, et celui-ci :

```
# Valeurs cibles
$FSMOSchemaMaster = "SRV-ADDS-01.it-connect.local"
$FSMODomainNamingMaster = "SRV-ADDS-01.it-connect.local"
$FSMOPDCEmulator = "SRV-ADDS-01.it-connect.local"
$FSMORIDMaster = "SRV-ADDS-01.it-connect.local"
$FSMOInfrastructureMaster = "SRV-ADDS-01.it-connect.local"

# Fonction pour envoyer un e-mail en cas de changement
function Send-EmailAlert{

    param($Role,
          $ValeurCible,
          $ValeurProd)

    $SMTPServer = "serveur.smtp.fr"
    $SMTPFrom = "expediteur@domaine.fr"
    $SMTPTo = "destinataire@domaine.fr"

    Send-MailMessage -SmtpServer $SMTPServer -From $SMTPFrom -To $SMTPTo `
                     -Subject "Active Directory - Alerte FSMO $Role!" `
                     -Body "Alerte ! Changement de propriétaire pour le rôle FSMO
$Role : $ValeurProd au lieu de $ValeurCible"

}

# Vérifier Role FSMO Schema Master
if((Get-ADForest it-connect.local).SchemaMaster -ne $FSMOSchemaMaster){
    Send-EmailAlert -Role "SchemaMaster" -ValeurCible $FSMOSchemaMaster -ValeurProd
(Get-ADForest it-connect.local).SchemaMaster
}

# Vérifier Role FSMO Domain Naming Master
if((Get-ADForest it-connect.local).DomainNamingMaster -ne $FSMODomainNamingMaster){
    Send-EmailAlert -Role "DomainNamingMaster" -ValeurCible $FSMODomainNamingMaster
-ValeurProd (Get-ADForest it-connect.local).DomainNamingMaster
}

# Vérifier Role FSMO PDC Emulator
if((Get-ADDomain it-connect.local).PDCEmulator -ne $FSMOPDCEmulator){
    Send-EmailAlert -Role "PDCEmulator" -ValeurCible $FSMOPDCEmulator -ValeurProd
(Get-ADDomain it-connect.local).PDCEmulator
}

# Vérifier Role FSMO RID Master
if((Get-ADDomain it-connect.local).RIDMaster -ne $FSMORIDMaster){
    Send-EmailAlert -Role "RIDMaster" -ValeurCible $FSMORIDMaster -ValeurProd (Get-
ADDomain it-connect.local).RIDMaster
}

# Vérifier Role FSMO Infrastructure Master
if((Get-ADDomain it-connect.local).InfrastructureMaster -ne
$FSMOInfrastructureMaster){
    Send-EmailAlert -Role "InfrastructureMaster" -ValeurCible
$FSMOInfrastructureMaster -ValeurProd (Get-ADDomain it-
connect.local).InfrastructureMaster
}
```

On vérifie la valeur actuelle de chaque rôle FSMO : en cas de différence avec la valeur attendue, un e-mail sera envoyé. Il faut adapter les 5 variables en début de script ainsi que les informations pour envoyer l'e-mail (serveur SMTP, expéditeur et destinataire).

3. Troubleshooting de la réplication Active Directory

Il y a plusieurs façons de réaliser un troubleshooting de la réplication Active Directory, à commencer par l'utilitaire en ligne de commande « repadmin ». Disponible depuis Windows Server 2003, il reste un outil pertinent aujourd'hui bien que PowerShell soit capable désormais d'afficher des informations pertinentes à ce sujet. Par ailleurs, l'outil graphique Active Directory Replication Status Tool est également intéressant.

Depuis Windows Server 2012, Microsoft offre aux administrateurs systèmes d'obtenir des informations sur la réplication Active Directory à l'aide de PowerShell. Comme pour le reste, c'est le module Active Directory qui apporte cette fonctionnalité.

Découvrons ensemble quelques cmdlets intéressants au sujet de la réplication Active Directory, que ce soit pour analyser l'état de la réplication Active Directory ou réaliser un troubleshooting.

I. Get-ADReplicationFailure

Cette première commande sert à vérifier si l'un de vos contrôleurs de domaine a une erreur de réplication, en indiquant son nom.

```
Get-ADReplicationFailure -Target <controleur-de-domaine>
```

Si la commande ne retourne pas de résultat, c'est qu'il n'y a pas d'erreurs de réplication pour le contrôleur de domaine en question. Sinon, le résultat contiendra le nom d'échec (*FailureCount*), le type d'échec (*FailureType*), la date et l'heure du premier échec (*FirstFailureTime*), ainsi que le serveur et le partenaire.

En regardant de plus près cette commande, elle contient un paramètre nommé « Scope » et qui permet d'avoir un périmètre plus ou moins large pour vérifier les erreurs de réplication. Avec la valeur « SITE », on peut vérifier les erreurs de réplication concernant un site. Par exemple, avec le site de « CAEN » :

```
Get-ADReplicationFailure –Scope SITE –Target CAEN
```

Le paramètre -Scope peut avoir deux autres valeurs : « DOMAIN » pour cibler le domaine ou « FOREST » pour cibler la forêt. La syntaxe n'est pas plus compliquée :

```
Get-ADReplicationFailure –Scope DOMAIN –Target it-connect.local
```

Passons à la suite.

II. Get-ADReplicationPartnerMetadata

Cette commande très pratique sert à obtenir des informations sur les métadonnées des partenaires de réplication et **détecter une erreur de réplication au sein de votre forêt**. En complément de la commande précédente, elle permettra d'affiner votre debug et d'obtenir des informations supplémentaires.

Elle retourne notamment les propriétés suivantes :

- **LastChangeUsn** : obtenir l'USN (*Update Sequence Number*) du DC
- **LastReplicationAttempt** : date et heure de la dernière réplication
- **LastReplicationResult** : résultat de la dernière réplication
- **LastReplicationSuccess** : date et heure de la dernière réplication réussie

Pour l'exécuter sur un contrôleur de domaine cible, la syntaxe est la suivante :

```
Get-ADReplicationPartnerMetadata -Target <nom-du-controleur-de-domaine>
```

Vous verrez qu'elle retourne de nombreuses informations, y compris les 4 propriétés citées précédemment. Maintenant, allons droit au but et utilisons cette commande pour regarder s'il y a un contrôleur de domaine qui a des problèmes de réplication :

```
Get-ADReplicationPartnerMetadata –Target * -Scope Server | `
      Where {$_.LastReplicationResult –ne "0"} | `
      Format-Table Server, LastReplicationAttempt, LastReplicationResult, Partner,
      Site
```

Cette commande va interroger tous les contrôleurs de domaine puisque nous indiquons la valeur « * » pour le paramètre -Target. En complément, nous ajoutons un filtre sur « LastReplicationResult » afin de prendre en compte seulement le cas où il est différent de « 0 ». Autrement dit, si le résultat est différent de « 0 », c'est qu'il y a une erreur !

La commande va retourner un tableau avec plusieurs valeurs : le nom du serveur, la date et l'heure de la dernière tentative de réplication, le résultat de la dernière réplication, le nom du partenaire avec lequel la réplication a échoué, ainsi que le site Active Directory du contrôleur de domaine.

Une bonne base pour orienter le troubleshooting dans le cas où il y a un incident, comme celui-ci par exemple :

```
PS C:\> Get-ADReplicationPartnerMetadata -Target * -Scope Server | where {$_.LastReplicationResult -ne "0"} | Format-Tab
le Server, LastReplicationAttempt, LastReplicationResult, Partner

Server                 LastReplicationAttempt LastReplicationResult Partner
------                 ---------------------- -------------------- -------
[redacted]             03/11/2020 13:20:23                  1722 CN=NTDS Settings,CN=[redacted]
```

Pour consulter la liste des opérations de réplication en attente sur un contrôleur de domaine, le cmdlet « Get-ADReplicationQueueOperation » doit être utilisé. Il suffit de préciser un nom de DC :

```
Get-ADReplicationQueueOperation -Server SRV-ADDS-01
```

Si la commande retourne un résultat vide, c'est qu'il n'y a rien dans la file d'attente pour ce DC.

