POWERSHELL 3.0

I.Présentation

A.Préambule

Ce document est un support de cours dont l'objet d'essayer de fournir les clés de compréhension du PowerShell.
Il est essentiellement constitué d'exemple de commandes, il vous suffit de les copier-coller dans votre console Powershell pour les tester.

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 langages tels 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.PowerShell 3

Windows PowerShell 3.0 nécessite Microsoft .NET Framework 4.0.
La nouvelle version de PowerShell est disponible sur Windows 7 Service Pack 1, Windows Server 2008 R2 SP1 ou encore Windows Server 2008 Service Pack 2 par simple mise à jour.
Elle est native sur Windows 8 et sur Windows Server 2012.

Pour déterminer la version de votre Powershell :

```
Get-Host | Select-Object Version
```

D.Les outils

- Windows PowerShell ISE, intégré à Windows 7
- PowerShell Scriptomatic (wmi)
- Visual Studio ou
- Power GUI ou
- Power Plus

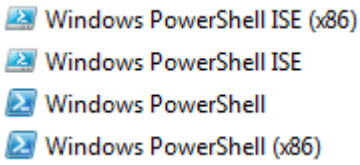
II.Premiers Pas

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, Microsoft Office et bien d'autres outils Microsoft.

B.L'interpréteur

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



C.Protection

1.Le niveau de sécurité : Get-ExecutionPolicy

```
Get-ExecutionPolicy -List
```

```
PS C:\Users\tondeur-h.CHV> Get-ExecutionPolicy -List

                                Scope                          ExecutionPolicy
                                -----                          ---------------
                        MachinePolicy                               Undefined
                           UserPolicy                               Undefined
                              Process                               Undefined
                          CurrentUser                               Undefined
                         LocalMachine                                  Bypass
```

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

```
Set-ExecutionPolicy -Scope LocalMachine -ExecutionPolicy unrestricted
Set-ExecutionPolicy -Scope CurrentUser -ExecutionPolicy remotesigned
```

3.Signature

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

Permet d'obtenir la signaure d'une applicaton quand celle ci est définie.

4.Voir aussi l'Aidesur ce sujet

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

5.Autorité de certification

La commande makecert.exe est installée avec Office ou Visual Studio.
makecert.exe -n "CN=ISTV" -a sha1 -eku 1.0 -r -sv private.pvk certificat.cer -ss Root -sr localMachine

L'outil Certificate Creation (Création de certificats) génère des certificats X.509 uniquement à des fins de tests. Il crée une paire de clés publique/privée pour les signatures numériques et

l'enregistre dans un fichier de certificat. Il associe également cette paire de clés à un nom d'éditeur spécifié et crée un certificat X.509 liant un nom spécifié par l'utilisateur à la partie publique de la paire de clés.

6.Associer un certificate à un script

```
$cert=@(Get-ChildItem cert:\Currentuser\My)[0]
Set-AuthenticodeSignature d:\test.ps1 $cert
```

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
Get-Help Set-Service –online
Get-Help *Service*
Get-Help *s* -Category Alias
Get-Command -Verb Get
Get-Command -Module NetTcpIp
```

```
Get-Help * -Parameter ComputerName
```

4.Actualiser l'aide 3.0

Update-Help (Vous devez être administrateur)

5.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
(Get-Process).ProcessName
(Get-Host).CurrentCulture | format-list -property *
(Get-Host).CurrentCulture.TextInfo.ANSICodePage
Get-Process | Sort-Object -Property CPU
Get-Process | Sort-Object -Property CPU -Descending
Get-Process | Sort CPU
```

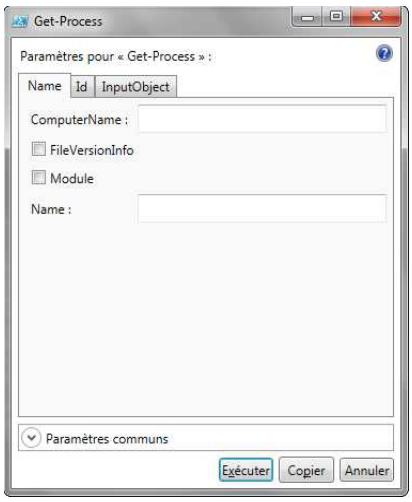
Ces commandes retournent les propriétés ou les méthodes associées à une cmdlet.

6.Afficher les propriétés des processus

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

7.Mode GUI

```
Show-Command
Show-Command -Name Get-Process
```



8.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
```

9.Les fournisseurs PowerShell : Get-PSProvider

Get-PSProvider
Get-ChildItem Env:
Set-Location Env:
New-Item –Name Test –Value 'Mon test à moi'
Get-Content Env:Test
Remove-Item Env:Test

L'ensemble de ces cmdlets, donnent des informations sur les chemins par défaut de powerShell ou des variables d'environnements.

10.Historique (historiser les cmdlets)

Start-Transcript
Stop-Transcript

E.Exécution des scripts

1.Exécution d'un script

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

Attention : pour pouvoir éxécuter un script powerShell, il faut posseder les droits.

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.Récupération du contenu de l'exécution d'une commande système

```
clear
$res=&hostname
$res.Trim
#$res=&{. 'c:\windows\system32\ipconfig.exe'}
$res=&'c:\windows\system32\ipconfig.exe'
clear
If($res|Where {$_ -match 'IPv4[ \.]+:[ ]+(\d+\.\d+\.\d+\.\d+)'})
{
$Matches[1]
}
```

4.Mac Address

```
clear
$cmd=&c:\windows\system32\ipconfig.exe /all
#$cmd[10]
$cmd|Foreach{
if($_ -match '([0-9a-f\-]{17})')
{
$matches[1]
break
}
}
```

5.Variable d'environnement

```
Foreach($item in (Get-ChildItem env:\))
{
"$($item.Key) : $($item.value)"
}
$env:COMPUTERNAME
```

6.Appel d'un programme

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

7.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}
```

8.Temporisation

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

9.Trigger

```
$DailyTrigger = New-JobTrigger -At 17:25 -Daily
Register-ScheduledJob -Name RestartFaultyService -ScriptBlock {Restart-
Service FaultyService }-Trigger $DailyTrigger
```

```
Get-ScheduledJob
Get-ScheduledJob -Name RestartFaultyService
Disable-ScheduledJob -Name RestartFaultyService
Enable-ScheduledJob -Name RestartFaultyService
Unregister-ScheduledJob -Name RestartFaultyService
```

