TP PowerShell – Utilisation/modification de Scripts

Note :
Puisque j'ai utilisé instinctivement la VM Windows Server, les résultats seront toujours faux dû au fait qu'on ne puisse pas créer, modifier et supprimer des utilisateurs locaux sur Windows Server du moment que l'AD est activé. Cependant, il est possible de modifier les scripts pour utiliser les cmdlet New-ADUser ainsi que Remove-ADUser.

Prérequis : VM Windows 10

1. Présentation

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

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

Pour réaliser l'activité, vous devez copier des scripts Powershell, disponible sur ClassRoom, dans le dossier scriptPowershell.

PowerShell dispose d'une collection de 15 cmdlets appelée **Microsoft.PowerShell.LocalAccounts** pour gérer les utilisateurs et groupes locaux.

Lien :
https://blog.netwrix.fr/2019/01/16/comment-ajouter-supprimer-et-modifier-des-utilisateurs-et-groupes-locaux-avec-powershell/

Pour visualiser la liste de ces cmdlets utiliser la commande suivante :

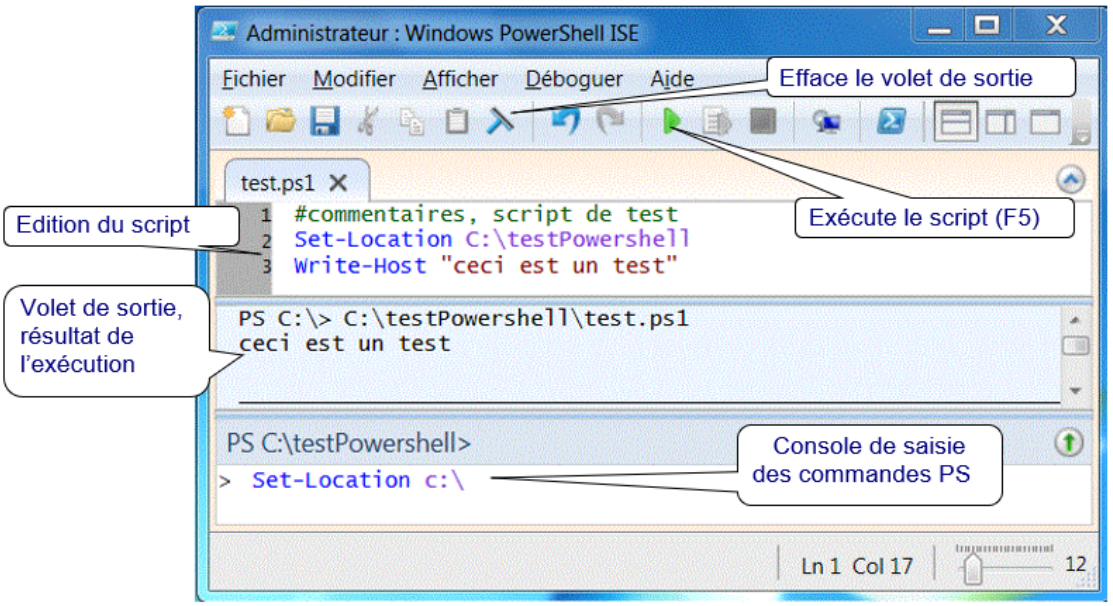
```
PS > Get-Command -Module Microsoft.PowerShell.LocalAccounts

CommandType     Name
-----------     ----
Cmdlet          Add-LocalGroupMember     => Ajouter un utilisateur au groupe local
Cmdlet          Disable-LocalUser        => Désactiver un compte d'utilisateur local
Cmdlet          Enable-LocalUser         => Activer un compte d'utilisateur local
Cmdlet          Get-LocalGroup           => Afficher les préférences du groupe local
Cmdlet          Get-LocalGroupMember     => Afficher la liste des membres du groupe local
Cmdlet          Get-LocalUser            => Afficher les informations d'un compte d'utilisateur
local
Cmdlet          New-LocalGroup           => Créer un nouveau groupe local
Cmdlet          New-LocalUser            => Créer un nouveau compte d'utilisateur local
Cmdlet          Remove-LocalGroup        => Supprimer un groupe local
Cmdlet          Remove-LocalGroupMember  => Supprimer un membre d'un groupe local
Cmdlet          Remove-LocalUser         => Supprimer un compte d'utilisateur local
Cmdlet          Rename-LocalGroup        => Renommer un groupe local
Cmdlet          Rename-LocalUser         => Renommer un compte d'utilisateur local
Cmdlet          Set-LocalGroup           => Modifier les paramètres d'un groupe local
Cmdlet          Set-LocalUser            => Modifier les paramètres de compte d'un utilisateur
local
```

Utilisation de Windows PowerShell ISE
Lancer Windows PowerShell ISE:
Démarrer\Tous les programmes\Accessoires\Windows PowerShell\Windows PowerShell ISE

Présentation des différentes fenêtres :

TP PowerShell – Utilisation/modification de Scripts

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



Dans PowerShell, il existe quatre paramètres de stratégie d'exécution des scripts qui sont :

Restricted	: paramètre par défaut, n'autorise pas l'exécution des scripts,
AllSigned	: n'exécute que les scripts de confiance, donc signés,
RemoteSigned	: exécute les scripts locaux sans obligation de confiance et les scripts de confiance issus d'Internet,
Unrestricted	: autorise l'exécution de tous les scripts.