III. Get-ADReplicationAttributeMetadata

Pour obtenir des informations quant à la réplication d'un objet spécifique de votre annuaire, il est nécessaire d'utiliser la commande Get-ADReplicationAttributeMetadata.

Pour chaque attribut de cet objet, on obtient différentes informations, notamment : le nom de l'attribut, sa valeur, le dernier DC depuis lequel il a était modifié, la date et l'heure de la modification, ainsi que son USN local sur le serveur et l'USN suite au changement, ce qui permettra de voir s'il est à jour ou pas.

Pour utiliser cette commande, il faut spécifier le DistinguishedName de l'objet et le contrôleur de domaine à interroger. Par exemple :

```
Get-ADReplicationAttributeMetadata -Object "CN=Florian,OU=Personnel,DC=it-
connect,DC=local" -Server SRV-ADDS-01 -ShowAllLinkedValues
```

Cette commande sera particulièrement intéressante si vous rencontrez des problèmes de réplication sur un objet spécifique.

IV. Informations sur les sites Active Directory

En complément des commandes que nous venons de voir, sachez qu'il existe d'autres commandes pour obtenir des informations sur les sites Active Directory.

Ces commandes sont simples à utiliser pour obtenir des informations :

Commande	Description
Get-ADReplicationSite -Filter *	Obtenir la liste des sites
Get-ADReplicationSiteLink -Filter *	Obtenir la liste des liens de site pour la forêt
Get-ADReplicationSiteLinkBridge -Filter *	Obtenir la liste des ponts de site (bridge)
Get-ADReplicationSubnet -Filter *	Obtenir la liste des sous-réseaux déclarés

Si vous rencontrez des problèmes liés à la réplication Active Directory, toutes ces commandes PowerShell vont vous rendre le troubleshooting plus facile grâce aux informations précieuses qu'elles fournissent.

4. Obtenir des informations sur les ordinateurs avec PowerShell

Sur le même principe que Get-ADUser, le cmdlet Get-ADComputer permet de récolter des informations sur les ordinateurs intégrés à l'annuaire Active Directory. Cette commande va permettre d'obtenir toutes les informations au sujet des objets de type « ordinateurs » de votre annuaire Active Directory, que ce soit les PC ou les serveurs.

I. Get-ADComputer : nom, OS et adresse IP

Il y a une commande que j'affectionne tout particulièrement, c'est celle-ci :

```
Get-ADComputer -Filter * -Properties Name,OperatingSystem,IPv4Address | Select-Object Name,OperatingSystem,IPv4Address
```

Son résultat est très intéressant puisqu'elle va permettre d'afficher une liste de tous les ordinateurs (et serveurs) intégrés au domaine, en indiquant trois informations :

- Le nom de la machine via la propriété « **Name** »
- Le système d'exploitation via la propriété « **OperatingSystem** »
- L'adresse IP utilisée via la propriété « **IPv4Address** »

```
PS C:\> Get-ADComputer -Filter * -Properties Name,OperatingSystem,IPv4Address | Select-Object Name,OperatingSystem,IPv4Address

Name        OperatingSystem             IPv4Address
----        ---------------             -----------
SRV-ADDS-01 Windows Server 2019 Standard 172.31.76.108
PC-W10      Windows 10 Éducation        192.168.1.204
SRV-WS-01   Windows Server 2019 Standard 192.168.1.51
```

L'adresse IP ne correspond pas à un attribut de l'annuaire Active Directory directement. En fait, cette propriété s'appuie sur les enregistrements DNS pour retrouver cette information.

Dans le même esprit, on peut obtenir l'adresse IPv6 de la machine (sous réserve que ce soit utilisé) en recherchant la propriété **IPv6Address**.

Si l'on veut affiner la requête, par exemple pour prendre uniquement toutes les machines dont le nom commence par « PC- », il faut ajouter un filtre basé sur la propriété « Name », comme ceci :

```
Get-ADComputer -Filter 'Name -like "PC-*"' -Properties Name,OperatingSystem,IPv4Address | Select-Object Name,OperatingSystem,IPv4Address
```

Pour la suite de ce chapitre, je vous propose de découvrir quelques requêtes intéressantes avec Get-ADComputer.

II. Combien est-ce qu'il y a de machines Windows 10 dans mon AD ?

Vous avez surement la réponse à cette question par l'intermédiaire de votre outil d'inventaire. Néanmoins, l'Active Directory peut également vous apporter une réponse à cette interrogation.

Tout d'abord, on peut lister les machines Windows 10 intégrées à l'AD avec cette commande :

```
Get-ADComputer -Filter {OperatingSystem -like '*Windows 10*'}
```

Ensuite, nous allons coupler cette commande à Measure-Object pour compter le nombre d'éléments retourné.

```
(Get-ADComputer -Filter {OperatingSystem -like '*Windows 10*'} | Measure-Object).Count
```

De la même façon, on peut compter le nombre total d'objets « ordinateurs » dans l'annuaire en retirant le filtre sur l'OS :

```
(Get-ADComputer -Filter * | Measure-Object).Count
```

Si l'on repart de notre filtre précédent, on peut **l'affiner pour rechercher uniquement les machines Windows 10 actives sur les 6 derniers mois.** Pour cela, nous devons ajouter une condition Where basée sur l'attribut « LastLogonDate » à savoir la date de dernière connexion de l'ordinateur au domaine.

```
Get-ADComputer -Filter {OperatingSystem -like '*Windows 10*'} -Properties
lastLogonDate | where { $_.lastLogonDate -gt (Get-Date).AddMonths(-6) }
```

Si vous souhaitez prendre une période plus longue, par exemple 12 mois, remplacez « -6 » par « -12 » pour prendre une date de référence correspondant à : la date du jour – 12 mois.

Bien sûr, cette méthode s'applique également aux autres systèmes d'exploitation, y compris Windows Server : il suffit d'ajuster le filtre.

III. Quelle est la date de dernière connexion au domaine d'une machine ?

Reprenons la propriété « *LastLogonDate* » évoquée précédemment. Elle correspond à un attribut du même nom présent au sein de chaque objet de type « ordinateurs ».

Si l'on veut obtenir une liste avec le nom des machines et la date de dernière connexion, il suffira d'exécuter la commande suivante :

```
Get-ADComputer -Filter * -Properties * | ft Name, LastLogonDate
```

```
PS C:\> Get-ADComputer -Filter * -Properties * | FT Name, LastLogonDate

Name        LastLogonDate
----        -------------
SRV-ADDS-01 19/04/2021 09:39:15
PC-W10      19/04/2021 09:45:50
SRV-WS-01   02/04/2021 13:13:27
```

C'est la fin de ce chapitre ! Dans le prochain chapitre, nous allons nous approfondir la notion d'objets inactifs, que ce soit pour les utilisateurs ou les ordinateurs.

5. Utiliser PowerShell pour identifier les ordinateurs et utilisateurs inactifs

On crée des utilisateurs, on ajoute des postes de travail et des serveurs au domaine, mais l'opération inverse est moins évidente : est-ce que l'on pense à nettoyer son annuaire Active Directory ? Notamment, afin d'identifier et de gérer les ordinateurs et les utilisateurs inactifs. D'une part, ceci permettra d'avoir un annuaire à jour, et d'autre part ceci évite de laisser des comptes actifs s'ils ne sont plus utilisés : un risque en termes de sécurité.

Dans son document « *Recommandations de sécurité relatives à Active Directory* », l'ANSSI recommande d'appliquer les règles suivantes pour la gestion des comptes inactifs, notamment pour vos utilisateurs :

- Définir une politique de désactivation automatique des comptes non utilisés pendant un certain laps de temps
- Conserver les comptes désactivés et inutilisés afin d'avoir un historique des comptes utilisateurs au sein du système d'information
- Placer les comptes désactivés dans une unité d'organisation (« OU ») spécifique
- Retirer les droits et privilèges des comptes désactivés

Passons maintenant à la pratique pour rechercher les objets inactifs, puis dans un second temps nous verrons comment gérer ces objets.

