Table des matières

I. Présentation

A. Préambule

Ce document est un support de cours dont l'objet est de fournir les clés de compréhension du PowerShell. Il ne peut pas faire l'objet de reproductions à des fins commerciales sans le consentement de son auteur.

B. Technologies de scripting

Tout système d'exploitation nécessite l'emploi de technologies complémentaires pour automatiser des tâches récurrentes. Unix et Linux disposent de différents shells. Avec Dos, puis Windows, Microsoft a développé différentes technologies de scripting. Initialement, il y a eu les commandes autour du DOS. Sous Windows NT, nous avions eu droit à Kix. Avec Windows, Bill Gates voulait faire de Visual Basic le langage universel. Nous avons eu droit à Vbscript utilisé dans Windows Scripting Host. Et puis, avec l'avènement de .Net, Microsoft a décidé de mettre en avant le PowerShell. Certains langages tels que Perl, Python présentent l'avantage de la portabilité.
Le PowerShell, d'un point de vue syntaxique, emprunte à différents langagestels que le Perl et aussi le Shell Unix.
La critique qu'on peut faire à Powershell est la lenteur de l'exécution due à l'utilisation du Framework .Net.

C. Pré-requis

Windows PowerShell nécessite Microsoft .NET Framework 2.0. Certains composants tels que Windows PowerShell ISE, l'éditeur graphique, les cmdlets Out-GridView , Get-WinEvent (disponible sur Windows Vista et supérieur) nécessitent Microsoft .NET Framework 3.5 avec le Service Pack 1. La cmdlet Export-Counter fonctionne uniquement sur Windows 7 et supérieur

D. Nouvelles fonctionnalités de la version Windows PowerShell 2

La nouvelle version de PowerShell, présente sur Windows 7, présente de nouvelles fonctionnalités, dont certains requièrent le framework.net. 3.5 :

100 nouvelles applets de commande, les cmdlets
Exécution à distance.
Environnement d'écriture de scripts intégré (ISE) PowerShell Windows.
Travaux en arrière-plan.
Débogueur.
Modules.
Transactions.
Événements.
Fonctions avancées.
Internationalisation du script.
Aide en ligne.

E. Les outils

- Windows ISE, intégré à Windows 7
- Sapien's PrimalScript IDE
- PowerShell Scriptomatic

II. Premiers pas

A. Les applets de commande ou cmdlets

Le langage PowerShell s'appuie sur un jeu de commandes qui peut être enrichi par l'installation de logiciels comme Microsoft Exchange 2007.

B. L'interpréteur

A partir de la ligne de commande, tapez *powershell* !

C. Protection

1. Le niveau de sécurité : Get-ExecutionPolicy

2. Changer le niveau de sécurité : Set-ExecutionPolicy

Le paramètre *scope* permet de limiter le niveau de sécurité à l'utilisateur courant, à la machine, etc.

AllSigned	Seul les scripts "signés" fonctionnent
RemoteSigned	Les scripts locaux fonctionne, ceux d'internet doivent être "signés"
Restricted	Aucun script externe autorisé
Unrestricted	Aucune limite pour l'exécution des scripts

3. Signature

```
Get-AuthenticodeSignature "C:\windows\notepad.exe"
```

4. Voir aussi

```
GetHelp about_Execution_Policies
GetHelp about_Profiles
Get-ExecutionPolicy
Set-ExecutionPolicy
Set-AuthenticodeSignature
```

D. Aide

1. Informations de plate-forme : *Get-Host*

Get-Host fournit, notamment, la version du PowerShell.

2. La liste des commandes : *Get-Command*

3. L'aide : Get-Help

```
Get-Help about
get-help Set-Service -examples
get-help Set-Service -detailed
get-help Set-Service -full
```

4. Méthodes et propriétés associées à une cmdlet

```
Get-Date|Get-Member
Get-Date | Get-Member -membertype methods
Get-Date | Get-Member -membertype properties
Get-Process | Get-Member -membertype aliasproperty
```

5. Afficher les propriétés d'un cmdlet

```
Get-Process |Select-Object ProcessName,PrivateMemorySize
```

6. Afficher les méthodes et propriétés d'un objet

L'utilisation du connecteur MySQL .Net suppose que vous l'ayez téléchargé et installé au préalable.

```
[void][system.reflection.Assembly]::LoadFrom("C:\Program Files\MySQL\MySQL Connector Net
6.3.6\Assemblies\v2.0\MySql.Data.dll")
New-Object MySql.Data.MySqlClient.MySqlConnection | Get-Member
```

7. Les fournisseurs PowerShell : Get-PSProvider

E. Exécution des scripts

1. Exécution d'un script

```
powershell d:\scripts\monscript.ps1
```

2. Appel d'un autre script

```
Invoke-Expression d:\scripts\monscript.ps1
& d:\scripts\monscript.ps1
d:\scripts\monscript.ps1
Invoke-Expression "d:\ scripts\monscript.ps1"
```

3. Appel d'un programme

```
Invoke-Item c:\windows\system32\calc.exe
```

4. Mesurer le temps d'exécution : Measure-Command

```
Clear
Write-Output "Ceci est un test"
$temps=Measure-Command { sleep -Seconds 1}
Write-Output "Mesure n°1: $temps"
$temps=Measure-Command {Write-Output "La commande est exécuté. Le message n'est pas
affiché." }
Write-Output "Mesure n°2: $temps"
$temps=Measure-Command {Write-host "La commande est exécuté. Et, cette fois, vous pouvez
le voir." }
Write-Output "Mesure n°3: $temps"
Measure-Command {d:\scripts\monscript.ps1}
```

5. Tempo

```
Start-Sleep -s 10
Start-Sleep -m 10000
```

F. Historique

1. Visualiser l'historique

```
Get-History
Get-History 32 -count 32
$MaximumHistoryCount = 150
```

2. Récupérer l'historique

```
Get-History | Export-Clixml "d:\scripts\my_history.xml"
Import-Clixml "d:\scripts\my_history.xml" | Add-History
```