Démarrer Windows PowerShell en tant qu'administrateur, puis utiliser la commande suivante pour définir la stratégie :

- Commande pour connaître la stratégie en cours : *Get-ExecutionPolicy*
- Commande pour modifier la stratégie : Set-ExecutionPolicy RemoteSigned

Première utilisation Microsoft.PowerShell.LocalAccounts
Utilisez la commande suivante pour obtenir la liste des utilisateurs :

```
PS > Get-LocalUser

Name               Enabled Description
----               ------- -----------
Administrateur     False   Compte d'utilisateur d'administration
Charles            True
DefaultAccount     False   Compte utilisateur géré par le système.
Invité             False   Compte d'utilisateur invité
WDAGUtilityAccount False   Compte d'utilisateur géré et utilisé par le
système pour les scénarios Windows Defender A...
```

La colonne Enabled indique les compte activés (True) et désactivé (False).

Utilisez la commande suivante pour obtenir la liste des propriétés d'un compte :

```
PS > Get-LocalUser -Name 'Charles' | Select-Object *
AccountExpires         :
Description            :
Enabled                : True
FullName               :
PasswordChangeableDate : 30/07/2022 14:42:13
PasswordExpires        :
UserMayChangePassword  : True
PasswordRequired       : False
PasswordLastSet        : 30/07/2022 14:42:13
LastLogon              : 17/01/2023 20:49:20
Name                   : Charles
SID                    : S-1-5-21-4085183622-348947280-3669370899-1001
PrincipalSource        : Local
ObjectClass            : Utilisateur
```

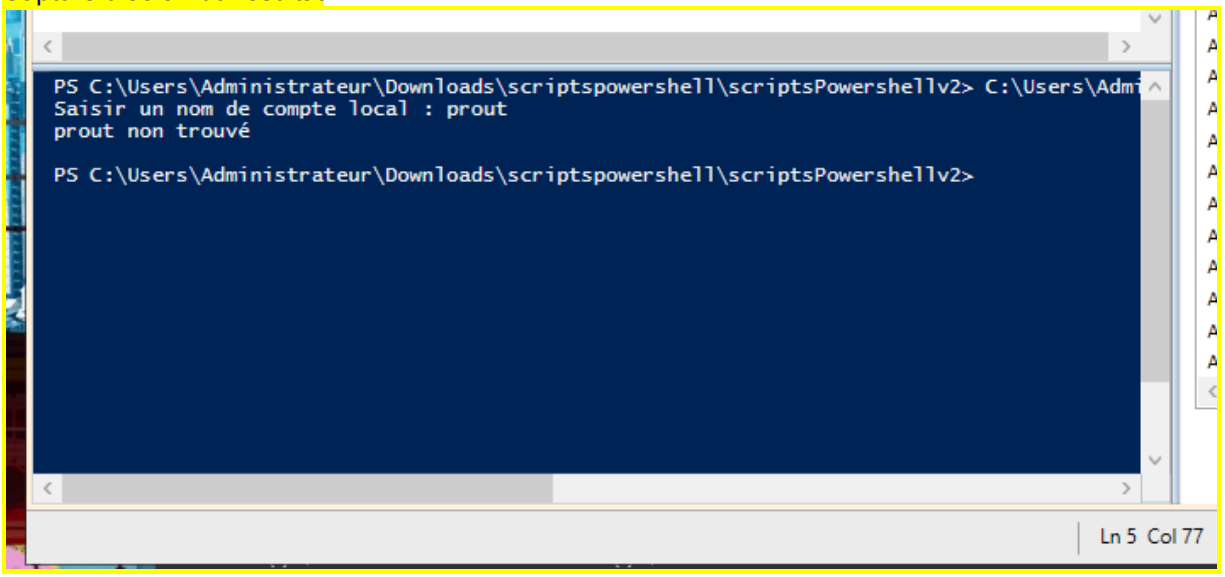
2. Exercice A : Accès aux comptes locaux du système