I. Comment identifier les utilisateurs inactifs ?

Pour atteindre notre objectif, on va s'appuyer sur le cmdlet nommé "*Search-ADaccount*". Il va permettre de rechercher les objets inactifs au sein de l'annuaire Active Directory, que ce soit les utilisateurs, mais également les ordinateurs.

Dans ce cmdlet, Le paramètre *Timespan* sert à indiquer un nombre de jours, par exemple si l'on indique 180 jours (6 mois) : la requête nous retournera les objets inactifs depuis au moins 180 jours. Grâce à ce cmdlet, nous allons pouvoir obtenir la liste des utilisateurs inactifs assez facilement.

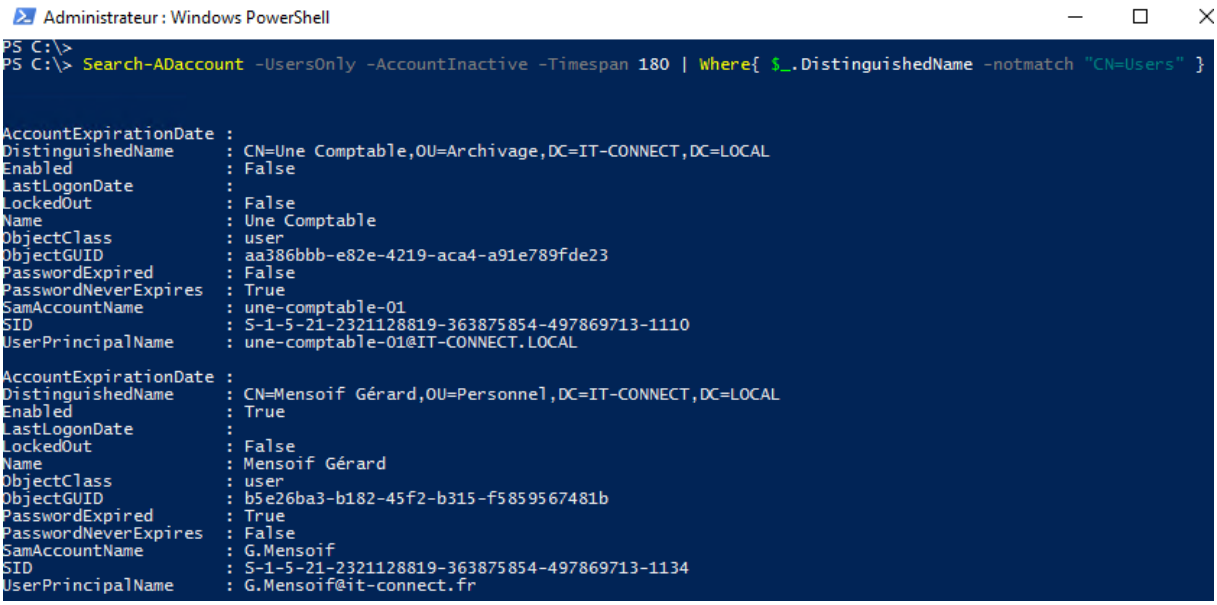
Pour rechercher les comptes utilisateurs inactifs depuis 180 jours :

```
Search-ADaccount -UsersOnly -AccountInactive -Timespan 180
```

Attention, les comptes qui ne se sont jamais connectés seront considérés comme étant inactifs. Autrement dit, un compte qui vient d'être créé sera considéré comme inactif.

Cette commande va probablement retourner certains comptes built-in comme étant inactifs, par exemple le compte "Invité" ou le compte "krbtgt" qui sert à distribuer les tickets Kerberos. Pour éviter une mésaventure, il peut être intéressant d'exclure de la recherche l'OU "Users". Ce qui nous donne :

```
Search-ADaccount -UsersOnly -AccountInactive -Timespan 180 | Where{
$_.DistinguishedName -notmatch "CN=Users" }
```



Ah oui, au fait, j'allais oublier : nous allons ajouter une autre condition dans la clause Where pour récupérer uniquement les comptes activés dans l'annuaire. Il n'y a pas d'intérêt à ce que la commande retourne les comptes désactivés, car il y a des chances qu'ils soient déjà gérés.

Voici la commande mise à jour pour récupérer seulement les comptes actifs :

```
Search-ADaccount -UsersOnly -AccountInactive -Timespan 180 | Where{
($_.DistinguishedName -notmatch "CN=Users") -and ($_.Enabled -eq $true) }
```

Maintenant que l'on est en mesure d'identifier les utilisateurs inactifs, faisons de même avec les ordinateurs.

II. Comment identifier les ordinateurs inactifs ?

Sur le même principe que pour les utilisateurs, nous allons utiliser le cmdlet « **Search-ADaccount** ». La différence, c'est **que le paramètre -UsersOnly sera remplacé par -ComputersOnly** pour s'intéresser aux objets ordinateurs.

Pour rechercher les comptes ordinateurs inactifs depuis 180 jours :

```
Search-ADaccount -ComputersOnly -AccountInactive -Timespan 180
```

Cette commande retournera également les éventuels gMSA (*Group Managed Service Account*) inactifs bien qu'ils aient une classe différente au niveau de l'Active Directory : « msDS-GroupManagedServiceAccount » contre la classe « computer » pour les objets ordinateurs.

Ce qui m'amène à vous suggérer d'appliquer un filtre supplémentaire sur la commande pour conserver seulement les objets avec la classe « computer », comme ceci :

```
Search-ADaccount -ComputersOnly -AccountInactive -Timespan 180 | Where{
$_.objectClass -eq "computer" }
```

Cette commande vous donnera la liste des ordinateurs inactifs depuis 180 jours ou plus. Quand je dis « ordinateurs », je fais référence aux postes de travail, aux serveurs et aux contrôleurs de domaines puisqu'ils s'appuient tous sur la classe « computer ».

III. Les options complémentaires de Search-ADAccount

Le cmdlet **Search-ADAccount**, que nous utilisons depuis le début de ce chapitre dispose d'autres paramètres qui vont permettre d'identifier les comptes dans différents états, et pas seulement les comptes inactifs.

Voici des informations au sujet de ces paramètres que je vous invite à tester sur le même principe que -AccountInactive :

Nom du paramètre	Description
-AccountExpired	Identifier les comptes expirés, c'est-à-dire les comptes qui ont une date d'expiration définie et qu'elle est passée.
-AccountDisabled	Identifier les comptes désactivés.
-AccountExpiring	Identifier les comptes qui vont expirer dans les X prochaines heures ou à partir d'une date spécifique. Implique l'utilisation de -DateTime ou -TimeSpan en complément.
-LockedOut	Identifier les comptes bloqués, suite à trop de tentatives de connexion avec un mauvais mot de passe, par exemple.
-PasswordExpired	Identifier les comptes dont le mot de passe est expiré.
-PasswordNeverExpires	Identifier les comptes où le mot de passe n'expire jamais.

IV. Search-ADAccount : comment exclure les nouveaux objets ?

Lorsque l'on utilise le cmdlet Search-ADAccount avec le paramètre -AccountInactive, on risque d'avoir des faux positifs. En effet, si l'on vient de créer un compte pour un utilisateur et que l'on exécute cette commande, le compte sera considéré comme étant inactif. L'attribut « lastLogon » sera vide, car l'utilisateur ne s'est pas encore connecté, ce qui explique ce cas de figure.

Pour éviter les faux-positifs, il faut que l'on effectue un filtre complémentaire sur la commande Search-ADAccount. Malheureusement, la commande ne remonte pas la propriété « whenCreated » des objets : dommage, cela aurait pu nous simplifier la tâche pour détecter si un utilisateur a été créé récemment ou non.

