Windows PowerShell

Source : https://www.supinfo.com/articles/single/4224-debuter-avec-powershell

Introduction

PowerShell est un langage de script développé par Microsoft et est inclus dans toutes les versions de Windows à partir de Windows 7 et Windows Server 2008. Cependant, PowerShell peut être installé sur toutes les machines ayant la version 2.0 du .NET framework installée.

L'utilisation de PowerShell est extrêmement utile voire indispensable pour les administrateurs systèmes. En effet, ce langage de script va permettre d'automatiser de nombreuses tâches, de manipuler des fichiers, d'administrer un Active directory, de faire des opérations sur un serveur exchange, de gérer un serveur SharePoint, d'administrer des machines client à distance ou encore d'effectuer des actions sur de nombreuses machines en même temps.

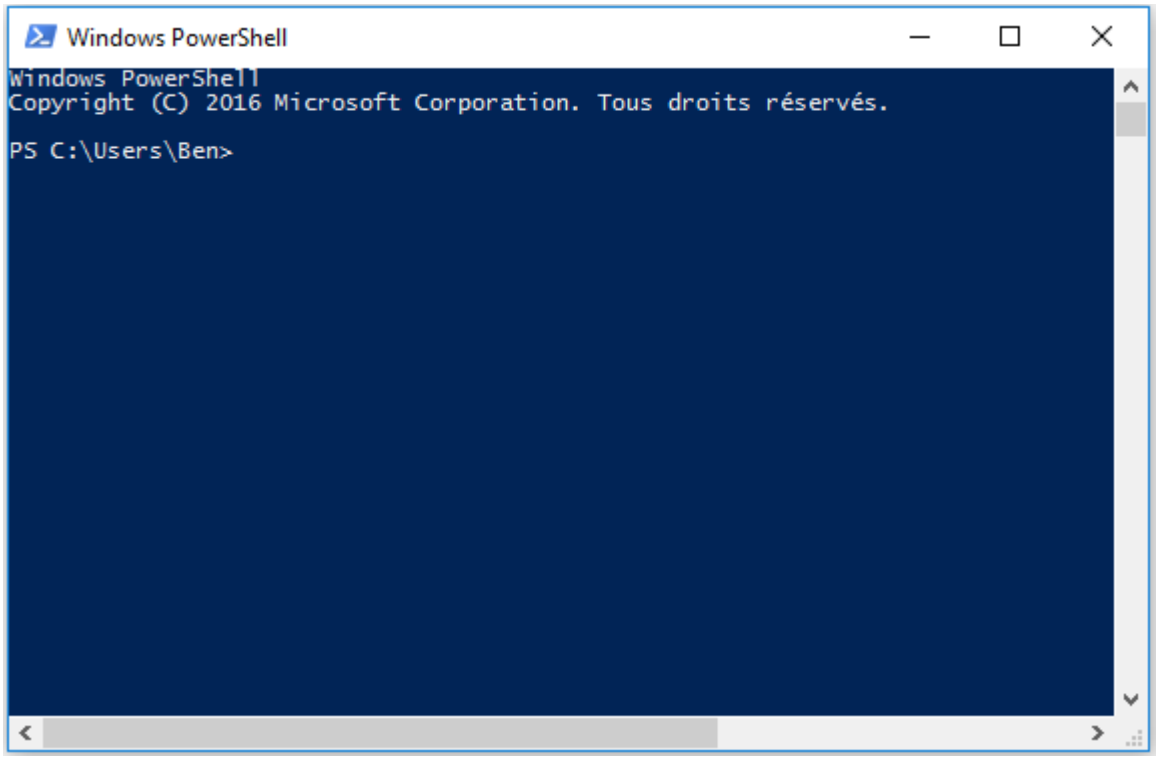
Si vous êtes familier avec un environnement Unix et l'utilisation de script batch ou avec l'utilisation de VBscript, l'apprentissage de ce langage vous sera grandement facilité.

Les chapitres de cet article ne se suivent pas nécessairement, ils abordent tous une utilisation différente de PowerShell pour des besoins spécifiques.

