Grégoire Roumache

Laboratoires de Powershell

Niveau 1

niveau-1-ex-01.ps1

```
# Trouvez la commande originale dont les alias sont dir et ls.
Get-Alias dir   # => Get-ChildItem
Get-Alias ls    # => Get-ChildItem

# Utilisez la pour renvoyer le listing du répertoire courant dans une variable.
$variable = Get-ChildItem

# Affichez ensuite le contenu de cette variable
$variable

# Affichez maintenant uniquement la première ligne de la variable
$variable[0]

# Affichez les méthodes et propriétés associées à cette variable
$variable.GetType()
$variable | Get-Member

# Utilisez la propriété adéquate pour afficher le mode d'accès autorisé,
# la date de dernière écriture ainsi que le répertoire associés à la
```

```
# première ligne de la variable
$variable[0] | Get-Member
$variable.GetAccessControl()
$variable.LastWriteTime
$variable.Parent
```

niveau-1-ex-02.ps1

```
# Lister les fichiers d'un répertoire qui ont pour extension
# txt via la commande Where-Object (?).
# Utilisez un pipeline pour réaliser la commande en une seule ligne.

Get-ChildItem | Where-Object {$_.Extension -eq ".txt"}
# ls | ? {$_.Extension -eq ".txt"}
```

niveau-1-ex-03.ps1

```
# Lister les fichiers d'un répertoire dans l'ordre de dernière modification
Get-ChildItem | Sort-Object LastWriteTime
```

niveau-1-ex-04.ps1

```
# Créez un tableau recensant tous les processus Notepad
$variable = Get-Process | Where-Object {$_.ProcessName -eq "notepad"}

# Tuez tous les processus notepad en utilisant la méthode
# adéquate de la variable

# $variable | Get-Member  # => get the properties and methods
$variable.Kill()
```

niveau-1-ex-05.ps1

```
# Sur base d'un fichier texte contenant une liste de noms, créez des répertoires
associés
# Astuces:
# - Créez un fichier texte contenant la liste de noms
# - Copiez le contenu du fichier dans une variable
# - Utilisez la commande New-Item pour créez les répertoire

$dir_path =  "C:\Users\groum.LAPTOP-MUKCF9T7\Downloads"
$file_path = "C:\Users\groum.LAPTOP-MUKCF9T7\Documents\Programmation\Scripting-
Unix-Powershell\Prog-OS-Powershell\liste-nom.txt"
$variable = Get-Content $file_path
```

```
# $variable | % {echo $_}
$variable | ForEach-Object {Write-Output ($dir_path + "\" + $_)}
$variable | ForEach-Object {New-Item -ItemType "directory" ($dir_path + "\" + $_)}
```

Niveau 2

niveau-2-ex-01.ps1

```
# Créez un script qui inverse les lettres des mots d'un texte
# Inversez les lettres, pas les mots
# Exemple :
# « Ceci est un tout petit script »
# « iceC tse nu tuot titep tpircs »

$text = Read-Host -Prompt 'Input text to reverse'
$text = $text.Split() | ForEach-Object {
    $word = $_.ToCharArray()
    for ($i = $_.ToCharArray().Count - 1; $i -ge 0 ; $i--) {
        $letter = $word[$i]
        $letter
    }
    ' '
}

Write-Host -Separator "" -ForegroundColor "Green" $text
```

niveau-2-ex-02.ps1

```
# Créez un script réalisant les tâches suivantes :
# 1. Ajout de deux clés de registre
#     - Test-Version
#         - Valeur : 2.0
#     - Test-Region
#         - Valeur : Belgium
#     - Chemin : HKEY_LOCAL_MACHINE\SOFTWARE
#     - Remarque : Stockez les données dans une variable tableau (hashtable)

$hashtable_register_keys = @{"Test-Version" = "2.0"; "Test-Region" = "Belgium"}

function Set-RegKey {
    param (
        $KeyName,
        $KeyValue
    )
    $path = "HKLM:\SOFTWARE"

    Write-Output "New register key in: $path, name: $KeyName, value: $KeyValue"
    New-ItemProperty -Path $path -Name $KeyName -Value $KeyValue
```

```
}

# 2. Enregistrement des actions dans un fichier log
#     - Chemin : vous stockez vos logs dans un répertoire « logs » à la racine
d'un disque
#     - Information à enregistrer
#         - Démarrage du script
#         - Création de la clé de registre « Test-Version », Valeur : 2.0
#         - Création de la clé de registre « Test-Région », Valeur : Belgium
#         - Fin de script
#     - Utilisez des variables pour les données

function Write-Log {
    param (
        $message = "This is not an exercise."
    )
    $log_path = "E:\logs-niveau2-02-ex.txt"
    $start_log_msg = "[ $(Get-Date) ] "
    $log_msg = $start_log_msg + $message

    Write-Output $log_msg
    $log_msg | Out-File -FilePath $log_path -Append
}

# affiche les valeurs de clé de registres
# Get-PSDrive
# on voit qu'il y a deux entrées pour le registre:
# HKCU (HKEY_CURRENT_USER) et HKLM (HKEY_LOCAL_MACHINE)

# Information à enregistrer
#    - Démarrage du script
#    - Création de la clé de registre « Test-Version », Valeur : 2.0
#    - Création de la clé de registre « Test-Région », Valeur : Belgium
#    - Fin de script
Write-Log -message "Start of the script"

$hashtable_register_keys.Keys | ForEach-Object {
    $Name = $_
    $Value = $hashtable_register_keys.$_
    Write-Log -message "Create register key: $Name : $Value"
    Set-RegKey -KeyName $Name -KeyValue $Value
}

Write-Log -message "End of the script"

Remove-ItemProperty -Path "HKLM:\SOFTWARE" -Name "Test-Version"
Remove-ItemProperty -Path "HKLM:\SOFTWARE" -Name "Test-Region"
```