Nous allons nous adapter et faire usage d'une boucle Foreach pour filtrer le résultat de la commande Search-ADAccount. Ce qui donne le bout de code suivant pour exclure aussi bien les nouveaux utilisateurs que les nouveaux ordinateurs.

```
$InactivesObjects = Search-ADaccount -AccountInactive -Timespan 180 | Where{
($_.DistinguishedName -notmatch "CN=Users") -and ($_.Enabled -eq $true) } |
foreach{

        if(($_.objectClass -eq "user") -and (Get-ADUser -Filter "Name -eq
'$($_.Name)'" -Properties WhenCreated).WhenCreated -lt (Get-Date).AddDays(-7)){ $_
}
        if(($_.objectClass -eq "computer") -and (Get-ADComputer -Filter "Name -eq
'$($_.Name)'" -Properties WhenCreated).WhenCreated -lt (Get-Date).AddDays(-7)){ $_
}
}
```

Dans l'exemple ci-dessus, on fait une exclusion des comptes créés sur les 7 derniers jours. Pour ajuster ce seuil, il suffit de modifier la valeur de la méthode « AddDays(-7) ». Remplacez le « 7 » par le chiffre que vous souhaitez mais veillez à bien laisser le signe moins devant, il est nécessaire pour soustraire X jours à la date actuelle.

V. Retirer l'utilisateur des groupes dont il est membre

Dans la suite de ce chapitre, je vais m'intéresser plus particulièrement aux utilisateurs afin de vous aider à appliquer les recommandations de l'ANSSI.

Ainsi, maintenant que nous sommes en capacité de récupérer la liste des objets inactifs, nous devons le supprimer des groupes auxquels il appartient. Prenons l'exemple d'un utilisateur nommé "*une-comptable-01*" et qui appartient à deux groupes : "*Comptabilité*" et "*Utilisateurs du domaine*". Nous allons lui laisser seulement le groupe par défaut, à savoir le groupe « Utilisateurs du domaine ».

Commençons par stocker le SamAccountName de l'utilisateur dans une variable :

```
$SamAccountName = "ma-comptable-01"
```

Puis, nous allons le retirer des groupes grâce à la commande ci-dessous :

```
Get-AdPrincipalGroupMembership -Identity $SamAccountName | where-Object { $_.Name -
Ne "Utilisateurs du domaine" } | Remove-AdGroupMember -Members $SamAccountName -
Confirm:$false
```

Nous ne demanderons pas de confirmation pour plus de simplicité. Le tour est joué !

VI. Désactiver l'utilisateur et le déplacer

L'utilisateur est identifié, il n'est plus membre des groupes spécifiques auxquels il appartenait, il est temps désormais de le désactiver. Pour réaliser cette action, la commande Set-ADUser sera bien utile :

```
Set-ADUser -Identity $SamAccountName -Enabled:$false -Description "Désactivé le
$(Get-Date -Format dd/MM/yyyy)"
```

Profitons-en pour mettre à jour la description de cet utilisateur en intégrant dans l'attribut "description", la date à laquelle nous l'avons désactivé afin d'avoir une traçabilité 👍

Il ne reste plus qu'à isoler cet utilisateur dans une unité d'organisation dédiée aux comptes inactifs, que l'on pourrait qualifier de comptes archivés.

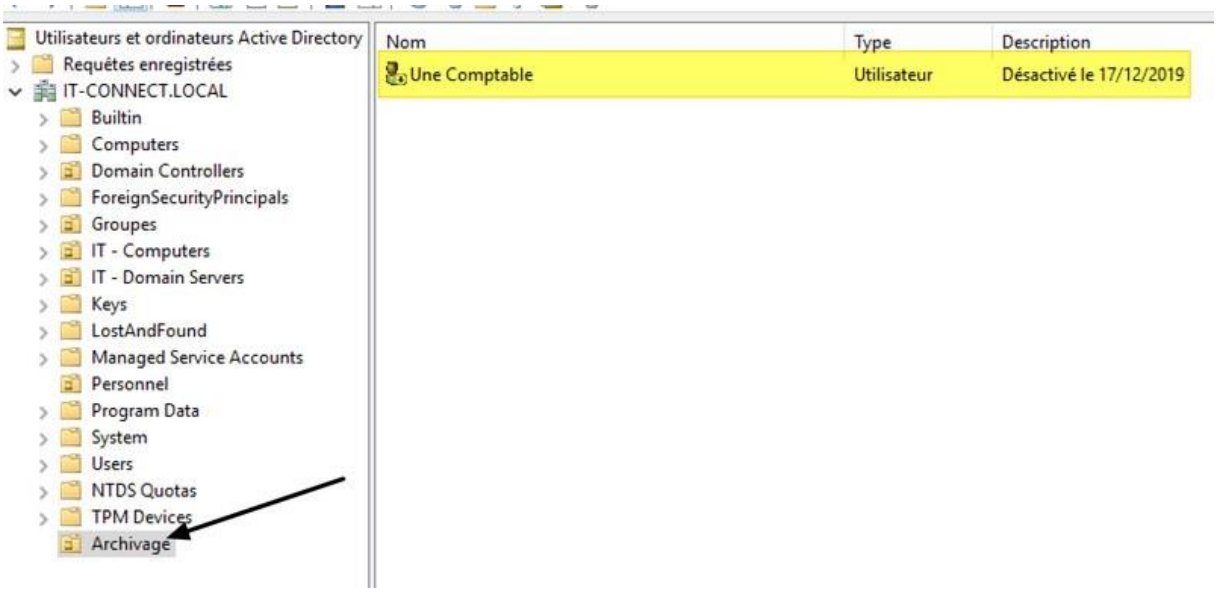
Avec le cmdlet *Move-ADObject*, nous allons utiliser le *DistinguishedName* de l'utilisateur, via la variable $DN. *Cette valeur sera récupérée automatiquement dans le script final que je vous propose ci-dessous.*

Par exemple :

```
$DN = "CN=Une Comptable,OU=Personnel,DC=IT-CONNECT,DC=LOCAL"
```

On déplace l'utilisateur vers l'OU *"OU=Archivage,DC=IT-CONNECT,DC=LOCAL" :*

```
Move-ADObject -Identity "$DN" -TargetPath "OU=Archivage,DC=IT-CONNECT,DC=LOCAL"
```



VII. Script de gestion des comptes inactifs

Pour finir sur ce chapitre et maintenant que nous avons vu les différentes actions à réaliser traduites en commandes PowerShell, on peut regrouper tout cela dans un script afin de se simplifier la tâche.

Dans le but d'avoir un seul script qui gère aussi bien les ordinateurs que les utilisateurs inactifs, il faut penser à ne pas indiquer *UsersOnly* ou *ComputersOnly* dans la commande *Search-ADAccount*.

Il faut également adapter certaines commandes, en fonction de s'il s'agit un objet utilisateur ou ordinateur à traiter. Pour cela, on va se baser sur l'attribut *ObjectClass* qui nous indique si l'objet est de type *user* ou *computer*.

```
$InactivesObjects = Search-ADaccount -AccountInactive -Timespan 180 | Where{
($_.DistinguishedName -notmatch "CN=Users") -and ($_.Enabled -eq $true) } |
foreach{

        if(($_.objectClass -eq "user") -and (Get-ADUser -Filter "Name -eq
'$($_.Name)'" -Properties whenCreated).whenCreated -lt (Get-Date).AddDays(-7)){ $_
}
        if(($_.objectClass -eq "computer") -and (Get-ADComputer -Filter "Name -eq
'$($_.Name)'" -Properties whenCreated).whenCreated -lt (Get-Date).AddDays(-7)){ $_
}
}

Foreach($Object in $InactivesObjects){

    $SamAccountName = $Object.SamAccountName
    $DN = $Object.DistinguishedName
    $ObjectClass = $Object.ObjectClass

    Write-Output "L'objet $SamAccountName est inactif !"

    # Si c'est un utilisateur...
    if($ObjectClass -eq "user"){

      # Retirer l'utilisateur des groupes (sauf "Utilisateurs du domaine")
      Get-AdPrincipalGroupMembership -Identity $SamAccountName | Where-Object {
$_.Name -Ne "Utilisateurs du domaine" } | Remove-AdGroupMember -Members
$SamAccountName -Confirm:$false -ErrorVariable ClearObject

      # Désactiver l'utilisateur
      Set-ADUser -Identity $SamAccountName -Enabled:$false -Description "Désactivé
le $(Get-Date -Format dd/MM/yyyy)" -ErrorVariable +ClearObject

    # Sinon, si c'est un ordinateur...
    }elseif($ObjectClass -eq "computer"){

      # Retirer l'ordinateur des groupes (sauf "Ordinateurs du domaine")
      Get-AdPrincipalGroupMembership -Identity $SamAccountName | Where-Object {
$_.Name -Ne "Ordinateurs du domaine" } | Remove-AdGroupMember -Members
$SamAccountName -Confirm:$false -ErrorVariable ClearObject

      # Désactiver l'ordinateur
      Set-ADComputer -Identity $SamAccountName -Enabled:$false -Description
"Désactivé le $(Get-Date -Format dd/MM/yyyy)" -ErrorVariable +ClearObject
    }

    # Déplacer l'utilisateur/ordinateur
    Move-ADObject -Identity "$DN" -TargetPath "OU=Archivage,DC=IT-CONNECT,DC=LOCAL"
-ErrorVariable +ClearObject

    if($ClearUser){
      Write-Output "ERREUR ! objet concerné : $SamAccountName ($ObjectClass)"
    }else{
      Write-Output "Traitement de l'objet $SamAccountName de type $ObjectClass avec
succès ! :-)"
    }

    Clear-Variable ClearUser
}
```