3. Exécuter une commande de l'historique

```
Invoke-History 3
```

4. Voir aussi

```
about_history
Invoke-History
Add-History
Clear-History
```

G. Informations de langue

```
Get-Culture
```

H. Passage d'arguments

1. Par tableau

```
$res=0
foreach($argument in $args)
{
      Write-Host $argument
```

```
}
```

2. Par la méthode Param

```
./monscript.ps1  -path "c:\windows" -value 1
Param ([string]$path, [int]$value)
Write-host   “le chemin est : $path et la valeur est : $value"
```

I. Commentaires

Commenter une ligne : #
Commenter un bloc : <# … #>

III. Cmdlets système

A. Le journal d'événements

```
Get-EventLog -list
Get-EventLog -list | Where-Object {$_.logdisplayname -eq "System"}
Get-EventLog system -newest 3
```

B. Les services

1. La liste des services

```
Get-Service
Get-Service | Where-Object {$_.status -eq "stopped"}
Get-Service | Where-Object {$_.status -eq "running"} |Select-Object Name, DisplayName
Get-Service | Sort-Object status,displayname
```

2. Démarrer, arrêter un service

```
Stop-Service MySQL
Start-Service MySQL
Restart-Service MySQL
Restart-Service -displayname "MySQL"
```

3. Mettre en suspens, reprendre un service

Le service en état suspendu ne permet plus des connexions supplémentaires.

```
Suspend-Service MySQL
Resume-Service tapisrv
```

4. Modifier les propriétés des services

```
set-service -name lanmanworkstation -DisplayName "LanMan Workstation"
get-wmiobject win32_service -filter "name = 'SysmonLog'"
set-service sysmonlog -startuptype automatic
Startuptype : manual, stopped
Set-Service clipsrv -startuptype "manual"
Set-Service "ati hotkey poller" -description "This is ATI HotKey Poller service."
```

C. Les process

1. Liste des process

```
Get-Process
Get-Process winword
Get-Process winword,explorer
Get-Process w*
Get-Process | Select-Object name,fileversion,productversion,company
```

2. Arrêter un process

```
Stop-Process 3512
Stop-Process -processname notepad
Stop-Process -processname note*
```

D. WMI

```
Get-WmiObject win32_bios
Get-WmiObject win32_bios -computername atl-fs-01
Get-WmiObject win32_bios | Select-Object *
Get-WmiObject win32_bios | Select-Object -excludeproperty "_*"
$data = Get-WmiObject Win32_OperatingSystem
$share = Get-WmiObject Win32_Share
$cpu = (Get-WmiObject win32_processor | select-object loadpercentage).loadpercentage
$availMem =( Get-WmiObject win32_perfFormattedData_perfos_memory | select-object
availableMbytes).availableMBytes / 1024
```

IV. Gestion des heures et des dates

A. Obtenir la date et l'heure : Get-Date

```
Get-Date
Get-Date -displayhint date
Get-Date -displayhint time
$A = Get-Date 5/1/2006
$A = Get-Date "5/1/2006 7:00 AM"
(Get-Date).AddMinutes(137)
$date = Get-Date -f "dd-MM-yyyy"
```

1. Méthodes associées à la cmdlet Get-Date

```
AddSeconds
AddMinutes
AddHours
AddDays
AddMonths
AddYears
```

B. Changer la date et l'heure : Set-Date

```
Set-Date -date "6/1/2006 8:30 AM"
Set-Date (Get-Date).AddDays(2)
Set-Date (Get-Date).AddHours(-1)
Set-Date -adjust 1:37:0
```

C. Calculs sur date

```
New-TimeSpan $(Get-Date) $(Get-Date -month 12 -day 31 -year 2006)
$(Get-Date)
New-TimeSpan $(Get-Date) $(Get-Date -month 12 -day 31 -year 2006)
New-TimeSpan $(Get-Date) $(Get-Date -month 12 -day 31 -year 2006 -hour 23 -minute 30)
New-TimeSpan  $(Get-Date 1/1/2011) $(Get-Date 31/12/2011)
```

V. Gestion des fichiers

PowerShell propose les mêmes commandes pour manipuler le système de fichiers et la base de registre.

A. Suppression de fichiers : Remove-Item

```
Remove-Item d:\scripts\test.txt
Remove-Item d:\scripts\*
Remove-Item d:\scripts\* -recurse
Remove-Item c:\*.tmp -recurse
Remove-Item d:\scripts\* -exclude *.wav
Remove-Item d:\scripts\* -include .wav,.mp3
Remove-Item d:\scripts\* -include *.txt -exclude *test*
```

B. Copie de fichiers : Copy-Item

```
Copy-Item d:\scripts\test.txt c:\test
Copy-Item d:\scripts\* c:\test
Copy-Item d:\scripts\*.txt c:\test
Copy-Item d:\scripts c:\test -recurse
```

C. Création de fichiers : New-Item

```
New-Item d:\scripts\Windows PowerShell -type directory
New-Item d:\scripts\new_file.txt -type file
New-Item d:\scripts\new_file.txt -type file -force
```

D. Déplacer les fichiers

```
Move-Item d:\scripts\test.zip c:\test
Move-Item d:\scripts\*.zip c:\test
Move-Item d:\scripts\test.zip c:\test -force
Move-Item d:\scripts\950.log c:\test\mylog.log
```

E. Renommer les fichiers

```
Rename-Item d:\scripts\test.txt new_name.txt
```

F. Informations sur les fichiers, répertoires et clés de registres

```
$(Get-Item c:\).lastaccesstime
$(Get-Item hklm:\SYSTEM\CurrentControlSet\services).subkeycount
```

G. Tester l'existence d'un chemin

```
Test-Path d:\scripts\test.txt
Test-Path d:\scripts\*.wma
Test-Path HKCU:\Software\Microsoft\Windows\CurrentVersion
```

