

PowerShell offre la possibilité de personnaliser l'affichage des colonnes.

Get-Help about_Calculated_Properties

Format-List

- **name/label** - optionnel
- **expression**
- **formatstring** - optionnel

Format-Table

- **name/label** - optionnel
- **expression**
- **formatstring** - optionnel
- **width** - optionnel
- **alignment** - optionnel

Select-Object

- **name/label** - optionnel
- **expression**

Sort-Object

- **expression**
- **ascending/descending** - optionnel

name/label

- Spécifie le nom de la propriété en cours de création.
- Vous pouvez utiliser **name** ou son alias **label**.

expression

- Un bloc de script utilisé pour calculer la valeur de la nouvelle propriété.

alignment

- Utilisé par les cmdlets qui produisent une sortie tabulaire pour définir comment les valeurs sont affichées dans une colonne.
- La valeur doit être **left**, **center**, ou **right**.

formatstring

- Spécifie une chaîne de format qui définit comment la valeur est formatée pour la sortie.
- Pour plus d'informations sur les chaînes de format, voir Types de format dans .NET.

width

- Spécifie la largeur maximale de la colonne d'un tableau lorsque la valeur est affichée.
- La valeur doit être supérieure à 0.

ascending / descending

- Les deux paramètres permettent de spécifier l'ordre de tri pour une ou plusieurs propriétés.
- Ce sont des valeurs booléennes.

Exemple 1a - Exemple avec des colonnes personnalisées avec Format-List

```
Get-ADUser -Filter * -Properties DisplayName | `
    Format-List Name,DisplayName,UserPrincipalName,DistinguishedName
```

On affiche le DistinguishedName de l'unité d'organisation parent des utilisateurs.

```
Get-ADUser -Filter * -Properties DisplayName | `
    Format-List Name,DisplayName,UserPrincipalName,DistinguishedName,
        @{label="OU parent"
        expression={$PSItem.DistinguishedName -split ', ',2}[1]}
    }
```

Exemple 1b - Exemple avec des colonnes personnalisées avec Format-List

```
$OU = "OU=FORMATION,DC=FORMATION,DC=LOCAL"
```

```
Get-ADOrganizationalUnit -Filter * `
    -SearchBase $OU | `
    Format-List Name,DistinguishedName
```

On affiche le DistinguishedName de l'unité d'organisation parent des unités d'organisation.

```
$OU = "OU=FORMATION,DC=FORMATION,DC=LOCAL"
```

```
Get-ADOrganizationalUnit -Filter * `
    -SearchBase $OU | `
    Format-List Name,DistinguishedName,
        @{label="OU parent"
        expression={$PSItem.DistinguishedName -split ', ',2}[1]}
    }
```

Exemple 2 - Exemple avec des colonnes personnalisées avec Format-List

```
$NOM_VM = "routeur"
Get-VMNetworkAdapter -VMName $NOM_VM `
    | Format-List Name,SwitchName,MacAddress,IPAddresses
```

Utilisation du paramètre -ExpandProperty avec plusieurs propriétés.

```
$NOM_VM = "routeur"
Get-VMNetworkAdapter -VMName $NOM_VM `
    | Format-List Name,SwitchName,MacAddress,
        @{label="IP"
        expression={$PSItem | Select-Object -ExpandProperty IPAddresses} -Join ', ' }
    }
```

Exemple 3 - Exemple avec des colonnes personnalisées avec Format-List

Par défaut la capacité des barrettes de mémoire est affichée en octets.

```
Get-CIMInstance -Class Win32_PhysicalMemory `
    | Format-List Caption, Capacity,
    Speed, DeviceLocator, Manufacturer, SerialNumber
```

note: pas besoin d'utiliser le paramètre FormatString pour afficher la taille des barrettes de mémoire

```
Get-CIMInstance -Class Win32_PhysicalMemory `
    | Format-List Caption,
    @{label='Size (GO) '
      expression={$PSItem.Capacity / 1GB}
    },
    Speed, DeviceLocator, Manufacturer, SerialNumber
```

Exemple 4 - Exemple avec des colonnes personnalisées avec Format-Table

Par défaut la taille des disques est affichée en octets.

```
$partition = 'C:'

Get-CIMInstance -Class Win32_LogicalDisk `
    -Filter "DeviceId='$partition'" `
    | Format-Table SystemName,DeviceID,VolumeName,Size,FreeSpace -AutoSize
```

note 1: on modifie le titre de plusieurs colonnes du tableau avec le paramètre Label

note 2: on utilise le paramètre FormatString='N2' pour afficher le résultat des calculs

```
$partition = 'C:'

Get-CIMInstance -Class Win32_LogicalDisk `
    -Filter "DeviceId='$partition'" `
    | Format-Table SystemName, `
        @{label='Partition'
          expression={$PSItem.DeviceId}
        }, `
        @{label='Description'
          expression={$PSItem.VolumeName}
        }, `
        @{label='Size (GO)'
          expression={$PSItem.Size / 1GB}
          formatstring='N2'
          alignment='center'
        }, `
        @{label='FreeSpace (GO)'
          expression={$PSItem.FreeSpace / 1GB}
          formatstring='N2'
          alignment='center'
        } -AutoSize
```

Exemple 5 - Exemple avec des colonnes personnalisées avec Format-Table

Utilisation des paramètres width et alignment avec Format-Table.

```
$chemin = "F:\_VIRTUEL\DISQUE"

$nom = @{label = 'Nom'
        expression= {$PSItem.Name}
        width=100
        alignment='left'
        }

$taille = @{label='KB'
            expression={ ($PSItem.Length / 1KB) }
            width=20
            alignment='right'
            }

Get-ChildItem -Path $chemin | Format-Table $nom,$taille
```

Exemple 6 - Exemple avec des colonnes personnalisées avec Sort-Object

Pour utiliser **Sort-Object** avec plusieurs paramètres, il faut utiliser des "hash table" pour trier par ordre croissant, décroissant, ou une combinaison d'ordres de tri.

```
Get-Service | Sort-Object -Property @{expression = "Status"
                                     descending = $true
                                   },
                               @{expression = "DisplayName"
                                     ascending = $true
                                   }
}
```

Exemple 7 - Exemple avec des colonnes personnalisées avec Select-Object

```
Get-NetAdapter | Select-Object Name,MacAddress
```

On modifie l'affichage de l'adresse MAC.

```
Get-NetAdapter | Select-Object Name,
                               @{label='MacAddress'
                                 expression={ $PSitem.MacAddress -replace ':','-'}
                               }
```

Exemple 8 - Exemple avec des colonnes personnalisées avec Select-Object

```
$OU = "OU=FORMATION,DC=FORMATION,DC=LOCAL"
```

```
Get-ADOrganizationalUnit -Filter * `
    -SearchBase $OU -Properties CanonicalName | Sort-Object CanonicalName | `
    Select-Object Name,DistinguishedName
```

On affiche le DistinguishedName de l'unité d'organisation parent des unités d'organisation.
Les unités d'organisation sont triées en ordre alphabétique sur la propriété CanonicalName.
On enregistre le résultat de la commande dans un fichier CSV.

```
$OU = "OU=FORMATION,DC=FORMATION,DC=LOCAL"
```

```
Get-ADOrganizationalUnit -Filter * `
    -SearchBase $OU -Properties CanonicalName | Sort-Object CanonicalName | `
    Select-Object @{label="NOM"
                  expression={$PSitem.Name}
                },
                @{label="OU_PARENT"
                  expression={$PSitem.DistinguishedName -split ',',[1]}
                } | `
    Export-Csv -Delimiter ";" `
        -NoTypeInformation `
        -Path "E:\_TEMP\OU_FORMATION.csv"
```

Pour le cours, il n'est pas nécessaire de comprendre le contenu des annexes 1 et 2.

Le contenu de ce document est un complément pour les prochains cours.

Informations sur le module ActiveDirectory

Active Directory

<https://learn.microsoft.com/en-us/powershell/module/activedirectory>

Le module ActiveDirectory contient 147 cmdlets.

```
(Get-Command -Module ActiveDirectory).Count
```

Utilisation de plusieurs GET-AD* et SET-AD*

Utiliser plusieurs cmdlets de PowerShell afin de se familiariser avec les objets d'un domaine Active Directory

- Utiliser plusieurs cmdlets GET du module ActiveDirectory
`Get-ADForest, Get-ADDomain, Get-ADOrganizationalUnit, Get-ADComputer, Get-ADUser, Get-ADGroup, Get-ADObject`
- Utiliser plusieurs cmdlets SET du module ActiveDirectory
`Set-ADDomain, Set-ADObject`
- Utiliser le cmdlet `Move-ADObject` pour déplacer un objet de l'Active Directory

Les opérateurs du paramètre -Filter avec des objets de l'Active Directory

Commande pour afficher des informations sur le paramètre "-Filter" avec des objets de l'Active Directory.
`Get-Help about_ActiveDirectory_Filter`

Mahheureusement, depuis plusieurs années la commande ne fonctionne pas.

Le contenu de la commande est disponible sur le site
[https://docs.microsoft.com/en-us/previous-versions/windows/server/hh531527\(v=ws.10\)](https://docs.microsoft.com/en-us/previous-versions/windows/server/hh531527(v=ws.10))

Voici la liste des opérateurs du paramètre **-Filter** pour les objets de l'Active Directory.

Les opérateurs du paramètre -Filter pour les objets de l'Active Directory	Description
<code>-eq</code>	Égal à
<code>-ne</code>	Pas égal à
<code>-le</code>	Plus petit que ou égal à
<code>-lt</code>	Plus petit que
<code>-ge</code>	Plus grand que ou égal à
<code>-gt</code>	Plus grand que
<code>-like</code>	Similaire à • supporte le caractère *
<code>-notlike</code>	Pas similaire à • supporte le caractère *
<code>-approx</code>	Approximativement égal à
<code>-bor</code>	OU binaire
<code>-band</code>	ET binaire
<code>-recursivematch</code>	Le filtre est appliqué récursivement
<code>-and</code>	ET logique
<code>-or</code>	OU logique
<code>-not</code>	NON logique

Les opérateurs du paramètre -Filter pour les objets de l'Active Directory.

-Filter

Specifies a query string that retrieves Active Directory objects. This string uses the PowerShell Expression Language syntax. The PowerShell Expression Language syntax provides rich type-conversion support for value types received by the *Filter* parameter. The syntax uses an in-order representation, which means that the operator is placed between the operand and the value. For more information about the *Filter* parameter, type `Get-Help about_ActiveDirectory_Filter`.

Syntax:

The following syntax uses Backus-Naur form to show how to use the PowerShell Expression Language for this parameter.

`<filter> ::= "{" <FilterComponentList> "}"`

`<FilterComponentList> ::= <FilterComponent> | <FilterComponent> <JoinOperator> <FilterComponent> | <NotOperator> <FilterComponent>`

`<FilterComponent> ::= <attr> <FilterOperator> <value> | "(" <FilterComponent> ")"`

`<FilterOperator> ::= "-eq" | "-le" | "-ge" | "-ne" | "-lt" | "-gt" | "-approx" | "-bor" | "-band" | "-recursivematch" | "-like" | "-notlike"`

`<JoinOperator> ::= "-and" | "-or"`

`<NotOperator> ::= "-not"`

`<attr> ::= <PropertyName> | <LDAPDisplayName of the attribute>`

`<value> ::= <compare this value with an <attr> by using the specified <FilterOperator>>`

For a list of supported types for <value>, type `Get-Help about_ActiveDirectory_ObjectModel`.

Note: For String parameter type, PowerShell will cast the filter query to a string while processing the command. When using a string variable as a value in the filter component, make sure that it complies with the [PowerShell Quoting Rules](#). For example, if the filter expression is double-quoted, the variable should be enclosed using single quotation marks: `Get-ADUser -Filter "Name -like '$UserName'"`. On the contrary, if curly braces are used to enclose the filter, the variable should not be quoted at all: `Get-ADUser -Filter {Name -like $UserName}`.

Note: PowerShell wildcards other than *, such as ?, are not supported by the *Filter* syntax.

Obtenir des informations sur la forêt avec Get-ADForest

Get-ADForest

Trouvez la valeur des propriétés suivantes:

Nom de la propriété	Valeur de la propriété
ForestMode	Windows2016Forest
DomainNamingMaster	SERVEUR1.FORMATION.LOCAL
GlobalCatalogs	{SERVEUR1.FORMATION.LOCAL}
SchemaMaster	SERVEUR1.FORMATION.LOCAL

Obtenir des informations sur le domaine avec Get-ADDomain

Get-ADDomain

Trouvez la valeur des propriétés suivantes:

Nom de la propriété	Valeur de la propriété
DistinguishedName	DC=FORMATION,DC=LOCAL
DNSRoot	FORMATION.LOCAL
DomainMode	Windows2016Domain
Name	FORMATION
Forest	FORMATION.LOCAL

La commande qui retourne SEULEMENT la valeur de la propriété DistinguishedName de votre domaine:
`(Get-ADDomain).DistinguishedName`

La commande qui retourne SEULEMENT la valeur de la propriété DNSRoot de votre domaine:
`(Get-ADDomain).DNSRoot`

La commande qui retourne SEULEMENT la valeur du SID (Security Identifier) d'un domaine Active Directory.
`(Get-ADDomain).DomainSID.Value`

Get-ADDefaultDomainPasswordPolicy

La commande affiche les propriétés sur la configuration des mots de passe au niveau du domaine.

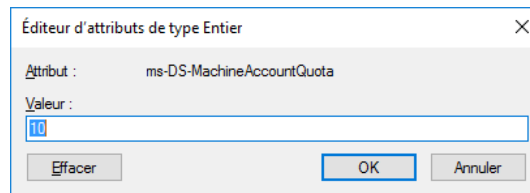
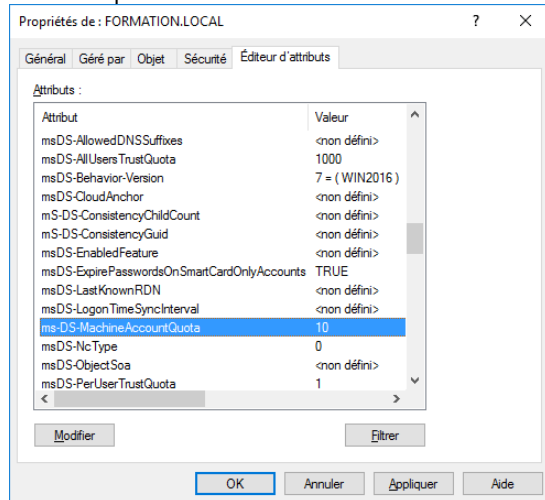
`Get-ADDefaultDomainPasswordPolicy`

Empêcher un utilisateur de joindre un ordinateur au domaine avec Set-ADDomain

Par défaut, un utilisateur authentifié qui n'est pas membre du groupe Administrateurs peut joindre 10 ordinateurs au domaine.

Dans la console **UOAD**

- Afficher les attributs du domaine **FORMATION.LOCAL** et sélectionner l'onglet "Éditeur d'attributs"
- Sélectionner l'attribut **ms-DS-MachineAccountQuota**
- La valeur par défaut de l'attribut **ms-DS-MachineAccountQuota** est 10



Pour empêcher un utilisateur authentifié de joindre des ordinateurs au domaine on doit modifier la valeur de l'attribut **ms-DS-MachineAccountQuota** pour 0.

Par programmation PowerShell

```
Set-ADDomain -Identity ((Get-ADDomain).DistinguishedName) `
-Replace @{'ms-DS-MachineAccountQuota'=0}
```

Obtenir des informations sur les unités d'organisation d'un domaine avec Get-ADOrganizationalUnit

La commande qui affiche les principales propriétés de l'unité d'organisation FORMATION

```
Get-ADOrganizationalUnit -Identity "OU=formation,DC=formation,DC=local"
```

La commande qui affiche toutes les propriétés de l'unité d'organisation FORMATION

```
Get-ADOrganizationalUnit -Identity "OU=formation,DC=formation,DC=local" `
    -Properties *
```

La commande qui affiche les principales propriétés de toutes les unités d'organisation de votre domaine

```
Get-ADOrganizationalUnit -Filter *
```

La commande qui affiche toutes les propriétés de toutes les unités d'organisation de votre domaine

```
Get-ADOrganizationalUnit -Filter * `
    -Properties *
```

La commande qui affiche les principales propriétés des unités d'organisation dont le nom contient "gestion"

```
Get-ADOrganizationalUnit -Filter {Name -like "*gestion*"}
```

La commande qui affiche les principales propriétés des unités d'organisation dont le nom commence par "inf"

```
Get-ADOrganizationalUnit -Filter {Name -like "inf*"}
```

La commande qui affiche les principales propriétés des unités d'organisation dont le nom est similaire à "gestionnaire?"

Le paramètre -Filter ne fonctionne pas avec le caractère ? qui remplace un seul caractère.

```
et-ADOrganizationalUnit -Filter {Name -like "gestionnaire?"}
```

SOLUTION À CE PROBLÈME

On doit obligatoirement utiliser **Where-Object** qui s'exécute sur le résultat de la commande.

```
Get-ADOrganizationalUnit -Filter * | Where-Object { $PSItem.Name -like "gestionnaire?" }
```

La commande qui affiche les propriétés "Name" et "DistinguishedName" de toutes les unités d'organisation de votre domaine

```
Get-ADOrganizationalUnit -Filter * | Format-Table Name,DistinguishedName -AutoSize
```

La commande qui affiche les propriétés "Name" et "Created" de toutes les unités d'organisation de votre domaine

```
Get-ADOrganizationalUnit -Filter * `
    -Properties created | Format-Table Name,Created -AutoSize
```

La commande qui affiche en ordre alphabétique la propriété CanonicalName de toutes les unités d'organisation de votre domaine

```
Get-ADOrganizationalUnit -Filter * `
    -Properties CanonicalName `
    | Select-Object -Property CanonicalName `
    | Sort-Object CanonicalName
```

Le paramètre -Filter et l'attribut DistinguishedName

Exemple 1

Trouver l'unité d'organisation dont le DistinguishedName est ou=formation,dc=formation,dc=local

Commande PowerShell avec le paramètre -Identity

```
$OU = "ou=formation,dc=formation,dc=local"
```

```
Get-ADOrganizationalUnit -Identity $OU
```

Trouver l'unité d'organisation dont le DistinguishedName est ou=formation,dc=formation,dc=local

Commande PowerShell avec le paramètre -Filter

```
$OU = "ou=formation,dc=formation,dc=local"
```

```
Get-ADOrganizationalUnit -Filter {DistinguishedName -eq $OU}
```

La commande avec le paramètre -Filter fonctionne parce qu'on vérifie avec -eq et que \$OU ne contient pas le caractère générique *

Exemple 2

Trouver la liste des unités d'organisation dont le DistinguishedName se termine par ou=formation,dc=formation,dc=local

```
$OU = "*ou=formation,dc=formation,dc=local"
```

```
Get-ADOrganizationalUnit -Filter {DistinguishedName -like $OU}
```

La commande avec le paramètre -Filter ne fonctionne pas parce qu'on vérifie avec -like et que \$OU contient le caractère générique *

Vous ne pouvez pas utiliser de caractères génériques lors du filtrage basé sur le DistinguishedName.