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

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

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

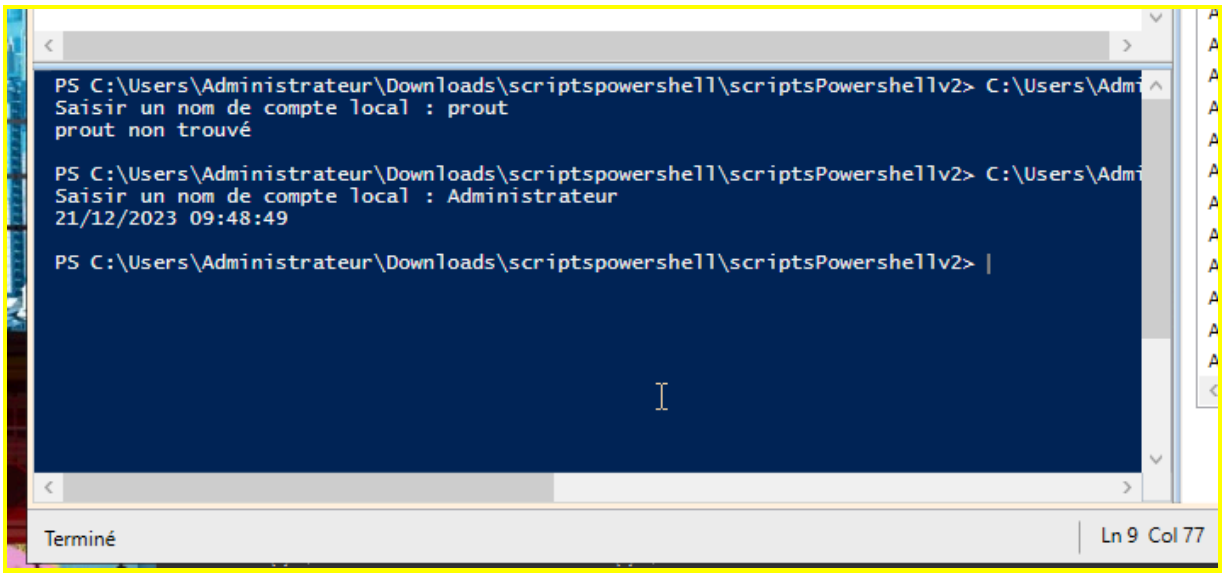
Capture d'écran du résultat



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

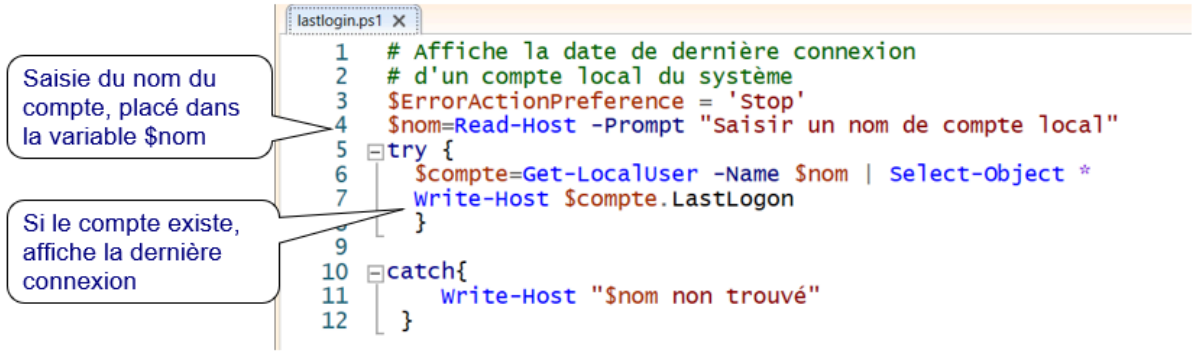
Capture d'écran du résultat
Voir note plus haut

TP PowerShell – Utilisation/modification de Scripts



Le script :

Explications de la ligne 4 : L'accès à la base locale de comptes utilisateurs d'un système Windows est réalisé avec la cmdlet **Get-LocalUser**.



La gestion des erreurs est réalisée avec les blocs Try .. Catch. Cela permet, si le compte n'existe pas :

- De ne pas interrompre le script,
- D'intercepter l'erreur,
- D'afficher un message d'erreur.

Remarque :

- Pour filtrer les éléments de la base de comptes, le nom de compte recherché est spécifié avec le paramètre **Name** et le nom de l'élément recherché contenu dans la variable $nom.
- le caractère | permet de chainer (d'envoyer) le résultat de la cmdlet **Get-LocalUser** vers la cmdlet **Select-Object** pour sélectionner ici toutes (caractère *) les propriétés de l'objet obtenu avec **Get-LocalUser**.

Il faut bien sûr avoir des droits d'administration sur l'ordinateur distant.