H. Lire un répertoire

```
Get-ChildItem -recurse
Get-ChildItem HKLM:\SYSTEM\CurrentControlSet\services
Get-ChildItem d:\scripts\*.* -include *.txt,*.log
Get-ChildItem d:\scripts\*.* | Sort-Object length
Get-ChildItem d:\scripts\*.* | Sort-Object length -descending
```

I. La sécurité

```
Get-Acl d:\scripts | Format-List
Get-Acl HKCU:\Software\Microsoft\Windows
Get-Acl d:\scripts\*.log | Format-List
```

VI. Manipulation des fichiers

A. Ajout à u fichier

```
Add-Content d:\scripts\test.txt "The End"
Add-Content d:\scripts\test.txt "`nThe End"
```

B. Caractères spéciaux

`0 Null
`a Beep
`b Backspace
`n New line
`r Carriage return
`t Horizontal tab
`' Single quote
`" Double quote

C. Recherche dans un fichier

```
Get-Content d:\scripts\test.txt | Select-String "Failed" -quiet
Get-Content c:\config.sys |Select-String files
Get-Content d:\scripts\test.txt | Select-String "Failed" -quiet -casesensitive
```

D. Création d'un fichier

```
Get-Process | Tee-Object -file d:\scripts\test.txt
```

E. Effacer le contenu d'un fichier

```
Clear-Content d:\scripts\test.txt
$A = Get-Date; Add-Content d:\test.log $A+`n
```

F. Convertir en Html

```
Get-Process | ConvertTo-Html | Set-Content d:\scripts\test.htm
Get-Process | ConvertTo-Html name,path,fileversion | Set-Content d:\scripts\test.htm
Get-Process | ConvertTo-Html name,path,fileversion -title "Process Information" | Set-
Content d:\scripts\test.htm
Get-Process |
ConvertTo-Html name,path,fileversion -title "Process Information" -body "Information
about the processes running on the computer." |
Set-Content d:\scripts\test.htm
Get-Process |
ConvertTo-Html name,path,fileversion -title "Process Information" -body "<H2>Information
about the processes running on the computer.</H2>" |
Set-Content d:\scripts\test.htm
Get-ChildItem c:\windows\*.exe | ConvertTo-Html name, length| Set-Content d:\index.html
```

G. Compter les lignes d'un fichier

```
Get-Content c:\config.sys | Measure-Object
Get-Content d:\scripts\test.txt | Select-Object -last 5
```

H. Lire un fichier CSV

```
Import-Csv d:\scripts\test.txt
Import-Csv d:\scripts\test.txt | Where-Object {$_.department -eq "Finance"}
Import-Csv d:\scripts\test.txt | Where-Object {$_.department -ne "Finance"}
Import-Csv d:\scripts\test.txt | Where-Object {$_.department -eq "Finance" -and $_.title
-eq "Accountant"}
Import-Csv d:\scripts\test.txt | Where-Object {$_.department -eq "Research" -or $_.title
-eq "Accountant"}
```

I. Les fichiers XML

```
Get-ChildItem d:\scripts | Export-Clixml d:\scripts\files.xml
$A = Import-Clixml d:\scripts\files.xml
$A | Sort-Object length
```

J. Export CSV

```
Get-Process | Export-Csv d:\scripts\test.txt
Get-Process | Export-Csv d:\scripts\test.txt -encoding "unicode"
#TYPE System.Diagnostics.Process
Get-Process | Export-Csv d:\scripts\test.txt -notype
Get-Process | Export-Csv d:\scripts\test.txt -force
```

K. Sauvegarde d'un fichier

```
Set-Content d:\scripts\test.txt "This is a test"
Get-Process|Set-Content d:\test.txt
```

L. Export Xml

```
Get-Process | Export-Clixml d:\scripts\test.xml
```

M. Sauvegarder dans un fichier texte

```
Get-Process | Out-File d:\scripts\test.txt
Get-Process | Out-File d:\scripts\test.txt -width 120
```

VII. Eléments du langage

A. Les variables et les constantes

1. Les variables

```
$Mem= WmiObject Win32_ComputerSystem
$Mbyte =1048576 # Another variable
"Memory Mbyte " + [int]($Mem.TotalPhysicalMemory/$Mbyte)
[int]$a =7
$a +3
$a
$DriveA, $DriveB, $DriveC, $DriveD = 250, 175, 330, 200
$i=0
[string]$Type = "Win32"
$WMI = Get-wmiobject -list | Where-Object {$_.name -match $Type}
Foreach ($CIM in $WMI) {$i++}
Write-Host 'There are '$i' types of '$Type
```

2. Les constantes

```
Set-Variable Thermometer 32 -option constant.
Set-Variable AllOverPlace 99 -scope global
$global:runners = 8
$alert = Get-Service NetLogon
$alert.status
```

3. Autres méthodes

```
Set-Variable server -option None -force
Set-Variable server -option Constant -value '10.10.10.10'
Remove-Variable server -force
```

B. Opérateurs

1. Comparaison

-lt	Less than
-le	Less than or equal to
-gt	Greater than
-ge	Greater than or equal to
-eq	Equal to
-ne	Not equal to
-like	Like; uses wildcards for pattern matching
-match	Expression régulière

2. Logiques

-and	Et
-or	Ou
-xor	Ou exclusif

C. Structures de contrôle

1. Do

```
$a = 1
do {$a; $a++}
while ($a -lt 10)
$a = 1
do {$a; $a++} until ($a -eq 10)
```

2. While

```
$a = 1
while ($a -lt 10) {$a; $a++}
```

3. For

```
for ($a = 1; $a -le 10; $a++) {$a}
```

4. Break

```
$a = 1,2,3,4,5,6,7,8,9
foreach ($i in $a)
{
    if ($i -eq 3)
    {
        break
    }
    else
    {
        $i
    }
}
```

5. If

```
$a = "white"
if ($a -eq "red")
    {"The color is red."}
elseif ($a -eq "white")
    {"The color is white."}
else
    {"The color is blue."}
```

