

Table des matières

I.	Présentation	4
A.	Préambule.....	4
B.	Technologies de scripting.....	4
C.	Pré-requis.....	4
D.	Nouvelles fonctionnalités de la version Windows PowerShell 2	4
E.	Les outils.....	4
II.	Premiers pas.....	5
A.	Les applets de commande ou cmdlets.....	5
B.	L'interpréteur	5
C.	Protection.....	5
1.	Le niveau de sécurité : Get-ExecutionPolicy.....	5
2.	Changer le niveau de sécurité : Set-ExecutionPolicy	5
3.	Signature	5
4.	Voir aussi.....	5
D.	Aide.....	5
1.	Informations de plate-forme : <i>Get-Host</i>	5
2.	La liste des commandes : <i>Get-Command</i>	5
3.	L'aide : <i>Get-Help</i>	5
4.	Méthodes et propriétés associées à une cmdlet	5
5.	Afficher les propriétés d'un cmdlet.....	5
6.	Afficher les méthodes et propriétés d'un objet.....	5
7.	Les fournisseurs PowerShell : <i>Get-PSProvider</i>	6
E.	Exécution des scripts.....	6
1.	Exécution d'un script.....	6
2.	Appel d'un autre script.....	6
3.	Appel d'un programme	6
4.	Mesurer le temps d'exécution : <i>Measure-Command</i>	6
5.	Tempo.....	6
F.	Historique.....	6
1.	Visualiser l'historique.....	6
2.	Récupérer l'historique.....	6
3.	Exécuter une commande de l'historique	6
4.	Voir aussi.....	6
G.	Informations de langue.....	6
H.	Passage d'arguments	6
1.	Par tableau.....	6
2.	Par la méthode <i>Param</i>	7
I.	Commentaires	7
III.	Cmdlets système.....	8
A.	Le journal d'événements.....	8
B.	Les services.....	8
1.	La liste des services	8
2.	Démarrer, arrêter un service.....	8
3.	Mettre en suspens, reprendre un service	8
4.	Modifier les propriétés des services.....	8
C.	Les process	8
1.	Liste des process.....	8
2.	Arrêter un process	8
D.	WMI	8
IV.	Gestion des heures et des dates	9
A.	Obtenir la date et l'heure : <i>Get-Date</i>	9
1.	Méthodes associées à la cmdlet <i>Get-Date</i>	9

B.	Changer la date et l'heure : Set-Date	9
C.	Calculs sur date	9
V.	Gestion des fichiers.....	10
A.	Suppression de fichiers : Remove-Item	10
B.	Copie de fichiers : Copy-Item	10
C.	Création de fichiers : New-Item	10
D.	Déplacer les fichiers	10
E.	Renommer les fichiers	10
F.	Informations sur les fichiers, répertoires et clés de registres	10
G.	Tester l'existence d'un chemin	10
H.	Lire un répertoire	10
I.	La sécurité.....	10
VI.	Manipulation des fichiers.....	11
A.	Ajout à u fichier.....	11
B.	Caractères spéciaux	11
C.	Recherche dans un fichier	11
D.	Création d'un fichier	11
E.	Effacer le contenu d'un fichier.....	11
F.	Convertir en Html	11
G.	Compter les lignes d'un fichier	11
H.	Lire un fichier CSV	11
I.	Les fichiers XML.....	11
J.	Export CSV	12
K.	Sauvegarde d'un fichier.....	12
L.	Export Xml	12
M.	Sauvegarder dans un fichier texte	12
VII.	Elements du langage	13
A.	Les variables et les constantes	13
1.	Les variables	13
2.	Les constantes.....	13
3.	Autres méthodes.....	13
B.	Opérateurs	13
1.	Comparaison.....	13
2.	Logiques.....	13
C.	Structures de contrôle	13
1.	Do	13
2.	While	13
3.	For	14
4.	Break.....	14
5.	If.....	14
6.	Foreach	14
7.	Switch	14
D.	Cmdlets.....	14
1.	Comptage	14
2.	Stats.....	14
3.	Sélection	14
4.	Tri.....	14
5.	Différence.....	15
6.	Affichage.....	15
7.	Filtre.....	15
8.	Valeurs unique.....	15
9.	Propriétés	15
10.	Impressions.....	15
11.	Boucle	15
12.	Tri.....	15

13. Message.....	15
14. Interaction	16
VIII. Quelques exemples.....	17
A. ADSI	17
B. Liste des fichiers exécutés sur la machine.....	17
C. Liste des services à partir du registre	17
D. Utilisation du composant WSH.....	17
1. Wscript.Shell.....	17
2. Wscript.Network.....	17
3. Scripting.FileSystemObject.....	18
E. MySQL : lecture de tables.....	18
F. MySQL : inventaire	19
1. La table.....	19
2. Le script.....	19
IX. Quelques sites	20
A. Références.....	20
B. Exemples de scripts	20
C. Documentations.....	20
D. Téléchargements.....	20
E. Blogs	20
X. Annexe 1 : les commandes.....	21
XI. Annexe 2 : les alias.....	23
XII. Annexe 3 : de Vbs à Powershell, documentation adaptée d'un document Microsoft.....	24