Lien :
https://learn.microsoft.com/fr-fr/powershell/module/microsoft.powershell.utility/select-object?view=powershell-7.2

A faire (seulement si le compte a été trouvé) :
- Dans la console de saisie des commandes (Volet 3), afficher les propriétés de la variable $compte, en utilisant la commande suivante : *$compte | Get-Member*

Capture d'écran du résultat

TP PowerShell – Utilisation/modification de Scripts

```
PS C:\Users\Administrateur\Downloads\scriptspowershell\scriptsPowershellv2> $compte


AccountExpires         :
Description            : Compte d'utilisateur d'administration
Enabled                : True
FullName               :
PasswordChangeableDate : 10/11/2023 14:05:54
PasswordExpires        :
UserMayChangePassword  : True
PasswordRequired       : True
PasswordLastSet        : 09/11/2023 14:05:54
LastLogon              : 21/12/2023 09:48:49
Name                   : Administrateur
SID                    : S-1-5-21-2436572114-224129429-3453362755-500
PrincipalSource        : ActiveDirectory
ObjectClass            : Utilisateur



PS C:\Users\Administrateur\Downloads\scriptspowershell\scriptsPowershellv2>
```

```
PS C:\Users\Administrateur\Downloads\scriptspowershell\scriptsPowershellv2> $compte | Get-Member


   TypeName : Selected.Microsoft.PowerShell.Commands.LocalUser

Name                   MemberType   Definition
----                   ----------   ----------
Equals                 Method       bool Equals(System.Object obj)
GetHashCode            Method       int GetHashCode()
GetType                Method       type GetType()
ToString               Method       string ToString()
AccountExpires         NoteProperty object AccountExpires=null
Description            NoteProperty string Description=Compte d'utilisateur d'administration
Enabled                NoteProperty bool Enabled=True
FullName               NoteProperty string FullName=
LastLogon              NoteProperty datetime LastLogon=21/12/2023 09:48:49
Name                   NoteProperty string Name=Administrateur
ObjectClass            NoteProperty string ObjectClass=Utilisateur
PasswordChangeableDate NoteProperty datetime PasswordChangeableDate=10/11/2023 14:05:54
PasswordExpires        NoteProperty object PasswordExpires=null
PasswordLastSet        NoteProperty datetime PasswordLastSet=09/11/2023 14:05:54
PasswordRequired       NoteProperty bool PasswordRequired=True
PrincipalSource        NoteProperty PrincipalSource PrincipalSource=ActiveDirectory
SID                    NoteProperty SecurityIdentifier SID=S-1-5-21-2436572114-224129429-345336...
UserMayChangePassword  NoteProperty bool UserMayChangePassword=True



PS C:\Users\Administrateur\Downloads\scriptspowershell\scriptsPowershellv2>
```

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

Capture d'écran du résultat

```
Enabled                NoteProperty bool Enabled=True
FullName               NoteProperty string FullName=
LastLogon              NoteProperty datetime LastLogon
ToString               Method       string ToString()
AccountExpires         NoteProperty object AccountExpires=null
Description            NoteProperty string Description=Compte d'utilisateur d'administration
Enabled                NoteProperty bool Enabled=True
FullName               NoteProperty string FullName=
```

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

Script modifié et capture d'écran du résultat

```
$nom=Read-Host -Prompt "Saisir un nom de compte local"
try {
   $compte=Get-LocalUser -Name $nom | Select-Object *
   Write-Host $compte.LastLogon, " | ", $compte.Description, " | ", $compte.FullName
   }
catch{
```

```
PS C:\Users\Administrateur\Downloads\scriptspowershell\scriptsPowershellv2> C:\Users\Administrat
Saisir un nom de compte local : Administrateur
21/12/2023 09:48:49  |  Compte d'utilisateur d'administration  |
```

3. Exercice B : Ajout d'un compte local du système

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

Attention :
Lorsque de la création d'un compte d'utilisateur local, il ne fait jamais tapez le mot de passe en texte simple mais le convertir en une chaîne sécurisée à l'aide du paramètre AsSecureString ou ConvertTo-SecureString.

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

Capture d'écran du résultat

```
1  # Ajoute un compte dans la base locale du système
2  # à partir de la saisie du nom, du mopt de passe et de la description
3  # puis ajout du compte au groupe local Utilisateurs
4
5  $nom=Read-Host -Prompt "Saisir un nom"
6  $UserPassword=Read-Host -AsSecureString -Prompt "Saisir un mot de passe"
7  $description=Read-Host -Prompt "Saisir une description"
8
9  try
10 {
11     $compte=New-LocalUser $nom -Password $UserPassword -Description $description
12     Write-Host "$nom ajouté"
13     # ajout du compte au groupe local Utilisateurs
14     Add-LocalGroupMember -Group "Utilisateurs" -Member $nom
15 }
16 catch{
17      Write-Host "$nom existe déjà"
18 }
```

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

Capture d'écran de réalisation du script

```
PS C:\Users\Administrateur\Downloads\scriptspowershell\scriptsPowershellv2> C:\Users\Administrateur\D
Saisir un nom : Testas
Saisir une description : Testas
Testas existe déjà

PS C:\Users\Administrateur\Downloads\scriptspowershell\scriptsPowershellv2>
```

Voir note plus haut

- Vérifier la création du compte avec la cmdlet Get-LocalUser.