6. Foreach

```
foreach ($i in get-childitem c:\windows)
{$i.extension}
```

7. Switch

```
$a = 5
switch ($a)
    {
        1 {"The color is red."}
        2 {"The color is blue."}
        3 {"The color is green."}
        4 {"The color is yellow."}
        5 {"The color is orange."}
        6 {"The color is purple."}
        7 {"The color is pink."}
        8 {"The color is brown."}
        default {"The color could not be determined."}
    }
```

D. Cmdlets

1. Comptage

```
Get-Service | Group-Object status
Get-ChildItem c:\windows | Group-Object extension
Get-ChildItem c:\windows | Group-Object extension | Sort-Object count
```

2. Stats

```
Get-Process | Measure-Object CPU -ave -max -min -sum
```

3. Sélection

```
Get-Process|Select-Object ProcessName -first 5
```

4. Tri

```
Get-Process|Select-Object ProcessName, Id |Sort-Object Id
```

5. Différence

a) Process

```
Clear
$A = Get-Process
Stop-Service MySQL
$B = Get-Process
Start-Service MySQL
Compare $A $B
```

b) Fichiers

```
$A = Get-Content d:\scripts\x.txt
$B = Get-Content d:\scripts\y.txt
Compare-Object A$ B$
```

6. Affichage

```
Get-Service | Format-List
Get-Service|Format-Table
Get-Service|Format-Wide -autosize
Get-Eventlog PowerShell | Out-Host -paging
Get-Eventlog PowerShell | Out-Host -p
Get-Eventlog PowerShell | more
```

7. Filtre

a) Exemples

```
Get-Service|Where-Object {$_.Status -eq 'Running'}|Select-Object Name,
DisplayName|Format-Table -autosize
Get-ChildItem c:\windows|Where-Object {$_.Name -like '*.exe'}|Select-Object Name
```

8. Valeurs unique

```
Get-Content d:\scripts\test.txt | Sort-Object | Get-Unique
Get-Process|Sort-Object ProcessName|Get-Unique|Select-Object ProcessName
```

9. Propriétés

```
Get-ItemProperty  "hklm:\SYSTEM\CurrentControlSet\services\MySQL"
```

10. Impressions

```
Get-Process | Output-Printer
Get-Process | Output-Printer "HP LaserJet 6P"
```

11. Boucle

```
Get-Process |ForEach-Object {Write-Host $_.ProcessName -foregroundcolor cyan}
#$rows = get-wmiobject -class Win32_QuickFixEngineering
#foreach ($objItem in $rows)
#{
#      write-host "HotFix ID: " $objItem.HotFixID
#}
#get-wmiobject -class Win32_QuickFixEngineering|Select-Object HotFixID
get-wmiobject -class Win32_QuickFixEngineering|ForEach-Object {Write-Host $_.HotFixID}
```

12. Tri

```
Get-ChildItem c:\windows\*.* | Sort-Object length -descending | Select-Object -first 3
Get-EventLog system -newest 5 | Sort-Object eventid
```

13. Message

```
Write-Warning "The folder D:\scripts2 does not exist."
Write-Host "This is red text on a yellow background" -foregroundcolor red -
backgroundcolor yellow
```

a) Couleurs

Black
DarkBlue

DarkGreen
DarkCyan
DarkRed
DarkMagenta
DarkYellow
Gray
DarkGray
Blue
Green
Cyan
Red
Magenta
Yellow
White

14. Interaction

```
$Name = Read-Host "Please enter your name"
Write-Host $Name
```

VIII. Quelques exemples

A. ADSI

Les méthodes, propriétés utilisables sont indiquées dans mon support consacré à cette technologie sur mon site.

```
Clear
$oDom = [ADSI] "WinNT://$env:computername"
$oUser=$oDom.Create("User","denis")
$oUser.SetInfo
$oUser.SetPassword("Denis;544{556")
$oUser.SetInfo
$oUser.Dispose
$oDom.Dispose
```

B. Liste des fichiers exécutes sur la machine

Ce script a pour objet de lireles fichiers qui ont été exécutés au moins une fois sur la machine. Cette liste associée au mécanisme du *Prefetcher* se situe dans le dossier *c:\windows\prefetch* de votre disque dur.

```
$rows=Get-ChildItem c:\windows\prefetch |Where-Object {$_.Name -match '\.EXE'}|Select-
Object Name
Foreach($row in $rows)
{
    $i = $row.Name.IndexOf(".")
    $a = $row.Name.substring(0,$i+4)
    Write-Host $a
}
```

C. Liste des services à partir du registre

```
Clear
$keys=Get-ChildItem hklm:SYSTEM\CurrentControlSet\services|Select-Object Name
$t = "boot","system","auto","manual"
Foreach($key in $keys)
{
    $a=$key.Name.Replace("HKEY_LOCAL_MACHINE\","hklm:")
    $s=(Get-ItemProperty $a).Start
    If($s -lt 4 -and $s -ge 0)
    {
        $p=$a.LastIndexOf('\')+1
        $l=$a.Length
        Write-Host $t[$s] `t $a.SubString($p,$l-$p)
        #
    }
}
```

D. Utilisation des composants WSH Windows Scripting Host

L'intérêt du PowerShell est de vous permettre d'employer les objets associés à la technologie Windows Scripting Host. : Wscript.NetWork et Wscript.Shell. Vous les retrouverez dans mon support consacré à cette technologie sur mon site.