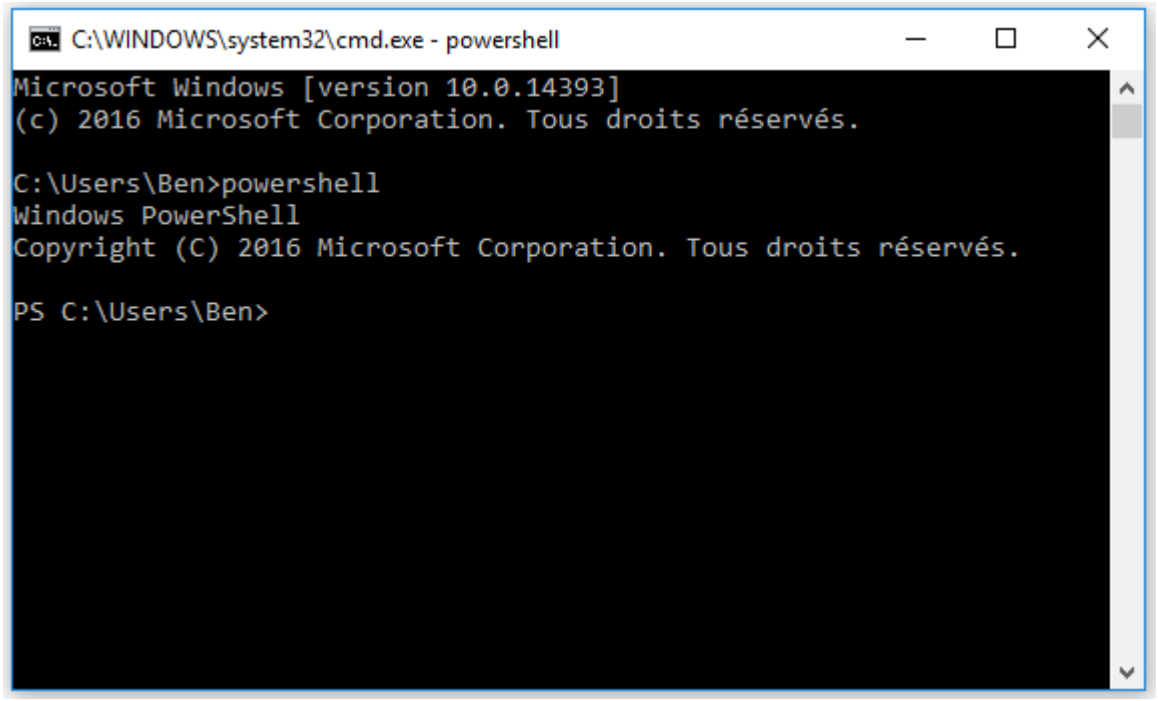
Utilisation des logiciels et manipulation du code

Le langage PowerShell peut être utilisé directement avec l'interpréteur de commande PowerShell, avec la console d'édition de script "PowerShell ISE" intégrée à Windows ou encore avec l'invite de commande Windows.