Capture d'écran du résultat

```
PS C:\Users\Administrateur\Downloads\scriptspowershell\scriptsPowershellv2> Get-LocalUser

Name           Enabled Description
----           ------- -----------
Administrateur True    Compte d'utilisateur d'administration
Invité         False   Compte d'utilisateur invité
krbtgt         False   Compte de service du centre de distribution de clés


PS C:\Users\Administrateur\Downloads\scriptspowershell\scriptsPowershellv2>
```

Bah du coup non..
Voir note plus haut

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

Le script :

Explications :
Ligne 6 : Saisie sécurisée du mot de passe avec le paramètre **AsSecureString**.

Ligne 11 : La cmdlet New-LocalUser permet la création du compte avec les paramètres :
- premier paramètre : le nom du compte avec le contenu de la variable $nom,
- deuxième paramètre : le mot de passe avec le contenu de la variable $UserPassword,
- troisième paramètre : la description avec le contenu de la variable $description,
- Le résultat est une variable nommée $compte.

Ligne 14 : Ajout du compte nouvellement créé dans le groupe Utilisateurs avec la cmdlet **Add-LocalGroupMember** :
- Cela est **nécessaire** pour que le compte et puisse **ouvrir une session** ;
- Vous pouvez choisir de mettre le compte dans un autre groupe, comme le groupe **Administrateurs**.

Saisie du nom, du mot de passe et de la description

Test de la réussite de la création

ajoutCompte.ps1

```
1  # Ajoute un compte dans la base locale du système
2  # à partir de la saisie du nom, du mopt de passe et de la description
3  # puis ajout du compte au groupe local Utilisateurs
4
5  $nom=Read-Host -Prompt "Saisir un nom"
6  $UserPassword=Read-Host -AsSecureString -Prompt "Saisir un mot de passe"
7  $description=Read-Host -Prompt "Saisir une description"
8
9  try
10 {
11     $compte=New-LocalUser $nom -Password $UserPassword -Description $description
12     Write-Host "$nom ajouté"
13     # ajout du compte au groupe local Utilisateurs
14     Add-LocalGroupMember -Group "Utilisateurs" -Member $nom
15 }
16 catch{
17      Write-Host "$nom existe déjà"
18 }
```

A faire :
- Modifier le script ajoutCompte.ps1 pour que le nom complet soit également saisi et renseigné au moment de l'ajout du compte.

Script modifié

```powershell
# Ajoute un compte dans la base locale du système
# à partir de la saisie du nom, du mopt de passe et de la description
# puis ajout du compte au groupe local Utilisateurs

$nom=Read-Host -Prompt "Saisir un nom"
$nomComplet=Read-Host -Prompt "Saisir un nom complet (Ex : Testas Poire)"
$UserPassword=Read-Host -AsSecureString -Prompt "Saisir un mot de passe"
$description=Read-Host -Prompt "Saisir une description"

try
{
    $compte=New-LocalUser $nom -Password $UserPassword -Description $description -FullName $nomComplet
    Write-Host "$nom ajouté"
    # ajout du compte au groupe local Utilisateurs
    Add-LocalGroupMember -Group "Utilisateurs" -Member $nom
}
catch{
     Write-Host "$nom existe déjà"
}
```

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

Capture d'écran du résultat

```
PS C:\Users\Administrateur\Downloads\scriptspowershell\scriptsPowershellv2> C:\Users\Adminis
Saisir un nom : test02
Saisir un nom complet (Ex : Testas Poire) : Pomme Poire
Saisir une description : Fruits
test02 existe déjà

PS C:\Users\Administrateur\Downloads\scriptspowershell\scriptsPowershellv2> Get-LocalUser

Name           Enabled Description
----           ------- -----------
Administrateur True    Compte d'utilisateur d'administration
Invité         False   Compte d'utilisateur invité
krbtgt         False   Compte de service du centre de distribution de clés


PS C:\Users\Administrateur\Downloads\scriptspowershell\scriptsPowershellv2>
```

Voir note plus haut

4. Exercice C : Parcours d'un fichier texte contenant les informations des comptes utilisateurs

Ce script permet d'afficher tous les noms des comptes utilisateurs contenus dans un fichier texte.
Les informations sont sous la forme : nomCompte/nomComplet/Description