1. Wscript.Shell

```
$oShell = New-Object -com Wscript.Shell
$oShell.Run("c:\windows\system32\calc.exe")
```

2. Wscript.Network

```
$oNetwork = New-Object -com Wscript.Network
Write-Host $oNetWork.UserName
Write-Host $oNetwork.ComputerName
Try
{
    $oNetwork.RemoveNetworkDrive("X:")
}
Catch
```

```
{
    Write-Warning "Et prout... en hommage aux TSAR09 Rouen La Vatine"
}
Finally
{
    $oNetwork.MapNetworkDrive("X:", "\\vapedago\apps")
    Get-ChildItem x:\
}
$oNetWork.Dispose
```

3. Scripting.FileSystemObject

```
$oFso = New-Object -com Scripting.FileSystemObject
$oFile=$oFso.GetFile("c:\config.sys")
Write-Host $oFile.DateLastAccessed
```

E. MySQL : lecture de tables

```
[void][system.reflection.Assembly]::LoadFrom("C:\Program Files\MySQL\MySQL Connector Net
6.3.6\Assemblies\v2.0\MySql.Data.dll")
Cls
$strConn="DataSource=localhost;Database='veille';User ID='root';Password='"
Try
{
    $oConn = New-Object MySql.Data.MySqlClient.MySqlConnection
    $oConn.ConnectionString = $strConn
    $oConn.Open()
    #$oConn  = New-Object MySql.Data.MySqlClient.MySqlConnection($strConn)
}
Catch [System.Exception]
{
    $e = $_.Exception
    Write-Host $e.Message
}
Finally
{
}
$oSql = New-Object MySql.Data.MySqlClient.MySqlCommand
$oSql.Connection = $oConn
$oSql.CommandText = "SELECT * from moteur"
$oReader = $oSql.ExecuteReader()
while($oReader.Read())
{
#    Write-Host $oReader.GetString('moteur_url')
    for ($i= 0; $i -lt $oReader.FieldCount; $i++)
    {
        Write-Host $oReader.GetValue($i).ToString()
    }
}
$oReader.Close()
$oReader.Dispose()
$oAdapter = New-Object MySql.Data.MySqlClient.MySqlDataAdapter($oSql)
$oDataSet = New-Object System.Data.DataSet
$oAdapter.Fill($oDataSet,"data")
$data = $oDataSet.Tables["data"]
$data | Format-Table
$data.Dispose()
$oDataSet.Dispose()
$oAdapter.Dispose()
$oSql.Dispose()
$oConn.Close()
$oConn.Dispose()
#    $sql = New-Object MySql.Data.MySqlClient.MySqlCommand
#    $sql.Connection = $oConn
#    $sql.CommandText = "INSERT INTO computer_details (computer_id, mac, dhcp, model,
domain, manufacturer, type, memory, ip, servicetag, lastimagedate, servicepack, os,
```

```
biosrev, scriptversion, lastrun, ou) VALUES (‘$resultID’, ‘$macAddress’, ‘$dhcp’,
‘$model’, ‘$domain’, ‘$manufacturer’, ‘$systemType’, ‘$memory’, ‘$ipAddress’,
‘$servicetag’, NOW(), ‘$servicePack’, ‘$operatingSystem’, ‘$biosrev’, ‘$version’, NOW(),
‘$ou’ )”
#    $sql.ExecuteNonQuery()
#    $dbconnect.Close()
```

F. MySQL : inventaire

1. La table

```
CREATE TABLE `logiciel` (
  `logiciel_nom` varchar(255) DEFAULT NULL,
  `logiciel_machine` varchar(15) DEFAULT NULL,
  `logiciel_date` varchar(20) DEFAULT NULL,
  UNIQUE KEY `uk_logiciel` (`logiciel_nom`,`logiciel_machine`)
)
```

2. Le script

```
Clear
[void][system.reflection.Assembly]::LoadFrom(“C:\Program Files\MySQL\MySQL Connector Net
6.3.6\Assemblies\v2.0\MySql.Data.dll”)
$strConn="DataSource=localhost;Database='inventaire';User ID='root';Password=''"
$oConn = New-Object MySql.Data.MySqlClient.MySqlConnection
$oConn.ConnectionString = $strConn
Try
{
    $oConn.Open()
}
Catch [System.Exception]
{
    $e = $_.Exception
    Write-Host $e.Message
}
$req = New-Object MySql.Data.MySqlClient.MySqlCommand
$req.Connection=$oConn
$content=Get-ChildItem c:\windows\prefetch\*.pf
$oNetwork = New-Object -com Wscript.Network
$c=$oNetwork.ComputerName
ForEach($row in $content)
{
    $n=$row.Name
    $d=[datetime](Get-Item $row).LastAccessTime
    $p=$n.LastIndexOf('-')
    $s=$n.SubString(0,$p)
    $sql=“INSERT INTO logiciel VALUES('"+$s+"','"+$c+"','"+$d+"')”
    $req.CommandText = $sql
    Try
    {
        $req.ExecuteNonQuery()
    }
    Catch
    {
        $sql=“UPDATE logiciel SET logiciel_date='"+$d+"'
        WHERE logiciel_nom='"+$s+"' AND logiciel_machine='"+$c+"'"
        $req.CommandText = $sql
        $req.ExecuteNonQuery()
    }
}
$req.Dispose()
$oConn.Close()
$oConn.Dispose()
```

IX. Quelques sites

A. Références

http://fr.wikipedia.org/wiki/Windows_PowerShell
http://technet.microsoft.com/fr-fr/library/bb978526.aspx
http://technet.microsoft.com/fr-fr/scriptcenter/dd742419.aspx
http://technet.microsoft.com/fr-fr/scriptcenter/dd793612.aspx
http://technet.microsoft.com/fr-fr/scriptcenter/
http://technet.microsoft.com/en-us/scriptcenter/default.aspx
http://windows.developpez.com/cours/?page=powershell
http://www.powershell-scripting.com/
http://www.powershellpro.com/

B. Exemples de scripts

http://technet.microsoft.com/fr-fr/scriptcenter/dd772285.aspx
http://www.sapien.com/downloads#
http://syskb.com/powershell-pour-les-nuls/
http://gallery.technet.microsoft.com/ScriptCenter/
http://syskb.com/powershell-pour-les-nuls/

C. Documentations

http://www.manning.com/payette/
http://www.xaml.fr/powershell.html
http://www.robvanderwoude.com/powershell.php

D. Téléchargements

http://support.microsoft.com/?kbid=968930
http://support.microsoft.com/kb/968930
http://www.microsoft.com/downloads/en/details.aspx?FamilyID=d87daf50-e487-4b0b-995c-f36a2855016e&displaylang=en