10.Envoi de mail

a)Méthode Send-MailMessage

```
$motdepasse = ConvertTo-SecureString "monmdp" -AsPlainText -Force
$authentification = New-Object System.Management.Automation.PSCredential
("herve@istv.fr", $motdepasse)
Send-MailMessage -To 'herve@univ-valenciennes.fr' -Subject 'test PS'
-From 'tondeur-h@istv.fr' -Body 'test PS' -SmtpServer 'smtp.istv.fr'
-Credential $authentification
```

Méthode .Net

```
$CredUser = "tondeur-h"
$CredPassword = "mypwd$"
$EmailFrom = "tondeur-h@istv.fr"
$EmailTo = "benameur-n@istv.fr"
$Subject = "Test PS3"
$Body = "Test PS3"
$SMTPServer = "smtp.istv.fr"
$SMTPClient = New-Object Net.Mail.SmtpClient($SmtpServer, 587)
$SMTPClient.EnableSsl = $true
$SMTPClient.Credentials = New-Object
System.Net.NetworkCredential($CredUser, $CredPassword);
$SMTPClient.Send($EmailFrom, $EmailTo, $Subject, $Body)
```

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
Get-UICulture
```

H.Passage d'arguments

1.Par tableau

```
./monscript.ps1 "c:\windows" 1
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
Get-EventLog -LogName application | where entrytype -eq 'error'
```

Exploiter les journaux du systéme d'exploitation.

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
Get-Service | Sort-Object status | Group-Object -Property status
```

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
Get-Process | Where-Object WorkingSet -gt 100MB | Select-Object Name
Get-Process | sort name | group name -NoElement | sort count -Descending
Get-Process | Where { $_.starttime.minute -lt 30} | select name,
starttime
```

2.Arrêter un process

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

3.Verbosité/Erreur

```
Stop-Process -processname notepad -Verbose
Get-Process -Name notepad -ErrorAction SilentlyContinue
```

D.Informations

```
Get-Host
Get-Hotfix
Get-HotFix|Where InstalledOn -lt 12/09/2014
```

E.CIM

Get-CIMClass –Class *network*

```
(Get-CimClass -Class Win32_NetworkAdapterConfiguration).CimClassMethods
(Get-CimClass
-ClassWin32_NetworkAdapterConfiguration).CimClassProperties
Get-CimClass -PropertyName speed
Get-CimClass -MethodName reboot
Get-CimClass -Class Win32_BIOS
Get-CimInstance -ClassName Win32_BIOS
(Get-CimInstance -ClassName Win32_BIOS).SerialNumber
```

F.WMI

```
Get-WmiObject -List
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.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 types

'Texte' –is [string]
$a = 55.86768
$b = $a.GetType().name

3.Les chaînes

Les chaînes de caractère peuvent être encadrées de guillemets ou d'apostrophes.
Les guillemets peuvent interpréter des variables

```
$a='test'
$b="$a"
Write-Output $b
```

#Here-String
$texte=@'
hgfhgh
gjgjjgj
'@

4.Caractères spéciaux

`0	Null
`a	Beep
`b	Backspace
`n	Saut de ligne
`r	Retour chariot
`t	Horizontal tab
`'	Single quote
`"	Double quote
`f	Saut de page
`v	Tabulation verticale

5.Substitution de variables

```
$fichier=Get-ChildItem c:\windows\system32\drivers\etc\services
$l=$fichier.Length
$n=$fichier.FullName

clear
```

```
"Taille du fichier $n : $l octets"
"Taille du fichier {1} : {0} octets" -f $l,$n
```

6.Les variables prédéfinies

$$	Dernière commande
$?	True si la commande a réussie / False si échouée
$Args	Tableau des paramètres passés à partir de la ligne de commande
$ConsoleFileName	Chemin du dernier fichier utilisé dans la session
$Error	Liste des erreurs de la session
$Event	Evénement traité par Register-ObjectEvent
$EventArgs	Arguments relatifs à Event
$Foreach	Enumerateur d'une boucle ForEach
$Home	Répertoire de base de l'utilisateur
$Host	Informations sur l'hôte
$LastExitCode	Code de sortie de la dernière commande du système execute
$PID	Process du script PowerShell
$Profile	Chemin du profil PowerShell
$PSHome	Répertoire d'installation du PowerShell
$PSItem	ou $_Objet courant
$PSScriptRoot	Répertoire du script
$PSVersionTable	Information sur PowerShell
$PWD	Répertoire courant
$ShellID	Identificateur du Shell
$MyInvocation	`$MyInvocation.MyCommand.Name`

Les constantes

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

B.Les tableaux

1.Principes de base

L'indice d'un tableau commence à 0.

$tab=1,2,3,4
$tab=0..99
$Jours="Lu","Ma","Me","Je","Ve","Sa","Di"
[int[]]$tab=1,2,3,4
$tab=[string]'Texte',[int]8,[double]3.47,[char]'z'

$tab[0]	Lit le 1er élément du tableau
$tab[$tab.length-1]	Dernier élément du tableau
$tab.length	Nombre d'éléments du tableau
$tab[0..2]	Affiche les éléments de l'indice 0 à 2
$tab[-1]	Dernier élément
$tab1+$tab2	Concaténation de tableau
$tab+=4	Ajout d'un élément au tableau

Pas de suppression de tableau

$tab=1,2,3,4
$tab=$tab[0..1+3]
$tab=$tab|Where-Object {$_ -ne 3}

Exemple

```
clear
[string[]]$Jours='Lu','Ma','Me','Je','Ve','Sa','Di'
$Jours[0]
$Jours[-1]
$jours.Length
$jours+='Dredi'
$Jours[-1]
#$Jours=$Jours|Sort
#$Jours=$Jours[0..4+7]
$Jours=$Jours|Where {$_ -match 'e'}
clear
$Jours
```

2.Effacer un élément avec méthode .Net

```
Clear
$a = New-Object System.Collections.ArrayList
$a.Add("red")
$a.Add("yellow")
$a.Add("orange")
$a.Add("green")
$a.Add("blue")
$a.Add("purple")
$a.Remove("yellow")
$a
$a=$null
```