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 avons 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. 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 fonctionnent, 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'avez 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 "hkml:\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 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
}
```

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

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

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

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

VBScript Function	Windows PowerShell Equivalent
Abs	<code>\$a = [math]::abs(-15)</code>
Array	<code>\$a = "red","orange","yellow","green","blue","indigo","violet"</code>
Asc	<code>\$a = [byte][char] "A"</code>
Atn	<code>\$a = [math]::atan(90)</code>
CBool	<code>\$a = 0</code> <code>\$a = [bool] \$a</code>
CByte	<code>\$a = "11.45"</code> <code>\$a = [byte] \$a</code>
CCur	<code>\$a = "{0:C}" -f 13</code>
CDate	<code>\$a = "11/1/2006"</code> <code>\$a = [datetime] \$a</code>
CDbl	<code>\$a = "11.45"</code> <code>\$a = [double] \$a</code>
Chr	<code>\$a = [char]34</code>
CInt	<code>\$a = "11.57"</code> <code>\$a = [int] \$a</code>
CLng	<code>\$a = "123456789.45"</code> <code>\$a = [long] \$a</code>
Cos	<code>\$a = [math]::cos(45)</code>
CreateObject	<code>\$a.visible = \$True</code> <code>\$a = new-object -comobject Excel.Application -strict</code>
CSng	<code>\$a = "11.45"</code> <code>\$a = [single] \$a</code>
CStr	<code>\$a = 17</code> <code>\$a = [string] \$a</code>
Date	<code>\$a = get-date -format d</code>
DateAdd	<code>\$a = (get-date).AddDays(37)</code> <code>(get-date).AddHours(37)</code> <code>(get-date).AddMilliseconds(37)</code> <code>(get-date).AddMinutes(37)</code> <code>(get-date).AddMonths(37)</code> <code>(get-date).AddSeconds(37)</code> <code>(get-date).AddTicks(37)</code> <code>(get-date).AddYears(37)</code> <code>\$a = ((get-date).AddHours(2)).AddMinutes(34)</code>
DateDiff	<code>\$a = New-TimeSpan \$(Get-Date) \$(Get-Date -month 12 -day 31 -year 2006 -hour 23 -minute 30)</code> <code>\$a.Days</code> Days : 109 Hours : 3 Minutes : 55 Seconds : 0 Milliseconds : 0 Ticks : 9431700000000 TotalDays : 109.163194444444 TotalHours : 2619.91666666667 TotalMinutes : 157195 TotalSeconds : 9431700 TotalMilliseconds : 9431700000
DatePart	<code>\$a = (get-date).day</code>

	<pre>\$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</pre>
DateSerial	<pre>MyDate1 = DateSerial(2006, 12, 31) \$a = get-date -y 2006 -mo 12 -day 31</pre>
DateValue	<pre>\$a = [datetime] "12/1/2006"</pre>
Day	<pre>\$a = (get-date).day</pre>
Eval	<pre>\$a = 2 + 2 -eq 45</pre>
Exp	<pre>\$a = [math]::exp(2)</pre>
Filter	<pre>\$a = "Monday","Month","Merry","Mansion","Modest" \$b = (\$a where-object {\$_ -like "Mon*"})</pre>
FormatCurrency	<pre>\$a = 1000 \$a = "{0:C}" -f \$a</pre>
FormatDateTime	<pre>\$a = (get-date).tolongdatestring() \$a = (get-date).toshortdatestring() \$a = (get-date).tolongtimestring() \$a = (get-date).toshorttimestring()</pre>
FormatNumber	<pre>\$a = 11 \$a = "{0:N6}" -f \$a</pre>
FormatPercent	<pre>\$a = .113 \$a = "{0:P1}" -f \$a</pre>
GetLocale	<pre>\$a = (get-culture).lcid \$a = (get-culture).displayname</pre>
Hex	<pre>\$a = 4517 \$a = "{0:X}" -f \$a</pre>
Hour	<pre>\$a = (get-date).hour</pre>
InputBox	<pre>\$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")</pre>
InStr	<pre>\$a = "wombat" \$b = \$a.contains("m") \$b = \$a.indexof("m")</pre>
InStrRev	<pre>\$a = "1234x6789x1234" \$b = \$a.lastindexofany("x")</pre>
Int/Fix	<pre>\$a = 11.98 \$a = [math]::truncate(\$a)</pre>
isArray	<pre>\$a = 22,5,10,8,12,9,80 \$b = \$a -is [array]</pre>
IsDate	<pre>\$a = 11/2/2006 \$a -is [datetime] \$a = [datetime] "11/2/2006"</pre>
IsEmpty	<pre>\$a = "" \$b = \$a.length -eq 0</pre>
IsNull	<pre>\$a = \$z -eq \$null</pre>
IsNumeric	<pre>\$a = 44.5 [reflection.assembly]::LoadWithPartialName("Microsoft.VisualBasic")</pre>

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

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

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