E. Blogs

http://www.leeholmes.com/blog/?s=powershell
http://powershellmasters.blogspot.com/
http://scriptingof.blogspot.com/
http://www.via-powershell.fr/
http://blogs.msdn.com/b/powershell/

X. Annexe 1 : les commandes

Add-Computer
Add-Content
Add-History
Add-Member
Add-PSSnapin
Add-Type
Checkpoint-Computer
Clear-Content
Clear-EventLog
Clear-History
Clear-Item
Clear-ItemProperty
Clear-Variable
Compare-Object
Complete-Transaction
Connect-WSMan
ConvertFrom-Csv
ConvertFrom-SecureString
ConvertFrom-StringData
Convert-Path
ConvertTo-Csv
ConvertTo-Html
ConvertTo-SecureString
ConvertTo-Xml
Copy-Item
Copy-ItemProperty
Debug-Process
Disable-ComputerRestore
Disable-PSBreakpoint
Disable-PSSessionConfiguration
Disable-WSManCredSSP
Disconnect-WSMan
Enable-ComputerRestore
Enable-PSBreakpoint
Enable-PSRemoting
Enable-PSSessionConfiguration
Enable-WSManCredSSP
Enter-PSSession
Exit-PSSession
Export-Alias
Export-Clixml
Export-Console
Export-Counter
Export-Csv
Export-FormatData
Export-ModuleMember
Export-PSSession
ForEach-Object
Format-Custom
Format-List
Format-Table
Format-Wide
Get-Acl
Get-Alias
Get-AuthenticodeSignature
Get-ChildItem
Get-Command
Get-ComputerRestorePoint
Get-Content
Get-Counter
Get-Credential
Get-Culture
Get-Date
Get-Event
Get-EventLog
Get-EventSubscriber
Get-ExecutionPolicy
Get-FormatData
Get-Help
Get-History
Get-Host
Get-HotFix
Get-Item
Get-ItemProperty
Get-Job
Get-Location
Get-Member
Get-Module
Get-PfxCertificate
Get-Process
Get-PSBreakpoint
Get-PSCallStack
Get-PSDrive
Get-PSProvider
Get-PSSession
Get-PSSessionConfiguration
Get-PSSnapin
Get-Random
Get-Service
Get-TraceSource
Get-Transaction
Get-UICulture
Get-Unique
Get-Variable
Get-WinEvent
Get-WmiObject
Get-WSManCredSSP
Get-WSManInstance
Group-Object
Import-Alias
Import-Clixml
Import-Counter
Import-Csv
Import-LocalizedData
Import-Module
Import-PSSession
Invoke-Command
Invoke-Expression
Invoke-History
Invoke-Item
Invoke-WmiMethod
Invoke-WSManAction
Join-Path
Limit-EventLog
Measure-Command
Measure-Object
Move-Item
Move-ItemProperty
New-Alias
New-Event
New-EventLog
New-Item
New-ItemProperty
New-Module
New-ModuleManifest
New-Object
New-PSDrive
New-PSSession
New-PSSessionOption
New-Service
New-TimeSpan
New-Variable
New-WebServiceProxy
New-WSManInstance
New-WSManSessionOption
Out-Default
Out-File
Out-GridView
Out-Host
Out-Null
Out-Printer
Out-String
Pop-Location
Push-Location
Read-Host
Receive-Job
Register-EngineEvent
Register-ObjectEvent
Register-PSSessionConfiguration
Register-WmiEvent
Remove-Computer
Remove-Event
Remove-EventLog
Remove-Item
Remove-ItemProperty
Remove-Job
Remove-Module
Remove-PSBreakpoint
Remove-PSDrive

Remove-PSSession
Remove-PSSnapin
Remove-Variable
Remove-WmiObject
Remove-WSManInstance
Rename-Item
Rename-ItemProperty
Reset-ComputerMachinePassword
Resolve-Path
Restart-Computer
Restart-Service
Restore-Computer
Resume-Service
Select-Object
Select-String
Select-Xml
Send-MailMessage
Set-Acl
Set-Alias
Set-AuthenticodeSignature
Set-Content
Set-Date
Set-ExecutionPolicy
Set-Item
Set-ItemProperty
Set-Location
Set-PSBreakpoint
Set-PSDebug
Set-PSSessionConfiguration
Set-Service
Set-StrictMode
Set-TraceSource
Set-Variable
Set-WmiInstance
Set-WSManInstance
Set-WSManQuickConfig
Show-EventLog
Sort-Object
Split-Path
Start-Job
Start-Process
Start-Service
Start-Sleep
Start-Transaction
Start-Transcript
Stop-Computer
Stop-Job
Stop-Process
Stop-Service
Stop-Transcript
Suspend-Service
Tee-Object
Test-ComputerSecureChannel
Test-Connection
Test-ModuleManifest
Test-Path
Test-WSMan
Trace-Command
Undo-Transaction
Unregister-Event
Unregister-PSSessionConfiguration
Update-FormatData
Update-List
Update-TypeData
Use-Transaction
Wait-Event
Wait-Job
Wait-Process
Where-Object
Write-Debug
Write-Error
Write-EventLog
Write-Host
Write-Output
Write-Progress
Write-Verbose
Write-Warning

XI. Annexe 2 : les alias

Possibilité fournie par le langage, une utilisation excessive des alias se heurte à la mémorisation des principales commandes du langage.