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

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

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

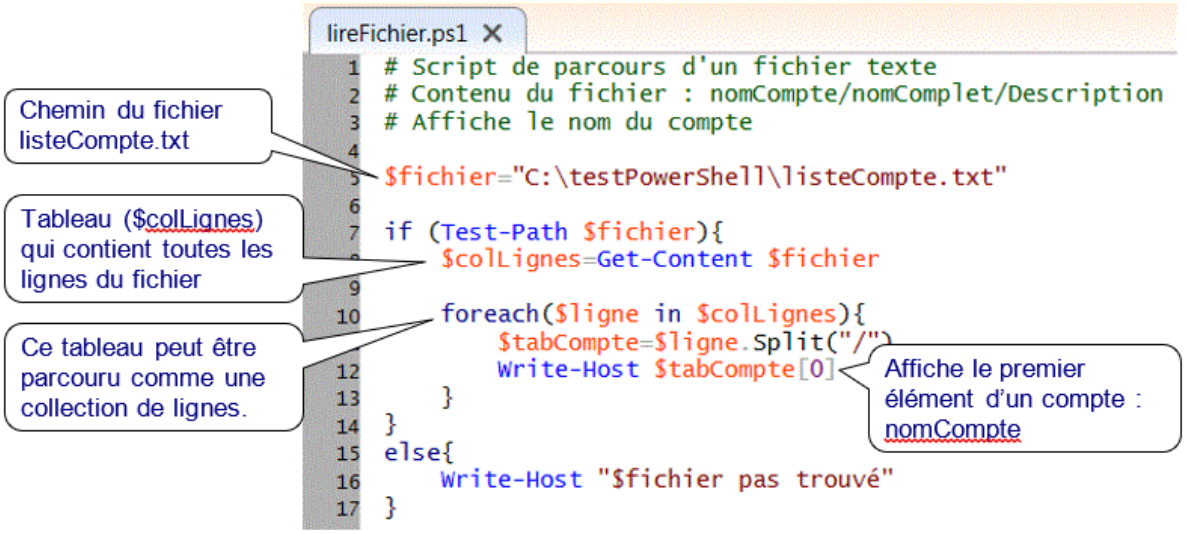
Capture d'écran du résultat

```
PS C:\Users\Administrateur\Downloads\scriptspowershell\scriptsPowershellv2> C:\Users\Administrateur\D
nom01
nom02
nom03

PS C:\Users\Administrateur\Downloads\scriptspowershell\scriptsPowershellv2>
```

Le script :

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

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



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

Capture du Script modifié et capture d'écran du résultat

```
1  # Script de parcours d'un fichier texte
2  # Contenu du fichier : nomCompte/nomComplet/Description
3  # Affiche le nom du compte
4
5  $fichier="C:\testPowerShell\listeCompte.txt"
6
7  if (Test-Path $fichier){
8      $colLignes=Get-Content $fichier
9
10     foreach($ligne in $colLignes){
11         $tabCompte=$ligne.Split("/")
12         Write-Host $tabCompte[0], $tabCompte[1]
13     }
14 }
15 else{
16     Write-Host "$fichier pas trouvé"
17 }
```

TP PowerShell – Utilisation/modification de Scripts

```
PS C:\Users\Administrateur\Downloads\scriptspowershell\scriptsPowershellv2> C:\User
nom01 nom01 prenom01
nom02 nom02 prenom02
nom03 nom03 prenom03

PS C:\Users\Administrateur\Downloads\scriptspowershell\scriptsPowershellv2>
```

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

Capture du Script modifié

```
$fichier="C:\testPowerShell\listeCompte.txt"

if (Test-Path $fichier){
    $colLIgnes=Get-Content $fichier

    foreach($ligne in $colLignes){
        $tabCompte=$ligne.Split("/")

        $nom=$tabCompte[0]
        $nomComplet=$tabCompte[1]
        $description=$tabCompte[2]
        $password=$tabCompte[3]
        try
        {
           $compte=New-LocalUser $nom -Password $password -Description $description -FullName $nomComplet
           Write-Host "$nom ajouté"
           # ajout du compte au groupe local Utilisateurs
           Add-LocalGroupMember -Group "Utilisateurs" -Member $nom
        }
        catch{
            Write-Host "$nom existe déjà"
        }
    }
}
else{
    Write-Host "$fichier pas trouvé"
}
```

- Tester l'ajout de ces comptes :

Capture d'écran du résultat

```
PS C:\Users\Administrateur\Downloads\scriptspowershell\scriptsPowershellv2> C:\User
nom01 existe déjà
nom02 existe déjà
nom03 existe déjà

PS C:\Users\Administrateur\Downloads\scriptspowershell\scriptsPowershellv2>
```

Voir note plus haut

- Faire un deuxième test avec le même fichier, que se passe-t-il pour les noms déjà existants ?
La même chose, car c'est bugué (Windows moment), en tout cas le script me semble être bon. J'ai testé de redémarrer et j'ai toujours le même soucis.

TP PowerShell – Utilisation/modification de Scripts

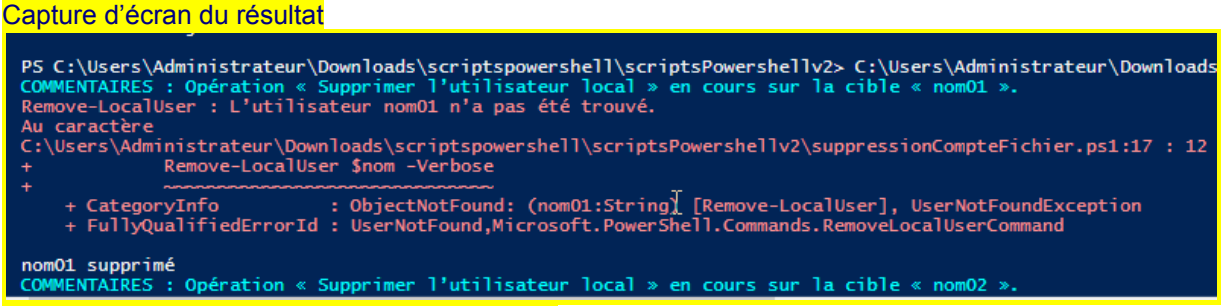
5. Exercice D : Suppressions de comptes utilisateurs