SOLUTION À CE PROBLÈME

On doit obligatoirement utiliser **Where-Object** qui s'exécute sur le résultat de la commande.

```
$OU = "*ou=formation,dc=formation,dc=local"
```

```
Get-ADOrganizationalUnit -Filter * | `
    Where-Object {$PSItem.DistinguishedName -like $OU}
```

Obtenir des informations sur les ordinateurs d'un domaine avec Get-ADComputer

La commande qui affiche les principales propriétés de l'ordinateur SERVEUR2.

```
Get-ADComputer -Identity "SERVEUR2"
```

La commande qui affiche toutes les propriétés de l'ordinateur SERVEUR2.

```
Get-ADComputer -Identity "SERVEUR2" `
-Properties *
```

Chaque ordinateur a un mot de passe qui est géré par le contrôleur de domaine.

Le mot de passe est automatiquement modifié tous les 30 jours.

PasswordLastSet affiche la date à laquelle le mot de passe de l'ordinateur a été modifié.

```
Get-ADComputer -Identity SERVEUR2 `
-Properties PasswordLastSet
```

La commande qui affiche les principales propriétés des ordinateurs dont le nom commence par "SERVEUR"

```
Get-ADcomputer -Filter {Name -like "SERVEUR*"}
```

La commande qui affiche SEULEMENT le nom de tous les ordinateurs de votre domaine

```
(Get-ADComputer -Filter *) .Name
```

La commande qui affiche les principales propriétés des ordinateurs dont le nom est similaire à "SERVEUR?"

Le paramètre -Filter ne fonctionne pas avec le caractère ? qui remplace un seul caractère.

```
Get-ADcomputer -Filter {Name -like "SERVEUR?"}
```

SOLUTION À CE PROBLÈME

On doit obligatoirement utiliser **Where-Object** qui s'exécute sur le résultat de la commande.

```
(Get-ADComputer -Filter * | Where-Object { $PSItem.Name -like "SERVEUR?" }).Name
```

La commande qui affiche seulement six propriétés de l'ordinateur SERVEUR2

```
Get-ADComputer -Identity SERVEUR2 `
-Properties IPv4Address,OperatingSystem,OperatingSystemVersion | `
Format-List Name,DNSHostName,SamAccountName,
IPv4Address,OperatingSystem,OperatingSystemVersion
```

```
Name : SERVEUR2
DNSHostName : SERVEUR2.FORMATION.LOCAL
SamAccountName : SERVEUR2$
IPv4Address : 192.168.1.20
OperatingSystem : Windows Server 2019 Datacenter
OperatingSystemVersion : 10.0 (17763)
```

Le SamAccountName d'un ordinateur de l'Active Directory se termine toujours par un \$.

Voici une commande qui affiche plusieurs propriétés de tous les ordinateurs de votre domaine en triant deux paramètres en ordre croissant et en triant un paramètre en ordre décroissant.

```
Get-ADComputer -Filter * `
    -Properties * | `
    Sort-Object -Property @{Expression = "Description"; Ascending = $true},
        @{Expression = "OperatingSystem"; Ascending = $true},
        @{Expression = "OperatingSystemVersion"; Descending = $true} | `
    Format-Table Description,Name,OperatingSystem,OperatingSystemVersion,WhenChanged -AutoSize
```

Voici le code qui permet d'afficher le nom de l'utilisateur qui a joint un ordinateur à l'Active Directory.

Si l'utilisateur a une délégation pour créer des objets dans l'Active Directory ou est un membre du groupe "Admins du domaine" alors mS-DS-CreatorSID est vide.

```
$nom = Get-ADComputer -Identity SERVEUR2 `
    -Properties mS-DS-CreatorSID | `
    Select-Object -Expandproperty mS-DS-CreatorSID | `
    Select-Object -ExpandProperty Value | `
    Foreach-Object {Get-ADUser -Filter {SID -eq $_PSItem}}

if ([string]::IsNullOrEmpty($nom))
{
    Write-Host "mS-DS-CreatorSID est vide" -ForegroundColor Yellow
}
else
{
    Write-Host "mS-DS-CreatorSID correspond à $nom" -ForegroundColor Cyan
}
```

Obtenir des informations sur les utilisateurs d'un domaine avec Get-ADUser

La commande qui affiche les principales propriétés de l'utilisateur dont le nom d'ouverture de session est EMP01

```
Get-ADUser -Identity "EMP01"
```

La commande qui affiche toutes les propriétés de l'utilisateur dont le nom d'ouverture de session est EMP01

```
Get-ADUser -Identity "EMP01" `
    -Properties *
```

Le nom de l'utilisateur dont le SID se termine par 500 varie selon la langue.

- En français, le nom de l'utilisateur est "Administrateur".
- En anglais, le nom de l'utilisateur est "Administrator".
- En espagnol, le nom de l'utilisateur est "Administrador".
- ...

```
$rep = Get-ADUser -Filter * | Where-Object { $PSItem.SID -like "S-1-5-21-*-500" }
$rep.Name
```

La commande qui affiche les principales propriétés des utilisateurs dont le nom de famille est Coutu.

```
Get-ADUser -Filter 'Surname -eq "Coutu"'
```

La commande qui affiche les principales propriétés des utilisateurs dont le nom d'ouverture de session débute par "EMP".

```
Get-ADUser -Filter {SamAccountName -like "EMP*"}
```

La commande qui affiche les principales propriétés des utilisateurs dont le nom d'ouverture de session est similaire à "EMP3?"

Le paramètre -Filter ne fonctionne pas avec le caractère ? qui remplace un seul caractère.

```
Get-ADUser -Filter {SamAccountName -like "EMP3?"}
```

SOLUTION À CE PROBLÈME

On doit obligatoirement utiliser **Where-Object** qui s'exécute sur le résultat de la commande.

```
Get-ADUser -Filter * | Where-Object { $PSItem.SamAccountName -like "EMP3?" }
```

La commande qui affiche les principales propriétés des utilisateurs qui sont dans l'unité d'organisation "FORMATION" qui est directement sous le domaine "FORMATION.LOCAL".

```
$sb= "OU=formation,DC=formation,DC=local"
```

```
Get-ADUser -Filter * -SearchBase $sb -SearchScope Subtree
ou
Get-ADUser -Filter * -SearchBase $sb
# Subtree est la valeur par défaut de SearchScope
```

Voici le code qui permet d'afficher les utilisateurs dont le nom débute par 0,1,2,3,4,5,6,7,8,9.

```
for ($i = 0; $i -le 9; $i++)  
{  
    $nom = -join ($i, "*")  
    Get-ADUser -Filter {Name -like $nom}  
}
```

Trouver les utilisateurs dans l'Active Directory qui ont une "adresse de messagerie" en utilisant -Filter.

```
$sb = (Get-ADDomain).DistinguishedName  
  
Get-ADUser -SearchBase $sb -Filter {mail -like "*"} -Properties *  
ou  
Get-ADUser -SearchBase $sb -Filter * -Properties * | `  
    Where-Object {$PSItem.mail -like "*"}
```

Trouver les utilisateurs dans l'Active Directory qui ont un "gestionnaire" en utilisant -Filter.

L'attribut "manager" contient le "DistinguishedName" du gestionnaire.

Vous ne pouvez pas utiliser de caractères génériques lors du filtrage basé sur le DistinguishedName.

SOLUTION À CE PROBLÈME

On doit obligatoirement utiliser **Where-Object** qui s'exécute sur le résultat de la commande.

```
$sb = (Get-ADDomain).DistinguishedName  
  
Get-ADUser -SearchBase $sb -Filter * -Properties * | `  
    Where-Object {$PSItem.manager -ne $null}
```


Obtenir des informations sur les groupes d'un domaine avec Get-ADGroup

La commande qui affiche les principales propriétés du groupe Administrateurs

```
Get-ADGroup -Identity "Administrateurs"
```

La commande qui affiche toutes les propriétés du groupe Administrateurs

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

Le nom du groupe dont le SID se termine par 513 varie selon la langue

- En français le nom du groupe est "Utilisateurs du domaine"
- En anglais le nom du groupe est "Domain Users"
- ...

```
$rep = Get-ADGroup -Filter * | Where-Object { $PSItem.SID -like "S-1-5-21-*-513" }
$rep.Name
```

La commande qui affiche les principales propriétés des groupes dont le nom commence par "Adm"

```
Get-ADGroup -Filter {Name -like "Adm*"}
```

La commande qui affiche SEULEMENT le nom des groupes dont le nom commence par "Adm".

```
(Get-ADGroup -Filter {Name -like "Adm*"}).Name
```

La commande qui affiche les principales propriétés des groupes dont le nom est similaire à "gr??"

Le paramètre -Filter ne fonctionne pas avec le caractère ? qui remplace un seul caractère.

```
Get-ADGroup -Filter {Name -like "gr??"}
```

SOLUTION À CE PROBLÈME

On doit obligatoirement utiliser **Where-Object** qui s'exécute sur le résultat de la commande.

```
Get-ADGroup -Filter * | Where-Object { $PSItem.Name -like "gr??" }
```

Obtenir des informations sur des objets de l'Active Directory avec Get-ADObject

Il est souvent plus avantageux d'utiliser Get-ADForest, Get-ADDomain, Get-ADOrganizationalUnit, Get-ADComputer, Get-ADUser et Get-ADGroup.

Dans certaines situations, nous devons utiliser Get-ADObject.

Il n'est pas possible d'afficher le contenu de l'attribut ms-DS-MachineAccountQuota du domaine en utilisant Get-ADDomain parce que les paramètres -Filter et -Properties n'existent pas.

Requête qui permet d'afficher le contenu de l'attribut ms-DS-MachineAccountQuota du domaine

```
Get-ADObject -Identity ((Get-ADDomain).DistinguishedName) `
-Properties ms-DS-MachineAccountQuota
```

Modifier des informations sur des objets de l'Active Directory avec Set-ADObject

Il est souvent plus avantageux d'utiliser Set-ADForest, Set-ADDomain, Set-ADOrganizationalUnit, Set-ADComputer, Set-ADUser et Set-ADGroup.

Dans certaines situations, nous devons utiliser Set-ADObject.

Il n'est pas possible de protéger les utilisateurs, les groupes et les ordinateurs contre une suppression accidentelle en utilisant Get-ADComputer, Get-ADUser, Get-ADGroup.

Requête qui permet de protéger l'ordinateur SERVEUR2 d'une suppression accidentelle.

```
Get-ADComputer -Identity SERVEUR2 | Set-ADObject -ProtectedFromAccidentalDeletion:$true
```

Requête qui permet de protéger l'utilisateur TECH d'une suppression accidentelle.

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

Requête qui permet de protéger le groupe grFormation d'une suppression accidentelle.

```
Get-ADGroup -Identity grFormation | Set-ADObject -ProtectedFromAccidentalDeletion:$true
```

Requête qui permet de trouver tous les objets du domaine dont le nom débute par "Adm"

```
$sb = "DC=formation,DC=local"
```

```
(Get-ADObject -SearchBase $sb -Filter {Name -like "Adm*"}) | `
Sort-Object ObjectClass | `
Format-Table ObjectClass,DistinguishedName -AutoSize
```

Déplacer des objets de l'Active Directory avec Move-ADObject

Pour déplacer un objet de l'Active Directory, il faut utiliser Move-ADObject.

```
# Nous avons besoin du DistinguishedName de l'objet et
# du DistinguishedName du nouvel emplacement.
$ordi = "CN=S9,OU=WEB,OU=SERVEURS,OU=FORMATION,DC=FORMATION,DC=LOCAL"
$emplacement = "OU=SQL,OU=SERVEURS,OU=FORMATION,DC=FORMATION,DC=LOCAL"

Move-ADObject -Identity $ordi -TargetPath $emplacement
```

Le nom d'un utilisateur est unique dans l'Active Directory.
Le nom d'un groupe est unique dans l'Active Directory.
Le nom d'un ordinateur est unique dans l'Active Directory.

```
# Nous avons besoin du DistinguishedName du nouvel emplacement.
$emplacement = "OU=SQL,OU=SERVEURS,OU=FORMATION,DC=FORMATION,DC=LOCAL"

Get-ADComputer -Identity S9 | Move-ADObject -TargetPath $emplacement
```

Pour le déplacement d'une unité d'organisation, nous devons utiliser le DistinguishedName.
Le nom d'une unité d'organisation n'est pas unique dans l'Active Directory.

Le déplacement d'une unité d'organisation implique le déplacement de tous les objets qui sont dans l'unité d'organisation qui sera déplacée.

```
# Nous avons besoin du DistinguishedName de l'unité d'organisation et
# du DistinguishedName du nouvel emplacement.
$OU = "OU=SRVTEST,DC=FORMATION,DC=LOCAL"
$emplacement = "OU=SERVEURS,OU=FORMATION,DC=FORMATION,DC=LOCAL"

Move-ADObject -Identity $OU -TargetPath $emplacement
```

ANNEXE 1

Utilisation du paramètre -LDAPFilter avec Get-ADUser

Il est possible d'effectuer des requêtes en utilisant la syntaxe LDAP (Lightweight Directory Access Protocol).

- LDAP est un protocole ouvert et multiplateforme utilisé pour l'authentification des services d'annuaire.
- LDAP est un moyen de communiquer avec Active Directory.

Get-ADUser permet d'utiliser le paramètre **-LDAPFilter**.

Les opérateurs sont différents selon l'utilisation du paramètre -Filter ou l'utilisation du paramètre -LDAPFilter.

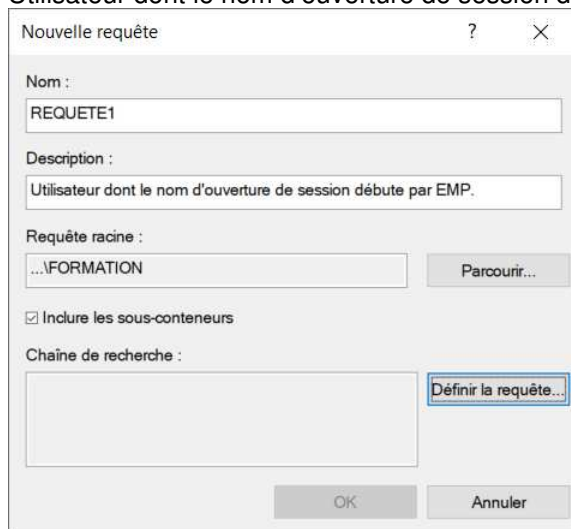
Les opérateurs du paramètre -Filter	Les opérateurs du paramètre -LDAPFilter
-eq	=
-ne	! x=y
-le	<=
-lt	! x >= y
-ge	>=
-gt	! x <= y
-like	=
-notlike	! x = y
-and	&
-or	
-not	!

Utilisation de "Requêtes enregistrées" pour générer le code LDAP



Dans la console UOAD, "**Requêtes enregistrées**" permet de générer le code LDAP d'une requête.

Utilisateur dont le nom d'ouverture de session débute par EMP.

A screenshot of the 'Nouvelle requête' (New Query) dialog box. It has a title bar with a question mark and a close button. The 'Nom :' field contains 'REQUETE1'. The 'Description :' field contains 'Utilisateur dont le nom d'ouverture de session débute par EMP.'. The 'Requête racine :' field contains '...FORMATION' and has a 'Parcourir...' button next to it. There is a checkbox 'Inclure les sous-conteneurs' which is checked. The 'Chaîne de recherche :' field is empty, and there is a 'Définir la requête...' button next to it. At the bottom are 'OK' and 'Annuler' buttons.

Rechercher Recherche personnalisée

Rechercher : Recherche personnalisée

Recherche personnalisée Avancé

Champ	Condition	Valeur

Liste des conditions :

Nom d'ouvertur...	Commence par	EMP

Ajouter Supprimer

OK Annuler

Nouvelle requête

Nom : REQUETE1

Description : Utilisateur dont le nom d'ouverture de session débute par EMP.

Requête racine : ...\\FORMATION Parcourir...

☒ Inclure les sous-conteneurs

Chaîne de recherche : (&(objectCategory=user)(objectClass=user)(userPrincipalName=EMP*)) Définir la requête...

OK Annuler

`(&(objectCategory=user)(objectClass=user)(userPrincipalName=EMP*))`

Voici plusieurs exemples qui utilisent le paramètre -LDAPFilter avec Get-ADUser

```
$sb = (Get-ADDomain).DistinguishedName  
$user_LDAP = "&(objectCategory=user) (objectClass=user) "
```

le nom d'ouverture de session débute par EMP

```
$q1 = "(" + $user_LDAP +  
      "(userPrincipalName=EMP*)" +  
      ")"
```

```
Get-ADUser -SearchBase $sb -LDAPFilter $q1 -Properties *
```

le nom d'ouverture de session débute par EMP **ET** l'adresse de messagerie se termine par @formation.local

```
$q2 = "(" + $user_LDAP +  
      "(userPrincipalName=EMP*) (mail=*@formation.local)" +  
      ")"
```

```
Get-ADUser -SearchBase $sb -LDAPFilter $q2 -Properties *
```

le nom = Richard **OU** le nom = Michelle **OU** le nom = Patrick

```
$q3 = "(" + $user_LDAP +  
      "(|(cn=Richard) (cn=Michelle) (cn=Patrick))" +  
      ")"
```

```
Get-ADUser -SearchBase $sb -LDAPFilter $q3 -Properties *
```

le prénom = Richard **ET** (la ville = Laval **OU** la ville = Verdun)

```
$q4 = "(" + $user_LDAP +  
      "(&(givenName=EMP01) (|(l=Laval) (l=Verdun)))" +  
      ")"
```

```
Get-ADUser -SearchBase $sb -LDAPFilter $q4 -Properties *
```

La liste des utilisateurs qui n'ont pas de gestionnaire en utilisant -LDAPFilter

```
$q5 = "(" + $user_LDAP +  
      "(!(manager=*))" +  
      ")"
```

```
Get-ADUser -SearchBase $sb -LDAPFilter $q5 -Properties *
```

ANNEXE 2

Utilisation de LDAP pour rechercher un ou plusieurs utilisateurs de l'Active Directory

Les deux exemples n'utilisent pas le module ActiveDirectory.