Ce script est fonctionnel et à votre disposition pour l'adapter selon vos besoins, notamment pour gérer uniquement les ordinateurs, ou les ordinateurs, ou alors ajouter l'envoi d'une notification si un compte est basculé en état « inactif ».

Actuellement, le script renvoie une sortie comme celle-ci :

```
L'objet une-comptable-01 est inactif !
Traitement de l'objet une-comptable-01 de type user avec succès ! :-)
L'objet ORDINATEUR-01$ est inactif !
Traitement de l'objet ORDINATEUR-01$ de type computer avec succès ! :-)
```

Ce chapitre est terminé, il est temps de poursuivre dans la gestion des objets inutiles, inactifs, en s'intéressant de plus près à l'utilisation de la Corbeille Active Directory.

Référez-vous au script « EBOOK-ADDS-Module-4-Chapitre-5-01.ps1 » sur GitHub

6. Gérer la corbeille Active Directory avec PowerShell

La corbeille Active Directory est une fonctionnalité présente sous Windows Server depuis Windows Server 2008 R2. Au même titre que la corbeille de Windows, elle permet de récupérer des éléments supprimés, notamment des objets ordinateurs, utilisateurs ou des groupes. Une fonctionnalité indispensable pour se protéger contre les suppressions accidentelles, qui peuvent s'avérer désastreuses.

Cette fonctionnalité nécessite que le niveau fonctionnel de votre domaine et votre forêt soit au minimum « Windows Server 2008 R2 », sinon vous ne pourrez pas l'activer. L'occasion de vous communiquer deux nouvelles commandes pour obtenir à la fois le niveau fonctionnel du domaine et de la forêt :

```
(Get-ADDomain).DomainMode
(Get-ADForest).ForestMode
```

Dans ce chapitre, je vous propose d'apprendre à gérer la corbeille Active Directory à partir de PowerShell, que ce soit pour l'activer si elle ne l'est pas déjà, mais aussi pour restaurer un objet de la corbeille ainsi que pour la purger.

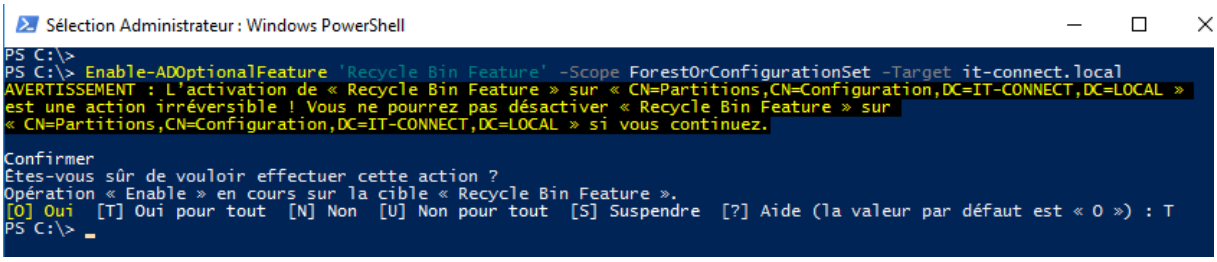
I. Activer la corbeille Active Directory

Le fait d'activer la corbeille Active Directory va altérer votre schéma Active Directory, ce qui implique que cette action est irréversible.

Ci-dessous, la commande à exécuter depuis votre contrôleur de domaine afin d'activer la corbeille. Il suffit de remplacer le domaine « it-connect.local » au sein du paramètre -Target, par votre domaine.

```
Enable-ADOptionalFeature 'Recycle Bin Feature' -Scope ForestOrConfigurationSet -
Target it-connect.local
```

Lors de l'exécution de la commande, vous devrez confirmer l'opération avec « T » ou « O ».



II. Restaurer un objet de la corbeille Active Directory

Afin de vous exercer, je vous invite à créer un utilisateur dans l'Active Directory puis à le supprimer directement : il va rejoindre la corbeille Active Directory. Pour ma part, il s'agit d'un utilisateur avec le « sAMAccountName » égal à « Florian ».

Une fois supprimé, nous pouvons lister le contenu de la corbeille Active Directory avec cette commande :

```
Get-ADObject -Filter 'isdeleted -eq $true' -IncludeDeletedObjects
```

Sur la copie d'écran ci-dessous, nous pouvons voir que l'utilisateur « florian » est bien supprimé. Il s'agit de l'utilisateur avec le DistinguishedName qui commence par « CN=Florian IT-Connect ». Nous remarquons également sa présence dans la corbeille puisqu'il est dans le container « Deleted Objects ».

```
PS C:\> Get-ADObject -Filter 'isdeleted -eq $true' -IncludeDeletedObjects


Deleted           : True
DistinguishedName : CN=Deleted Objects,DC=IT-CONNECT,DC=LOCAL
Name              : Deleted Objects
ObjectClass       : container
ObjectGUID        : 07abe11a-068e-44c4-ae06-7e74d17d1331

Deleted           : True
DistinguishedName : CN=Florian IT-Connect\0ADEL:2d6d4b4c-01c3-4201-85b9-83353efb3870,CN=Deleted
                    Objects,DC=IT-CONNECT,DC=LOCAL
Name              : Florian IT-Connect
                    DEL:2d6d4b4c-01c3-4201-85b9-83353efb3870
ObjectClass       : user
ObjectGUID        : 2d6d4b4c-01c3-4201-85b9-83353efb3870
```

Grâce aux commandes Get-ADObject et Restore-ADObject, il ne reste plus qu'à restaurer cet objet. Ci-dessous, la commande pour restaurer l'utilisateur « florian » :

```
Get-ADObject -Filter 'sAMAccountName -eq "florian"' -IncludeDeletedObjects |
Restore-ADObject
```

Ensuite, si l'on consulte le contenu de la corbeille Active Directory, on peut constater que l'utilisateur n'est plus dedans ! Une bonne nouvelle… Il est de retour dans l'Active Directory à son emplacement d'origine au moment de la suppression.

```
PS C:\> Get-ADObject -Filter 'samaccountname -eq "florian"' -IncludeDeletedObjects | Restore-ADObject
PS C:\> Get-ADObject -filter 'isdeleted -eq $true' -includeDeletedObjects


Deleted           : True
DistinguishedName : CN=Deleted Objects,DC=IT-CONNECT,DC=LOCAL
Name              : Deleted Objects
ObjectClass       : container
ObjectGUID        : 07abe11a-068e-44c4-ae06-7e74d17d1331


PS C:\> _
```

III. Purger la corbeille Active Directory

Désormais, je vais vous expliquer **comment purger la corbeille Active Directory** pour supprimer définitivement les éléments qui s'y trouvent. Cela va permettre d'effacer définitivement toutes les traces au sujet d'un compte utilisateur ou d'un compte ordinateur, par exemple. Comme nous l'avons vu dans le chapitre précédent, il est recommandé de conserver les comptes correspondants aux utilisateurs qui ont quitté votre entreprise, en appliquant des règles spécifiques (désactiver l'objet, notamment).