Ce script permet de supprimer tous les comptes utilisateurs dont les noms sont contenus dans un fichier texte.
Les informations sont toujours sous la forme : nomCompte/nomComplet/Description

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

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

Capture d'écran du résultat



(C'est la même sortie pour les trois utilisateurs)
On n'obtient aucun résultat probant, car les utilisateurs n'existent pas. Cependant on peut voir qu'il le confirme bien car il vérifie si les utilisateurs sont bien supprimés juste après.

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

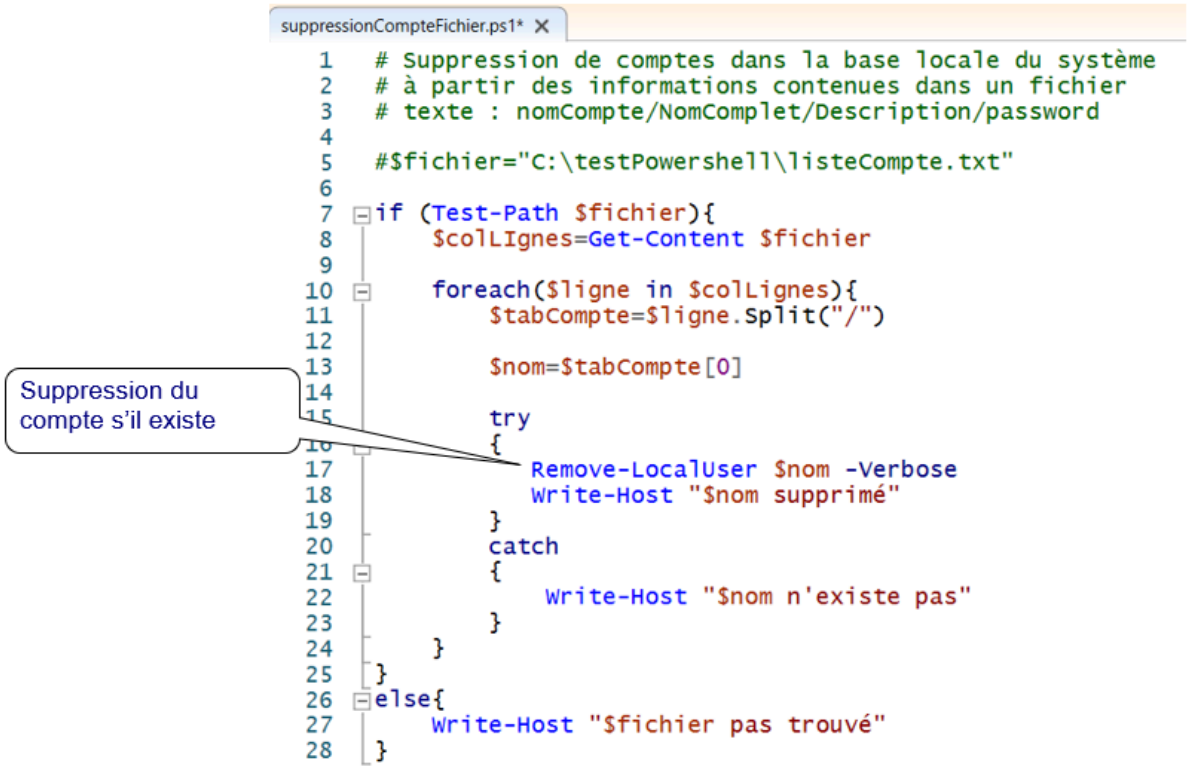
Capture d'écran du résultat

Voir note plus haut

Le script :

Explications :
Ligne 17 : La cmdlet Remove-LocalUser permet de supprimer un compte dont le nom du compte est spécifié par le premier paramètre, ici la variable $nom.