Les deux exemples utilisent la classe "System.DirectoryServices.DirectorySearcher" du ".NET Framework".

```
# Code pour rechercher l'utilisateur EMP01 qui est dans la OU "FORMATION"
# $nom correspond au nom d'ouverture de session
$nom = " EMP01"

# ADSI signifie (Active Directory Service Interfaces)
$root = [ADSI]"LDAP://OU=formation,DC=formation,DC=local"

$searcher = New-Object -TypeName System.DirectoryServices.DirectorySearcher -ArgumentList ($root)
$searcher.filter = "(&(objectCategory=person)(objectClass=User)(name=$nom))"

$resultat = ($searcher.findOne()).GetDirectoryEntry()

$login      = $user.sAMAccountName
$prenom     = $user.givenName
$nom        = $user.sn
$nomComplet = $user.displayName

$info = "$login`t$prenom`t$nom`t$nomComplet"
Write-Host $info -ForegroundColor Yellow

$info = "-" * 80
Write-Host $info -ForegroundColor Cyan
```

```
# Code pour rechercher les utilisateurs dont le nom débute par EMP
# et qui sont dans la OU "FORMATION"
# $nom correspond au nom d'ouverture de session
$nom = "EMP*"

# ADSI signifie (Active Directory Service Interfaces)
$root = [ADSI]"LDAP://OU=formation,DC=formation,DC=local"

$searcher = New-Object -TypeName System.DirectoryServices.DirectorySearcher # -ArgumentList ($root)
$searcher.filter = "(&(objectCategory=person)(objectClass=User)(name=$nom))"

$searcher.findAll() | ForEach-Object {
    $user = $PSItem.GetDirectoryEntry()

    $login      = $user.sAMAccountName
    $prenom     = $user.givenName
    $nom        = $user.sn
    $nomComplet = $user.displayName

    $info = "$login`t$prenom`t$nom`t$nomComplet"
    Write-Host $info -ForegroundColor Yellow

    $info = "-" * 80
    Write-Host $info -ForegroundColor Cyan
}
```

Pour le cours, il n'est pas nécessaire de comprendre le contenu des annexes 1 et 2.

PowerShell - WMI - CIM

Vous devez exécuter les commandes sur le serveur réel

Objectifs

- Utilisation de **Get-CimInstance** et de **Set-CimInstance** pour utiliser les classes du namespace **ROOT\CIMV2**
- Utiliser les cmdlets **Format-Table** et **Format-List**
note: **-AutoSize** et **-Wrap** sont deux paramètres du cmdlet **Format-Table**

Documentation

Computer System Hardware Classes

<https://learn.microsoft.com/en-us/windows/win32/cimwin32prov/computer-system-hardware-classes>

Operating System Classes

<https://learn.microsoft.com/en-us/windows/win32/cimwin32prov/operating-system-classes>

Outils pour explorer les classes WMI

WMI Explorer 2.0.0.2

<https://github.com/vinaypamnani/wmie2>

Les avantages de CIM

- **CIM** est basé sur des standards ouverts définis par la DMTF (Distributed Management Task Force).
- Les cmdlets **CIM** utilisent le protocole **WS-Man** (Web Services for Management) pour la communication, qui est plus moderne et sécurisé.

Les désavantages de WMI

- **WMI** est spécifique à Windows.
- **WMI** utilise **DCOM** (Distributed Component Object Model), qui est plus ancien et peut être plus compliqué à configurer et à sécuriser à travers des pare-feux.

Les classes Win32*

Pour afficher le nom des classes Win32_* en ordre alphabétique.

```
(Get-CimClass -Namespace "root\cimv2" -ClassName "Win32_*").CimClassName | Sort-Object
```

Pour afficher le nombre de classes Win32_*.

```
(Get-CimClass -Namespace "root\cimv2" -ClassName "Win32_*").Count
```

Il existe près de huit cents classes Win32 sous "root\cimv2".

Avantage de CimInstance versus WmiObject

Utilisation de Win32_OperatingSystem pour afficher la date d'installation du système d'exploitation

Retourne la date d'installation du système d'exploitation

```
$rep1 = (Get-WmiObject -Class Win32_OperatingSystem).InstallDate
```

\$rep1 contient "20160910092428.000000-240"

Utilisation de Win32_OperatingSystem pour afficher la date d'installation du système d'exploitation

Retourne la date d'installation du système d'exploitation

```
$rep2 = (Get-CimInstance -ClassName Win32_OperatingSystem).InstallDate
```

\$rep2 contient "10 septembre 2016 09:24:28"

Conclusion: La date est plus facile à lire avec Get-CimInstance.

Exemples avec Win32_OperatingSystem

Voici la commande pour afficher toutes les propriétés de Win32_OperatingSystem
`Get-CimInstance -ClassName Win32_OperatingSystem | Select-Object *`

Voici trois requêtes PowerShell qui permettent de vérifier si le système d'exploitation est 'Microsoft Windows Server 2019 Datacenter' et si la version est '10.0.17763'.

```
Get-CimInstance -ClassName Win32_OperatingSystem | `
    Where-Object { $PSItem.Caption -eq 'Microsoft Windows Server 2019 Datacenter' -and
                  $PSItem.Version -eq '10.0.17763' }
```

```
Get-CimInstance -ClassName Win32_OperatingSystem `
    -Filter "(Caption = 'Microsoft Windows Server 2019 Datacenter' and
            Version = '10.0.17763')"
```

```
Get-CimInstance -Query "select * from Win32_OperatingSystem `
                        where (Caption = 'Microsoft Windows Server 2019 Datacenter' and
                               Version = '10.0.17763')"
```

Exemples avec Win32_OperatingSystem

Voici la commande pour afficher la propriété "Description" de Win32_OperatingSystem
`Get-CimInstance -ClassName Win32_OperatingSystem | Select-Object Description`

Voici trois requêtes PowerShell qui permettent de vérifier la description d'un ordinateur
`Get-CimInstance -ClassName Win32_OperatingSystem | Where-Object Description -eq "ORDI1"`

```
Get-CimInstance -ClassName Win32_OperatingSystem -Filter "Description = 'ORDI1'"
```

```
Get-CimInstance -Query "Select * from Win32_OperatingSystem where Description = 'ORDI1'"
```

Exemples avec Win32_OperatingSystem

Exemple de l'utilisation de Set-CimInstance avec la classe Win32_OperatingSystem pour modifier la description d'un ordinateur.

Voici quatre requêtes PowerShell qui permettent de modifier la description d'un ordinateur

```
# EXEMPLE 1
$nom = Get-CimInstance -ClassName Win32_OperatingSystem
$nom.Description = "ORDI1"
Set-CimInstance -CimInstance $nom
```

```
# EXEMPLE 2
$nom = Get-CimInstance -ClassName Win32_OperatingSystem
Set-CimInstance -CimInstance $nom -Property @{Description="ORDI2"}
```

```
# EXEMPLE 3
Set-CimInstance -Query 'Select * from Win32_OperatingSystem' `
    -Property @{Description="ORDI3"}
```

```
# EXEMPLE 4
Get-CimInstance -ClassName Win32_OperatingSystem | `
    Set-CimInstance -Property @{Description = "ORDI4"}
```

Exemples avec Win32_ComputerSystem

```
# Voici la commande pour afficher toutes les propriétés de Win32_ComputerSystem
Get-CimInstance -ClassName Win32_ComputerSystem | Select-Object *
```

Voici la commande qui affiche

- Le nom de votre ordinateur
- Le nom du domaine ou du groupe de travail
- Le nom de l'utilisateur qui est connecté pour exécuter le test
- L'affichage utilise Format-Table avec le paramètre -AutoSize

```
Get-CimInstance -ClassName Win32_ComputerSystem | `
    Format-Table -AutoSize Caption,Domain,UserName
```

Exemples avec Win32_Group

```
# Voici la commande pour afficher toutes les propriétés de Win32_Group
Get-CimInstance -ClassName Win32_Group | Select-Object *
```

```
# Voici la commande pour afficher le nom et le SID de tous les groupes du serveur réel
```

- L'affichage utilise Format-Table avec le paramètre -AutoSize

```
Get-CimInstance -ClassName Win32_Group | `
    Format-Table -AutoSize Name,SID
```

Exemples avec Win32_UserAccount

```
# Voici la commande pour afficher toutes les propriétés de Win32_UserAccount
Get-CimInstance -ClassName Win32_UserAccount | Select-Object *
```

```
# Voici la commande pour afficher le nom et le SID de tous les usagers du serveur réel
```

- L'affichage utilise Format-Table avec le paramètre -AutoSize

```
Get-CimInstance -ClassName Win32_UserAccount | `
    Format-Table -AutoSize Name,SID
```

Exemples avec Win32_UserAccount

VERSION 1

NOTE: il n'y a pas de cmdlet pour **déverrouiller** le compte d'un utilisateur local

Voici le code pour déverrouiller le compte d'un utilisateur local

```
$liste = "ETU", "TEST", "TECH"

$users = Get-CimInstance -ClassName Win32_UserAccount `
    -Filter "LocalAccount=True and Lockout=True" | `
    Where-Object Name -in $liste

foreach ($user in $users)
{
    # Affiche le nom de l'utilisateur
    $nom = $user.name
    Write-Warning "Nom de l'utilisateur: $nom"

    # Affiche le contenu de la propriété Lockout avant la modification
    $user.Lockout

    # Modification de la propriété Lockout
    $user.Lockout = "False"

    # Affiche le contenu de la propriété Lockout après la modification
    $user.Lockout

    # Mise à jour de l'utilisateur
    Set-CimInstance -CimInstance $user
}
```

VERSION 2

NOTE: il n'y a pas de cmdlet pour **déverrouiller** le compte d'un utilisateur local

Voici le code pour déverrouiller le compte d'un utilisateur local

```
$liste = "ETU", "TEST", "TECH"

$users = Get-CimInstance -ClassName Win32_UserAccount `
    -Filter "LocalAccount=True and Lockout=True" | `
    Where-Object Name -in $liste

foreach ($user in $users)
{
    # Mise à jour de l'utilisateur pour déverrouiller son compte
    Set-CimInstance -CimInstance $user `
        -Property @{LockOut=$false}

    $nom = $user.name
    Write-Warning "Le compte de l'utilisateur est déverrouillé: $nom"
}
```

Exemples avec Win32_VideoController

Voici la commande pour afficher des informations sur la carte vidéo de votre ordinateur

- Le nom de votre ordinateur
- Le nom du modèle de la carte vidéo
- Le nom du processeur vidéo
- La résolution de l'écran utilisée et le nombre de couleurs
- La quantité de mémoire sur la carte vidéo
- La version du pilote de la carte vidéo

```
Get-CimInstance -ClassName Win32_VideoController | `
    Select-Object -Property SystemName,
                        Name,
                        VideoProcessor,
                        AdapterRAM,
                        VideoModeDescription,
                        CurrentRefreshRate,
                        MaxRefreshRate,
                        DriverDate,
                        DriverVersion | Format-List
```

```
SystemName      : VM70035316
Name            : NVIDIA Quadro M2000
VideoProcessor  : Quadro M2000
AdapterRAM      : 4293918720
VideoModeDescription : 1680 x 1050 x 4294967296 couleurs
CurrentRefreshRate : 59
MaxRefreshRate  : 75
DriverDate      : 2021-02-22 19:00:00
DriverVersion   : 27.21.14.6172
```

Exemples avec Win32_Processor

Voici la commande pour afficher toutes les propriétés de Win32_Processor

```
Get-CimInstance -Class Win32_Processor | Select-Object *
```

Exemples avec Win32_LogicalDisk

Comment afficher la liste des disques locaux présents avec les informations suivantes:

- Leur nom (lettre)
- Leur type
- Leur système de fichier
 - Les lecteurs de disque qui ne contiennent pas de média n'ont pas de système de fichier.
 - Consulter le site <https://learn.microsoft.com/en-us/windows/win32/cimwin32prov/win32-logicaldisk> pour connaître les valeurs de l'attribut DriveType.
- La taille
- L'espace libre
- L'affichage utilise Format-Table avec les paramètres -AutoSize et -Wrap

Voici trois réponses en utilisant trois façons différentes de faire cette requête (where-object, -filter, -query)

```
Get-CimInstance -ClassName Win32_LogicalDisk | Where-Object drivetype -eq 3 | `
    Format-Table -AutoSize -Wrap DeviceID,Description,FileSystem,Size,FreeSpace
```

```
Get-CimInstance -ClassName Win32_LogicalDisk -Filter "DriveType = 3" | `
    Format-Table -AutoSize -Wrap DeviceID,Description,FileSystem,Size,FreeSpace
```

```
Get-CimInstance -Query "select * from Win32_LogicalDisk where DriveType = 3" | `
    Format-Table -AutoSize -Wrap DeviceID,Description,FileSystem,Size,FreeSpace
```

Exemples avec Win32_Service

Comment afficher la liste des services dont l'état est à "démarrer" avec les informations suivantes:

- Le nom du service
- Le nom complet du service
- Le chemin
- Le mode de démarrage
- L'affichage utilise Format-List

Voici trois réponses en utilisant trois façons différentes de faire cette requête (where-object, -filter, -query)

```
Get-CimInstance -ClassName Win32_Service | Where-Object state -eq "Running" | `
    Format-List name,displayname,pathname,startmode
```

```
Get-CimInstance -ClassName Win32_Service -Filter 'state = "running"' | `
    Format-List name,displayname,pathname,startmode
```

```
Get-CimInstance -Query 'select * from Win32_Service where state = "running"' | `
    Format-List name,displayname,pathname,startmode
```

Exemples avec Win32_NetworkConnection

Comment afficher le nom et le chemin des disques réseaux présentement connectés

- L'affichage utilise Format-Table avec le paramètre -AutoSize

```
Get-CimInstance -ClassName win32_NetworkConnection | `
    Format-Table -AutoSize LocalName,RemoteName
```

Comment afficher le nom et le chemin des disques réseaux présentement connectés

- L'affichage utilise Out-GridView

```
Get-CimInstance -ClassName win32_NetworkConnection | `
    Select-Object LocalName,RemoteName | Out-GridView
```

Exemples avec Win32_NetworkAdapter

Comment afficher l'index, l'adresse MAC et le nom des cartes réseaux (NetConnectionID).

- Le paramètre MACAddress ne doit pas être nul
- Le paramètre NetConnectionID ne doit pas être nul
- L'affichage utilise Format-Table avec les paramètres -AutoSize et -Wrap

Voici trois réponses en utilisant trois façons différentes de faire cette requête (where-object, -filter, -query)

```
Get-CimInstance -ClassName Win32_NetworkAdapter | `
    Where-Object { $PSItem.MACAddress -ne $null -and
        $PSItem.NetConnectionID -ne $null } | `
    Format-Table -AutoSize -Wrap Index,MACAddress,NetConnectionID
```

```
Get-CimInstance -ClassName Win32_NetworkAdapter `
    -Filter "MACAddress is not null and NetConnectionID is not null" | `
    Format-Table -AutoSize -Wrap Index,MACAddress,NetConnectionID
```

```
Get-CimInstance -Query "select * from Win32_NetworkAdapter `
    where MACAddress is not null and NetConnectionID is not null" | `
    Format-Table -AutoSize -Wrap Index,MACAddress,NetConnectionID
```

Exemples avec Win32_NetworkAdapterConfiguration

Comment afficher l'index, la description des cartes réseaux, l'adresse MAC, l'adresse IP, le masque de sous-réseau, la passerelle, les adresses des serveurs DNS

- Le paramètre MACAddress ne doit pas être nul
- Le paramètre Description net doit pas être nul
- L'affichage utilise Format-List

Voici trois réponses en utilisant trois façons différentes de faire cette requête (where-object, -filter, -query)

```
Get-CimInstance -ClassName Win32_NetworkAdapterConfiguration | `
    Where-Object { $PSItem.MACAddress -ne $null -and
        $PSItem.Description -ne $null } | `
    Format-List Index,Description,MACAddress,IPAddress,IPSubnet,
        DefaultIPGateway,DNSServerSearchOrder
```

```
Get-CimInstance -ClassName Win32_NetworkAdapterConfiguration `
    -Filter "MACAddress is not null and Description is not null" | `
    Format-List Index,Description,MACAddress,IPAddress,IPSubnet,
        DefaultIPGateway,DNSServerSearchOrder
```

```
Get-CimInstance -Query "select * from Win32_NetworkAdapterConfiguration `
    where MACAddress is not null and Description is not null" | `
    Format-List Index,Description,MACAddress,IPAddress,IPSubnet,
        DefaultIPGateway,DNSServerSearchOrder
```

Exemples avec Win32_BIOS

Les propriétés de **Win32_BIOS** sont en lecture seulement.

```
Clear-Host
$computer = $env:COMPUTERNAME
$namespace = "ROOT\CIMV2"
$classname = "Win32_BIOS"

Write-Output "====="
Write-Output "Computer : $computer"
Write-Output "NameSpace : $namespace"
Write-Output "ClassName : $classname"
Write-Output "====="

Get-CimInstance -Namespace $namespace -ClassName $classname
```


ANNEXE 1

Modules PowerShell pour modifier les propriétés du BIOS

La compagnie "Hewlett Packard Enterprise" offre le module PowerShell "HPEBIOSCmdlets 4.0.0.0" pour administrer le BIOS/UEFI des serveurs.

<https://www.powershellgallery.com/packages/HPEBIOSCmdlets>

HPEBIOSCmdlets 4.0.0.0

Scripting Tools for Windows PowerShell : BIOS Cmdlets creates an interface to HPE BIOS ROM-Based Setup Utility (RBSU) or UEFI System Utilities. These cmdlets can be used to configure the BIOS settings on HPE ProLiant servers.

Le module "HPEBIOSCmdlets" est offert avec des scripts pour faciliter la gestion du BIOS/UEFI.

- ConfigureAdminInfo.ps1
- ConfigureAMDCorePerformanceBoosting.ps1
- ConfigureBIOSAdminPassword.ps1
- ConfigureBootMode.ps1
- ConfigureBootOrder.ps1
- ConfigureEMSConsoleAndSerialPort.ps1
- ConfigureIntelCoreBoosting.ps1
- ConfigureIntelTurboBoost.ps1
- ConfigureNetworkBootsettings.ps1
- ConfigureNVDIMMConfiguration.ps1
- ConfigurePCIDeviceWithPCIeLinkSpeedForGen10.ps1
- ConfigurePowerOnPassword.ps1
- ConfigureProcessorJitterControl.ps1
- ConfigureProcessorPower.ps1
- ConfigureServerAvailability.ps1
- ConfigureServerSecurity.ps1
- ConfigureThermalAndFanOption.ps1
- ConfigureTPM.ps1
- ConfigureUEFIOptimizedBoot.ps1
- ConfigureVirtualInstallDisk.ps1
- ConfigureWorkloadProfileForGen10servers.ps1
- ResetBIOSAdminPassword.ps1
- ResetBIOSDefaultManufacturingSettings.ps1
- ResetPowerOnPassword.ps1

La compagnie DELL offre le module PowerShell "DellBIOSProvider 2.8.0" pour administrer le BIOS des ordinateurs Dell Optiplex, Latitude, Precision, XPS Notebook et Venue 11.