Note : par défaut, un objet sera définitivement supprimé après 180 jours passés dans la corbeille afin d'assurer un nettoyage régulier et éviter qu'elle cumule des objets indéfiniment. L'attribut *msDS-deletedObjectLifetime* contient cette valeur correspondante au « *Deleted Object Lifetime* » c'est-à-dire la durée de vie d'un objet supprimé.

Pour effectuer cette action sur la corbeille, le cmdlet "Get-ADObject" sera utile pour récupérer la liste des objets supprimés (et que l'on peut purger) en se basant sur deux critères :

- L'attribut **isDeleted égal à vrai (true)**
- **La présence de "DEL:" dans le nom de l'objet**.

Il faut également penser à inclure le paramètre **-IncludeDeletedObjects** pour que la commande retourne aussi les objets supprimés : indispensable compte tenu de ce que l'on cherche à effectuer comme action.

Ci-dessous, la commande permettant de visualiser l'ensemble des éléments supprimés, ce qui permet de visualiser le **contenu de la corbeille Active Directory**.

```
Get-ADObject -Filter 'isDeleted -eq $true -and Name -like "*DEL:*"' -
IncludeDeletedObjects
```

Si l'on veut visualiser seulement les objets ordinateurs présents dans la corbeille, c'est-à-dire les objets avec la classe « computer », on va ajouter un filtre afin de cibler cette classe. En PowerShell, ceci se traduit par l'ajout de la condition suivante pour filtrer sur la classe « computer » : *-and ObjectClass -eq "computer"*

Si l'on reprend la commande Get-ADObject, nous obtenons :

```
Get-ADObject -Filter 'isDeleted -eq $true -and Name -like "*DEL:*" -and ObjectClass
-eq "computer"' -IncludeDeletedObjects
```

Pour les utilisateurs, le principe est le même, sauf que l'on remplace "computer" par "user" pour cibler une classe différente. La commande Get-ADObject devient :

```
Get-ADObject -Filter 'isDeleted -eq $true -and Name -like "*DEL:*" -and ObjectClass
-eq "user"' -IncludeDeletedObjects
```

Pur **supprimer les objets de la corbeille Active Directory**, par exemple tous les utilisateurs ou tous les utilisateurs, il faudra coupler **Get-ADObject** avec **Remove-ADObject**. La première commande va récupérer la liste des objets comme nous l'avons vu, pour ensuite envoyer son résultat à la seconde commande grâce au pipeline. Vous l'aurez compris, la commande « Remove-ADObject » va supprimer les objets.

Ci-dessous, la commande complète pour supprimer les ordinateurs de la corbeille :

```
Get-ADObject -Filter 'isDeleted -eq $true -and Name -like "*DEL:*" -and ObjectClass
-eq "computer"' -IncludeDeletedObjects | Remove-ADObject -Confirm:$false
```

Les utilisateurs et les autres objets dans la corbeille ne seront pas altérés par cette commande. Pour vider toute la corbeille, il suffit de retirer le filtre sur les classes d'objets (premier exemple) et d'ajouter le Remove-ADObject.

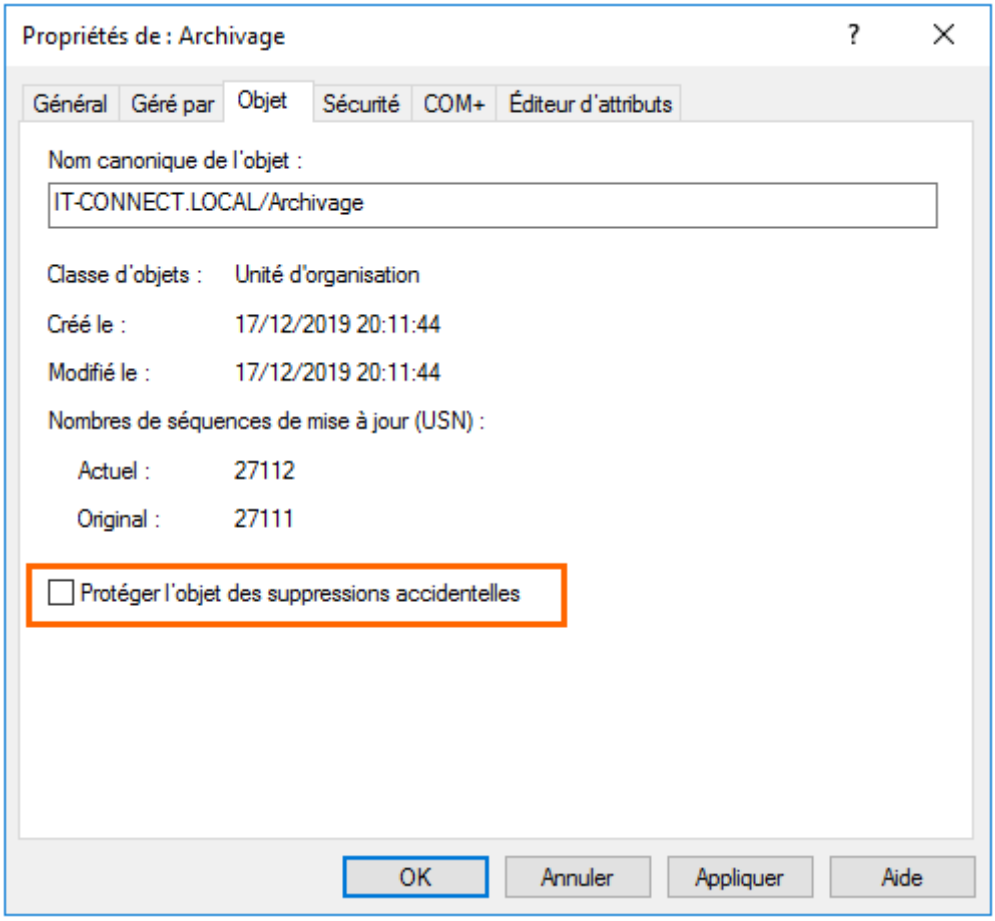
Avec ces différentes commandes, vous êtes en mesure d'activer la corbeille Active Directory sur votre domaine et de gérer les objets supprimés, notamment pour réaliser la restauration d'un objet si le cas se présente.

7. Activer la protection contre la suppression accidentelle sur vos objets

Dans le but de protéger le contenu d'un annuaire Active Directory, notamment contre les suppressions accidentelles, Microsoft a intégré une protection qu'il est possible de déployer sur les objets de son annuaire.

Par défaut, lorsque l'on crée une unité d'organisation (OU – *Organizational Unit*), cette option est activée. Cependant, sur les annuaires plus anciens et qui sont migrés au fur et à mesure des évolutions serveurs, il est possible que cette option ne soit pas active sur certaines unités d'organisation.

L'option en question s'appelle : **Protéger l'objet des suppressions accidentelles.**



Nous verrons comment protéger l'intégralité des unités d'organisation, et nous verrons également comment protéger les comptes utilisateurs, ordinateurs et groupes sensibles de votre organisation.

Comme nous l'avons, il existe la corbeille Active Directory pour récupérer des objets supprimés, mais il vaut mieux éviter d'en arriver là.

I. Lister les OU non protégées

Commençons par lister les unités d'organisation où cette option n'est pas active : bien que cette étape facultative, elle permettra de se rendre compte s'il y a beaucoup d'OUs concernées ou non. Une sorte d'audit rapide de la situation… C'est également l'occasion de découvrir une commande PowerShell supplémentaire.