niveau-2-ex-03.ps1

```
# Dans l'exercice précédent, on suppose que les clés sont inexistantes avant le
lancement du script.
# Dans le cas contraire, PowerShell génère une erreur.
# Exercice : Implémentez la gestion des erreurs via try-catch-finally

$hashtable_register_keys = @{"Test-Version" = "2.0"; "Test-Region" = "Belgium"}

function Set-RegKey {
    param (
        $KeyName,
        $KeyValue
    )
    $path = "HKLM:\SOFTWARE"

    try {
        New-ItemProperty -Path $path -Name $KeyName -Value $KeyValue
        Write-Output "New register key in: $path, name: $KeyName, value:
$KeyValue"
    }
    catch {
        Write-Host -ForegroundColor "Magenta" "The key $KeyName already exists."
    }
}

function Write-Log {
    param (
        $message = "This is not an exercise."
    )
    $log_path = "E:\logs-niveau2-02-ex.txt"
    $start_log_msg = "[ $(Get-Date) ] "
    $log_msg = $start_log_msg + $message

    Write-Output $log_msg
    $log_msg | Out-File -FilePath $log_path -Append
}

Write-Log -message "Start of the script"

$hashtable_register_keys.Keys | ForEach-Object {
    $Name = $_
    $Value = $hashtable_register_keys.$_
    Write-Log -message "Create register key: $Name : $Value"
    Set-RegKey -KeyName $Name -KeyValue $Value
}

Write-Log -message "End of the script"

Remove-ItemProperty -Path "HKLM:\SOFTWARE" -Name "Test-Version"
Remove-ItemProperty -Path "HKLM:\SOFTWARE" -Name "Test-Region"
```

Niveau 3

niveau-3-ex-01.ps1

```
# Créer un script sur un client qui permet d'afficher les services d'une machine
distante.
# Fonctionnalités du script :
# - Demande le nom de la machine à interroger
#     - Si rien n'est entré, utiliser la variable d'environnement locale
#     - Utilisez le paramètre ComputerName
# - Demande si on cherche les services "Stopped" ou "Running" ==> Vérifier que
l'utilisateur entre bien 1 ou 2
# - Demande le nombre de services à afficher ==> Validation entre 2 et 10
# - Utilisez une fonction avec des paramètres avancés       ====>>
??????????????????

# /!\ Désactiver les parefeux /!\
# les noms des machines sont dispo dans "active directory users and computers
management"

# 1. get the machine's name
$machine_name = Read-Host "Machine name (leave blank for localhost)"
if ($machine_name.Length -eq 0) { $machine_name = "localhost" }

# 2. get the service type/status (running/stopped)
$service_type = Read-Host "Choose between running (1) or stopped (2) services.
Please type 1 or 2."
$status = "Stopped"
if ($service_type -eq 1) { $status = "Running" }

# 3. get the number of services to display
$nb_services = Read-Host "Choose the number of services to display (between 1 and
10)"

# 4. get the services and filter them
Get-Service -ComputerName $machine_name `
    | Where-Object {$_.Status -eq $status} `
    | Select-Object -First $nb_services `
    | Write-Host
```

niveau-3-ex-02.ps1

```
# Créez un script qui permet de vérifier l'état d'un service.
# Selon son état, demander pour le démarrer/arrêter.
# Fonctionnalités du script :
#     - Demande le nom de la machine à interroger
#     - Demande à l'utilisateur le nom du service
#     - Affiche l'état du service
#     - Si le service est "Stopped", demander s'il faut le démarrer
#     - Si le service est "Running", demander s'il faut le stopper
#     - Affiche l'état final du service
```

```
$machine_name = Read-Host "Name of the machine" # ex : localhost
$service_name = Read-Host "Name of the service" # ex : WinDefend

$service = Get-Service -ComputerName $machine_name | Where-Object {$_.Name -eq
$service_name}
Write-Host $service " -- " $service.Status

if ($service.Status -eq "Running") {
    $change_status = Read-Host "Should we stop this service (y/n)"
} else {
    $change_status = Read-Host "Should we start this service  (y/n)"
}

if ($change_status -eq "y" -and $service.Status -eq "Running") {
    $service | Set-Service -Status "Stopped"
} elseif ($change_status -eq "y" -and $service.Status -eq "Stopped") {
    $service | Set-Service -Status "Running"
}

Write-Host $service " -- " $service.Status
```