<https://www.powershellgallery.com/packages/DellBIOSProvider>

DellBIOSProvider 2.8.0

The 'Dell Command | PowerShell Provider' provides native configuration capability of Dell Optiplex, Latitude, Precision, XPS Notebook and Venue 11 systems within PowerShell.

ANNEXE 2

Exemples pour trouver la valeur UUID du BIOS

Voici le code pour afficher la valeur UUID du BIOS de chaque ordinateur virtuel.

Le code doit s'exécuter sur le serveur réel.

Le code affiche le BiosGUID même si l'ordinateur virtuel n'est pas démarré.

Clear-Host

```
$VMNames = (Get-VM).Name

ForEach ($VMName in $VMNames)
{
    Get-CimInstance -Namespace Root\Virtualization\V2 `
        -ClassName Msvm_VirtualSystemSettingData `
        -Filter "ElementName = '$VMName'" `
        | Select-Object ElementName, BiosGUID
}
```

Le code doit s'exécuter dans un ordinateur virtuel.

La valeur de UUID est identique à la valeur BiosGUID.

Clear-Host

```
$computerSystemProduct = Get-CimInstance -Namespace root\cimv2 `
    -ClassName Win32_ComputerSystemProduct | Select-Object *
```

```
$computerSystemProduct.UUID
```

Introduction à PowerShell

Objectifs

- Explorer les environnements de programmation de PowerShell
- Maîtriser les cmdlets de bases

PowerShell Documentation
PowerShell Gallery

<https://learn.microsoft.com/en-us/powershell>
<https://www.powershellgallery.com>

Mise en place

Je vous conseille de créer un raccourci sur la barre des tâches, en exécution "administrateur" pour la console

- PowerShell ISE (Attention: choisir la version 64 bits)

Informations sur PowerShell

"Windows 10", "Windows 11", "Windows Server 2016", "Windows Server 2019" et "Windows Server 2022" utilise "Windows PowerShell version 5.1".

"PowerShell 7.4.4" ne remplace pas "Windows PowerShell version 5.1".

"PowerShell 7.4.4" s'installe en parallèle à "Windows PowerShell version 5.1".

"PowerShell 7.4.4" peut s'installer sur Windows, Linux et OSX

- Windows (x64), Windows (x86)
- Debian, Red Hat
- OSX

"PowerShell 7.4.4" est la plus récente version de PowerShell.

"PowerShell 7.4.4" est disponible depuis le 2024-07-18.

"PowerShell 7.4.4" est basé sur ".NET 8 version 8.0.303".

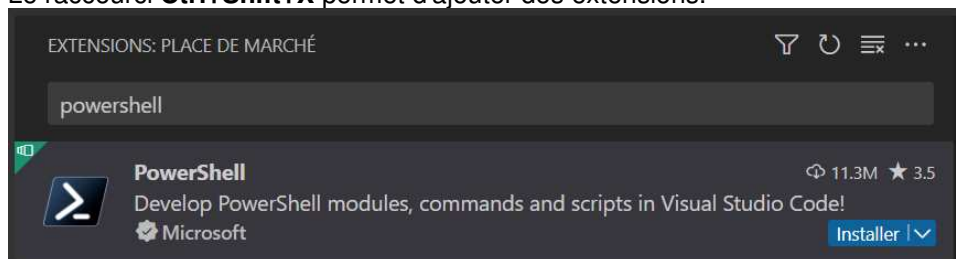
Le code source de "PowerShell 7.4.4" est disponible.

"PowerShell" est programmé en C#.

<https://github.com/PowerShell/PowerShell/tags>

Pour utiliser PowerShell avec "Visual Studio Code", vous devez ajouter l'extension "PowerShell".

Le raccourci **Ctrl+Shift+X** permet d'ajouter des extensions.



Modules supplémentaires

Il existe des modules supplémentaires pour gérer les ressources qui sont sur "**Microsoft Azure**".

Microsoft Azure PowerShell

<https://www.powershellgallery.com/packages/Az>

Pour gérer les ressources qui sont sur "**Microsoft Azure**" vous devez avoir un compte Azure.

Le module Az.Accounts contient deux cmdlets pour se connecter ou se déconnecter.

```
(Get-Command -Name *connect* -Module Az.Accounts).Name
```

```
Connect-AzAccount
```

```
Disconnect-AzAccount
```

Voici la liste des cmdlets que vous pouvez utiliser pour gérer les machines virtuelles sur Azure.

```
(Get-Command -Module Az.Compute -Name *AzVm).Name
```

```
Get-AzVM
```

```
New-AzVM
```

```
Remove-AzVM
```

```
Restart-AzVM
```

```
Set-AzVM
```

```
Start-AzVM
```

```
Stop-AzVM
```

```
Update-AzVM
```

Il existe des modules supplémentaires pour gérer les ressources qui sont sur "**Amazon Web Services**".

Outils AWS pour PowerShell

<https://aws.amazon.com/fr/powershell>

<https://www.powershellgallery.com/packages/AWSPowerShell>

Il existe des modules supplémentaires pour gérer les ressources des serveurs **ESXi** et **vCenter**.

VMware PowerCLI

<https://www.powershellgallery.com/packages/VMware.PowerCLI>

Il existe des modules supplémentaires pour gérer les autorisations NTFS.

NTFSSecurity

<https://www.powershellgallery.com/packages/NTFSSecurity>

Le site "PowerShell Gallery" contient plusieurs milliers de modules.

Introduction à PowerShell

La programmation avec PowerShell permet de manipuler des objets du système d'exploitation.

cmdlet (commandlet)

Un cmdlet est une commande fournie par PowerShell.

Les cmdlets sont écrits en C# ou en VB.NET et sont inclus dans les modules PowerShell.

function

Une fonction PowerShell est une commande personnalisée créée par un utilisateur.

Les fonctions sont écrites en PowerShell.

alias

Un alias est un raccourci vers un cmdlet ou une fonction.

Pour afficher la liste des cmdlet.

Get-Command -CommandType cmdlet

Pour afficher le nombre de cmdlet.

(Get-Command -CommandType cmdlet).Count

Pour afficher la liste des cmdlet qui sont dans un module particulier

Get-Command -Module Hyper-V -CommandType cmdlet

Pour afficher le nom du module qui contient le cmdlet **Get-VM**

(Get-Command -Name Get-VM).ModuleName

Pour afficher la liste des cmdlets qui contiennent **vm**

Get-Command -Name *vm* -CommandType Cmdlet

Pour afficher la liste des fonctions qui contiennent **vm**

Get-Command -Name *vm* -CommandType Function

Pour afficher la liste des alias qui contiennent **vm**

Get-Command -Name *vm* -CommandType Alias

Un cmdlet est constitué d'un verbe (VERB) suivi d'un nom (NOUN).

Pour afficher la liste des verbes utilisés par Windows PowerShell

Get-Verb

Pour afficher la liste des cmdlets, fonctions et alias si le verbe est **convert**

Get-Command -Verb convert

Pour afficher la liste des cmdlets, fonctions et alias si le verbe débute par **convert**

Get-Command -Verb convert*

Pour afficher la liste des cmdlets, fonctions et alias qui contiennent le nom **vm**

Get-Command -Noun *vm*

Pour trouver la relation entre l'alias **ls** et son raccourci

Get-Command -Name ls | Select-Object Name, ResolvedCommandName

Effectuer des calculs avec PowerShell

3 + 2	# le résultat est 5
3 - 2	# le résultat est 1
3 * 2	# le résultat est 6
3 / 2	# le résultat est 1.5
2 / 3	# le résultat est 0,666666666666667
5%4	# le résultat est 1
	# % est l'opérateur MODULO qui calcule le reste de la division
"-" * 80	# affiche 80 tirets
1kb	# affiche 1024
1mb	# affiche 1048576
1gb	# affiche 1073741824
1tb	# affiche 1099511627776
1pb	# affiche 1125899906842624
1gb / 1mb	# le résultat est 1024
1tb / 1gb	# le résultat est 1024
1pb / 1gb	# le résultat est 1048576
2gb * 5	# le résultat est 10737418240
0xffff	# affiche 65535
16 + 0x10	# le résultat est 32

Utilisation du cmdlet Get-Help

Dans PowerShell, on peut afficher de l'aide sur plus d'une centaine de sujets.

La commande suivante permet d'afficher la liste complète des sujets:

```
Get-Help -Name about
```

```
# Commande pour afficher en ordre alphabétique la liste complète des sujets
$info = (Get-Help -Name about).Name | Sort-Object -Unique
$info
"-" * 80
$total = $info.Count
Write-Host "Nombre de sujet = $total" -ForegroundColor Yellow
"-" * 80
```

Voici une liste de plusieurs sujets intéressants:

```
about_Arithmetic_Operators
about_Assignment_Operators
about_Comparison_Operators
about_Logical_Operators
about_Operator_Precedence
about_Operators
about_Type_Operators

about_Do
about_For
about_Foreach
about_If
about_Switch
about_While

about_Arrays
about_Functions
about_Functions_Advanced
about_Functions_Advanced_Methods
about_Functions_Advanced_Parameters
about_Functions_CmdletBindingAttribute
about_Functions_OutputTypeAttribute
about_Hash_Tables
about_Ref
about_Regular_Expressions
about_Scopes
about_Script_Blocks
about_Scripts
about_Try_Catch_Finally
about_Variables
```


Utilisation de plusieurs paramètres de Get-Help

Ces exemples affichent des informations d'aide plus détaillées du cmdlet **Format-Table**.

```
Get-Help -Name Format-Table -Detailed
```

```
Get-Help -Name Format-Table -Full
```

Ces exemples affichent les parties sélectionnées du cmdlet **Format-Table**.

```
Get-Help -Name Format-Table -Examples
```

```
Get-Help -Name Format-Table -Parameter *
```

```
Get-Help -Name Format-Table -Parameter AutoSize
```

Cet exemple montre comment afficher la version en ligne de l'article d'aide du cmdlet **Format-Table** dans votre navigateur web par défaut.

```
Get-Help -Name Format-Table -Online
```

Mise à jour de l'aide dans PowerShell

L'aide de PowerShell est disponible ou sera téléchargée au besoin.

Le cmdlet "**update-help**" permet d'effectuer la mise à jour de l'aide à condition d'avoir une connexion internet. La mise à jour de l'aide peut prendre un certain temps.

Cette commande force la mise à jour de l'aide pour PowerShell.

```
Update-help -Force
```

Utilisation des ALIAS

Un alias remplace le nom d'un cmdlet par un nom très court.

Le cmdlet "**Get-Alias**" permet d'afficher les alias.

Il n'est pas recommandé d'utiliser les alias dans des scripts parce qu'ils peuvent porter à confusion.

Les alias peuvent être difficiles à comprendre en particulier pour les programmeurs débutants.

Le code est plus difficile à maintenir en particulier pour un autre programmeur que l'auteur du script.

Exemple d'alias facile à comprendre

```
clear      Clear-Host
```

```
cp         Copy-Item
```

Plusieurs alias pour le même cmdlet

```
cd         Set-Location
```

```
chdir      Set-Location
```

```
copy       Copy-Item
```

```
cp         Copy-Item
```

```
cpi        Copy-Item
```

Exemple d'alias difficile à comprendre

```
%          ForEach-Object
```

```
?          Where-Object
```

Introduction à la programmation PowerShell

Les opérateurs arithmétiques

`Get-Help -Name about_Arithmetic_Operators`

PowerShell supports the following arithmetic operators:

Operator	Description	Example
+	Adds integers; concatenates strings, arrays, and hash tables.	<code>6 + 2</code> <code>"file" + "name"</code> <code>@(1, "one") + @(2.0, "two")</code> <code>@{"one" = 1} + @{"two" = 2}</code>
-	Subtracts one value from another value	<code>6 - 2</code>
-	Makes a number a negative number	<code>-6</code> <code>(Get-Date).AddDays(-1)</code>
*	Multiply numbers or copy strings and arrays the specified number of times.	<code>6 * 2</code> <code>@("!") * 4</code> <code>"!" * 3</code>
/	Divides two values.	<code>6 / 2</code>
%	Modulus - returns the remainder of a division operation.	<code>7 % 2</code>
-band	Bitwise AND	<code>5 -band 3</code>
-bnot	Bitwise NOT	<code>-bnot 5</code>
-bor	Bitwise OR	<code>5 -bor 0x03</code>
-bxor	Bitwise XOR	<code>5 -bxor 3</code>
-shl	Shifts bits to the left the specified number of times	<code>102 -shl 2</code>
-shr	Shifts bits to the right	<code>102 -shr 2</code>

The bitwise operators only work on integer types.

OPERATOR PRECEDENCE

PowerShell processes arithmetic operators in the following order:

Precedence	Operator	Description
1	()	Parentheses
2	-	For a negative number or unary operator
3	*, /, %	For multiplication and division
4	+, -	For addition and subtraction

PowerShell processes the expressions from left to right according to the precedence rules. The following examples show the effect of the precedence rules:

Expression	Result
<code>3+6/3*4</code>	11
<code>3+6/(3*4)</code>	3.5
<code>(3+6)/3*4</code>	12

Les opérateurs de comparaison

`Get-Help -Name about_Comparison_Operators`

PowerShell includes the following comparison operators:

Type	Operators	Description
Equality	<code>-eq</code> <code>-ne</code> <code>-gt</code> <code>-ge</code> <code>-lt</code> <code>-le</code>	equals not equals greater than greater than or equal less than less than or equal
Matching	<code>-like</code> <code>-notlike</code> <code>-match</code> <code>-notmatch</code>	Returns true when string matches wildcard pattern Returns true when string does not match wildcard pattern Returns true when string matches regex pattern; <code>\$matches</code> contains matching strings Returns true when string does not match regex pattern; <code>\$matches</code> contains matching strings
Containment	<code>-contains</code> <code>-notcontains</code> <code>-in</code> <code>-notin</code>	Returns true when reference value contained in a collection Returns true when reference value not contained in a collection Returns true when test value contained in a collection Returns true when test value not contained in a collection
Replacement	<code>-replace</code>	Replaces a string pattern
Type	<code>-is</code> <code>-isnot</code>	Returns true if both object are the same type Returns true if the objects are not the same type

Les opérateurs logiques

`Get-Help -Name about_Logical_Operators`

PowerShell supports the following logical operators.

Operator	Description	Example
-and	Logical AND. TRUE when both statements are TRUE.	(1 -eq 1) -and (1 -eq 2) False
-or	Logical OR. TRUE when either statement is TRUE.	(1 -eq 1) -or (1 -eq 2) True
-xor	Logical EXCLUSIVE OR. TRUE when only one statement is TRUE	(1 -eq 1) -xor (2 -eq 2) False
-not	Logical not. Negates the statement that follows.	-not (1 -eq 1) False
!	Same as -not	!(1 -eq 1) False

Voici les commandes pour obtenir de l'aide sur les instructions:

- **IF** `Get-Help -Name about_if`
- **FOR** `Get-Help -Name about_for`
- **FOREACH** `Get-Help -Name about_foreach`
- **SWITCH** `Get-Help -Name about_switch`
- **WHILE** `Get-Help -Name about_while`

L'opérateur IF est utilisé pour tester des conditions

```
if ($a -gt 2)
{
    Write-Host "La valeur $a est plus grande que 2."
}

if ($a -gt 2)
{
    Write-Host "La valeur $a est plus grande que 2."
}
else
{
    Write-Host ("La valeur $a est plus petite ou égale à 2," +
        " ou n'existe pas ou n'est pas initialisée.")
}

if ($a -gt 2)
{
    Write-Host "La valeur $a est plus grande que 2."
}
elseif ($a -eq 2)
{
    Write-Host "La valeur $a est égale à 2."
}
else
{
    Write-Host ("La valeur $a est plus petite que 2," +
        " ou n'existe pas ou n'est pas initialisée.")
}
```

L'opérateur FOR est utilisé pour effectuer une boucle

Une boucle FOR s'exécute en utilisant une valeur de départ, un test et un incrément.

La boucle s'exécute 10 fois et affiche les valeurs 1,2,3,4,5,6,7,8,9,10

```
for($i=1; $i -le 10; $i++)  
{  
    Write-Host $i  
}
```

La boucle s'exécute 10 fois et affiche les valeurs 10,9,8,7,6,5,4,3,2,1

```
for($i=10; $i -ge 1; $i--)  
{  
    Write-Host $i  
}
```

L'opérateur FOREACH est utilisé pour parcourir tous les éléments d'une collection

Une boucle FOREACH exécute une itération à partir des valeurs d'une collection.

```
$lettres = "a","b","c","d"  
  
foreach ($lettre in $lettres)  
{  
    Write-Host $lettre  
}
```

L'opérateur SWITCH est utilisé pour évaluer une expression

```
$i = 3  
switch ($i)  
{  
    1 {"La valeur est un."}  
    2 {"La valeur est deux."}  
    3 {"La valeur est trois."}  
    4 {"La valeur est quatre."}  
}
```

La valeur est trois.

```
$i = 3  
switch ($i)  
{  
    1 {"La valeur est un."}  
    2 {"La valeur est deux."}  
    3 {"La valeur est trois."}  
    4 {"La valeur est quatre."}  
    3 {"Encore trois."}  
}
```

La valeur est trois.