Pour récupérer cette liste, on s'appuie sur le commandlet **Get-ADOrganizationalUnit** où l'on filtrera les résultats afin d'afficher uniquement les Unités d'organisation où le paramètre **ProtectedFromAccidentalDeletion** (correspondant à l'option de protection) est à **false** (faux).

La commande est la suivante :

```
Get-ADOrganizationalUnit -Filter * -Property ProtectedFromAccidentalDeletion | `
                         Where{ $_.ProtectedFromAccidentalDeletion -eq $false }
```

Pour que ce soit plus lisible, nous pouvons optimiser la sortie :

```
Get-ADOrganizationalUnit -Filter * -Property ProtectedFromAccidentalDeletion | `
                         Where{ $_.ProtectedFromAccidentalDeletion -eq $false } | `
                         Format-Table DistinguishedName
```

Seulement une OU concernée dans mon exemple, sur laquelle j'ai volontairement désactivé cette option :

```
PS C:\Windows\system32> Get-ADOrganizationalUnit -Filter * -Property ProtectedFromAccide
                        Where{ $_.ProtectedFromAccidentalDeletion -eq $false } |
                        Format-Table DistinguishedName

DistinguishedName
-----------------
OU=Archivage,DC=IT-CONNECT,DC=LOCAL
```

Passons maintenant à l'activation de cette option sur l'ensemble des OUs où ce n'est pas actif.

II. Activer la protection sur toute les OU de l'annuaire

Plutôt que de devoir repointer une par une les OUs afin d'activer la protection manuellement là où elle n'est pas active, on va plutôt faire cela en PowerShell.

Pour cela, on reprend la commande qui permet de lister les OUs où l'option de protection n'est pas active, à laquelle on appliquera une commande supplémentaire sur la sortie : **Set-ADOrganizationalUnit** pour modifier une OU.

Autrement dit, à partir de la liste obtenue avec la commande précédente on indique que l'on souhaite activer l'option ProtectedFromAccidentalDeletion sur ces unités d'organisation, la « liste » étant envoyée via le pipeline.

Sans plus tarder, voici la commande :

```
Get-ADOrganizationalUnit -Filter * -Property ProtectedFromAccidentalDeletion | `
                         Where{ $_.ProtectedFromAccidentalDeletion -eq $false } | `
                         Set-ADOrganizationalUnit -ProtectedFromAccidentalDeletion $true
```

Bonne nouvelle ! Suite à l'exécution de cette commande, vous pouvez être sûr que l'ensemble des Unités d'organisations de votre annuaire Active Directory sont protégées. Si vous souhaitez vous en assurer, c'est tout simple : il suffit d'exécuter à nouveau la première commande pour lister les OUs, et logiquement elle ne doit pas retourner la moindre OU.

Cette commande, simple finalement, vous permettra de faire un gain de temps conséquent.

III. Protéger un autre objet : utilisateur, ordinateur, etc.

Maintenant que nos unités d'organisation sont protégées, il est intéressant d'ajouter cette protection sur les comptes utilisateurs et ordinateurs sensibles. Ce serait dommage que l'objet ordinateur correspondant au PC du DSI soit supprimé… Idem pour son compte utilisateur.

Pour protéger un objet, il va falloir le récupérer : par exemple via « Get-ADUser » pour un utilisateur, ou « Get-ADComputer » pour un ordinateur. Ensuite, l'objet sera modifié via Set-ADObject qui dispose de l'option ProtectedFromAccidentalDeletion comme le cmdlet dédié à la modification des OUs.

Pour protéger le compte Administrateur, voici la commande :

```
Get-ADUser -Identity "Administrateur" | `
           Set-ADObject -ProtectedFromAccidentalDeletion:$true
```

De la même façon, pour protéger l'ordinateur « PC-DSI » :

```
Get-ADComputer -Identity "PC-DSI" | `
               Set-ADObject -ProtectedFromAccidentalDeletion:$true
```

Grâce aux deux exemples ci-dessus, vous voyez bien que la logique reste la même et que c'est la récupération de l'objet ou des objets en entrée qui est différente.

Pour vérifier cette valeur, il faudra utiliser la commande Get-ADObject. Elle est utile, car elle permet de faire des requêtes très précises sur l'annuaire, peu importe le type d'objet.

Pour vérifier que notre compte « Administrateur » est bien protégé :

```
Get-ADObject -Filter {ObjectClass -eq "user" -and SamAccountName -eq
"Administrateur"} -Properties ProtectedFromAccidentalDeletion
```

Voici le résultat :

```
PS C:\Windows\system32> Get-ADObject -Filter {ObjectClass -eq "user" -and SamAccountName


DistinguishedName               : CN=Administrateur,CN=Users,DC=IT-CONNECT,DC=LOCAL
Name                            : Administrateur
ObjectClass                     : user
ObjectGUID                      : c083ace3-81af-4875-9825-35583db79fd5
ProtectedFromAccidentalDeletion : True
```

Pour le PC, il faudra seulement modifier la classe d'objet (ObjectClass) et modifier le SamAccountName pour indiquer « PC-DSI$ » : le dollar à la fin est obligatoire sur un identifiant d'objet ordinateur.

```
Get-ADObject -Filter {ObjectClass -eq "computer" -and SamAccountName -eq "PC-DSI$"}
-Properties ProtectedFromAccidentalDeletion
```

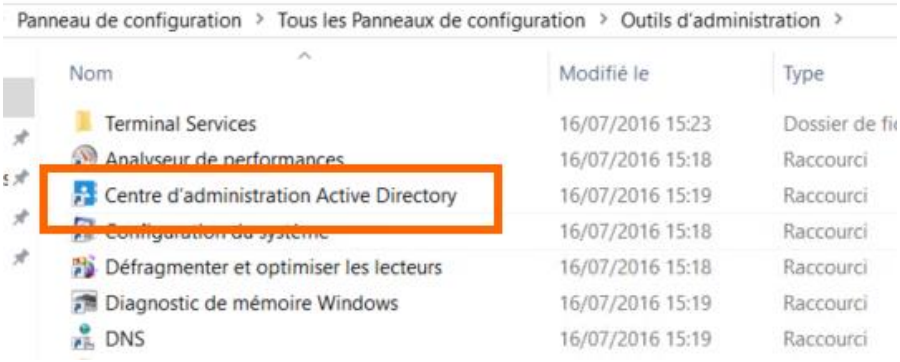
Le tour est joué, maintenant à vous d'activer ces options sur les objets qui sont précieux au niveau de votre infrastructure.

8. Apprendre PowerShell grâce au Centre d'administration Active Directory

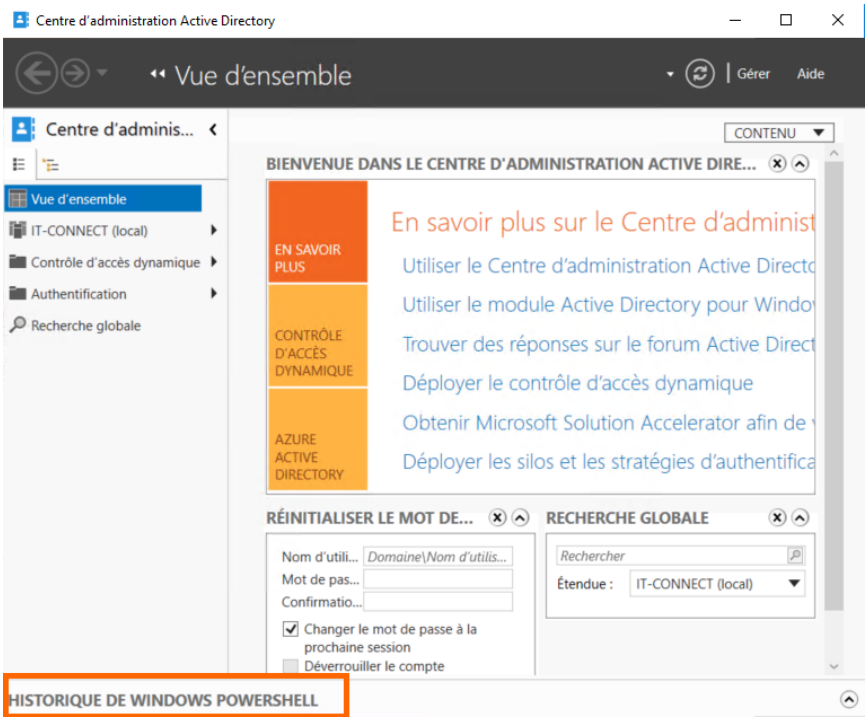
Ce chapitre est le seul où vous serez autorisé à manipuler l'interface graphique de votre contrôleur de domaine. Il faut dire que c'est pour la bonne cause : le Centre d'administration Active Directory (ADAC) est très intéressant pour apprendre PowerShell. Pourquoi ? Parce qu'il propose d'accéder à un historique des commandes PowerShell qui vous donne la commande PowerShell correspondante à chacune des opérations effectuées dans cette console d'administration.