niveau-3-ex-03.ps1

```
# Créez un script qui permet d'ajouter ou de modifier une clé de registre sur une
machine distante.
# Fonctionnalités du script :
#     - Utilisation des fonctions avancées          ====>>          Quelles
fonctions ??
#     - Gestion des erreurs avec try/catch
#     - Gestion du forçage en cas de clé déjà présente, ce qui produit une
modification de la valeur    ====>>      What ?
#     - Le script demande à l'utilisateur :
#         - D'entrer le nom de la machine distante
#         - Le nom de la clé à ajouter/modifier
#         - La valeur de la clé
#         - Donne le choix entre les chemins suivants :
#             - HKLM:\Software\
#             - HKLM:\System\

$machine_name = Read-Host "Name of the machine"
$key_name = Read-Host "Name of the key"
$key_value = Read-Host "Value of the key"
$path = Read-Host "Choose between path (1) 'HKLM:\Software\' and (2)
'HKLM:\System\'. Type 1 or 2."

if ($path -eq 1) {
    $path = "HKLM:\Software\"
} else {
    $path = "HKLM:\System\"
}
```

```
try {
    New-ItemProperty -ComputerName $machine_name -Path $path -Name $key_name -
Value $key_value
} catch {
    Write-Host -ForegroundColor "Magenta" "The key $key_name already exists."
}
```

Niveau 4

niveau-4-ex-01.ps1

```
# Modifiez la politique d'exécution de script de la machine locale
# pour qu'elle n'exécute que des scripts signés qu'ils soient locaux ou non.

Get-ExecutionPolicy
# - RemoteSigned => default = scripts can run, requires a signature for downloaded
scripts
# - AllSigned => every scripts needs to be signed
# Set-ExecutionPolicy AllSigned
# Set-ExecutionPolicy RemoteSigned
Get-ExecutionPolicy

# https://docs.microsoft.com/en-
us/powershell/module/microsoft.powershell.core/about/about_execution_policies?
view=powershell-7.1
```

niveau-4-ex-02.ps1

```
# Créez un script simple
#   1. Le signer
#   2. L'exécuter

$file = 'test.ps1'

# 1. création d'un certificat auto-signé
$cert = New-SelfSignedCertificate -Subject "greg's cert" -Type CodeSigningCert -
CertStoreLocation Cert:\CurrentUser\My

# 2. on a une erreur car le cert auto-signe n'est pas approuvé
#    ==> le déplacer dans le répertoire de l'ordinateur
Move-Item -Path $cert.PSPath -Destination "Cert:\CurrentUser\Root"

# 3. créer un script simple
New-Item -Path . -Name $file -ItemType "file" -Value "Write-Output 'Hello !!'"

# 4. signer ce script
Set-AuthenticodeSignatire -FilePath "$(pwd)\$file" -Certificate $cert
```

```
# 5. lancer le script
$file
```

niveau-4-ex-03.ps1

```
# Sur base d'un serveur Active Directory fonctionnel
#   - Domaine créé (henallux.local ou autre...)
#   - DNS fonctionnel sur votre domaine
#   1. Joindre un PC client au domaine
#   2. Créer les OU suivantes
#     -  Professeurs
#     -  Etudiants
#     -  Administration
#   3. En un script sur base d'un fichier csv, créer et ajouter
#      les utilisateurs en fonction des paramètres du fichier csv
#   4. Vérifier que les utilisateurs peuvent utiliser le pc client


$file = "C:\Users\groum.LAPTOP-MUKCF9T7\Documents\Programmation\Scripting-Unix-Powershell\Prog-OS-Powershell\PowerShellSecure.csv"


# 1. Joindre un PC client au domaine
Add-Computer -ComputerName "greg-laptop" -DomainName "greg-domain\domain-controller"

Get-ADComputer -Filter *


# 2. Créer les OU
New-ADOrganizationalUnit -Path "DC=henallux,DC=local" -Name "Professeurs"
New-ADOrganizationalUnit -Path "DC=henallux,DC=local" -Name "Etudiants"
New-ADOrganizationalUnit -Path "DC=henallux,DC=local" -Name "Administration"

Get-ADOrganizationalUnit -Filter *


# 3. En un script sur base d'un fichier csv
# firstname;lastname;username;departement;ChangePWlogon;ou
$csv = Import-Csv -Path $file -Delimiter ';'
$csv | ForEach-Object {
    New-ADUser -GivenName $_.firstname -Surname $_.lastname -Name $_.username `
        -Department $_.departement -ChangePasswordAtLogon $_.ChangePWlogon -Path $_.ou
}

Get-ADUser -Filter *


# 4. Vérifier que les utilisateurs peuvent utiliser le pc client
# se connecter avec un des utilisateurs
```

```
# https://docs.microsoft.com/en-
us/powershell/module/microsoft.powershell.management/add-computer?view=powershell-
5.1
```