Encore trois.

```
# Break permet d'arrêter immédiatement
$i = 3
switch ($i)
{
    1 {"La valeur est un."}
    2 {"La valeur est deux."}
    3 {"La valeur est trois."; Break}
    4 {"La valeur est quatre."}
    3 {"Encore trois."}
}
La valeur est trois.
```

```
# La commande SWITCH teste deux valeurs
switch (4,2)
{
    1 {"La valeur est un."}
    2 {"La valeur est deux."}
    3 {"La valeur est trois."; Break}
    4 {"La valeur est quatre."}
    3 {"Encore trois."}
}
La valeur est quatre.
La valeur est deux.
```

```
# Default est utilisé si aucun teste fonctionne
$i = 5
switch ($i)
{
    1 {"La valeur est un."; Break }
    2 {"La valeur est deux."; Break }
    3 {"La valeur est trois."; Break}
    4 {"La valeur est quatre."; Break }
    Default {"Aucune valeur."}
}
Aucune valeur.
```

```
# La commande switch permet d'utiliser des tests pour valider une valeur
$heure = $(Get-Date).hour
switch ($heure)
{
    { $heure -ge 0 -and $heure -lt 8 } { Write-Host "Nous sommes la nuit." }
    { $heure -ge 8 -and $heure -lt 18 } { Write-Host "Nous sommes le jour." }
    { $heure -ge 18 -and $heure -lt 24 } { Write-Host "Nous sommes le soir." }
}
```

L'opérateur WHILE est utilisé pour effectuer une boucle

Il ne faut pas oublier d'incrémenter la valeur de la variable dans la boucle WHILE.

```
$val =1
while($val -le 3)
{
    Write-Host $val
    $val++
}
```

Les variables

Une variable débute avec \$

- 1) Une variable peut contenir le résultat d'une commande Windows
`$resultat = ping 10.57.22.100`
`$resultat`
- 2) Une variable peut contenir les propriétés d'un objet.
`$col = Get-CimInstance -ClassName win32_processor`
`$col`
`$col.NumberOfCores`
`$col.NumberOfLogicalProcessors`
- 3) Normalement, une variable est en mémoire mais PowerShell permet qu'une variable soit un fichier.
`{C:\Temp\Test.txt} = "Test pour écrire dans un fichier."`
`{C:\Temp\Test.txt} += "`n" + "Ligne 2 !!!"`

Pour afficher la liste des variables d'environnement

- Get-ChildItem env:

Pour afficher le contenu d'une variable d'environnement

On doit ajouter **\$env:** devant le nom de la variable d'environnement de Windows

- \$env:computername

Pour afficher la liste des variables

- Get-Variable

Pour créer une nouvelle variable, variable en lecture seule ou constante

- New-Variable -Name pi -Value ([system.math]::PI) -Option Constant

Pour effacer le contenu d'une variable

- Clear-Variable -Name resultat

Pour supprimer une variable et son contenu

- Remove-Variable -Name resultat

La variable \$?

Le contenu de la variable **\$?** indique si l'exécution de la dernière commande a réussi ou échoué.

Les variables booléennes

- \$true
- \$false

Concaténation des chaînes de caractères

Guillemet simple

La substitution de la variable \$nombre ne fonctionne pas

```
$nombre = 5
```

```
'Nombre = $nombre'
```

```
résultat: Nombre = $nombre
```

Guillemet double

La substitution de la variable \$nombre fonctionne

```
$nombre = 5
```

```
"Nombre = $nombre"
```

```
résultat: Nombre = 5
```

L'opérateur + permet de concaténer des chaînes de caractères

```
$c = 'abc' + 'xyz'
```

```
$c
```

```
résultat: abcxyz
```

L'opérateur + permet de concaténer des variables

```
$c1 = 'abc'
```

```
$c2 = 'xyz'
```

```
$c3 = $c1 + $c2
```

```
$c3
```

```
résultat: abcxyz
```


Les tableaux

Un tableau contient plusieurs valeurs.
Chaque valeur est séparée par une virgule.

```
$data = @() # déclaration d'un tableau vide

$tab1 = @(1,2,3,4,5,6,7,8,9,10) # un tableau qui contient 10 valeurs
$tab1 = 1,2,3,4,5,6,7,8,9,10 # autre syntaxe pour un tableau

$tab2 = @("python","rust","C++") # un tableau qui contient 10 valeurs
$tab2 = "python","rust","C++" # autre syntaxe pour un tableau

$tab3 = @(1..10) # l'opérateur .. permet de générer plusieurs valeurs
$tab3 = 1..10 # autre syntaxe pour un tableau
```

Manipuler un tableau

Le premier élément d'un tableau commence à la position d'index [0].

```
$tab1[0] # affiche le premier élément du tableau $tab1
$tab1[4] # affiche le cinquième élément du tableau $tab1
$tab1[-1] # affiche le dernier élément du tableau $tab1
$tab1[-4..-1] # affiche les quatre derniers éléments du tableau $tab1

Get-Member -InputObject $tab1 # affiche les méthodes disponibles pour le tableau $tab1

$tab1.SetValue(500,5) # change la valeur de l'index 5
# le nouveau contenu de $tab1 est 1,2,3,4,5,500,7,8,9,10

$tab1[5]=6 # le nouveau contenu de $tab1 est 1,2,3,4,5,6,7,8,9,10

$tab1.Contains(5) # vérifie si la valeur 5 est présente dans le tableau
$tab1

$tab1.Length # affiche le nombre de valeurs dans le tableau $tab1

$tab2 += "Android" # ajout d'un élément dans le tableau $tab2
# $tab2 contient "python","rust","C++","Android"
```

Table de hachage (hash table)

Une table de hachage est une structure de données qui consiste à associer des paires [clé = valeur]. Il est important de savoir que l'ordre d'affichage des éléments ne correspond pas à celui de la définition de la table de hachage.

Déclaration d'une table de hachage sur une ligne

```
$var = @{ "cd"="ordi1";"routeur"="ordi6" }  
$var.cd  
$var.routeur
```

Déclaration d'une table de hachage sur plusieurs lignes

```
$var2 = @{ "cd"="ordi1"  
           "routeur"="ordi6" }  
$var2.cd  
$var2.routeur
```

Trier une table de hachage

```
$hash = @{a = 1; b = 2; c = 3; d = 4; e = 5; f = 6}  
  
foreach ($h in $hash.GetEnumerator() | Sort-Object Key)  
{  
    Write-Host "Nom= $($h.Key) Valeur= $($h.Value)" -ForegroundColor Green  
}
```

Créer une table de hachage en forçant le respect de l'ordre

```
$hash = [ordered]@{a = 1; b = 2; c = 3; d = 4; e = 5; f = 6}  
  
foreach ($h in $hash.GetEnumerator())  
{  
    Write-Host "Nom= $($h.Key) Valeur= $($h.Value)" -ForegroundColor Green  
}
```

GetEnumerator()	# permet de récupérer chaque combinaison clé/valeur
\$hash.keys	# liste les clés de la table de hachage \$hash
\$hash.values	# liste les valeurs de la table de hachage \$hash
\$hash.a	# récupère la valeur de la clé "a"
\$hash["a"]	# récupère la valeur de la clé "a"
\$hash["a","d"]	# récupère la valeur de la clé "a" et de la clé "d"
Get-Member -InputObject \$hash	# affiche les méthodes disponibles pour la table de hachage \$hash
\$hash.Add("g",7)	# ajoute une paire à la table de hachage \$hash
\$hash.Remove("a")	# supprime une paire à la table de hachage \$hash
\$hash.Clear()	# efface le contenu
\$ageList = @{}	# création d'une "Hasht Table" vide ou efface le contenu

Exemples

Voici une table de hachage qui contient plusieurs variables.

```
$messages = @{  
    # Les erreurs pour la variable "PATHS".  
    MSG_CHECKING_PATHS = "Checking paths..."  
    MSG_COMPUTING_PATHS = "Computing paths..."  
    MSG_CREATING_PATHS = "Creating paths..."  
  
    # Les erreurs pour la variable "IMAGE".  
    MSG_COPYING_IMAGE = "Copying image..."  
    MSG_MOUNTING_IMAGE = "Mounting image..."  
    MSG_CONVERTING_IMAGE = "Converting image..."  
    MSG_SKIPPING_IMAGE_CONVERSION = "Skipping image conversion..."  
}
```

Voici deux exemples qui affichent des messages qui sont dans la table de hachage

```
Write-Host $messages.MSG_CHECKING_PATHS  
Write-Host $messages.MSG_MOUNTING_IMAGE
```

Utilisation d'une table de hachage pour initialiser les paramètres d'un cmdlet

Exemple 1 – les paramètres sont sur une ligne

```
$params = @{ Name = "TEMP"; Path = "E:\_TEMP"; Description = "test ..."; FullAccess = "Tout le monde" }  
New-SmbShare @params
```

Exemple 2 – les paramètres sont sur plusieurs lignes

```
$params = @{  
    Name      = "TEMP"  
    Path      = "E:\_TEMP"  
    Description = "test ..."  
    FullAccess = "Tout le monde"  
}  
New-SmbShare @params
```

Exemple 3 – permet de mettre un paramètre en commentaire

```
$params = @{  
    Name      = "TEMP"  
    Path      = "E:\_TEMP"  
    # Description = "test ..."  
    FullAccess = "Tout le monde"  
}  
New-SmbShare @params
```

Pour utiliser **Sort-Object** avec plusieurs paramètres, il faut utiliser une table de hachage par paramètre pour trier par ordre croissant, décroissant, ou une combinaison d'ordres de tri.

```
Get-Service | Sort-Object -Property @{Expression = "Status"; Descending = $true},  
                                   @{Expression = "DisplayName"; Ascending = $true}
```

L'utilisation de la variable `$PSItem` lorsqu'on utilise un "PIPE"

note: "PowerShell 3.0" remplace `$_` par `$PSItem`

`PSItem` contient la valeur courante d'une commande "PIPE".

Syntaxe avant "PowerShell 3.0"

```
Get-Service | Where-Object { $_.Status -eq "Stopped" }
```

Syntaxe standard avec "PowerShell 3.0"

```
Get-Service | Where-Object { $PSItem.Status -eq "Stopped" }
```

Si on utilise un seul paramètre, on peut utiliser la syntaxe simplifiée

```
Get-Service | Where-Object Status -eq "Stopped"
```

Si on utilise deux paramètres, on ne peut pas utiliser la syntaxe simplifiée

```
Get-Service | Where-Object `
    { $PSItem.Status -eq "Stopped" -and $PSItem.DisplayName -like "*Windows*" }
```

Exemple intéressant

La variable `$serveurs` va contenir: HV01,HV02,HV03,HV04,HV05,HV06,HV07,HV08,HV09,HV10

```
$serveurs = 1..10 | ForEach-Object { "HV{0:D2}" -f $PSItem }
```

Write-Output permet à d'autres cmdlets de capturer et de traiter cette sortie

Avec **Write-Output**, le contenu du fichier output.txt ne sera pas vide.

```
$messages = "Cours", "C53"
```

```
$messages | ForEach-Object { Write-Output $PSItem } | Out-File -FilePath output.txt
```

Write-Host ne permet pas à d'autres cmdlets de capturer et de traiter cette sortie

Avec **Write-Host**, le contenu du fichier output.txt sera vide.

```
$messages = "Cours", "C53"
```

```
$messages | ForEach-Object { Write-Host $PSItem } | Out-File -FilePath output.txt
```

Out-Host force l'affichage du résultat à l'écran

Lors de l'exécution de plusieurs commandes, il arrive que le résultat des deux commandes soit fusionné. Les deux commandes affichent deux colonnes avec exactement les mêmes noms.

```
Get-LocalUser -Name Administrateur | Select-Object Name, SID  
Get-LocalGroup -Name Administrateurs | Select-Object Name, SID
```

```
Name          SID  
----          -  
Administrateur S-1-5-21-2975316056-3426304165-532087291-500  
Administrateurs S-1-5-32-544
```

Le résultat des deux commandes est simplement les deux colonnes Name et SID.

```
Get-LocalUser -Name Administrateur | Select-Object Name, SID | Out-Host  
Get-LocalGroup -Name Administrateurs | Select-Object Name, SID
```

```
Name          SID  
----          -  
Administrateur S-1-5-21-2975316056-3426304165-532087291-500
```

```
Name          SID  
----          -  
Administrateurs S-1-5-32-544
```

Out-Host force l'affichage du résultat de la première commande.

Le résultat de la deuxième commande s'affiche à la suite du résultat de la première commande.

Exemple de code

Voici le code qui affiche le nom de la carte réseau et sa vitesse de transmission.

note: le nom de la carte réseau doit être le même sur chaque serveur

```
Clear-Host

$carte = "Ethernet"

# La variable $serveurs va contenir: HV01,HV02,HV03,HV04,HV05,HV06,HV07,HV08,HV09,HV10
$serveurs = 1..10 | ForEach-Object { "HV{0:D2}" -f $PSItem }

foreach ($serveur in $serveurs)
{
    Write-Host $serveur -ForegroundColor Yellow

    Get-NetAdapter -Name $carte -CimSession $serveur | Format-Table Name,LinkSpeed

    "-"*100
}
```

Détails sur les boucles

```
1..10 | ForEach-Object { "HV{0:D2}" -f $PSItem }
```

Dans cet exemple ForEach est l'alias de ForEach-Object

```
1..10 | ForEach { "HV{0:D2}" -f $PSItem }
```

Dans cet exemple, foreach est une méthode de la collection

```
(1..10).foreach({"HV{0:D2}" -f $PSItem})
```

Exemple de code

Voici le code qui affiche la liste complète des fonctions, cmdlet, alias de tous les modules disponibles.

```
Clear-Host

$modules = (Get-Module -ListAvailable).Name

foreach ($module in $modules)
{
    Write-Host "Nom du module: $module" -ForegroundColor Green

    Get-Command -All -Module $module

    "*" * 80
}
```

Utilisation d'un workflow et d'une boucle FOREACH et du paramètre -Parallel

Ce script envoie la commande Restart-Computer en parallèle aux ordinateurs.

Ce script n'est pas ralenti par le fait qu'il peut y avoir des ordinateurs qui sont fermés.

```
# La variable $computers va contenir les noms 407P01 à 407P32
$computers = 1..32 | ForEach-Object { "407P{0:D2}" -f $_ }

Workflow Restart-AllComputers
{
    param([string[]]$Computers)

    ForEach-Object -Parallel ($computer in $computers)
    {
        Restart-Computer -PSComputerName $computer -Force -Verbose
    }
}

Restart-AllComputers -Computers $computers
```

IMPORTANT: On ne peut pas prédire l'ordre des résultats lorsqu'on exécute des tâches en parallèles.

Comment afficher le nom de l'ordinateur

Il existe plusieurs manières d'afficher le nom de l'ordinateur.

Méthode 1: utilisation de hostname.exe

```
hostname.exe
```

Méthode 2: utilisation de la variable d'environnement

```
$env:COMPUTERNAME
```

Méthode 3: utilisation d'un objet WMI

```
(Get-WMIObject Win32_ComputerSystem).Name
```

Méthode 4: utilisation d'une instance CIM

```
(Get-CIMInstance CIM_ComputerSystem).Name
```

Méthode 5: utilisation d'une méthode ".Net Framework"

```
[system.environment]::MachineName
```

Méthode 6: utilisation d'une méthode ".Net Framework"

```
[system.net.dns]::GetHostName()
```

Méthode 7: utilisation du cmdlet Get-ComputerInfo

```
(Get-ComputerInfo).CsName
```

Comment trouver la méthode la plus rapide

Le cmdlet Measure-Command permet de mesurer la vitesse d'exécution d'une commande.

exemple: Measure-Command { hostname.exe }

La propriété TotalMilliseconds permet de comparer facilement la vitesse d'exécution d'une méthode par rapport à une autre.

Le cmdlet Measure-Object permet d'effectuer des calculs comme la moyenne

exemple:

```
1..100 | Foreach-Object { Measure-Command { hostname.exe } } | Measure-Object -Average TotalMilliseconds
```

On exécute 100 fois le cmdlet Measure-Command et le cmdlet Measure-Object calcule la moyenne de la propriété "TotalMilliseconds".

Effectuons des tests pour déterminer la différence dans le temps d'exécution.

Clear-Host

IMPORTANT: on ne doit pas utiliser des variables qui contiennent les commandes
Foreach-Object possède deux alias: foreach et %
Les tests vont du plus rapide au plus lent.

```
1..100 | % {Measure-Command {[system.environment]::MachineName}} | `
Measure-Object -Average TotalMilliseconds
0,008832 ms
```

```
1..100 | % {Measure-Command {$env:computername}} | `
Measure-Object -Average TotalMilliseconds
0,024194 ms
```

```
1..100 | % {Measure-Command {[system.net.dns]::GetHostName()}} | `
Measure-Object -Average TotalMilliseconds
0,047328 ms
```

```
1..100 | % {Measure-Command {(Get-WMIObject Win32_ComputerSystem).Name}} | `
Measure-Object -Average TotalMilliseconds
8,362251 ms
```

```
1..100 | % {Measure-Command {hostname.exe}} | `
Measure-Object -Average TotalMilliseconds
9,12406 ms
```

```
1..100 | % {Measure-Command {(Get-CIMInstance CIM_ComputerSystem).Name}} | `
Measure-Object -Average TotalMilliseconds
9,330143 ms
```

```
1..100 | % {Measure-Command {(Get-ComputerInfo).CsName}} | `
Measure-Object -Average TotalMilliseconds
1659,054235 ms
```

Comparaison de la vitesse d'exécution des commandes

La commande 1 est toujours la plus rapide avec un temps d'exécution d'environ **0.008 ms**

Les commandes 2 et 3 sont environ 5 fois plus lentes que la commande 1

Les commandes 4, 5 et 6 sont environ 1000 fois plus lentes que la commande 1

La commande 7 est environ 200 000 fois plus lente que la commande 1

La commande la plus rapide

"[system.environment]::MachineName" est toujours extrêmement rapide.

La commande la plus lente

"(Get-ComputerInfo).CsName" est toujours extrêmement lente.

Stratégie d'exécution des scripts avec PowerShell

Les fichiers de script PowerShell doivent avoir l'extension PS1.

La commande **Get-ExecutionPolicy** est utilisée pour obtenir la valeur de la stratégie d'exécution.

La commande **Set-ExecutionPolicy** est utilisée pour modifier la stratégie d'exécution des scripts.

Les valeurs possibles pour le paramètre **-ExecutionPolicy** sont

- **AllSigned**
Nécessite que tous les scripts et tous les fichiers de configuration soient signés par un éditeur approuvé, y compris les scripts écrits sur l'ordinateur local.
- **Bypass**
Permet l'exécution de tous les scripts sans restriction ni avertissement.
- **Default**
Définis la stratégie d'exécution par défaut.
Restricted pour les clients Windows ou RemoteSigned pour les serveurs Windows.
- **RemoteSigned**
Permet l'exécution de scripts créés localement.
Les scripts téléchargés doivent être signés par un éditeur de confiance.
C'est le paramètre par défaut pour les serveurs Windows.
- **Restricted**
Ne permet pas l'exécution de scripts.
C'est le paramètre par défaut pour les clients Windows.
- **Undefined**
Supprime la stratégie d'exécution actuelle et la stratégie d'exécution effective est Restricted.
- **Unrestricted**
Permet l'exécution de tous les scripts, mais avertit avant d'exécuter des scripts téléchargés.