1. Interpréteur de commande PowerShell :

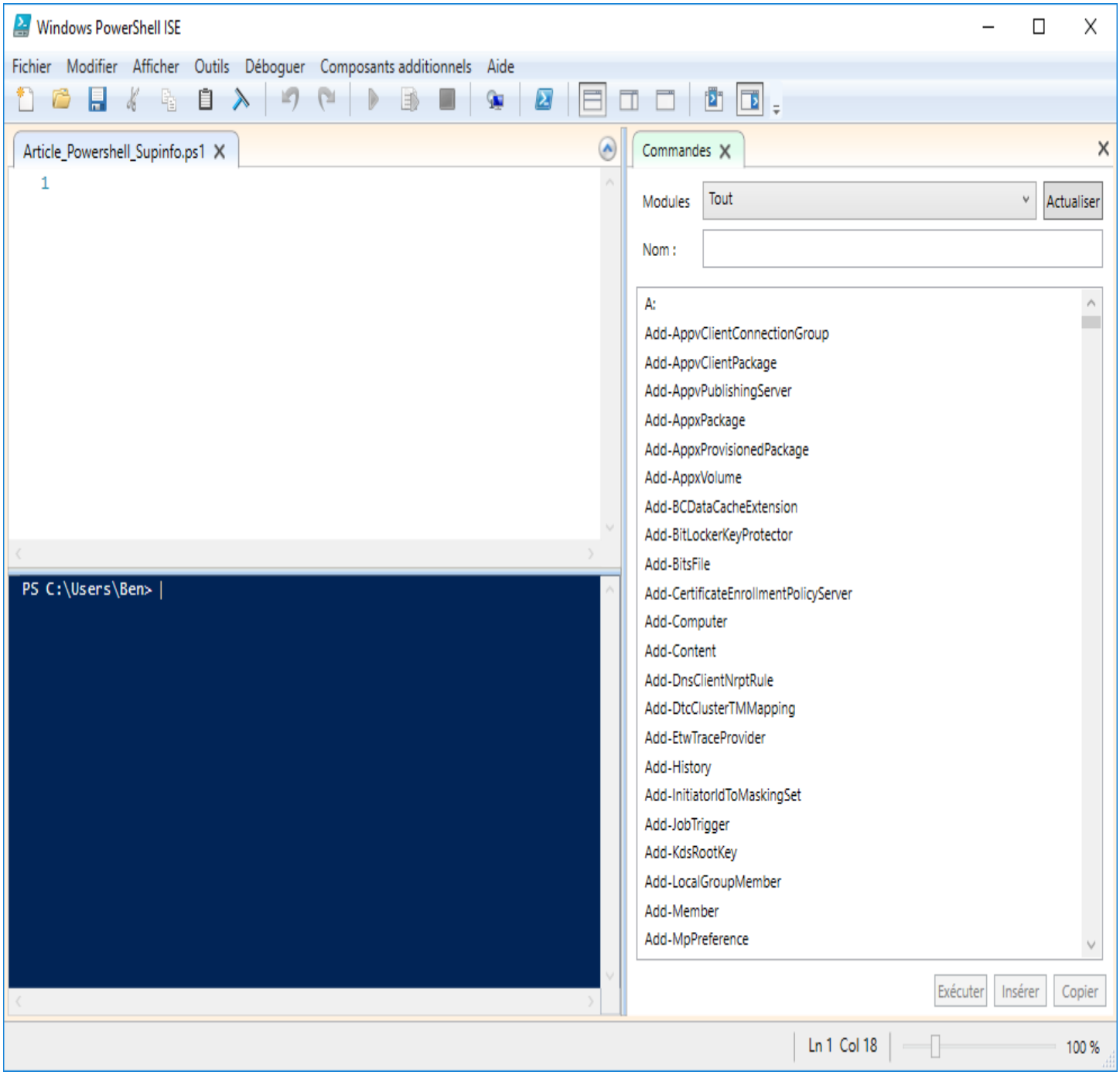


2. Invite de commande Windows :

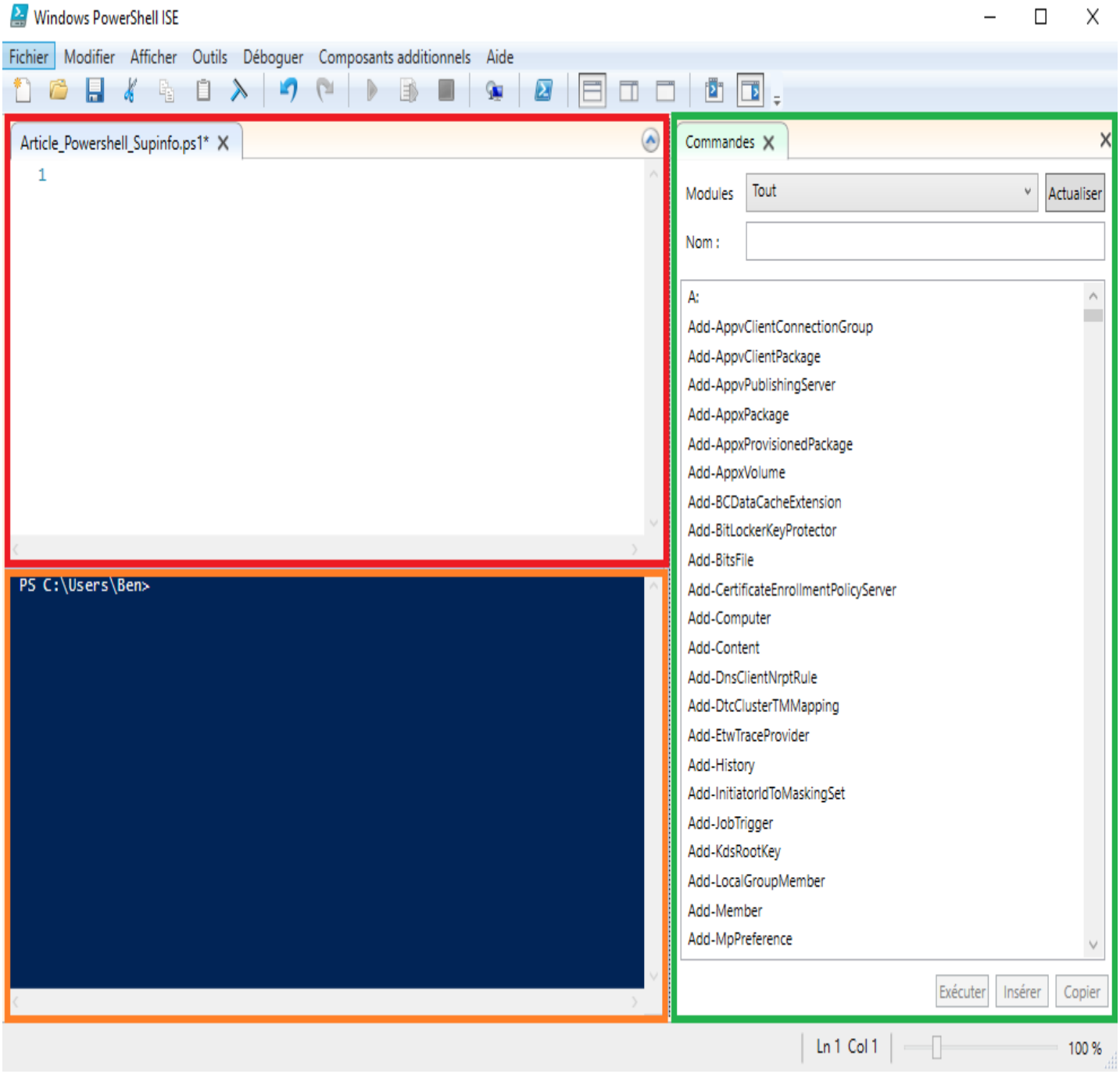


Il est nécessaire de taper la commande "powershell" dans le CMD pour pouvoir se servir du langage PowerShell dans celui-ci.

3. Powershell ISE :



Même si l'interpréteur de commande suffit amplement pour effectuer quelques commandes à la volée, celui-ci devient assez contraignant lorsque l'on doit exécuter un script plus important. En effet, l'interpréteur nous permet d'écrire ligne par ligne, en exécutant chacune de celles-ci, alors que l'IDE nous permet d'écrire un programme complet qu'on pourra exécuter dans son intégralité sans action supplémentaire de notre part et pourra être sauvegardé pour être relancé ultérieurement.



- Le bloc du haut (en rouge) : c'est l'espace dédié à l'écriture du code. Ici, il n'y a qu'un script ouvert. Si plusieurs scripts sont ouverts dans la même fenêtre PowerShell ISE, ils apparaîtront dans de nouveaux onglets.