I. L'historique PowerShell d'ADAC

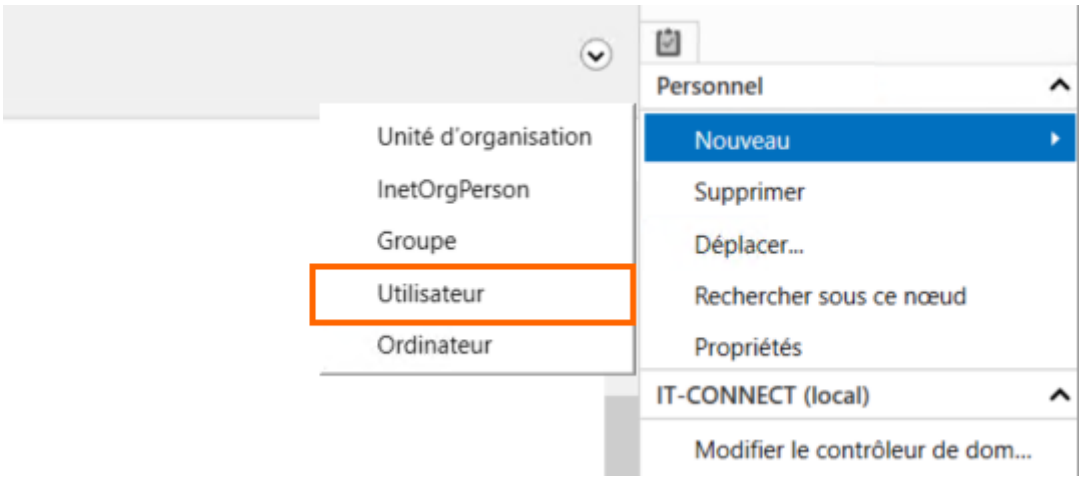
Je vous invite à ouvrir le Centre d'administration Active Directory, que vous pourrez trouver au sein des Outils d'administration.



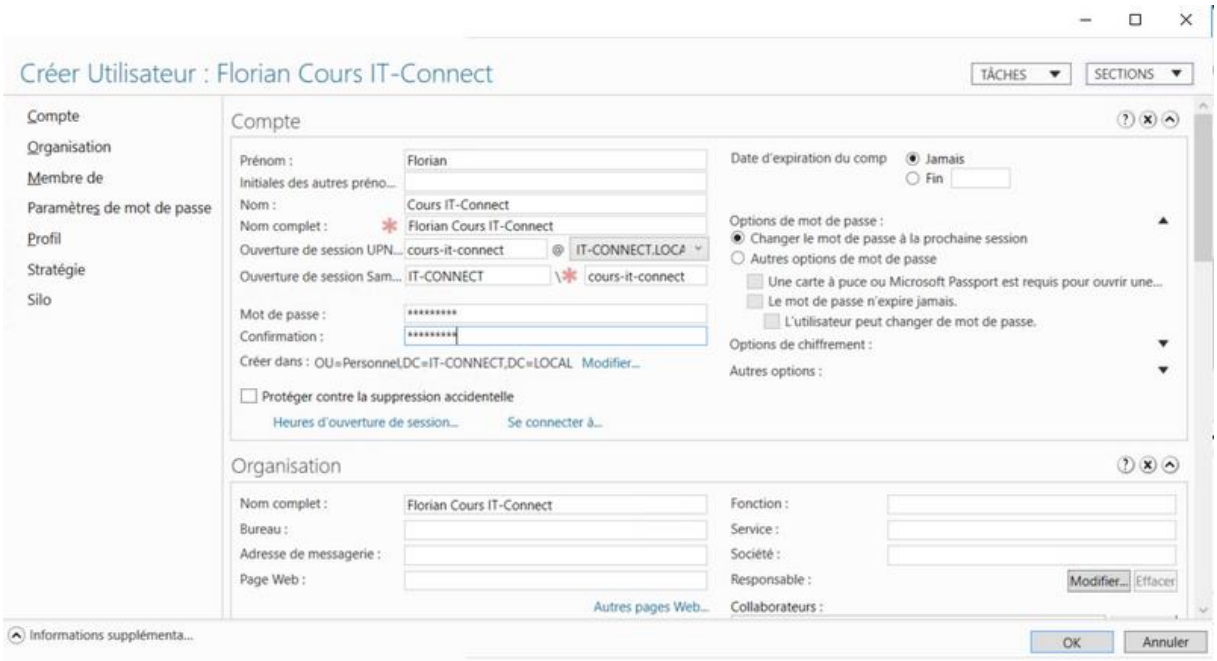
Une fois la console ouverte, dans le bas de la fenêtre, vous verrez « Historique de Windows PowerShell ». Cliquez dessus… Un panneau va s'afficher.



À titre d'exemple, ce que je vous invite à faire, c'est créer un nouvel utilisateur à partir de l'interface graphique. Positionnez-vous dans une unité d'organisation à partir de l'arborescence à gauche, puis créer un nouvel utilisateur : en haut à droite, cliquez sur « Nouveau » puis « Utilisateur ».



Renseignez les différents champs pour créer votre utilisateur, notamment le nom, le prénom, l'identifiant, ainsi que le mot de passe. Validez quand vous avez terminé… L'utilisateur va être créé dans votre annuaire.



Dans le même temps, l'historique des commandes PowerShell va afficher les commandes qui ont permis de créer l'utilisateur avec les options et les informations que vous avez choisies lors de la création avec l'interface graphique.

Vous pouvez copier ces commandes et même les rejouer : pratique ! Un bon moyen d'apprendre PowerShell et de faire la transition entre la création en mode graphique et via PowerShell. Il est à noter que l'historique des commandes n'affiche pas le mot de passe en clair, il affiche « System.Security.SecureString » là où le mot de passe devrait apparaître et c'est tant mieux !

Au-delà d'apprendre à utiliser PowerShell, cet historique permettra de visualiser les dernières actions réalisées sur votre annuaire Active Directory à l'aide de la console ADAC. Cet historique fonctionne pour l'ensemble des actions et pas seulement pour la création d'un utilisateur.

Conclusion

Tout au long de la lecture de ce livre numérique, vous avez pu découvrir de nouvelles commandes PowerShell pour gérer un annuaire Active Directory au quotidien, principalement en ce qui concerne la gestion des utilisateurs, des ordinateurs et des groupes. Désormais, vous êtes en mesure d'administrer votre annuaire Active Directory à l'aide de PowerShell et d'automatiser certaines tâches fastidieuses comme la création des utilisateurs.

Puisque PowerShell est un langage de programmation, il ne vous reste plus qu'à laisser vos connaissances et votre imagination s'exprimer pour développer des outils qui répondent précisément aux besoins de votre entreprise. J'ai toujours à l'esprit qu'il vaut mieux passer du temps à développer un script pour automatiser une tâche plutôt que de passer du temps à effectuer des actions manuellement de façon répétitive.

Je vous encourage à consulter notre site IT-Connect pour découvrir encore plus de contenus, en accès libre, que ce soit des tutoriels, des cours en ligne ou de l'actualité pour assurer une veille technologique.

https://www.it-connect.fr