Exemple

```
Set-ExecutionPolicy -ExecutionPolicy Unrestricted -Force
```

IMPORTANT: on ne peut pas exécuter un script PowerShell en double cliquant sur le fichier.

Pour exécuter un script PowerShell, il faut spécifier le chemin absolu ou le chemin relatif lors de l'appel.

Il est possible d'exécuter un script PowerShell sans modifier la stratégie d'exécution.

La ligne de code s'exécute dans une invite de commandes CMD.

```
powershell.exe -ExecutionPolicy Bypass X:\PS\votre_script.ps1
```

Les commentaires

Cette ligne est en commentaire à cause du symbole #.

```
<#  
Ceci est un commentaire  
sur plusieurs lignes.  
#>
```

Caractère de continuité

Il est possible d'écrire une ligne d'instruction sur plusieurs lignes en utilisant un caractère de continuité.

Le caractère de continuité correspond à l'accent de grave (**code ASCII 96**).

Les fichiers de configuration pour PowerShell

Un profil applicable à tous les utilisateurs et aux consoles powershell.exe et powershell_ise.exe.

- **\$PSHOME\profile.ps1**
-

Un profil applicable à tous les utilisateurs et à la console powershell.exe.

- **\$PSHOME\Microsoft.PowerShell_profile.ps1**

Un profil applicable à tous les utilisateurs et à la console powershell_ise.exe.

- **\$PSHOME\Microsoft.PowerShellISE_profile.ps1**
-

Un profil applicable à l'utilisateur courant et aux consoles powershell.exe et powershell_ise.exe.

- **\$HOME\Documents\WindowsPowerShell\profile.ps1**
-

Un profil applicable à l'utilisateur courant et à la console powershell.exe.

- **\$HOME\Documents\WindowsPowerShell\Microsoft.PowerShell_profile.ps1**

Un profil applicable à l'utilisateur courant et à la console powershell_ise.exe

- **\$HOME\Documents\WindowsPowerShell\Microsoft.PowerShellISE_profile.ps1**
-

L'ordre d'exécution des fichiers de configuration pour "powershell.exe"

\$PROFILE | Select-Object *

```
AllUsersAllHosts      : $PSHOME\profile.ps1
AllUsersCurrentHost   : $PSHOME\Microsoft.PowerShell_profile.ps1
CurrentUserAllHosts   : $HOME\Documents\WindowsPowerShell\profile.ps1
CurrentUserCurrentHost : $HOME\WindowsPowerShell\Microsoft.PowerShell_profile.ps1
```

L'ordre d'exécution des fichiers de configuration pour "powershell_ise.exe"

\$PROFILE | Select-Object *

```
AllUsersAllHosts      : $PSHOME\profile.ps1
AllUsersCurrentHost   : $PSHOME\Microsoft.PowerShellISE_profile.ps1
CurrentUserAllHosts   : $HOME\Documents\WindowsPowerShell\profile.ps1
CurrentUserCurrentHost : $HOME\Documents\WindowsPowerShell\Microsoft.PowerShellISE_profile.ps1
```

commande pour créer un fichier "profile" pour "CurrentUserAllHosts"

```
if (!(Test-Path -Path $PROFILE.CurrentUserAllHosts))
{ New-Item -Type File -Path $PROFILE.CurrentUserAllHosts -Force }
```

psedit permet d'éditer facilement les différents fichiers "profile" si les fichiers existent

```
psedit $profile.AllUsersAllHosts
psedit $profile.AllUsersCurrentHost
psedit $profile.CurrentUserAllHosts
psedit $profile.CurrentUserCurrentHost
```

Commande pour afficher les variables dont le nom débute par **ps** donc des variables de PowerShell
Get-Variable ps*

La variable **\$psISE** permet de configurer l'environnement ISE de PowerShell.

Exemple d'un fichier \$PSHOME\profile.ps1

```
# Modification de la variable BufferSize
$InfoHost = Get-Host
$InfoWindow = $InfoHost.UI.RawUI
$NewSize = $InfoWindow.BufferSize
$NewSize.Height = 8192
$NewSize.Width = 512
$InfoWindow.BufferSize = $NewSize

# Force l'affichage du volet de script dans "PowerShell ISE"
if ($psISE.CurrentPowerShellTab.ExpandedScript -eq $false)
{
    $psISE.CurrentPowerShellTab.ExpandedScript = $true
}

# Modification du texte dans la barre de titre
$InfoWindow.WindowTitle = "Console de " + [system.environment]::UserName
```

Exemple d'un fichier \$HOME\Documents\WindowsPowerShell\profile.ps1

```
# Change le dossier actif
Set-Location E:\scriptPS1
```

Pour démarrer "**PowerShell ISE**" sans utiliser les fichiers de configuration pour PowerShell.
powershell_ISE.exe -NoProfile

Pour le cours, il n'est pas nécessaire de comprendre le contenu des annexes 1, 2 et 3.

Notions avancées de PowerShell

L'opérateur -f est utilisé pour le formatage de l'affichage

{I,A:FS}.. -f nombre

I Indexe des items à afficher

A Alignement

Si A est un nombre positif alors l'alignement sera de n caractères vers la droite

Si A est un nombre négatif alors l'alignement sera de n caractères vers la gauche

FS Un paramètre optionnel de formatage qui agit sur l'item en fonction de son type

Voici la liste des paramètres FS valides

:c	Représentation monétaire
:d	Padded. (:dP precision=number of digits); if needed, leading zeros are added to the beginning of the (whole) number.
:e	Scientific (exp) notation
:f	Fixed point :f5 = fix to 5 places
:g	Most compact format, fixed or sci :g5 = 5 significant digits
:n	Number (:nP precision=number of decimal places), includes culture separator for thousands 1,000.00
:p	Pourcentage
:r	Précision réversible
:x	Représentation en format hexadécimal
:hh :mm :ss	Affiche l'heure, les minutes et les secondes d'une date "{0:hh}:{0:mm}:{0:ss}"
:HH	Affiche l'heure sous le format 24H
MM	Affiche le mois
MMMM	Affiche le nom au complet du mois
:dd	Affiche le jour du mois
:ddd	Affiche le nom du jour de la semaine
:dddd	Affiche le nom au complet du jour de la semaine
:yyyy	Affiche l'année au complet
#	Caractère spécial

Voici plusieurs exemples

```
"{0:c}" -f 5.25  
5,25 $
```

```
"{0:d5}" -f 123  
00123
```

```
"{0,10:d5}" -f 123  
00123
```

```
# Pour créer une liste de nom avec un suffixe (ORDI01 à ORDI10)  
1..10 | ForEach-Object {"ORDI{0:d2}" -f $PSItem}  
ORDI01  
ORDI02  
ORDI03  
ORDI04  
ORDI05  
ORDI06  
ORDI07  
ORDI08  
ORDI09  
ORDI10
```

```
"{0,15:n4}" -f 2123.24597  
2 123,2460
```

```
"{0:x}" -f 255  
ff
```

```
# Convertir le caractère 'A' en valeur hexadécimale  
'0x' + "{0:x}" -f [int][char]'A'  
0x41
```

```
"{0:###-###-###}" -f 123456789  
123-456-789
```

```
# Alignement de texte à gauche et à droite  
"{0,-10}|{1,10}" -f "Power", "Shell"  
Power      |      Shell
```

```
# Affichage de texte et de nombre  
"{1,10:d5}{0,10}{2,10:x}" -f "Salut", 64, 255  
00064      Salut      ff
```

```
# Affiche l'heure, les minutes et les secondes de la date
# note: n'indique pas AM ou PM
"{0:hh}:{0:mm}:{0:ss}" -f (Get-Date)
02:44:49
```

```
# Affiche l'heure (format 24H), les minutes et les secondes de la date
"{0:HH}:{0:mm}:{0:ss}" -f (Get-Date)
14:45:57
```

```
# Affiche le jour du mois de la date
"{0:dd}" -f (Get-Date)
11
```

```
# Affiche le nom du jour de la semaine de la date
"{0:ddd}" -f (Get-Date)
jeu.
```

```
# Affiche le nom au complet du jour de la semaine de la date
"{0:dddd}" -f (Get-Date)
Jeudi
```

```
# Affiche l'année de la date
"{0:yyyy}" -f (Get-Date)
2021
```

```
# Affiche le mois de la date
"{0:MM}" -f (Get-Date)
02
```

```
# Affiche le nom au complet du mois de la date
PS D:\_OUTILS> "{0:MMMM}" -f (Get-Date)
février
```

Utilisation de la classe "System.Math" du ".NET Framework"

Voici plusieurs méthodes de la classe **system.math** du ".NET Framework".

```
[system.math]::Abs(n)
[system.math]::Equals(m,n)
[system.math]::Exp(n)           # retourne la valeur de la constante "e" à la puissance "n"
[system.math]::Ceiling(n)
[system.math]::Floor(n)
[system.math]::Max(m,n)
[system.math]::Min(m,n)
[system.math]::Pow(m,n)        # retourne la valeur du nombre "m" à la puissance "n"
[system.math]::Round(n)
[system.math]::Sqrt(n)
[system.math]::Truncate(n)
```

Voici deux propriétés de la classe **system.math** du ".NET Framework".

```
[system.math]::E           # constante E = 2.71828182845905
[system.math]::PI          # constante PI = 3.14159265358979
```

Utilisation de la classe "System.Environment" du ".NET Framework"

NOTE: c'est la commande la plus rapide pour récupérer le nom d'un ordinateur

```
[system.environment]::MachineName

[system.environment]::UserName

[system.environment]::OSVersion
```


Utilisation de la classe "System.Windows.MessageBox" du ".NET Framework"

Voici comment afficher un message dans un **MessageBox** en utilisant la méthode **Show**.

NOTE: La méthode Show est surchargée.

Voici la syntaxe d'une surcharge de la méthode Show

```
Show (string messageBoxText,  
      string caption)
```

Exemple

```
[System.Windows.MessageBox]::Show('Le message.', 'Le titre')
```

Voici la syntaxe d'une surcharge de la méthode Show

```
Show (string messageBoxText,  
      string caption,  
      System.Windows.MessageBoxButton button,  
      System.Windows.MessageBoxImage icon)
```

Exemple

```
$rep = [System.Windows.MessageBox]::Show('Le message', 'Le titre', 'YesNoCancel', 'Error')
```

```
switch ($rep)  
{  
  'Yes'      { Write-Host 'Oui' }  
  'No'       { Write-Host 'Non' }  
  'Cancel'   { Write-Host 'Annuler' }  
}
```

Voici la liste des valeurs possibles pour l'objet "System.Windows.MessageBoxButton"

```
[System.Enum]::GetNames([System.Windows.MessageBoxButton])  
OK  
OKCancel  
YesNoCancel  
YesNo
```

Voici la liste des valeurs possibles pour l'objet "System.Windows.MessageBoxImage"

```
[System.Enum]::GetNames([System.Windows.MessageBoxImage])  
None  
Hand  
Error  
Stop  
Question  
Exclamation  
Warning  
Asterisk  
Information
```

Utilisation de la classe "Microsoft.VisualBasic.Interaction" du ".NET Framework"

Voici comment afficher un message dans un **MsgBox** qui est toujours au premier plan.

```
Add-Type -AssemblyName Microsoft.VisualBasic
$rep =
[Microsoft.VisualBasic.Interaction]::MsgBox('Message','YesNoCancel,SystemModal,Information','
Titre')
$rep

switch ($rep)
{
    'Yes'      { Write-Host 'Oui' }
    'No'       { Write-Host 'Non' }
    'Cancel'   { Write-Host 'Annuler' }
}
```

"Microsoft.VisualBasic.MsgBoxStyle" a plusieurs valeurs qu'il est possible de combiner par une virgule

```
[System.Enum]::GetNames([Microsoft.VisualBasic.MsgBoxStyle])
ApplicationModal
DefaultButton1
OkOnly
OkCancel
AbortRetryIgnore
YesNoCancel
YesNo
RetryCancel
Critical
Question
Exclamation
Information
DefaultButton2
DefaultButton3
SystemModal
MsgBoxHelp
MsgBoxSetForeground
MsgBoxRight
MsgBoxRtlReading
```

Utilisation de la classe "System.Convert" du ".NET Framework"

Convertit la représentation d'une chaîne de caractères d'un nombre dans une base spécifiée en un entier 32 bits signé.
ToInt32(String, Int32)

Conversion d'un nombre binaire en entier
[system.convert]::ToInt32('10000000', 2)

Conversion d'un nombre octal en entier
[system.convert]::ToInt32('10', 8)

Conversion d'un nombre hexadécimal en entier
[system.convert]::ToInt32('FF', 16)

Convertit la représentation d'une chaîne de caractères d'un nombre dans une base spécifiée en un entier 64 bits signé.
ToInt64(String, Int32)

Conversion d'un grand nombre hexadécimal en entier
[system.convert]::ToInt64('FFFFFFFF', 16)

Conversion d'un grand nombre binaire en entier
[system.convert]::ToInt64('10000000000000000000000000000000', 2)

Définir le nombre décimal
\$nombreDecimal = 1024

Conversion d'un nombre entier en binaire (chaîne de caractères)
\$nombreBinaire = [system.convert]::ToString(\$nombreDecimal, 2)

Affichage du nombre binaire sur 32 colonnes (32 bits)
\$nombreBinaire.PadLeft(32, '0')

Définir le nombre décimal
\$nombreDecimal = 255

Conversion d'un nombre entier en hexadécimal (chaîne de caractères)
\$nombreHexa = [system.convert]::ToString(255, 16)

Conversion en majuscule du nombre hexadécimal
\$nombreHexa = \$nombreHexa.ToUpper()

Affichage du nombre hexadécimal sur 8 colonnes (32 bits)
\$nombreHexa.PadLeft(8, '0')

Chargement d'une classe avant l'appel d'une méthode

```
Add-Type -AssemblyName System.Web  
[System.Web.Security.Membership]::GeneratePassword(8,1)
```

- le premier paramètre de GeneratePassword spécifie la longueur du mot de passe
- le deuxième paramètre de GeneratePassword spécifie le nombre minimum de caractères spéciaux

Get-Member permet d'obtenir la liste des propriétés et des méthodes des objets.

```
# $S contient la liste des services sur un ordinateur local  
$S = Get-Service
```

```
# Get-Member obtient le type System.ServiceProcess.ServiceController et la liste des membres  
# contient des méthodes comme Pause, Start, Stop et des propriétés comme StartType, Status.  
$S | Get-Member
```

```
# Get-Member obtient le type System.Object[] et la liste des membres contient des méthodes  
# comme Add, Remove et des propriétés comme Length.  
Get-Member -InputObject $S
```

Utilisation d'objet COM

Voici un exemple de code qui utilise l'objet COM **Wscript.Shell** pour créer un raccourci sur le Bureau.

```
$cible = "C:\_TEMP\info.txt"
$lien = "C:\Users\richard\Desktop\Mon_Lien.lnk"

$WshShell = New-Object -ComObject Wscript.Shell
$raccourci = $WshShell.CreateShortcut($lien)
$raccourci.TargetPath = $cible
$raccourci.Save()
```

Le fichier **COM_Object_Excel.ps1** qui est sur LÉA montre comment utiliser l'objet COM **Excel.Application** pour insérer des valeurs et effectuer des calculs dans Excel.

Le fichier **GUI_map_drive.ps1** qui est sur LÉA montre comment utiliser la classe **System.Windows.Forms** pour créer des objets: Form, Button, Label, TextBox, MaskedTextBox.

PSDrive

Un PSDrive a un comportement similaire à l'Explorateur de fichiers.
Un PSDrive est relié à un fournisseur.

Pour s'assurer que le module ActiveDirectory est accessible par un fournisseur PSDrive

```
Import-Module ActiveDirectory
```

Affiche la liste des fournisseurs

```
Get-PSProvider | Format-Table -AutoSize
```

Name	Capabilities	Drives
----	-----	-----
Registry	ShouldProcess, Transactions	{HKLM, HKCU}
Alias	ShouldProcess	{Alias}
Environment	ShouldProcess	{Env}
FileSystem	Filter, ShouldProcess, Credentials	{C, R}
Function	ShouldProcess	{Function}
Variable	ShouldProcess	{Variable}
Certificate	ShouldProcess	{Cert}
WSMan	Credentials	{WSMan}
ActiveDirectory	Include, Exclude, Filter, ShouldProcess, Credentials	{AD}

Affiche les lecteurs disponibles pour PSDrive

```
Get-PSDrive | Format-Table -AutoSize
```

Name	Used (GB)	Free (GB)	Provider	Root	CurrentLocation
----	-----	-----	-----	----	-----
AD			ActiveDirectory	//RootDSE/	
Alias			Alias		
C	15,69	110,76	FileSystem	C:\	Users\Administrateur.HV1
Cert			Certificate	\	
Env			Environment		
Function			Function		
HKCU			Registry	HKEY_CURRENT_USER	
HKLM			Registry	HKEY_LOCAL_MACHINE	
Variable			Variable		
WSMan			WSMan		

"AD" est le nom du lecteur qui permet d'accéder au fournisseur "ActiveDirectory".