3.Tableaux associatifs

```
$recettes=[ordered]@{Lu=100;Ma=800;Me=350;Je=560;Ve=340}
$recettes|Format-List
$recettes['Ve']
$recettes+=@{Sa=1230}
$recettes.keys
$recettes.values
$recettes.keys|Foreach {$recettes.$_}
```

4.Autres méthodes

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

5.Portée

$global:variable	Par défaut
$local:variable	Locale à la function, au script, au bloc d'instructions
$script:variable	Script
$using:variable	Exécution à distance

6.Nombre aléatoire

```
(New-Object system.random).next()
Get-Random
Get-Random -Maximum 21 -Minimum 1
Get-Random -InputObject (1..10) -Count 5
```

C.Opérateurs

1.Concaténation

signe +

2.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; utilises les caractéres jokers pour matcher des chaines.
-match Expression régulière

1 –lt 2

3.Expressions régulières

'PowerShell' –match 'l$'
'PowerShell' –notmatch 'l$'
$Matches, $Matches[i]
'Date: 02/09/2014' –match '^Date:\s(?<date>(?<jour>\d{2})/>(?<mois>\d{2})/>(?<annee>\d{4}))$'
$Matches.annee

```
clear
$Str="Henri est au boulot avec Hervé"
$Regex="(Henri)( est au boulot avec )(Hervé)"
$new=$Str –replace $Regex, '$3$2$1'
$new
$Str=$null;$Regex=$null
```

4.Logiques

-and Et
-or Ou
-xor Ou exclusif

5.Plages

1..99

6.Appartenance

'T' -in 'DSTC','Tondeur'
'T' -notin 'DSTC','Tondeur'
Contains, c'est l'inverse : 'DSTC','Tondeur' contains 'T'

7.Opérateurs binaires

-band
-bor
-bnot
-bxor

8.Affectation

$i=0
$i++
$i=$i+8 ou $i+=8

9.Cast

```
clear
```

```
$b=Read-Host 'Saisissez votre élément'
if($b -match '^\d+$')
{
$b=[int]$b
$b*100
}
else
{
'Ceci n''est pas une valeur'
}
$b.GetType().Name
```

10.Forcer la définition de variables

Set-PSDebug -Strict

D.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 ($item in Get-Process)
```

```
{
"$($item.CPU*1000)"
}
Get-Process|Foreach{
"$($_.CPU*1000)"
}
Get-Process|Foreach{$_.CPU*1000}
Get-Process|Foreach CPU

foreach ($i in get-childitem c:\windows)
{$i.extension}
"un vélo.","un ballon","une chouette." | ForEach-Object Insert
-ArgumentList 0,"C'est "
```

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."}
}
Switch -regex (chaine)
{
'^test'{'Ca commence par test';break}
'test$' {'Ca finit par test';break}
}
```

8.Exemple conditionnelle

```
Clear
$chaine=Read-Host 'Texte'
Switch -regex ($chaine)
{
'^test'{'Ca commence par test';break}
'test$' {'Ca finit par test';break}
Default {'Ni l''un, ni l''autre'}
}
If($chaine -Match '^test')
{
'Ca commence par test'
}
ElseIf($chaine -Match 'test$')
{
'Ca finit par test'
}
Else
{
'Ni l''un, ni l''autre'
}
```

E.Gestion d'erreurs

1.Préférence

$ErrorActionPreference='SilentlyContinue'
Valeurs possibles : SilentlyContinue, Continue, Stop, Inquire, Ignore (3.0 : non stockée dans $Error)

2.Cas par cas

```
Get-ChildItem c:\test.bat -ErrorAction SilentlyContinue -ErrorVariable
err
$err
```

Trap

```
clear
$ErrorActionPreference='SilentlyContinue'
trap { 'Erreur';exit}
100/0
Get-Process
```

Try…Catch

```
clear
Try
{
100/0
}
Catch
{
"Errare humanum est, sed...`n$($Error[0])"
}
Finally
{
'J''ai fait mon boulot'
}
```

Débogage

$VerbosePreference
Write-Verbose
Write-Debug
Set-PSDebug –Step
Set-PsBreakPoint –Command Get-Process : point de débogage à chaque exécution de la commande Get-Process
Commandes Débogeur : S (Suivant et retour),V,O,L,G (Stop),K (Pile)

F.Pipelining

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

a)Liste

```
Get-Service | Format-List -Property
Get-Service | Format-List *
```

b)Tableau

```
Get-Service|Format-Table
Get-Service | Where Status -eq 'Running'| Format-Table -Property
Name,DisplayName
Get-Service | Where Status -eq 'Running'| Format-Table -Property
Name,DisplayName -GroupBy Name
Get-Service | Where Status -eq 'Running'| Format-Table -Property
Name,DisplayName -AutoSize
```

c)Colonne

```
Get-Service|Format-Wide -Property Name -autosize
Get-Service|Format-Wide -Property Name -column 4 -autosize
```

d)Write-Output

C'est la commande implicite

```
Get-Eventlog PowerShell | Out-Host -paging
Get-Eventlog PowerShell | Out-Host -p
Get-Eventlog PowerShell | more
```

e)Write-Host

Il renvoie vers la console et ne peut pas renvoyer vers un fichier

f)Exemples

```
Get-Service|Where Status -eq 'Running'|Select Name,DisplayName|Format-
Table -AutoSize -HideTableHeaders
Get-Process|Where-Object { $_.Name -match '^S'}|Select Name,Handle|
Format-List -GroupBy Name
Get-Process|Out-GridView -Title 'Mon bô tableau, roi des …'
```

7.Filtre

a)Avec Where-Object

```
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
```

b)Avec filter

```
Filter Get-BigProcess
{
Begin
{
$conso=0
}
Process
{
If($_.CPU -gt 1)
{
$_
}
$conso+=$_.VM
}
End
{
"`nConso cumulée des process de plus de 100MB : $($conso/(1024*1024))
Mo"
}
}
Get-Process|Get-BigProcess
```

8.Valeurs unique

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

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 | Where Handle -gt 0
Get-Process | Where-Object Handle -gt 0
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
```

G.Fonctions

1.Sans retour

```
Function Set-Popup
{
param([string]$title,[string]$message)
$owsh=New-Object -ComObject Wscript.shell
$owsh.Popup($message,0,$title)
}
Set-Popup -title 'Ma boîte à moi' -message 'Mon texte à moi'
```

2.Avec retour

