**Documentation sur la programmation PowerShell**

**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 "Administrateur Hyper-V".
* 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 "B61\_SERVEUR1" `

-Credential $Cred `

-ScriptBlock { get-process }

**Exemple 3:**

**# on enregistre les informations pour l'authentification dans une variable**

$Cred = Get-Credential

$NOM\_VM = "B61\_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 `

-FilePath $nom\_fichier `

-Credential $Cred

**Utilisation d'une variable globale dans un ScriptBlock**

Il est possible d'utiliser une variable globale dans un ScriptBlock qui s'exécute sur un ordinateur distant à condition d'utiliser **$using:** devant le nom de la variable.

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

$prog = "win\*"

$NOM\_VM = "B61\_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 "B61\_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 "B61\_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 "B61\_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 "B61\_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 placer une virgule devant le nom de la variable lors de l'appel.

$p = ("power\*","win\*")

Invoke-Command -VMName "B61\_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 = "B61\_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 `

-ScriptBlock $code `

-ArgumentList **$siteFTP** `

-Credential $Cred

$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 "FORMATION\Administrateur"

Invoke-Command -VMName "B61\_SERVEUR1","B61\_SERVEUR2" `

-Credential $Cred `

-ScriptBlock { Get-LocalUser –Name T\* }

**Résultat de l'exécution du ScriptBlock**

Name Enabled Description PSComputerName

---- ------- ----------- --------------

TECH True B61\_SERVEUR1

TEST True B61\_SERVEUR2

**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

**Le paramètre –Password de New-SmbMapping est de type <String>**

**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"**

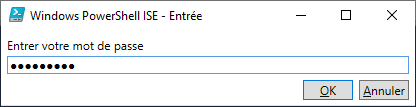
**Convertir une chaîne sécurisée en texte clair**

**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 votre nom d'utilisateur"**

**$mdp = Read-Host -Prompt "Entrer votre mot de passe" -AsSecureString**

**# Ces deux lignes de code permettent de récupérer le mot de passe en texte clair.**

**$BSTR = [System.Runtime.InteropServices.Marshal]::SecureStringToBSTR($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 votre nom d'utilisateur"**

**$mdp = Read-Host -Prompt "Entrer votre 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 le 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 *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

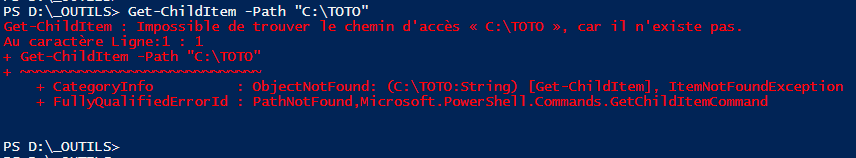
**Utilisation du cmdlet Enter-PSSession**

Enter-PSSession démarre une session interactive avec un ordinateur distant

* Enter-PSSession -computername NomOrdinateur
* Enter-PSSession -computername NomOrdinateur -Credential Domaine\utilisateur

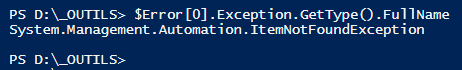
Pour terminer la session interactive on exécute la commande EXIT.

**Comment trouver le nom de l'exception pour "Try / Catch"**



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**



**$Error[0].Exception.GetType().FullName**

* **Cette commande permet de trouver 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 avec "Try and Catch"**

**try**

**{**

**Get-ChildItem -Name "C:\TOTO" -ErrorAction Stop**

**}**

**catch [System.Management.Automation.ItemNotFoundException]**

**{**

**Write-Verbose "Le chemin $($PSItem.TargetObject) n'existe pas!" –Verbose**

**}**

**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\srv.csv -Destination C:\Source\srv.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\_sha1 = (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