Déplacement au niveau du lecteur "AD"

```
cd AD:
```

Pour afficher les objets qui sont directement sous le lecteur "AD".

```
Get-ChildItem
```

Name	ObjectClass	DistinguishedName
----	-----	-----
formation	domainDNS	DC=formation,DC=local
Configuration	configuration	CN=Configuration,DC=formation,DC=local
Schema	dMD	CN=Schema,CN=Configuration,DC=formation,DC=local
DomainDnsZones	domainDNS	DC=DomainDnsZones,DC=formation,DC=local
ForestDnsZones	domainDNS	DC=ForestDnsZones,DC=formation,DC=local

Naviguer dans le lecteur "AD"

Déplacement au niveau du domaine FORMATION.LOCAL

```
cd "DC=formation,DC=local"
```

Pour afficher les objets qui sont directement sous le lecteur "AD:\DC=formation,DC=local".

```
Get-ChildItem
```

Name	ObjectClass	DistinguishedName
----	-----	-----
_DATA	organizationalUnit	OU=_DATA,DC=formation,DC=local
Builtin	builtinDomain	CN=Builtin,DC=formation,DC=local
Computers	container	CN=Computers,DC=formation,DC=local
Domain Controllers	organizationalUnit	OU=Domain Controllers,DC=formation,DC=local
ForeignSecurityPr...	container	CN=ForeignSecurityPrincipals,DC=formation,DC=local
Infrastructure	infrastructureUpdate	CN=Infrastructure,DC=formation,DC=local
Keys	container	CN=Keys,DC=formation,DC=local
LostAndFound	lostAndFound	CN=LostAndFound,DC=formation,DC=local
Managed Service A...	container	CN=Managed Service Accounts,DC=formation,DC=local
NTDS Quotas	msDS-QuotaContainer	CN=NTDS Quotas,DC=formation,DC=local
ORDINATEURS	organizationalUnit	OU=ORDINATEURS,DC=formation,DC=local
Program Data	container	CN=Program Data,DC=formation,DC=local
System	container	CN=System,DC=formation,DC=local
TPM Devices	msTPM-Information...	CN=TPM Devices,DC=formation,DC=local
Users	container	CN=Users,DC=formation,DC=local
UTILISATEURS	organizationalUnit	OU=UTILISATEURS,DC=formation,DC=local

Déplacement au niveau du conteneur "Users"

```
cd "CN=Users"
```

Pour sortir du lecteur PSDrive, il suffit de se déplacer dans le lecteur "C"

```
c:
```

ANNEXE 1 WINNT:// et LDAP://

Les deux fournisseurs les plus utilisés pour l'administration système et réseau sont

```
WINNT://      # c'est le fournisseur Windows et Windows Server
LDAP://       # c'est le fournisseur qui permet d'accéder à LDAP
              # LDAP (Lightweight Directory Access Protocol)
```

Syntaxe générique pour WINNT://

WINNT://<nom_domaine>,<nom_objet>,<nom_classe>

```
# L'utilisateur "Richard" est un utilisateur local
$a = [ADSI]"WinNT://127.0.0.1/RICHARD,user"
```

```
# L'utilisateur "Administrateur" est un utilisateur du domaine "FORMATION"
$b = [ADSI]"WinNT://formation/Administrateur,user"
```

```
# Liste les propriétés de l'objet $b
$b | Get-Member
```

Syntaxe générique pour LDAP://

LDAP://<nom_unique>

Le nom_unique correspond au DistinguishedName.
L'objet recherché doit être parfaitement connu.

LDAP://CN=Administrateur,CN=Users,DC=formation,DC=local

```
$c = [ADSI]"LDAP://CN=Administrateur,CN=Users,DC=formation,DC=local"
```

```
# Liste les propriétés de l'objet $c
$c | Get-Member
```

La liste des propriétés de "\$b | Get-Member" est différente de "\$c | Get-Member".

ANNEXE 2

Module Microsoft.Windows.Bcd.Cmdlets dans "Windows 11"

Pour configurer le fichier BCD (Boot Configuration Database) les administrateurs peuvent utiliser BCDEDIT.EXE. Avec la commande BCDEDIT.EXE, c'est difficile d'automatiser la modification du fichier BCD.

La configuration du "Windows Boot Manager" dans le UEFI de votre ordinateur modifie indirectement le contenu du fichier BCD.

Le module Microsoft.Windows.Bcd.Cmdlets permet d'automatiser la configuration du fichier BCD.

```
# Commande qui affiche la liste complète des cmdlets  
(Get-Command -Module Microsoft.Windows.Bcd.Cmdlets).Name
```

```
Copy-BcdEntry  
Disable-BcdElementBootDebug  
Disable-BcdElementBootEms  
Disable-BcdElementDebug  
Disable-BcdElementEms  
Disable-BcdElementEventLogging  
Disable-BcdElementHypervisorDebug  
Enable-BcdElementBootDebug  
Enable-BcdElementBootEms  
Enable-BcdElementDebug  
Enable-BcdElementEms  
Enable-BcdElementEventLogging  
Enable-BcdElementHypervisorDebug  
Export-BcdStore  
Get-BcdEntry  
Get-BcdEntryDebugSettings  
Get-BcdEntryHypervisorSettings  
Get-BcdStore  
Import-BcdStore  
New-BcdEntry  
New-BcdStore  
Remove-BcdElement  
Remove-BcdEntry  
Set-BcdBootDefault  
Set-BcdBootDisplayOrder  
Set-BcdBootSequence  
Set-BcdBootTimeout  
Set-BcdBootToolsDisplayOrder  
Set-BcdDebugSettings  
Set-BcdElement  
Set-BcdHypervisorSettings
```

ANNEXE 3

Conflits entre le module Hyper-V et le module VMware.VimAutomation.Core

La compagnie VMware utilise PowerShell pour gérer les machines virtuelles et son hyperviseur ESXi.

Malheureusement, le module VMware.VimAutomation.Core inclus dans PowerCLI contient plusieurs cmdlet qui portent le même nom que les cmdlet du module Hyper-V.

Pour utiliser le bon cmdlet, il faut utiliser la syntaxe suivante: **[ModuleName]\[Cmdlet_or_Function_Name]**

Par exemple, **Get-VM** existe dans le module Hyper-V et le module VMware.VimAutomation.Core

```
# Pour vérifier si une commande existe dans plusieurs modules,  
# ajouter le caractère * devant le nom de la commande.  
(Get-Command *Get-VM).Source  
VMware.VimAutomation.Core  
Hyper-V  
  
# Si le caractère * n'est pas devant le nom de la commande,  
# le résultat affiche seulement un module.  
(Get-Command Get-VM).Source  
VMware.VimAutomation.Core
```

Il est important de spécifier le nom du module pour utiliser le bon cmdlet.

```
Get-Help VMware.VimAutomation.Core\Get-VM
```

```
Get-Help Hyper-V\Get-VM
```

PowerShell Direct

Pour créer une session "PowerShell Direct" sur une machine virtuelle

- L'ordinateur virtuel doit s'exécuter localement sur l'ordinateur réel.
- Vous devez utiliser un compte qui est membre du groupe "Administrateurs".
- Vous devez fournir les informations pour s'authentifier à l'ordinateur virtuel.
- L'ordinateur réel doit exécuter le système d'exploitation Windows 10 ou Windows Server 2016.
- L'ordinateur virtuel doit exécuter le système d'exploitation Windows 10 ou Windows Server 2016.

Exemple 1:

```
Enter-PSSession -VMName "VMName"
```

- **Les commandes s'exécutent sur la machine virtuelle.**

```
Exit-PSSession
```

Exemple 2:

on enregistre les informations pour l'authentification dans une variable

```
$Cred = Get-Credential
```

```
Invoke-Command -VMName "C53-SERVEUR1" `
    -Credential $Cred `
    -ScriptBlock { get-process }
```

Exemple 3:

on enregistre les informations pour l'authentification dans une variable

```
$Cred = Get-Credential
```

```
$NOM_VM = "C53-SERVEUR1"
```

```
$nom_fichier = "D:\script\mon_script.ps1"
```

Le fichier de script doit utiliser le cmdlet Write-Output

si on veut récupérer les messages affichés

```
$rep = Invoke-Command -VMName $NOM_VM `
    -Credential $Cred `
    -FilePath $nom_fichier
```

Utilisation d'une variable globale dans un ScriptBlock

Il est possible d'utiliser une variable globale dans un ScriptBlock à condition d'utiliser **\$using:** devant le nom de la variable.

```
$Cred = Get-Credential -Credential "FORMATION\Administrateur"
```

```
$prog = "win*"
```

```
$NOM_VM = "C53-SERVEUR1"
```

```
Invoke-Command -VMName $NOM_VM `
    -Credential $Cred `
    -ScriptBlock { Get-Process -Name $using:prog }
```

Utilisation du paramètre -ArgumentList avec "Invoke-Command"

Utilisation de la variable \$args[0] dans "Invoke-Command"

\$args[0] accepte seulement une valeur

```
$p = "power*"
Invoke-Command -VMName "C53-SERVEUR1" `
  -Credential "FORMATION\Administrateur" `
  -ScriptBlock { Get-Process -Name $args[0] } `
  -ArgumentList $p
```

Utilisation de la variable \$args dans "Invoke-Command"

\$args peut accepter plus d'une valeur mais dans cet exemple \$p contient une seule valeur

```
$p = "power*"
Invoke-Command -VMName "C53-SERVEUR1" `
  -Credential "FORMATION\Administrateur" `
  -ScriptBlock { Get-Process -Name $args } `
  -ArgumentList $p
```

Utilisation de la variable \$args dans "Invoke-Command"

\$args permet de passer un tableau en paramètre comme dans l'exemple

```
$p = "power*", "win*"
Invoke-Command -VMName "C53-SERVEUR1" `
  -Credential "FORMATION\Administrateur" `
  -ScriptBlock { Get-Process -Name $args } `
  -ArgumentList $p
```

Utilisation d'une variable nommée dans "Invoke-Command"

Avec cette méthode, on doit utiliser **param** pour utiliser une variable nommée.

La variable nommée peut accepter plus d'une valeur mais dans cet exemple \$p contient une seule valeur

```
$p = "power*"
Invoke-Command -VMName "C53-SERVEUR1" `
  -Credential "FORMATION\Administrateur" `
  -ScriptBlock {
    param ($proc)
    Get-Process -Name $proc
  } `
  -ArgumentList $p
```

Utilisation d'une variable nommée dans "Invoke-Command"

La variable nommée permet de passer un tableau en paramètre comme dans l'exemple

Par contre, on doit obligatoirement respecter la syntaxe dans le paramètre -ArgumentList.

```
$p = "power*", "win*"
Invoke-Command -VMName "C53-SERVEUR1" `
  -Credential "FORMATION\Administrateur" `
  -ScriptBlock {
    param ($proc)
    Get-Process -Name $proc
  } `
  -ArgumentList (, $p)
```

Exemple 1 – Exécution d'un ScriptBlock qui utilise des commandes Write-Output

```
# on enregistre les informations pour l'authentification dans une variable
$Cred = Get-Credential

$NOM_VM = "C53-SERVEUR2"
$siteFTP= "FTP_ADR1"

# on doit utiliser le cmdlet Write-Output si on veut récupérer les messages affichés
$code = { Write-Output "====="
           Write-Output "Nom, état et dossier du site FTP"
           Write-Output "====="
           Get-Website -name $args[0] `
               | Select-Object Name,State,PhysicalPath `
               | Format-Table -AutoSize `
               | Out-String
           }

$siteInfo = Invoke-Command -VMName $NOM_VM `
    -Credential $Cred `s
    -ScriptBlock $code `
    -ArgumentList $siteFTP

$siteInfo
```

Résultat de l'exécution du ScriptBlock

Le contenu de la variable \$siteInfo

```
=====
Nom, état et dossier du site FTP
=====

name          state    physicalPath
----          -
FTP_ADR1      Started C:\_FTP\FTP_ADR1
```

Exemple 2 – Exécution d'un ScriptBlock sur plusieurs ordinateurs virtuels

```
# on enregistre les informations pour l'authentification dans une variable
$Cred = Get-Credential -Credential "Administrateur"

# Les VM "WIN10-1" et "WIN10-2" ne sont pas dans l'Active Directory.
Invoke-Command -VMName "WIN10-1","WIN10-2" `
    -Credential $Cred `
    -ScriptBlock { Get-LocalUser -Name t* | Out-Host }
```

Résultat de l'exécution du ScriptBlock

```
Name Enabled Description
----
TECH True

Name Enabled Description
----
TEST True
```

L'ordinateur virtuel "Server_Core_2019" est sur le SERVEUR2.
EXÉCUTION DU CODE À PARTIR DU SERVEUR1

Le SERVEUR1 et le SERVEUR2 sont dans un domaine "Active Directory".
L'ordinateur virtuel "Server_Core_2019" est sur le SERVEUR2.

Voici comment exécuter du code dans l'ordinateur virtuel qui est sur le SERVEUR2 mais en exécutant le code PowerShell sur le SERVEUR1,

```
# Voici le contenu du fichier "E:\core_info.ps1" qui est sur le SERVEUR1
Write-Output "======"
Write-Output "Configuration de la VM"
Write-Output "======"
(Get-ComputerInfo).CsName
(Get-NetIPAddress -AddressFamily IPv4).IPAddress

#-----
# Ce code doit s'exécuter sur le SERVEUR1
#-----
# Lire le contenu du fichier PS1 qui est sur le SERVEUR1
# '\n\r' correspond à un saut de ligne dans Windows
$fichier = Get-Content -Path "E:\core_info.ps1" `
            -Delimiter '\n\r'

$cred_VM = Get-Credential -UserName "Administrateur" `
            -Message "Server_Core_2019"

# $using:cred_VM permet de passer la variable $cred_VM dans la requête imbriquée.
# $using:fichier permet de passer la variable $fichier dans la requête imbriquée.

Invoke-Command -ComputerName "SERVEUR2" `
    -ScriptBlock { hostname.exe
        Invoke-Command -VMName "Server_Core_2019" `
            -Credential $using:cred_VM `
            -scriptblock { param($code)
                Invoke-Expression $code
            } `
            -ArgumentList ($using:fichier)
    }
```

Résultat de l'exécution du code

```
SERVEUR2
=====
Configuration du serveur 'CORE'
=====
CORE
192.168.53.140
127.0.0.1
```

Sauvegarder l'authentification d'un utilisateur dans une variable

Ce code permet de conserver dans une variable le nom de l'utilisateur et le mot de de passe qui est enregistré sous forme "System.Security.SecureString" dans une variable.

```
# Script qui utilise trois paramètres obligatoires
[CmdletBinding()]
Param
(
    [Parameter(Mandatory=$True)] $NOM_VM,
    [Parameter(Mandatory=$True)] $UserName,
    [Parameter(Mandatory=$True)] $MDP
)
Clear-Host

$pass = ConvertTo-SecureString -AsPlainText $MDP -Force
$Cred = New-Object -TypeName System.Management.Automation.PSCredential `
    -ArgumentList $Username,$pass

$code = { Start-VM -Name $args[0] }

Invoke-Command -ComputerName SERVEUR1 `
    -ScriptBlock $code `
    -ArgumentList $NOM_VM `
    -Credential $Cred
```

Plusieurs commandes de PowerShell utilisent le paramètre -Password qui est de type <System.Security.SecureString>.

Le paramètre -Password de New-LocalUser est de type <System.Security.SecureString>
si le paramètre est de type <System.Security.SecureString>
Read-Host permet facilement de sécuriser le mot de passe
\$mdp = Read-Host -Prompt "Entrer le mot de passe" -AsSecureString

```
New-LocalUser -Name "admin" `
    -Password $mdp `
    -FullName "Prénom Nom" `
    -Description "admin est membre du groupe Administrateurs"
```

Plusieurs commandes de PowerShell utilisent le paramètre -Password qui est de type <string>.

Le paramètre -Password de New-SmbMapping est de type <String>.

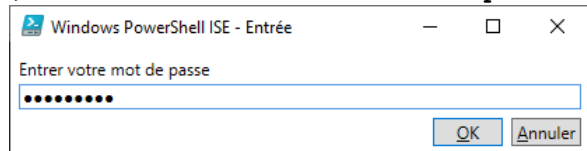
Convertir une chaîne sécurisée en texte clair

Ce code est utile pour des cmdlets qui ont besoin du mot de passe en texte clair.

Par exemple, pour se connecter à un partage réseau avec New-SmbMapping

Le paramètre -Password de New-SmbMapping est de type <String>

```
$SecurePassword = Read-Host -Prompt "Entrer votre mot de passe" -AsSecureString
```



```
# Ces deux lignes de code permettent de récupérer le mot de passe en texte clair.  
$BSTR = [System.Runtime.InteropServices.Marshal]::SecureStringToBSTR($SecurePassword)  
$UnsecurePassword = [System.Runtime.InteropServices.Marshal]::PtrToStringAuto($BSTR)
```

```
# Voici le contenu de la variable $SecurePassword pour le mot de passe AAAaaa111  
System.Security.SecureString
```

```
# Le contenu de la variable $BSTR n'est jamais le même  
1548010025976
```

```
# Voici le contenu de la variable $UnsecurePassword  
AAAaaa111
```

Exemple sécuritaire pour se connecter à un partage réseau avec New-SmbMapping

```
$utilisateur = Read-Host -Prompt "Entrer le nom de l'utilisateur"  
$mdp = Read-Host -Prompt "Entrer le mot de passe" -AsSecureString
```

```
# Ces deux lignes de code permettent de récupérer le mot de passe en texte clair.  
$BSTR = [System.Runtime.InteropServices.Marshal]::SecureStringToBSTR($mdp)  
$UnsecurePassword = [System.Runtime.InteropServices.Marshal]::PtrToStringAuto($BSTR)
```

```
New-SmbMapping -LocalPath R: \  
                -RemotePath \\127.0.0.1\C$ \  
                -UserName $utilisateur \  
                -Password $UnsecurePassword
```

```
Clear-Variable -Name UnsecurePassword
```

Exemple moins sécuritaire pour se connecter à un partage réseau avec New-SmbMapping

```
$utilisateur = Read-Host -Prompt "Entrer le nom de l'utilisateur"  
$mdp = Read-Host -Prompt "Entrer le mot de passe"
```

```
New-SmbMapping -LocalPath R: \  
                -RemotePath \\127.0.0.1\C$ \  
                -UserName $utilisateur \  
                -Password $mdp
```

```
Clear-Variable -Name mdp
```


Exécution du code PowerShell sur un ordinateur distant

Exécution de commandes PowerShell sur un ordinateur distant

Si on a des ordinateurs qui ne sont pas membres d'un domaine mais d'un "Groupe de travail", c'est plus difficile d'avoir accès à un autre ordinateur par programmation PowerShell.

Sur chaque ordinateur il faut exécuter la commande suivante:

- **Enable-PSRemoting -Force**
Cette commande démarre le service WinRM et active la fonctionnalité "Gestion à distance de Windows" dans le "Pare-feu Windows".
 - "Get-Service WinRM" permet de vérifier l'état du service WinRM

note: si vous avez une carte réseau de type "Réseau public" vous devez utiliser la commande suivante: **Enable-PSRemoting -SkipNetworkProfileCheck -Force**

Sur chaque ordinateur il faut exécuter la commande suivante:

- **Set-Item wsman:\localhost\client\trustedhosts -Value * -Force**
Cette commande ajoute des ordinateurs auxquels on a confiance.
On peut remplacer le paramètre * par une liste de noms ou d'adresses IP qui sont séparés par des virgules.

Sur chaque ordinateur il faut exécuter la commande suivante:

- **Restart-Service WinRM**
On redémarre le service WinRM pour s'assurer que les nouveaux paramètres sont utilisés.

Pour vérifier si WinRM est fonctionnel sur un ordinateur distant il faut exécuter la commande suivante:

- **Test-WSMan -ComputerName NomOrdinateurDistant**
NomOrdinateurDistant est le nom de l'ordinateur distant sur lequel on veut avoir accès à l'aide de PowerShell.