Alias	CmdLet
%	ForEach-Object
?	Where-Object
ac	Add-Content
asnp	Add-PSSnapIn
cat	Get-Content
cd	Set-Location
chdir	Set-Location
clc	Clear-Content
clear	Clear-Host
clhy	Clear-History
cli	Clear-Item
clp	Clear-ItemProperty
cls	Clear-Host
clv	Clear-Variable
compare	Compare-Object
copy	Copy-Item
cp	Copy-Item
cpi	Copy-Item
cpp	Copy-ItemProperty
cvpa	Convert-Path
dbp	Disable-PSBreakpoint
del	Remove-Item
diff	Compare-Object
dir	Get-ChildItem
ebp	Enable-PSBreakpoint
echo	Write-Output
epal	Export-Alias
epcsv	Export-Csv
epsn	Export-PSSession
erase	Remove-Item
etsn	Enter-PSSession
exsn	Exit-PSSession
fc	Format-Custom
fl	Format-List
foreach	ForEach-Object
ft	Format-Table
fw	Format-Wide
gal	Get-Alias
gbp	Get-PSBreakpoint
gc	Get-Content
gci	Get-ChildItem
gcm	Get-Command
gcs	Get-PSCallStack
gdr	Get-PSDrive
ghy	Get-History
gi	Get-Item
gjb	Get-Job
gl	Get-Location
gm	Get-Member
gmo	Get-Module
gp	Get-ItemProperty
gps	Get-Process
group	Group-Object
gsn	Get-PSSession
gsnp	Get-PSSnapIn
gsv	Get-Service
gu	Get-Unique
gv	Get-Variable
gwmi	Get-WmiObject
h	Get-History
history	Get-History
icm	Invoke-Command
iex	Invoke-Expression
ihy	Invoke-History
ii	Invoke-Item
ipal	Import-Alias
ipcsv	Import-Csv
ipmo	Import-Module
ipsn	Import-PSSession
ise	powershell_ise.exe
iwmi	Invoke-WMIMethod
kill	Stop-Process
lp	Out-Printer
ls	Get-ChildItem
man	help
md	mkdir
measure	Measure-Object
mi	Move-Item
mount	New-PSDrive
move	Move-Item
mp	Move-ItemProperty
mv	Move-Item
nal	New-Alias
ndr	New-PSDrive
ni	New-Item
nmo	New-Module
nsn	New-PSSession
nv	New-Variable
ogv	Out-GridView
oh	Out-Host
popd	Pop-Location
ps	Get-Process
pushd	Push-Location
pwd	Get-Location
r	Invoke-History
rbp	Remove-PSBreakpoint
rcjb	Receive-Job
rd	Remove-Item
rdr	Remove-PSDrive
ren	Rename-Item
ri	Remove-Item
rjb	Remove-Job
rm	Remove-Item
rmdir	Remove-Item
rmo	Remove-Module
rni	Rename-Item
rnp	Rename-ItemProperty
rp	Remove-ItemProperty
rsn	Remove-PSSession
rsnp	Remove-PSSnapin
rv	Remove-Variable
rvpa	Resolve-Path
rwmi	Remove-WMIObject
sajb	Start-Job
sal	Set-Alias
saps	Start-Process
sasv	Start-Service
sbp	Set-PSBreakpoint
sc	Set-Content
select	Select-Object
set	Set-Variable
si	Set-Item
sl	Set-Location
sleep	Start-Sleep
sort	Sort-Object
sp	Set-ItemProperty
spjb	Stop-Job
spps	Stop-Process
spsv	Stop-Service
start	Start-Process
sv	Set-Variable
swmi	Set-WMIInstance
tee	Tee-Object
type	Get-Content
where	Where-Object
wjb	Wait-Job
write	Write-Output

XII. Annexe 3 : de Vbs à Powershell, documentation adaptée d'un document Microsoft

VBScript Function	Windows PowerShell Equivalent
Abs	$a = [math]::abs(-15)
Array	$a = "red","orange","yellow","green","blue","indigo","violet"
Asc	$a = [byte][char] "A"
Atn	$a = [math]::atan(90)
CBool	$a = 0 $a = [bool] $a
CByte	$a = "11.45" $a = [byte] $a
CCur	$a = "{0:C}" -f 13
CDate	$a = "11/1/2006" $a = [datetime] $a
CDbl	$a = "11.45" $a = [double] $a
Chr	$a = [char]34
CInt	$a = "11.57" $a = [int] $a
CLng	$a = "123456789.45" $a = [long] $a
Cos	$a = [math]::cos(45)
CreateObject	$a.visible = $True $a = new-object -comobject Excel.Application -strict
CSng	$a = "11.45" $a = [single] $a
CStr	$a = 17 $a = [string] $a
Date	$a = get-date –format d
DateAdd	$a = (get-date).AddDays(37) (get-date).AddHours(37) (get-date).AddMilliseconds(37) (get-date).AddMinutes(37) (get-date).AddMonths(37) (get-date).AddSeconds(37) (get-date).AddTicks(37) (get-date).AddYears(37) $a = ((get-date).AddHours(2)).AddMinutes(34)
DateDiff	$a = New-TimeSpan $(Get-Date) $(Get-Date –month 12 -day 31 -year 2006 -hour 23 -minute 30) $a.Days Days : 109 Hours : 3 Minutes : 55 Seconds : 0 Milliseconds : 0 Ticks : 94317000000000 TotalDays : 109.163194444444 TotalHours : 2619.91666666667 TotalMinutes : 157195 TotalSeconds : 9431700 TotalMilliseconds : 9431700000
DatePart	$a = (get-date).day