- Le bloc du bas (en orange) : il sert à exécuter du code exactement comme dans l'interpréteur de commande vu précédemment, mais sert aussi à visualiser les retours du code lorsqu'il est lancé depuis PowerShell ISE via le bouton "Exécuter le script" (raccourci : touche F5).
- Le bloc de droite (en vert) est très utile car il contient toutes les commandes PowerShell ainsi que leurs descriptions, leurs paramètres, leurs syntaxes, les entrées et sorties, etc. Il est possible dans ce menu de rechercher une commande en particulier.

Si l'éditeur de script natif de Microsoft ne vous convient pas, sachez qu'il en existe de nombreux autres et je vous invite à vous renseigner à ce sujet.

Les scripts PowerShell peuvent être enregistrés dans des fichiers ayant pour extension « .ps1 » et ceux-ci peuvent être directement exécutés.

Si lors de la première utilisation vous recevez le message suivant : « Impossible de charger le fichier C:\NomDuFichierPowershell.ps1, car l'exécution de scripts est désactivée sur ce système. », c'est simplement que vous n'avez pas donné l'autorisation à votre système d'exécuter des commandes PowerShell. Pour ce faire, tapez la commande « Set-ExecutionPolicy RemoteSigned » et vous devriez pouvoir exécuter votre code.