```
Function Conso-Memoire
{
Param([string]$process)
Get-Process|Foreach{
if($process -eq $_.ProcessName)
{
[math]::round($_.VM/1048576)
break
}
}
0
}
```

```
Conso-Memoire -process 'firefox'
. 'C:\powershell\biblio.ps1'
Get-DriveFreeSpace -Letter 'c:'
```

H.Gestion des modules

1.Emplacement des modules

Ils sont déterminés par la variable d'environnement `$env:PSModulePath`.
%windir%\System32\WindowsPowerShell\v1.0\Modules
%UserProfile%\Documents\WindowsPowerShell\Modules

2.Télécharger des modules complémentaires

http://gallery.technet.microsoft.com/scriptcenter/site/search?f[0].Type=ProgrammingLanguage&f[0].Value=PowerShell&f[0].Text=PowerShell&sortBy=Downloads

3.Les modules liés à l'administration

```
Get-Module -ListAvailableListe tous les modules
```

4.Commandes d'un module

```
Get-command –module DnsServer
```

5.Charger automatiquement les modules

$PSModuleAutoloadingPreference='All' (None,ModuleQualified)

6.Décharger un module

Remove-Module Dnsserver

7.Créer un module

Créez un répertoire et un fichier psm1 du même nom dans l'un des répertoires défini par `$env:PSModulePath`

8.Exemple : devices.psm1

a)Définition des fonctions du module

```
<#
.Synopsis
Indique le taux d'espace libre.
.Description
La fonction Get-DriveFreeSpace indique le taux d'espace libre
calculé à partir de l'appel à WMI.
.Parameter Letter
Entrez la lettre de lecteur telle que C:.
.Example
Get-DriveFreeSpace 'C:'
.Example
Get-DriveFreeSpace -Letter 'C:'
.Link
Get-DriveFreeSpace
#>
Function Get-DriveFreeSpace
{
Param([string]$Letter)
#$res=$null
$Drives=Get-WmiObject Win32_LogicalDisk| Where Size -ne $Null
Foreach($Drive in $Drives)
{
If($Drive.DeviceID -eq $Letter)
```

```
{
$res=[Math]::Round($Drive.FreeSpace/$Drive.Size*100,2)
Return $res
#Break
}
}
#$res
}
```

b)Utilisation du module

```
Import-Module Devices
Devices\Get-DriveFreeSpace -Letter 'D:'
$env:PSModulePath
```

V.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
$Date=Get-Date –Year 2013 –Month 9 –Day 1
$A = Get-Date 30/3/2015
$A = Get-Date "30/3/2015 7:00 AM"
(Get-Date).AddMinutes(137)
$date = Get-Date -Format 'dd-MM-yyyy'
Get-Date -format 'yyyyMMddHHmmssfff'
Get-Date –Format d
Formats : d, D,f,F,g,G,m,M,r,R,s,t,T,u,U,y,Y
```

B.Méthodes associées à la cmdlet Get-Date

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

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

```
Set-Date -date "6/4/2015 8:30 AM"
Set-Date (Get-Date).AddDays(2)
Set-Date (Get-Date).AddHours(-1)
Set-Date -adjust 1:37:0
(Get-Date).addYears(1).dayOfWeek
([DateTime]'01/21/1971').DayOfWeek
```

D.Calculs sur date

```
New-TimeSpan $(Get-Date) $(Get-Date -month 12 -day 31 -year 2014)
$(Get-Date)
New-TimeSpan $(Get-Date) $(Get-Date -month 12 -day 31 -year 2014)
New-TimeSpan $(Get-Date) $(Get-Date -month 12 -day 31 -year 2014 -hour 23
-minute 30)
New-TimeSpan $(Get-Date 1/1/2016) $(Get-Date 31/12/2016)
```

E.Création de fichier

New-Item –Type file –Name "Rapport_$((Get-Date –Format 'yyyyMMdd')).txt"

VI.Gestion Des Fichiers

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

A.Système

1.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
```

2.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
```

3.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
```

4.Renommer les fichiers

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

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

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

C.Tester l'existence d'un chemin

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

D.Lire un répertoire

1.Commandes

```
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
Get-ChildItem | Where-Object { -not $_.PSIsContainer } : liste les
fichiers uniquement
Get-ChildItem -File : idem à la précédente
Get-ChildItem -Force | Where-Object { -not $_.PSIsContainer -and
$_.Attributes -band [IO.FileAttributes]::Archive }
Get-ChildItem -File -Hidden : idem à la précédente
Get-ChildItem -Attribute !Directory+Hidden,!Directory
```

2.Attributs (IO.FileAttributes)

- ReadOnly
- Hidden
- System
- Directory
- Archive
- Device
- Normal
- Temporary
- SparseFile
- ReparsePoint
- Compressed
- Offline
- NotContentIndexed
- Encrypted

E.La sécurité

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

```
$acls=Get-Acl -Path 'c:\test\fictest.txt'
ForEach($fic in Get-ChildItem 'd:\powershell')
{
$path=$fic.FullName
Set-Acl -Path $path -AclObject $acls
}
```

F.Ajout à un fichier

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

G.Recherche dans un fichier

```
Select-String -Path 'c:\windows\ntbtlog.txt' -Pattern 'Did not load
driver'
Select-String -Path 'c:\windows\ntbtlog.txt' -Pattern 'Did not load
driver' -List
Select-String -Path 'c:\windows\ntbtlog.txt' -Pattern 'Did not load
driver' -quiet
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
```

H.Les redirections

On peut créer des fichiers avec les opérateurs de redirection usuels : > et >>

I.Création d'un fichier

La différence entre Out-File et Set-Content est que le premier ne sait créer que des fichiers texte.

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

J.Effacer le contenu d'un fichier

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

K.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
```

1.Utiliser une page CSS

```
Get-Service|where`Status -eq 'running'|ConvertTo-HTML -Property
Name,DisplayName
-Title 'Liste des services' `
-Body '<h1>Services qui s''exécutent</h1>'|Out-file
c:\powershell\services.html

Get-Service|where`Status -eq 'running'|ConvertTo-HTML -Property
Name,DisplayName
-Head '<title>Areuhhh</title><link rel="stylesheet" type="text/css"
href="style.css"/>'`
-Body '<h1>Services qui s''exécutent</h1>'|Out-file
c:\powershell\services.html
```

L.Conversion en JSON

```
Get-Process|ConvertTo-JSON
'{ "Temps": "Lundi 6 avril 2015 17:45" }' | ConvertFrom-Json | Get-Member
-Name Temps
```

M.Compter les lignes d'un fichier

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

N.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"}
```

O.Les fichiers XML

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

P.Export CSV

La différence entre ConvertTo-CSV et Export-CSV est que la conversion pour ConvertTo est réalisée en mémoire. Attention aux gros tableaux !