	$a = (get-date).dayofweek $a = (get-date).dayofyear $a = (get-date).hour $a = (get-date).millisecond $a = (get-date).minute $a = (get-date).month $a = (get-date).second $a = (get-date).timeofday $a = (get-date).year $a = (get-date).hour
DateSerial	MyDate1 = DateSerial(2006, 12, 31) $a = get-date -y 2006 -mo 12 -day 31
DateValue	$a = [datetime] "12/1/2006"
Day	$a = (get-date).day
Eval	$a = 2 + 2 -eq 45
Exp	$a = [math]::exp(2)
Filter	$a = "Monday","Month","Merry","Mansion","Modest" $b = ($a | where-object {$_ -like "Mon*"})
FormatCurrency	$a = 1000 $a = "{0:C}" -f $a
FormatDateTime	$a = (get-date).tolongdatestring() $a = (get-date).toshortdatestring() $a = (get-date).tolongtimestring() $a = (get-date).toshorttimestring()
FormatNumber	$a = 11 $a = "{0:N6}" -f $a
FormatPercent	$a = .113 $a = "{0:P1}" -f $a
GetLocale	$a = (get-culture).lcid $a = (get-culture).displayname
Hex	$a = 4517 $a = "{0:X}" -f $a
Hour	$a = (get-date).hour
InputBox	$a = new-object -comobject MSScriptControl.ScriptControl $a.language = "vbscript" $a.addcode("function getInput() getInput = inputbox(`"Message box prompt`",`"Message Box Title`") end function") $b = $a.eval("getInput")
InStr	$a = "wombat" $b = $a.contains("m") $b = $a.indexof("m")
InStrRev	$a = "1234x6789x1234" $b = $a.lastindexofany("x")
Int/Fix	$a = 11.98 $a = [math]::truncate($a)
IsArray	$a = 22,5,10,8,12,9,80 $b = $a -is [array]
IsDate	$a = 11/2/2006 $a -is [datetime] $a = [datetime] "11/2/2006"
IsEmpty	$a = "" $b = $a.length -eq 0
IsNull	$a = $z -eq $null
IsNumeric	$a = 44.5 [reflection.assembly]::LoadWithPartialName("'Microsoft.VisualBasic")

	$b = [Microsoft.VisualBasic.Information]::isnumeric($a)
IsObject	$a = new-object -comobject scripting.filesystemobject $b = $a -is [object]
Join	$a = "h","e","l","l","o" $b = [string]::join("", $a)
LBound	$a = 1,2,3,4,5,6,7,8,9 $b = $a.getlowerbound(0)
LCase	$a = "ABCDEFGHIJKLMNOPQRSTUVWXYZ" $a = $a.ToLower()
Left	$a="ABCDEFGHIJKLMNOPQRSTUVWXYZ" $a = $a.substring(0,3)
Len	$a = "abcdefghijklmnopqrstuvwxyz" $b = $a.length
Log	$a = [math]::log(100)
LTrim	$a = "..........123456789.........." $a = $a.TrimStart()
RTrim	$a = "..........123456789.........." $a = $a.TrimEnd()
Trim	$a = "..........123456789.........." $a = $a.Trim()
Mid	$a="ABCDEFG" $a = $a.substring(2,3)
Minute	$a =(get-date).minute
Month	$a = get-date -f "MM" $a = [int] (get-date -f "MM")
MonthName	$a = get-date -f "MMMM"
MsgBox	$a = new-object -comobject wscript.shell $b = $a.popup("This is a test",0,"Test Message Box",1)
Now	$a = get-date
Oct	$a = [Convert]::ToString(999,8)
Replace	$a = "bxnxnx" $a = $a -replace("x","a")
RGB	$blue = 10 $green= 10 $red = 10 $a = [long] ($blue + ($green * 256) + ($red * 65536))
Right	$a = "ABCDEFGHIJKLMNOPQRSTUVWXYZ" $a = $a.substring($a.length - 9, 9)
Rnd	$a = new-object random $b = $a.next(1,100) $b = $a.next()
Round	$a = [math]::round(45.987654321, 2)
ScriptEngine	$a = (get-host).version
ScriptEngineBuildVersion	$a = (get-host).version.build
ScriptEngineMajorVersion	$a = (get-host).version.major
ScriptEngineMinorVersion	$a = (get-host).version.minor
Second	$a = (get-date).second
Sgn	$a = [math]::sign(-453)
Sin	$a = [math]::sin(45)
Space	$a = " " * 25 $a = $a + "x"
Split	$a = "atl-ws-01,atl-ws-02,atl-ws-03,atl-ws-04" $b = $a.split(",")

Sqr	$a = [math]::sqrt(144)
StrComp	$a = "dog" $b = "DOG" $c = [String]::Compare($a,$b,$True)
String	$a = "=" * 20
StrReverse	$a = "Scripting Guys" for ($i = $a.length - 1; $i -ge 0; $i--) {$b = $b + ($a.substring($i,1))}
Tan	$a = [math]::tan(45)
Time	$a = get-date -displayhint time
TimeSerial	$a = get-date -h 17 -mi 10 -s 45 -displayhint time
TimeValue	$a = [datetime] "1:45 AM"
TypeName	$a = 55.86768 $b = $a.gettype().name
UBound	$a = "a","b","c","d","e" $a.getupperbound(0) $a.length-1
UCase	$a = "abcdefghijklmnopqrstuvwxyz" $a = $a.ToUpper()
WeekdayName	$a = (get-date).dayofweek $a = (get-date "12/25/2007").dayofweek
Year	$a = (get-date).year $a = (get-date "9/15/2005").year
Const Statement	set-variable -name ForReading -value 1 -option constant
Dim Statement	$a = [string]
Execute Statement	$a = "get-date" invoke-expression $a
Function Statement	function multiplynumbers { $args[0] * $args[1] } multiplynumbers 38 99
On Error Statement	$erroractionpreference = "SilentlyContinue" Incidentally, your choices for this variable include: SilentlyContinue Continue (the default value) Inquire Stop
Option Explicit Statement	set-psdebug –strict set-psdebug -off
Private Statement	$Private:a = 5
Public Statement	$Global:a = 199
Randomize Statement	$a = new-object random $b = $a.next()
ReDim Statement	$a = 1,2,3,4,5 $a = $a + 100 $a = $a[0..2]
Set Statement	$a = new-object -comobject Excel.Application $a.visible = $True
Stop Statement	set-psdebug –step set-psdebug -off
Sub Statement	function multiplynumbers { $args[0] * $args[1] } multiplynumbers 38 99
Description Property	$a = $error[0].ToString()
HelpContext Property	$a = $error[0].helplink
HelpFile Property	$a = $error[0].helplink
Number Property	ScriptHalted $error[0].errorrecord
Source Property	$a = $error[0].source

Clear Method	$error[0] = "" $error.clear()
Raise Method	$b = "The file could not be found."; throw $b