Pour ceux qui ont déjà l'habitude du Visual Basic, voici le lien pour la correspondance entre les commandes vbscript et PowerShell : https://technet.microsoft.com/en-us/library/ee692947.aspx

Tips et raccourcis clavier

- Auto-complétion : comme tous les langages, PowerShell fournit une commande d'auto-complétion pour faciliter et accélérer l'écriture du code. Si vous êtes au milieu d'une ligne de code, vous avez la possibilité d'appuyer sur la touche « Tab », et la commande que vous écrivez va se compléter automatiquement. Si ce n'est pas la commande que vous désirez, vous avez la possibilité d'appuyer à nouveau sur Tab pour faire défiler toutes les possibilités de complétion pour votre commande. De plus, PowerShell nous permet d'afficher toutes les possibilités de complétions disponibles (au lieu de les passer une par une comme ci-dessus) par l'associations des touches Ctrl + Espace.
- Réexécuter du code : si vous voulez relancer une ligne de code déjà écrite précédemment sans avoir à la réécrire à nouveau, vous pouvez faire défiler toutes les anciennes commandes exécutées en appuyant sur la flèche du haut.
- Copier-Coller : dans l'interpréteur de commande PowerShell, vous ne pouvez pas utiliser votre clavier pour faire un copier/coller. Pour ce faire, vous devez sélectionner la zone que vous voulez copier, et faire un clic droit sur celle-ci. La sélection disparaîtra et le bout de code sélectionné s'enregistrera dans le presse-papier. Pour coller, il vous suffit simplement de faire un clic droit à nouveau.
- La touche F7 affiche l'historique de toutes les commandes tapées sous forme d'un tableau.
- PowerShell n'est pas sensible à la casse, aux espaces et aux tabulations en trop.
- Les commentaires se font par le biais du caractère "#" pour commenter une seule ligne et des caractères "<#" et "#>" en début et en fin pour commenter plusieurs lignes.

Les variables

Voici une très courte présentation du fonctionnement de base des variables PowerShell. Etant donné qu'il n'y a que quelques subtilités à retenir, nous ne nous y attarderons pas.

Les variables en PowerShell doivent toujours être précédées du symbole « $ » et celles-ci sont automatiquement typées lors de leur affectation.

Par exemple, pour déclarer une string, il faut simplement écrire : « $string = "Glados" ». Pour un int, il faut écrire = « $int = 42 ». Voici un exemple de déclaration pour une liste = « $list = 1, 3, 3, 7 ». Le procédé est le même pour toutes les autres variables.

En ce qui concerne les listes, pour ajouter un item dans celles-ci, il faut écrire « $list += $nouvelElement ». En effet dans le cas d'une liste, nous ne pouvons pas utiliser la commande « $list.Add($nouvelElement) » car une liste est une collection d'éléments de taille fixe et nous ne pouvons donc pas en rajouter. L'utilisation de la commande "+=" permet de créer une nouvelle liste avec tous les éléments contenus dans l'ancienne avec le nouvel élément en plus. Ce procédé écrase l'ancienne liste.

Evidemment, les exemples ci-dessus traitent uniquement les cas où on veut affecter une valeur à la variable lors de sa déclaration. Sachez qu'il est possible de créer des variables vides déjà typées.

Par exemple, une liste vide peut être créée avec la commande suivante : « $list = @() ».

Etant donné que la structure des fonctions de bases telles que « if », « else », « while », etc, sont communes a beaucoup de langages, elles ne seront pas présentées ici.

Les opérateurs

Les opérateurs de comparaisons PowerShell sont différents des autres langages. Il est donc nécessaire de faire un point rapide sur ceux-ci :

- "eq" : égal à (Equal)
- "ne" : différents (Not Equal)
- "lt" : plus petit que (Less Than)
- "gt" : plus grand que (Greater Than)
- "ge" : plus grand ou égal (Greater than or Equal to)
- "le" : plus petit ou égal (Less than or Equal to)

Un exemple simple pour visualiser :

```
if ($maVariable1 -eq $maVariable2) {
   #bloc de code
}
```

Ce qui se traduirait par "si la variable 1 est égale à la variable 2, alors on exécute le bloc de code."