```
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
```

Q.Sauvegarde d'un fichier

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

R.Export Xml

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

S.Sauvegarder dans un fichier texte

Outfile permet de choisir l'encodage avec le paramètre –Encoding.

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

T.Interactif

Get-Service|Out-GridView

VII.Registre

A.Lecture d'une clé

```
Get-ChildItem -Path hkcu:\
```

B.Créer une clé

Push-Location
Set-Location HKCU:
Test-Path .\Software\istv
New-Item -Path .\Software -Name istv
Pop-Location

C.Créer une valeur
New-ItemProperty -path HKLM:\SOFTWARE\Microsoft\Windows\CurrentVersion\Run -name "Notepad" -value "C:\WINDOWS\NOTEPAD.EXE" -type string

D.Suppression de clé
Remove-Item

E.Lecture / Ecriture
```
$val = Get-ItemProperty -Path
hklm:software\microsoft\windows\currentversion\policies\system -Name "EnableLUA"
if($val.EnableLUA -ne 0)
{
set-itemproperty -Path
hklm:software\microsoft\windows\currentversion\policies\system -Name "EnableLUA"
-value 0
}
```

VIII.Exécution Distante

A.Présentation
Powershell utilise le RPC. Il s'appuie sur le service WinRM (Gestion à distance de Windows).
Au niveau du pare-feu, vérifiez que les règles liées à la gestion distante soient activées.
Pour vérifier que le service s'exécute, tapez : netstat –ano|find "5985".Pour configurer le service, tapez sous Powershell : Enable-PSRemoting.
Vous disposez aussi de la commande
winrm –quickconfig.
Pour vérifier la configuration : winrm get winrm/config

1.Sécurité
```
Enable-PSRemoting
Enter-PSSession –ComputerName host –Credential domain\user
Get-PSSessionConfiguration
```

B.Authentification
Dans un domaine, elle est de type Kerberos. Sinon, elle est en mode Negotiate (NTLM de poste à poste)

C.Machines de confiance (Poste à poste)

C'est du côté client.
```
Set-Item WSMan:\localhost\client\trustedhosts -value ACERARIEN -force –
Concatenate
Get-Item WSMan:\localhost\client\trustedhosts
```

Pour verifier : winrm get winrm/configAu niveau du registre, passez le paramètre HKLM\SOFTWARE\Microsoft\windows\CurrentVersion\Policies\System\LocalAccountTokenFilterPolicy

En Powershell :
Set-ItempProperty –Path HKLM:\SOFTWARE\Microsoft\windows\CurrentVersion\Policies\System –name LocalAccountTokenFilterPolicy –Value 1 –Type DWord

D.Droits
Seuls les utilisateurs des groupes Administrateurs et Utilisateurs de gestion à distance peuvent se

connecter via WinRM.

```
Set-PSSessionConfiguration -ShowSecurityDescriptorUI -Name
Microsoft.PowerShell
```

E.Sessions

1.Session temporaire

Implicite par Invoke-Command et Enter-PSSession
Enter-PSSession –ComputerName ACERARIEN
Pour qu'elle soit permanente, ajoutez le paramètre – Session

2.Session permanente

New-PSSession –ComputerName ACERARIEN

3.Exécution distante

Invoke-Command –ComputerName ACERARIEN –ScriptBlock {$env::PATH}

4.Rappel de la session

$session=New-PSSession –ComputerName ACERARIEN
Invoke-Command –Session $session –ScriptBlock {$env::PATH}

F.Liste des commandes possibles

```
Get-Help * -Parameter ComputerName
```

G.Exemples

1.Invoke-Command

```
EXit-PSSession
$motdepasse = ConvertTo-SecureString "password" -AsPlainText -Force
$authentification = New-Object System.Management.Automation.PSCredential
`
("MF230\Administrateur",$motdepasse)
$session=New-PSSession –ComputerName MF230 -Credential $authentification
$path=Invoke-Command -ScriptBlock {$env:computername} -Session $session
$cmd=Invoke-Command -ScriptBlock {&ipconfig} -Session $session
clear
$path
$cmd
```

2.Get-Process

```
$motdepasse = ConvertTo-SecureString "password" -AsPlainText -Force
$authentification = New-Object System.Management.Automation.PSCredential
`
("MF230\Administrateur",$motdepasse)
Enter-PSSession –ComputerName MF230 -Credential $authentification
Get-Process -ComputerName MF230
```

IX.Modules Windows 8 Et Windows 2012

A.NetAdapter

1.Importer le module NetAdapter

```
Import-Module NetAdapter
```

2.Profil
```
Get-NetConnectionProfile
```

3.Lister les périphériques réseaux
```
Get-NetAdapter
```

4.Elements attachés à la carte réseau
```
Get-NetAdapterBinding Ether*|Where-Object Enabled
```

5.Désactiver IPv6
```
Get-NetAdapterBinding -DisplayName *TCP/IPv6* | Disable-NetAdapterBinding
```

B.Partage réseau SmbShare
```
Import-Module SmbShare
Get-SmbShare
New-SmbShare –Path C:\test –Name test
Remove-SmbShare –Name test
Get-SmbSession
Get-SmbSession -ClientUserName *admin* | Close-SmbSession
Get-SmbShareAccess -Name test
Get-SmbShareAccess –Name test | Revoke-SmbShareAccess – AccountName
Everyone
Block-SmbShareAccess –Name test –AccountName Everyone
Get-SMBOpenFile | Select-Object ClientComputerName,ClientUserName,Path
Get-SMBOpenFile | Select-Object ClientComputerName,ClientUserName,Path
Get-SmbOpenFile -ClientUserName mdn\administrator | Close-SmbOpenFile
```

C.Impression
```
Import-Module PrintManagement
Get-Printer -Name *Brother* | Select-Object Name,Type,DriverName,PortName
Get-Printer -Name *Brother* | Get-PrintJob | Remove-PrintJob
```

D.ODBC
```
Import-Module wdac
Get_OdbcDsn
Add-OdbcDsn -Name InternalDsn -DsnType User -DriverName "SQL Server"
-SetPropertyValue @("Database=LocalDatabase","Server=sql2008")
```

E.DNS
```
Resolve-DnsName –Name yahoo.fr | Format-List
Get-DnsClientCache| Select-Object -Property Name
Get-DNSClientServerAddress|Where-Object ServerAddresses
```

F.Disque
```
Import-Module Storage
Get-Disk
Get-Volume | Select-Object -Property DriveLetter,FileSystemLabel,Size
Initialize-Disk
New-Partition
Format-Volume -DriveLetter D|Format-List
```

G.Drivers
```
Get-WindowsDriver -Online | where date -gt 10/8/2014
```

H.Applications