Configurations dans le "Pare-feu Windows" de l'ordinateur distant

- Activé le paramètre "Partage de fichiers et d'imprimantes"

Modification dans le registre Windows de l'ordinateur distant

- Configurer le paramètre "**LocalAccountTokenFilterPolicy**"
[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Policies\System]
"LocalAccountTokenFilterPolicy"=dword:00000001

Modifier la liste des "TrustedHosts" pour autoriser tous les ordinateurs

```
set-item wsman:\localhost\Client\TrustedHosts -value *
```

Modifier la liste des "TrustedHosts" pour autoriser des ordinateurs

```
set-item wsman:\localhost\Client\TrustedHosts -value ordi1,ordi2
```

Modifier la liste des "TrustedHosts" pour autoriser tous les ordinateurs d'un domaine

```
set-item wsman:\localhost\Client\TrustedHosts *.decinfo.cvm
```

Pour ajouter un ordinateur à une liste existante des "TrustedHosts"

```
$curValue = (get-item wsman:\localhost\Client\TrustedHosts).value  
set-item wsman:\localhost\Client\TrustedHosts -value "$curValue,ordi99"
```

Modifier la liste des "TrustedHosts" pour autoriser des adresses IP

note: une adresse IPv6 doit être entre crochet

```
set-item wsman:\localhost\Client\TrustedHosts -value 192.168.0.100,[0:0:0:0:0:0:0:0]
```

Modifier la liste des "TrustedHosts" sur un ordinateur distant pour autoriser des adresses IP

```
connect-wsman -computername ordi99  
set-item wsman:\ordi99\Client\TrustedHosts -value 192.168.0.100,[0:0:0:0:0:0:0:0]  
disconnect-wsman -computername ordi99
```

Pour afficher la liste des "TrustedHosts"

```
get-item wsman:\localhost\Client\TrustedHosts
```

- Par défaut, l'item TrustedHosts existe mais sa valeur est vide.

Pour afficher la liste des "TrustedHosts" sur un ordinateur distant

```
connect-wsman -computername ordi99  
get-item Wsman:\ordi99\Client\TrustedHosts  
disconnect-wsman -computername ordi99
```

Utilisation du cmdlet invoke-command

invoke-command exécute des commandes ou des scripts sur un ordinateur distant

- invoke-command -computername ordi1,ordi2 {get-process}
- invoke-command -computername ordi1,ordi2 -filepath c:\scripts\MonScript.ps1

Utilisation du cmdlet Enter-PSSession

Enter-PSSession démarre une session interactive avec un ordinateur distant

- Enter-PSSession -computername NomOrdinateur
- Enter-PSSession -computername NomOrdinateur -Credential Domaine\utilisateur

Pour terminer la session interactive on exécute la commande EXIT.

Comment trouver le nom de l'exception pour "Try / Catch"

```
PS D:\_OUTILS> Get-ChildItem -Path "C:\TOTO"
Get-ChildItem : Impossible de trouver le chemin d'accès « C:\TOTO », car il n'existe pas.
Au caractère Ligne:1 : 1
+ Get-ChildItem -Path "C:\TOTO"
+ ~~~~~
+ CategoryInfo          : ObjectNotFound: (C:\TOTO:String) [Get-ChildItem], ItemNotFoundException
+ FullyQualifiedErrorId : PathNotFound,Microsoft.PowerShell.Commands.GetChildItemCommand

PS D:\_OUTILS>
```

Le message d'erreur affiche l'exception "**ItemNotFoundException**" mais ce n'est pas le nom complet de cette exception.

\$Error[0] contient toujours l'erreur la plus récente

```
PS D:\_OUTILS> $Error[0].Exception.GetType().FullName
System.Management.Automation.ItemNotFoundException

PS D:\_OUTILS>
```

\$Error[0].Exception.GetType().FullName

- Cette commande affiche le nom complet de l'exception

\$Error.Count

- Pour compter le nombre d'erreurs dans la variable **\$Error**

\$Error.Clear()

- Pour vider la liste des erreurs dans la variable **\$Error**

Le paramètre "**-ErrorAction Stop**" est souvent utilisé pour que "**Try / Catch**" fonctionne correctement.

Exemple sans "Try and Catch"

```
Remove-Item -Path "C:\TOTO"
```

\$Error[0].Exception.GetType().FullName

```
PS E:\ps> $Error[0].Exception.GetType().FullName
System.Management.Automation.ItemNotFoundException
```

\$Error.Clear()

Exemple avec "Try and Catch"

```
$chemin = "C:\TOTO"
```

```
try
{
    Remove-Item -Path $chemin -ErrorAction Stop
    Write-Host "'$chemin' existe !!!" -ForegroundColor Cyan
}
catch [System.Management.Automation.ItemNotFoundException]
{
    Write-Host "'$chemin' n'existe pas !!!" -ForegroundColor Yellow
}
```

Code pour afficher la liste complète des exceptions

```
$nbExceptions = 0
$nbErreurs = 0

$rep = [appdomain]::CurrentDomain.GetAssemblies() | ForEach-Object {
    Try
    {
        $PSItem.GetExportedTypes() | Where-Object {
            $PSItem.Fullname -Match 'Exception'
        }
    }
    Catch
    {
        $nbErreurs++
    }
}

$exceptions = $rep | Select-Object FullName | Sort-Object FullName
$nbExceptions = $rep.Count
$exceptions

Write-Host "$nbExceptions exceptions" -ForegroundColor Green
Write-Host "$nbErreurs erreurs" -ForegroundColor Yellow
```

Exemple avec les cmdlet Compress-Archive et Expand-Archive

Compress-Archive est un cmdlet du module Microsoft.PowerShell.Archive

Expand-Archive est un cmdlet du module Microsoft.PowerShell.Archive

```
Get-Command -Module Microsoft.PowerShell.Archive
```

```
Compress-Archive -Path c:\temp -DestinationPath c:\backup\temp.zip
```

```
Expand-Archive -Path c:\backup\temp.zip -DestinationPath c:\temp -Force
```

Exemple avec le cmdlet Copy-Item

Copy-Item est un cmdlet du module Microsoft.PowerShell.Management

Création d'une session vers l'ordinateur 407P33

```
$cs = New-PSSession -ComputerName 407P33
```

Le paramètre -ToSession est utilisé pour copier un fichier sur un ordinateur distant

```
Copy-Item -Path C:\Source\test.csv -Destination C:\Source\test.csv -ToSession $cs
```

Le paramètre -FromSession est utilisé pour copier un fichier à partir d'un ordinateur distant

```
Copy-Item -Path C:\Source\src.csv -Destination C:\Source\src.csv -FromSession $cs
```

Exemple avec le cmdlet Get-FileHash

Get-FileHash est un cmdlet du module Microsoft.PowerShell.Utility

Le cmdlet Get-FileHash permet d'utiliser plusieurs algorithmes de hachage sur un fichier

```
$info_shal = (Get-FileHash -Path win10.iso -Algorithm SHA1).HASH
```

Exemple avec le cmdlet Invoke-WebRequest

Invoke-WebRequest est un cmdlet du module Microsoft.PowerShell.Utility

Exemple pour afficher le contenu du fichier par défaut d'un site WEB

Le site "http://ifconfig.me" retourne l'adresse IP qui donne accès à internet.

```
$ip = (Invoke-WebRequest -Uri https://ifconfig.me/ip).Content
```

Voici le contenu de la variable \$ip

```
206.167.112.182
```

Exemple pour sauvegarder dans un fichier, le contenu du fichier par défaut d'un site WEB

```
Invoke-WebRequest -Uri https://ifconfig.me -OutFile C:\_TEMP\ADR1.HTML
```

Le fichier IP.TXT contient SEULEMENT l'adresse IP externe qui donne accès à internet

```
(Invoke-WebRequest -Uri https://ifconfig.me/ip).Content | Out-File C:\_TEMP\IP.TXT
```

Exemple avec le cmdlet Format-Hex

Format-Hex est un cmdlet du module **Microsoft.PowerShell.Utility**

Le cmdlet Format-Hex permet de convertir un caractère en une valeur hexadécimale

```
'PowerShell' | Format-Hex
```

Exemple avec le cmdlet New-TemporaryFile

New-TemporaryFile est un cmdlet du module **Microsoft.PowerShell.Utility**

Le cmdlet New-TemporaryFile permet de créer un fichier temporaire.

```
$fichier1 = New-TemporaryFile  
$fichier1
```

Le fichier temporaire est créé automatiquement dans le dossier de la variable d'environnement TEMP.
\$ENV:TEMP

Exemple avec le cmdlet Send-MailMessage

Send-MailMessage est un cmdlet du module **Microsoft.PowerShell.Utility**

```
Send-MailMessage -From "Nom@Compagnie.ca" `  
                -To "AutreNom@AutreCompagnie.ca" `  
                -Subject "WDS: $ENV:COMPUTERNAME" `  
                -Body "Fin de l'installation sur l'ordinateur: $ENV:COMPUTERNAME" `  
                -SmtpServer smtp.compagnie.ca
```

Continuer l'exécution d'un script après un redémarrage

Dans le registre de Windows, il y a quatre clés qui permettent d'exécuter du code selon quatre situations.

HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run

- permet d'exécuter du code à chaque ouverture de session d'un l'utilisateur spécifique

HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\RunOnce

- permet d'exécuter du code une seule fois lors de l'ouverture de session d'un utilisateur spécifique

HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run

- permet d'exécuter du code à chaque démarrage de l'ordinateur et à l'ouverture de session d'un utilisateur de l'ordinateur

HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\RunOnce

- permet d'exécuter du code une seule fois lors du démarrage de l'ordinateur et à l'ouverture de session d'un utilisateur de l'ordinateur

Exemple d'un script qui va exécuter du code après un redémarrage de l'ordinateur.

```
$RunOnceKey = "HKLM:\Software\Microsoft\Windows\CurrentVersion\RunOnce"
```

```
$code = "c:\windows\system32\WindowsPowerShell\v1.0\powershell.exe"
```

```
$code += " -ExecutionPolicy Unrestricted"
```

```
$code += " -File C:\script\JoindreDomaine.ps1"
```

```
Set-ItemProperty -Path $RunOnceKey `
    -Name 'JoindreDomaine' `
    -Value $code `
    -Force
```

```
Rename-Computer -NewName HV1 `
    -Restart
```

Fonction avancée dans PowerShell

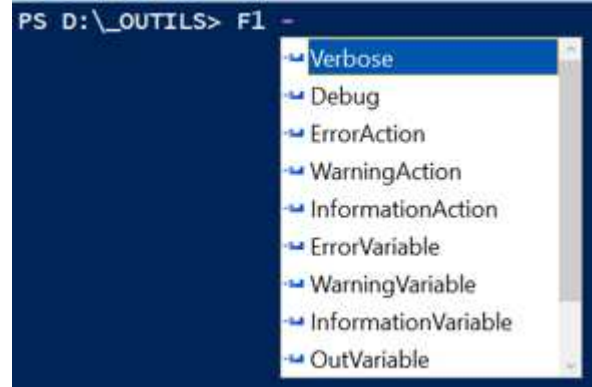
Utilisation de l'attribut **CmdletBinding** et de l'attribut **Parameter**

Voici la fonction avancée la plus simple

```
Function F1 {  
    [CmdletBinding()] Param()  
}
```

Si on enregistre la fonction F1 dans un fichier TEST_F1.PS1

L'attribut **CmdletBinding** permet d'avoir accès à plusieurs paramètres



L'attribut **CmdletBinding** permet d'utiliser **Write-Verbose** dans une fonction

```
Function F2 {  
    [CmdletBinding()] Param()  
    Write-Verbose "Ce message sera affiché en utilisant le paramètre -Verbose."  
    Write-Host "Message normal"  
}
```

Si on enregistre la fonction F2 dans un fichier TEST_F2.PS1

```
PS D:\_OUTILS> D:\_OUTILS\TEST_F2.ps1  
  
PS D:\_OUTILS> F2 -Verbose  
COMMENTAIRES : Ce message sera affiché en utilisant le paramètre -Verbose.  
Message normal  
  
PS D:\_OUTILS> f2  
Message normal  
  
PS D:\_OUTILS>  
PS D:\_OUTILS> |
```


La fonction F3 utilise un paramètre obligatoire

```
Function F3 {  
    [CmdletBinding()]  
    Param([Parameter(Mandatory=$True)]  
        [String]$Source  
    )  
    Write-Host $Source -ForegroundColor Yellow  
}
```

Si on enregistre la fonction F3 dans un fichier TEST_F3.PS1

PowerShell nous demande de fournir une valeur pour le paramètre de notre fonction

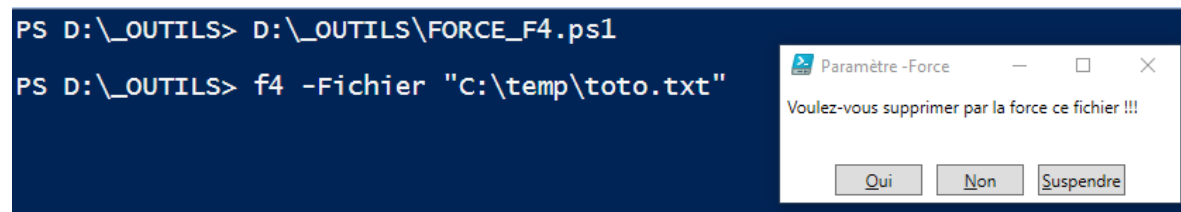


```
PS D:\_OUTILS> D:\_OUTILS\TEST_F3.ps1  
  
PS D:\_OUTILS> F3  
applet de commande F3 à la position 1 du pipeline de la commande  
Fournissez des valeurs pour les paramètres suivants :  
Source :
```

La fonction F4 utilise un paramètre pour afficher une fenêtre de confirmation.

```
Function F4 {  
    [CmdletBinding(SupportsShouldProcess)]  
    Param([String]$Fichier)  
  
    $ConfirmPreference = "Low"  
  
    $info = "Voulez-vous supprimer par la force ce fichier !!!"  
    If ($PSCmdlet.ShouldContinue($info, "Paramètre -Force"))  
    {  
        Remove-Item $Fichier -Force  
    } Else  
    {  
        Write-Host "Annulation de la suppression du fichier !!!" -ForegroundColor Cyan  
    }  
}
```

Si on enregistre la fonction F4 dans un fichier FORCE_F4.PS1



```
PS D:\_OUTILS> D:\_OUTILS\FORCE_F4.ps1  
  
PS D:\_OUTILS> f4 -Fichier "C:\temp\toto.txt"
```

The screenshot also shows a Windows dialog box titled "Paramètre -Force" with the message "Voulez-vous supprimer par la force ce fichier !!!". It has three buttons: "Oui", "Non", and "Suspendre".

Fonctions avec des variables de type "reference"

Exemple 1

```
function carre([ref]$x)
{
    $x.value = $x.value * $x.value
}

$nombre = 2
Write-Host "Valeur initiale = $nombre" -ForegroundColor Yellow

carre([ref]$nombre)
Write-Host "Valeur modifiée = $nombre" -ForegroundColor Yellow
```

Exemple 2

```
function double
{
    Param ([ref]$x)
    $x.value = $x.value * 2
}

$nombre = 8
Write-Host "Valeur initiale = $nombre" -ForegroundColor Yellow

double([ref]$nombre)
Write-Host "Valeur modifiée = $nombre" -ForegroundColor Yellow
```

Installation d'un module PowerShell

Cette section explique où installer un module PowerShell.

L'emplacement du module varie selon l'utilisation

- Pour un utilisateur spécifique
- Pour tous les utilisateurs

Un module est constitué d'une ou plusieurs fonctions.

Si votre script contient plusieurs fonctions (f1, f2, f3, f4, f5) mais que vous voulez rendre disponibles seulement les fonctions f4 et f5, à la fin du module vous devez ajouter les lignes de code suivantes:

```
Export-ModuleMember -Function f4
```

```
Export-ModuleMember -Function f5
```

Les différents dossiers pour les modules

```
$env:PSModulePath -split ';'
```

```
# La commande affiche les trois dossiers
```

```
C:\Users\TECH.FORMATION\Documents\WindowsPowerShell\Modules
```

```
C:\Program Files\WindowsPowerShell\Modules
```

```
C:\Windows\system32\WindowsPowerShell\v1.0\Modules
```

```
C:\Users\TECH.FORMATION\Documents\WindowsPowerShell\Modules
```

```
$HOME\Documents\WindowsPowerShell\Modules
```

Ce dossier est utilisé pour ajouter des modules à un utilisateur spécifique.

```
C:\Program Files\WindowsPowerShell\Modules
```

```
$env:ProgramFiles\WindowsPowerShell\Modules
```

Ce dossier est utilisé pour ajouter des modules à tous les utilisateurs.

```
C:\Windows\system32\WindowsPowerShell\v1.0\Modules
```

```
$PSHome\Modules
```

Ce dossier est réservé aux modules de Windows.

Il ne faut pas installer des modules dans ce dossier.

Ajout d'un module pour un "utilisateur spécifique"

Pour qu'un utilisateur puisse avoir accès à un module qu'il a créé ou téléchargé à partir d'un site web comme <https://www.powershellgallery.com>, il faut installer le module dans un dossier qui est spécifique à l'utilisateur.

\$HOME\Documents\WindowsPowerShell\Modules\MonModule\MonModule.psm1

IMPORTANT: le nom du dossier **MonModule** et le préfixe du nom du fichier **MonModule.psm1** doivent être exactement le même, sinon PowerShell ne trouvera pas le module.

Ajout d'un module pour "tous les utilisateurs"

Pour que tous les utilisateurs puissent avoir accès à un module qu'un administrateur a créé ou téléchargé à partir d'un site web comme <https://www.powershellgallery.com>, il faut installer le module dans le dossier "**C:\Program Files\WindowsPowerShell\Modules**".

\$env:ProgramFiles\WindowsPowerShell\Modules\MonModule\MonModule.psm1

IMPORTANT: le nom du dossier **MonModule** et le préfixe du nom du fichier **MonModule.psm1** doivent être exactement le même, sinon PowerShell ne trouvera pas le module.

Exemple d'un module PowerShell

Voici le code du fichier EmptyFile.psm1

Permet de créer rapidement un fichier vide en spécifiant sa taille

```
function New-EmptyFile
{
    param( [string]$FilePath, [double]$Size )

    # Utilisation d'une classe .NET
    $file = [System.IO.File]::Create($FilePath)
    $file.SetLength($Size)
    $file.Close()

    Get-Item $file.Name
}
```

Voici l'emplacement du fichier EmptyFile.psm1

\$HOME\Documents\WindowsPowerShell\Modules\EmptyFile\EmptyFile.psm1

Exemple d'utilisation de la fonction New-EmptyFile

New-EmptyFile -FilePath c:_temp\big_file.txt -Size 100gb

Répertoire : c:_temp

Mode	LastWriteTime	Length	Name
----	-----	-----	----
-a----	2023-04-30 08:06	107374182400	big_file.txt