TP PowerShell – Utilisation/modification de Scripts




```powershell
# Suppression de comptes dans la base locale du système
# à partir des informations contenues dans un fichier
# texte : nomCompte/NomComplet/Description/password

#$fichier="C:\testPowershell\listeCompte.txt"

if (Test-Path $fichier){
    $colLIgnes=Get-Content $fichier

    foreach($ligne in $colLignes){
        $tabCompte=$ligne.Split("/")

        $nom=$tabCompte[0]

        try
        {
            Remove-LocalUser $nom -Verbose
            Write-Host "$nom supprimé"
        }
        catch
        {
            Write-Host "$nom n'existe pas"
        }
    }
}
else{
    Write-Host "$fichier pas trouvé"
}
```

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

Capture du script modifié

```powershell
        $nom=$tabCompte[0]

        try
        {
            Remove-LocalUser $nom, test01, test02 -Verbose
            Write-Host "$nom supprimé"
        }
        catch
        {
            Write-Host "$nom n'existe pas"
        }
    }
```

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

Capture d'écran du résultat

```
COMMENTAIRES : Opération « Supprimer l'utilisateur local » en cours sur la cible « test02 ».
Remove-LocalUser : L'utilisateur test02 n'a pas été trouvé.
Au caractère
C:\Users\Administrateur\Downloads\scriptspowershell\scriptsPowershellv2\suppressionCompteFichier.ps1:17 : 12
+            Remove-LocalUser $nom, test01, test02 -Verbose
+            ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : ObjectNotFound: (test02:String) [Remove-LocalUser], UserNotFoundException
    + FullyQualifiedErrorId : UserNotFound,Microsoft.PowerShell.Commands.RemoveLocalUserCommand

nom03 supprimé
```

Voir note plus haut

TP PowerShell – Utilisation/modification de Scripts

Annexe

A. Le pipeline et la variable $_

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

Exemple :
Get-Children | Get-Member

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

Autre exemple, plus complexe :
Get-ChildItem -recurse | Where {$_.extension -eq ".txt"}

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

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

B. Les structures de contrôle

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

```
If (condition)
{
     #bloc d'instructions
}
```

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

```
$var = Read-Host "Entrez un caractère"
If ($var -eq 'A')
{
   "Le caractère saisi par l'utilisateur est un 'A'"
}
```

On peut aussi mettre un else :

```
$var = Read-Host "Entrez un caractère"
If ($var -eq 'A')
{
   "Le caractère saisi par l'utilisateur est un 'A'"
}
Else
{
   "Le caractère saisi par l'utilisateur n'est pas un 'A' !"
}
```

Et un ou plusieurs elseif :

```
$var = Read-Host "Entrez un caractère"
If ($var -eq 'A')
{
   "Le caractère saisi par l'utilisateur est un 'A'"
}
Elseif ($var -eq 'B')
{
   "Le caractère saisi par l'utilisateur est un 'B'"
}
Else
{
   "Le caractère saisi par l'utilisateur n'est pas un 'A' ni un 'B' !"
}
```

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

Les différents opérateurs de comparaison :

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

Et quelques autres opérateurs utiles :

-not	Not
!	Not
-and	And
-or	Or

On peut réaliser des conditions complexes :

```
If (($var1 -eq 15) -and -not ($var2 -eq 18))
{
   write-host "lkjlkj" }
```

Quelques variables automatiques :

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

C. Les répétitions

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

- While
- Do…While
- Do…Until
- For
- Foreach

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

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

```
Foreach(<élément> in <collection>)
{
   # bloc d'instructions qui traite un élément
}
```

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

```
$collection = get-childitem

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

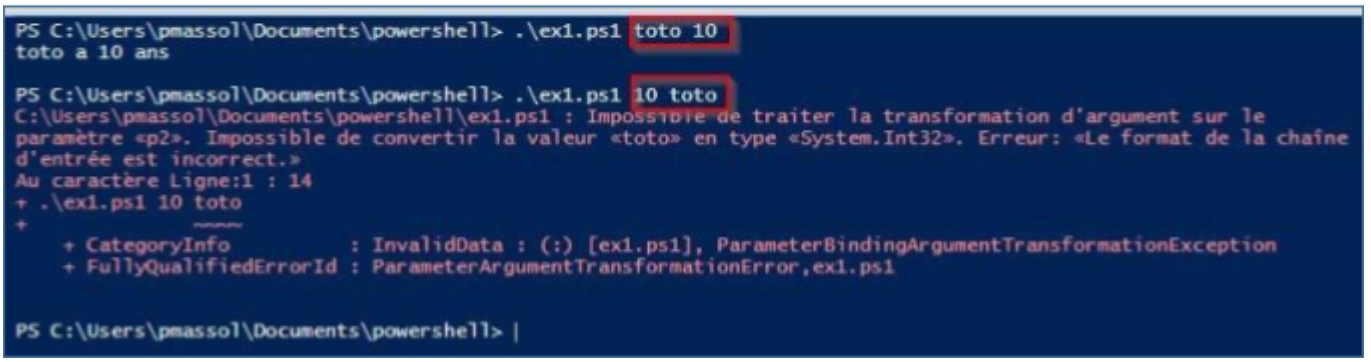
On obtient tous les éléments (dossiers et fichiers) contenus dans le dossier courant. On parcourt cette collection et on teste si c'est un dossier ou un fichier.

D. Les paramètres de script

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

```
param(
[string]$p1,
[int]$p2
)

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

On l'exécute de la façon suivante :



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