```
Get-AppxPackage | Select Name, Version, Publisher | Where Publisher
-Match Microsoft | Sort Name
```

X.A Tester

A.Panneau de configuration

```
Get-ControlPanelItem -Name Affichage
```

B.Renommer un ordinateur

```
Rename-Computer –ComputerName anciennom –NewName nouveaunom
–DomainCredential nouveaunom\administrateur –Force –Restart
```

C.Windows Core

```
Add-WindowsFeature Server-Gui-Shell, Server-Gui-Mgmt-Infra
Install-WindowsFeature Server-Gui-Shell, Server-Gui-Mgmt-Infra
```

XI.Active Directory

A.ADSI

Pour les versions antérieures à Windows 2008.
Il permet de gérer la base de comptes locaux.

1.Gestion des groupes locaux

a)Liste des groupes et des utilisateurs locaux

```
$conn=[ADSI]"WinNT://."
$conn.Children|Where SchemaClassName -eq 'group'|Select -ExpandProperty
Name
$conn.Children|Where SchemaClassName -eq 'user'|Select -ExpandProperty
Name
```

Membre d'un groupe

```
$conn=[ADSI]"WinNT://./Administrateurs,group"
$conn.Invoke('Members')|Foreach{
$_.GetType().InvokeMember('Name','GetProperty',$null,$_,$Null)
}
```

Ajout à un groupe

```
$conn=[ADSI]"WinNT://./Utilisateurs,group"
$conn.Add("WinNT://Administrateur")
```

b)Supprimer un membre d'un groupe

```
$conn=[ADSI]"WinNT://./Utilisateurs,group"
$conn.Remove("WinNT://Administrateur")
```

Lister les utilisateurs

```
$adsi = [ADSI]"WinNT://."
$adsi.psbase.children | where {$_.psbase.schemaClassName -match "user"}
| select @{n="Name";e={$_.name}}
```

Créer un groupe

```
$conn = [ADSI]"WinNT://."
```

```
$ogrp= $conn.Create('group','test')
$ogrp.Put('Description','Groupe de test')
$ogrp.SetInfo()
$ogrp.Dispose()
$conn.Dispose()
```

Renommer un groupe

```
$conn = [ADSI]"WinNT://./test,group"
$conn.PSBase.rename('test2')
$conn.setInfo()
$conn.Dispose()
```

Gestion des utilisateurs

c)Création d'un compte utilisateur

```
Clear

$oDom = [ADSI]"WinNT://."
$oUser=$oDom.Create("user","tondeur-h")
$oUser.PSBase.InvokeSet('Description','Big Boss')
$oUser.SetPassword("mdpnull")
$oUser.SetInfo()
$oUser.Dispose()
$oDom.Dispose()
```

d)Modifier un compte local

```
Clear
$oUser = [ADSI]"WinNT://./tondeur-h,user"
$oUser.PSBase.InvokeSet('Description','hervé')
$oUser.SetInfo()
$oUser.Dispose()
```

e)Lister les propriétés d'un utilisateur

```
Clear
$oUser = [ADSI]"WinNT://./Administrateur,user"
$oUser.PSAdapted
$oUser.PSBase.InvokeGet('LastLogin').DateTime
$oUser.PSBase.InvokeGet('PasswordAge')
```

B.Module (à partir de Windows Server 2008)

1.Import

```
Import-Module ActiveDirectory
Get-Module ActiveDirectory
Get-command –Module ActiveDirectory
```

2.Liste des lecteurs

AD apparaît dans la liste des lecteurs !

```
Get-PSDrive
```

3.Gestion de l'annuaire

a)Lister l'annuaire

```
Get-ChildItem 'AD:\OU=Domain Controllers,DC=istv,DC=net'
```

Requêtes

```
Get-ADObject -LDAPFilter '(&(objectCategory=person)(objectClass=user))'
```

```
Get-ADObject -LDAPFilter '(name=*acer*)'
```

b)filtres
```
Get-ADObject -Filter {objectClass -eq 'computer'}
```

Pour la liste des comptes désactivés :
```
Get-ADObject -Filter {(userAccountControl -eq 514) -and (objectClass -eq
'user')}
```

c)Vitesse d'interrogation
```
Measure-Command{Get-ADObject -Filter {(name -like '*admin*') -and
(ObjectClass -eq 'group')}}
Measure-Command{Get-ADObject -LDAPFilter '(name=*admin*)' | Where
ObjectClass -eq 'group'}
Measure-Command{Get-ADObject -LDAPFilter '(name=*admin*)'}
Measure-Command{Get-ADObject -Filter {name -like '*admin*'}}
```

Lire les propriétés
```
Get-ItemProperty -Path 'AD:\CN=tondeur-h,OU=Informatique,OU=Services
généraux,DC=istv,DC=net' -name displayName
Get-ItemProperty -Path 'AD:\CN=tondeur-h,OU=Informatique,OU=Services
généraux,DC=istv,DC=net' -name displayName|Select-Object -ExpandProperty
displayName
(Get-ItemProperty -Path 'AD:\CN=tondeur-h,OU=Informatique,OU=Services
généraux,DC=istv,DC=net' -name displayName).displayName
```

d)Modifier une propriété
```
$path='AD:\CN=tondeur-h,OU=Informatique,OU=Services
généraux,DC=istv,DC=net'
Set-ItemProperty -Path $path -name displayName -value 'Tondeur Hervé'
(Get-ItemProperty -Path $path -name displayName).displayName
```

e)Déplacement d'un objet
```
$old='AD:\OU=Informatique,DC=istv,DC=net'
$new='AD:\OU=Services généraux,DC=istv,DC=net'
Move-Item -Path $old -Destination $new
```

4.Les utilisateurs

a)Liste des utilisateurs
```
Get-ADUser -Filter * |Select name
Get-ADUser -Filter * |Select name
Get-ADUser -Filter * -Properties WhenCreated|Select Name,WhenCreated
Get-ADUser -Filter * -SearchBase 'OU=Informatique,OU=Services
généraux,DC=istv,DC=net'|Select name
```

b)Création d'un utilisateur
```
$mdp=ConvertTo-SecureString 'paul' -AsPlainText -Force
New-ADUser -SamAccountName paul -Name paul -Path
'OU=Informatique,OU=Services généraux,DC=istv,DC=net' -AccountPassword
$mdp
```

c)Modifier un mot de passe
```
$mdp=ConvertTo-SecureString 'paul' -AsPlainText -Force
Set-ADAccountPassword -Identity paul -NewPassword $mdp -Reset
Set-ADUser -Identity paul -Enabled $true
```

Effacer un utilisateur
```
Remove-ADUser -identity:paul -Confirm:$false
```

lire les attributs