Les commandes (cmdlet)

Voici une liste des commandes les plus utiles en PowerShell :

Get-Help

PowerShell contient sa propre documentation. Cette commande permet d’afficher l’aide en rapport avec une commande en particulier.

```
PS C:\Users\Ben> Get-Help Get-Member

NOM
    Get-Member

SYNTAXE
    Get-Member [[-Name] <string[]>]  [<CommonParameters>]


ALIAS
    gm


REMARQUES
    Get-Help ne parvient pas à trouver les fichiers d'aide de cette applet de commande sur cet ordinateur.
    trouve qu'une aide partielle.
        -- Pour télécharger et installer les fichiers d'aide du module comportant cette applet de commande,
    Update-Help.
        -- Pour afficher en ligne la rubrique d'aide de cette applet de commande, tapez : «Get-Help Get-Mem
    » ou
           accédez à http://go.microsoft.com/fwlink/?LinkID=113322.
```

Get-Command

Permet d'afficher la liste de toutes les commandes PowerShell.

```
PS G:\Work\Supinfo\Articles\Powershell> Get-Command

CommandType     Name                                               Version    Source
-----------     ----                                               -------    ------
Alias           Add-ProvisionedAppxPackage                         3.0        Dism
Alias           Apply-WindowsUnattend                              3.0        Dism
Alias           Disable-PhysicalDiskIndication                     2.0.0.0    Storage
Alias           Disable-StorageDiagnosticLog                       2.0.0.0    Storage
Alias           Enable-PhysicalDiskIndication                      2.0.0.0    Storage
Alias           Enable-StorageDiagnosticLog                        2.0.0.0    Storage
Alias           Flush-Volume                                       2.0.0.0    Storage
Alias           Get-DiskSNV                                        2.0.0.0    Storage
Alias           Get-PhysicalDiskSNV                                2.0.0.0    Storage
Alias           Get-ProvisionedAppxPackage                         3.0        Dism
Alias           Get-StorageEnclosureSNV                            2.0.0.0    Storage
Alias           Initialize-Volume                                  2.0.0.0    Storage
Alias           Move-SmbClient                                     2.0.0.0    SmbWitness
Alias           Remove-ProvisionedAppxPackage                      3.0        Dism
Alias           Write-FileSystemCache                              2.0.0.0    Storage
Function        A:
Function        Add-BCDataCacheExtension                           1.0.0.0    BranchCache
Function        Add-BitLockerKeyProtector                          1.0.0.0    BitLocker
Function        Add-DnsClientNrptRule                              1.0.0.0    DnsClient
Function        Add-DtcClusterTMMapping                            1.0.0.0    MsDtc
Function        Add-EtwTraceProvider                               1.0.0.0    EventTracingManagement
Function        Add-InitiatorIdToMaskingSet                        2.0.0.0    Storage
Function        Add-MpPreference                                   1.0        Defender
Function        Add-NetEventNetworkAdapter                         1.0.0.0    NetEventPacketCapture
Function        Add-NetEventPacketCaptureProvider                  1.0.0.0    NetEventPacketCapture
Function        Add-NetEventProvider                               1.0.0.0    NetEventPacketCapture
Function        Add-NetEventVFPProvider                            1.0.0.0    NetEventPacketCapture
Function        Add-NetEventVmNetworkAdapter                       1.0.0.0    NetEventPacketCapture
Function        Add-NetEventVmSwitch                               1.0.0.0    NetEventPacketCapture
Function        Add-NetEventVmSwitchProvider                       1.0.0.0    NetEventPacketCapture
Function        Add-NetEventWFPCaptureProvider                     1.0.0.0    NetEventPacketCapture
Function        Add-NetIPHttpsCertBinding                          1.0.0.0    NetworkTransition
Function        Add-NetLbfoTeamMember                              2.0.0.0    NetLbfo
Function        Add-NetLbfoTeamNic                                 2.0.0.0    NetLbfo
Function        Add-NetNatExternalAddress                          1.0.0.0    NetNat
Function        Add-NetNatStaticMapping                            1.0.0.0    NetNat
Function        Add-NetSwitchTeamMember                            1.0.0.0    NetSwitchTeam
```