```
Get-ADUser -Identity tondeur-h -Properties *
Get-ADUser -Identity tondeur-h -Properties CN,displayName
```

Modifier des attributs

```
Set-ADUser -identity tondeur-h -Replace @{
Description='Utilisateur Powershell';
TelephoneNumber='0327234567';
OtherTelephone=@('0607080910')}
```

effacer un attribut

```
Set-ADUser -identity tondeur-h -Clear OtherTelephone
```

Les groupes

Commandes relatives aux groups

Get-Command –Module ActiveDirectory –Name *group*

Liste des groupes

```
Get-AdGroup -Filter *|Select Name
Get-AdGroup -Filter {groupScope -eq 'DomainLocal'}|Select Name
Get-AdGroup -Filter * -SearchBase 'OU=Informatique,OU=Services
généraux,DC=istv,DC=net'
```

d)Création de groupes

```
New-ADGroup -Name Formateurs -GroupScope DomainLocal -GroupCategory
Security -Path 'OU=Informatique,OU=Services généraux,DC=dutout,DC=net'
```

e)Membres d'un groupe

```
Get-ADGroupMember -Identity Administrateurs|Select name
```

f)Ajout à un groupe

```
Add-ADGroupMember -Identity Administrateurs -Members denis,thierry
Add-ADPrincipalGroupMembership thierry -MemberOf Administrateurs
```

Supprimer les membres d'un groupe

Pour ces deux commandes, vous pouvez utilizer le paramètre –Confirm:$false
Remove-ADGroupMember
Remove-ADPrincipalGroupMemberShip

Suppression d'un groupe

Remove-ADGroup

C.Déploiement (2012)

```
Import-Module ADDSDeployment
```

1.Ajout de la forêt

```
Install-ADDSForest –DomainName dsfc.local –DomainMode Win2008R2 –
ForestMode Win2008R2 –RebootOnCompletion
```

2.Ajout du DC

```
Install-ADDSDomainController –DomainName dsfc.local
```

3.Désinstallation du DC