Get-Member

L'applet de commande Get-Member sert à récupérer les propriétés et les méthodes disponibles pour un objet spécifié. Pour l'exemple suivant, nous utiliserons le pipeline "|" que nous verrons un peu plus loin.

```
PS C:\Windows\system32> Get-Process | Get-Member


   TypeName : System.Diagnostics.Process

Name                       MemberType     Definition
----                       ----------     ----------
Handles                    AliasProperty  Handles = Handlecount
Name                       AliasProperty  Name = ProcessName
NPM                        AliasProperty  NPM = NonpagedSystemMemorySize64
PM                         AliasProperty  PM = PagedMemorySize64
SI                         AliasProperty  SI = SessionId
VM                         AliasProperty  VM = VirtualMemorySize64
WS                         AliasProperty  WS = WorkingSet64
Disposed                   Event          System.EventHandler Disposed(System.Object, System.EventArgs)
ErrorDataReceived          Event          System.Diagnostics.DataReceivedEventHandler ErrorDataReceived(Sys
Exited                     Event          System.EventHandler Exited(System.Object, System.EventArgs)
OutputDataReceived         Event          System.Diagnostics.DataReceivedEventHandler OutputDataReceived(Sy
BeginErrorReadLine         Method         void BeginErrorReadLine()
BeginOutputReadLine        Method         void BeginOutputReadLine()
CancelErrorRead            Method         void CancelErrorRead()
CancelOutputRead           Method         void CancelOutputRead()
Close                      Method         void Close()
CloseMainWindow            Method         bool CloseMainWindow()
CreateObjRef               Method         System.Runtime.Remoting.ObjRef CreateObjRef(type requestedType)
Dispose                    Method         void Dispose(), void IDisposable.Dispose()
Equals                     Method         bool Equals(System.Object obj)
GetHashCode                Method         int GetHashCode()
GetLifetimeService         Method         System.Object GetLifetimeService()
GetType                    Method         type GetType()
InitializeLifetimeService  Method         System.Object InitializeLifetimeService()
Kill                       Method         void Kill()
Refresh                    Method         void Refresh()
Start                      Method         bool Start()
ToString                   Method         string ToString()
WaitForExit                Method         bool WaitForExit(int milliseconds), void WaitForExit()
WaitForInputIdle           Method         bool WaitForInputIdle(int milliseconds), bool WaitForInputIdle()
__NounName                 NoteProperty   string __NounName=Process
BasePriority               Property       int BasePriority {get;}
Container                  Property       System.ComponentModel.IContainer Container {get;}
```

Dans l'exemple ci-dessus, nous avons récupéré tous les attributs et toutes les méthodes de la commande "Get-Process".

Get-ChildItem

Cette commande permet de lister tous les éléments contenus dans un répertoire (équivalent à la commande "ls" sous Linux ou "dir" sous Windows).

```
Windows PowerShell
Copyright (C) 2016 Microsoft Corporation. Tous droits réservés.

PS G:\Work\Supinfo\Articles\Powershell> Get-ChildItem


    Répertoire : G:\Work\Supinfo\Articles\Powershell


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----        05/03/2017     18:50                Dossier_Article_PowerShell
-ar---        05/03/2017     18:51              0 Archive_Article_PowerShell - Copie.txt
-a----        05/03/2017     18:52              0 Article.ps1
-a----        05/03/2017     18:52              0 Article.xml
```

Comme on peut le voir sur le screen ci-dessus, il y a une colonne « Mode » dans le résultat de la commande. Cette colonne indique le type du fichier courant.

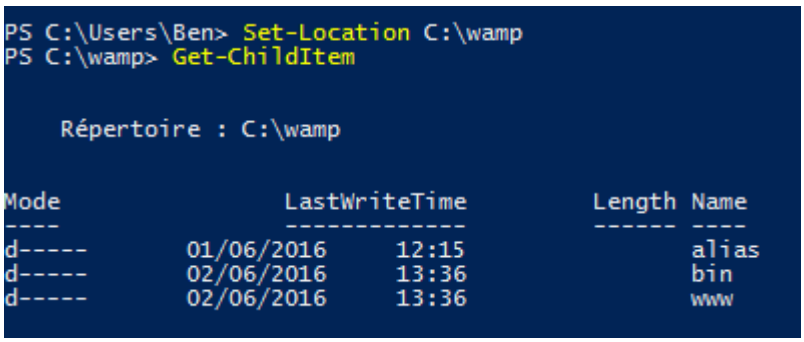
Voici les différents types :

- d : Répertoire
- a : Archive
- r : Elément en lecture seule
- h : Elément caché
- s : Elément système

Bien évidemment, les commandes possèdent plusieurs options pour des utilisations spécifiques. Etant donné que celles-ci sont assez nombreuses et extrêmement utiles, je vous invite à aller vous renseigner sur leurs fonctionnements.