```
Uninstall-ADDSDomainController -LastDomainControllerInDomain
-RemoveApplicationPartitions
```

XII.PowerShell Sous Windows 2008 R2

A.Source

http://technet.microsoft.com/fr-fr/library/dd378843%28WS.10%29.aspx

B.La listes des cmdlets

Cmdlet
Description
Add-ADComputerServiceAccount
Adds one or more service accounts to an Active Directory computer.
Add-ADDomainControllerPasswordReplicationPolicy
Adds users, computers, and groups to the Allowed List or the Denied List of the read-only domain controller (RODC) Password Replication Policy (PRP).
Add-ADFineGrainedPasswordPolicySubject
Applies a fine-grained password policy to one more users and groups.
Add-ADGroupMember
Adds one or more members to an Active Directory group.
Add-ADPrincipalGroupMembership
Adds a member to one or more Active Directory groups.
Clear-ADAccountExpiration
Clears the expiration date for an Active Directory account.
Disable-ADAccount
Disables an Active Directory account.
Disable-ADOptionalFeature
Disables an Active Directory optional feature.
Enable-ADAccount
Enables an Active Directory account.
Enable-ADOptionalFeature
Enables an Active Directory optional feature.
Get-ADAccountAuthorizationGroup
Gets the Active Directory security groups that contain an account.
Get-ADAccountResultantPasswordReplicationPolicy
Gets the resultant password replication policy for an Active Directory account.
Get-ADComputer
Gets one or more Active Directory computers.
Get-ADComputerServiceAccount
Gets the service accounts that are hosted by an Active Directory computer.
Get-ADDefaultDomainPasswordPolicy
Gets the default password policy for an Active Directory domain.
Get-ADDomain
Gets an Active Directory domain.
Get-ADDomainController
Gets one or more Active Directory domain controllers, based on discoverable services criteria, search parameters, or by providing a domain controller identifier, such as the NetBIOS name.
Get-ADDomainControllerPasswordReplicationPolicy

Gets the members of the Allowed List or the Denied List of the RODC PRP.
Get-ADDomainControllerPasswordReplicationPolicyUsage
Gets the resultant password policy of the specified ADAccount on the specified RODC.
Get-ADFineGrainedPasswordPolicy
Gets one or more Active Directory fine-grained password policies.
Get-ADFineGrainedPasswordPolicySubject
Gets the users and groups to which a fine-grained password policy is applied.
Get-ADForest
Gets an Active Directory forest.
Get-ADGroup
Gets one or more Active Directory groups.
Get-ADGroupMember
Gets the members of an Active Directory group.
Get-ADObject
Gets one or more Active Directory objects.
Get-ADOptionalFeature
Gets one or more Active Directory optional features.
Get-ADOrganizationalUnit
Gets one or more Active Directory OUs.
Get-ADPrincipalGroupMembership
Gets the Active Directory groups that have a specified user, computer, or group.
Get-ADRootDSE
Gets the root of a domain controller information tree.
Get-ADServiceAccount
Gets one or more Active Directory service accounts.
Get-ADUser
Gets one or more Active Directory users.
Get-ADUserResultantPasswordPolicy
Gets the resultant password policy for a user.
Install-ADServiceAccount
Installs an Active Directory service account on a computer.
Move-ADDirectoryServer
Moves a domain controller in AD DS to a new site.
Move-ADDirectoryServerOperationMasterRole
Moves operation master (also known as flexible single master operations or FSMO) roles to an Active Directory domain controller.
Move-ADObject
Moves an Active Directory object or a container of objects to a different container or domain.
New-ADComputer
Creates a new Active Directory computer.
New-ADFineGrainedPasswordPolicy
Creates a new Active Directory fine-grained password policy.
New-ADGroup
Creates an Active Directory group.
New-ADObject
Creates an Active Directory object.
New-ADOrganizationalUnit
Creates a new Active Directory OU.
New-ADServiceAccount
Creates a new Active Directory service account.

New-ADUser
Creates a new Active Directory user.
Remove-ADComputer
Removes an Active Directory computer.
Remove-ADComputerServiceAccount
Removes one or more service accounts from a computer.
Remove-ADDomainControllerPasswordReplicationPolicy
Removes users, computers, and groups from the Allowed List or the Denied List of the RODC PRP.
Remove-ADFineGrainedPasswordPolicy
Removes an Active Directory fine-grained password policy.
Remove-ADFineGrainedPasswordPolicySubject
Removes one or more users from a fine-grained password policy.
Remove-ADGroup
Removes an Active Directory group.
Remove-ADGroupMember
Removes one or more members from an Active Directory group.
Remove-ADObject
Removes an Active Directory object.
Remove-ADOrganizationalUnit
Removes an Active Directory OU.
Remove-ADPrincipalGroupMembership
Removes a member from one or more Active Directory groups.
Remove-ADServiceAccount
Removes an Active Directory service account.
Remove-ADUser
Removes an Active Directory user.
Rename-ADObject
Changes the name of an Active Directory object.
Reset-ADServiceAccountPassword
Resets the service account password for a computer.
Restore-ADObject
Restores an Active Directory object.
Search-ADAccount
Gets Active Directory user, computer, and service accounts.
Set-ADAccountControl
Modifies user account control (UAC) values for an Active Directory account.
Set-ADAccountExpiration
Sets the expiration date for an Active Directory account.
Set-ADAccountPassword
Modifies the password of an Active Directory account.
Set-ADComputer
Modifies an Active Directory computer.
Set-ADDefaultDomainPasswordPolicy
Modifies the default password policy for an Active Directory domain.
Set-ADDomain
Modifies an Active Directory domain.
Set-ADDomainMode
Sets the domain functional level for an Active Directory domain.
Set-ADFineGrainedPasswordPolicy
Modifies an Active Directory fine-grained password policy.

Set-ADForest
Modifies an Active Directory forest.
Set-ADForestMode
Sets the forest mode for an Active Directory forest.
Set-ADGroup
Modifies an Active Directory group.
Set-ADObject
Modifies an Active Directory object.
Set-ADOrganizationalUnit
Modifies an Active Directory OU.
Set-ADServiceAccount
Modifies an Active Directory service account.
Set-ADUser
Modifies an Active Directory user.
Uninstall-ADServiceAccount
Uninstalls an Active Directory service account from a computer.
Unlock-ADAccount
Unlocks an Active Directory account.

XIII.Quelques Exemples

A.Liste des fichiers exécutés sur la machine

Ce script a pour objet de lire les 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
}
```

B.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)
#
}
}
```

C.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")
Pour disposer de toutes les methods :
$oShell|Get-Member
```

2.Wscript.Network

```
$oNetwork = New-Object -com Wscript.Network
#$oNetWork.UserName
#$env:USERNAME
#$oNetwork.ComputerName
Try
{
$oNetwork.RemoveNetworkDrive('P:')
}
Catch
{
'Ca marche pas'
}
Finally
{
$oNetwork.MapNetworkDrive('P:', '\\10.114.3.152\PatchWin7',$false,`
'MF231\Administrateur','password')
}
$oNetwork=$null
Get-ChildItem x:\
}
$oNetWork.Dispose
```

3.Partage d'imprimante

```
$Path = "\\10.114.3.153\hpjpp"
$oNw = New-Object -com Wscript.Network
Try
{
$oNw.RemoveWindowsPrinterConnection($path)
}
Catch
{
}
Finally
{
$oNw.AddWindowsPrinterConnection($path)
}
```

4.Scripting.FileSystemObject

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

D.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()
```

E.Les compteurs

```
do
{
(Get-Counter -Counter '\Interface réseau(*)\Octets
reçus/s').CounterSamples|Where InstanceName -like 'broadcom*'|Select
CookedValue
}
While($true)
```

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()
```

XIV.Quelques Sites

PowerShell 3.0 est en passe de s'imposer comme technologie de scripting dans les environnements Windows. Derrière une simplicité apparente, se cache parfois une réelle complexité. Ces quelques liens vous permettront, je l'espère, de progresser dans un langage qui s'appuie sur le Framework .Net 4.0.

A.Sites en français

•[Windows PowerShell] https://technet.microsoft.com/fr-fr/library/bb978526.aspx (**site officiel**) : guide
•[Centre de scripts Windows PowerShell] http://www.microsoft.com/powershell (**site officiel**) : téléchargements, scripts, mémento
•[Galerie de scripts PowerShell] https://gallery.technet.microsoft.com/scriptcenter/site/search?f%5B0%5D.Type=ProgrammingLanguage&f%5B0%5D.Value=PowerShell&f%5B0%5D.Text=PowerShell&sortBy=Downloads (**site officiel**) : téléchargements, scripts

B.Sites en anglais

•[PowerShell.com] http://powershell.com/cs/media/default.aspx : scripts, tutoriaux
•[Precision Computing] http://www.leeholmes.com/blog/?s=powershell : scripts

C.Téléchargements

•[Windows Management Framework 3.0] http://www.microsoft.com/en-us/download/details.aspx?id=34595 **(site officiel)**
•[PowerShell Scriptomatic] http://www.microsoft.com/en-us/download/details.aspx?id=24121 (en)

D.Éditeur gratuit

•[PowerShell Plus] https://www.idera.com/productssolutions/freetools/powershellplus

XV.Annexe

1 : 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 = [System.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"

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