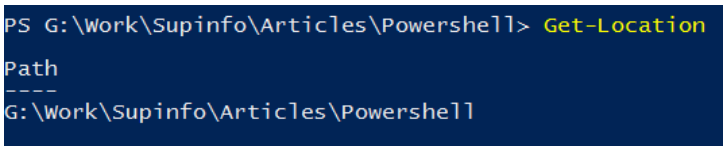
Set-Location<Path>

Cette commande permet de changer de répertoire (équivalent à la commande « cd »). Pour ce faire, il suffit de mettre le chemin que vous voulez atteindre en paramètre de la commande, comme par exemple : « Set-Location C:\wamp »



Get-Location

Affiche le répertoire courant :



Get-Service & Get-Process

Ces deux commandes permettent respectivement d'afficher la liste de tous les services Windows et la liste de tous les processus Windows.

1. Get-Service :

```
PS C:\wamp> Get-Service

Status   Name               DisplayName
------   ----               -----------
Running  AdobeARMservice    Adobe Acrobat Update Service
Running  AGSService         Adobe Genuine Software Integrity Se...
Stopped  AJRouter           Service de routeur AllJoyn
Stopped  ALG                Service de la passerelle de la couc...
Running  AlienFusionService Alienware Fusion Service
Stopped  AppIDSvc           Identité de l'application
Running  Appinfo            Informations d'application
Running  Apple Mobile De... Apple Mobile Device Service
Stopped  AppMgmt            Gestion d'applications
Stopped  AppReadiness       Préparation des applications
Stopped  AppVClient         Microsoft App-V Client
Stopped  AppXSvc            Service de déploiement AppX (AppXSVC)
Stopped  aspnet_state       ASP.NET State Service
Running  AtherosSvc         AtherosSvc
Running  AudioEndpointBu... Générateur de points de terminaison...
Running  Audiosrv           Audio Windows
Stopped  AxInstSV           Programme d'installation ActiveX (A...
Stopped  BDESVC             Service de chiffrement de lecteur B...
Running  BFE                Moteur de filtrage de base
Stopped  BITS               Service de transfert intelligent en...
```

Pour récupérer un service en particulier, il nous est possible de spécifier son nom lors de l'appel de la commande :

```
PS C:\wamp> Get-Service Dnscache

Status   Name               DisplayName
------   ----               -----------
Running  Dnscache           Client DNS
```

On peut également démarrer ou arrêter un service en utilisant respectivement les commandes suivantes :

```
Start-Service -Name "NomDuService"
Stop-Service -Name "NomDuService"
```

2. Get-Process

```
PS C:\wamp> Get-Process

Handles  NPM(K)    PM(K)      WS(K)     CPU(s)      Id  SI ProcessName
-------  ------    -----      -----     ------      --  -- -----------
    137      11     1648       8064       0,06   14108   6 acrotray
    287       9     5616       5724              2920   0 AdminService
    261      15     3332       9880              2196   0 AGSService
    195      17     9700      17572       0,06    1840   6 AlienFusionController
    288      27    25080      19384              5756   0 AlienFusionService
    493      36    29428      47420       0,81   15476   6 AlienwareAlienFXController
    392      32    27944      37812       2,50    6928   6 AlienwareTactXMacroController
    194      19     3312       3596              2912   0 AppleMobileDeviceService
    316      18     9164      26040       1,36    7320   6 ApplicationFrameHost
    121       8     1316       1712              2860   0 armsvc
    253      14     3700       1096       0,06    7868   6 AsusSmartGestureDetector64
    283      18     3836       3880       0,16    9724   6 AsusTPCenter
     69       7     1212       1080       0,08   14476   6 AsusTPHelper
    210      12     2816        972       0,14    7296   6 AsusTPLoader
    184      11     7032      13388       0,25    1660   0 audiodg
    177      12     5452       9396       0,03    7272   6 AWCCApplicationWatcher32
    160      22     8876      10184       0,02    7948   6 AWCCApplicationWatcher64
    469      47    42344      51556       0,56    8584   6 AWCCServiceController
    358      44    98004     115472       7,25     608   6 chrome
    257      24    27168      34640       4,16    1204   6 chrome
    158      11     1836       9252       0,06    4084   6 chrome
```

Comme pour la commande Get-Service, il est possible ici de récupérer un processus en particulier lors de l'appel de la commande en spécifiant son nom.

New-Item

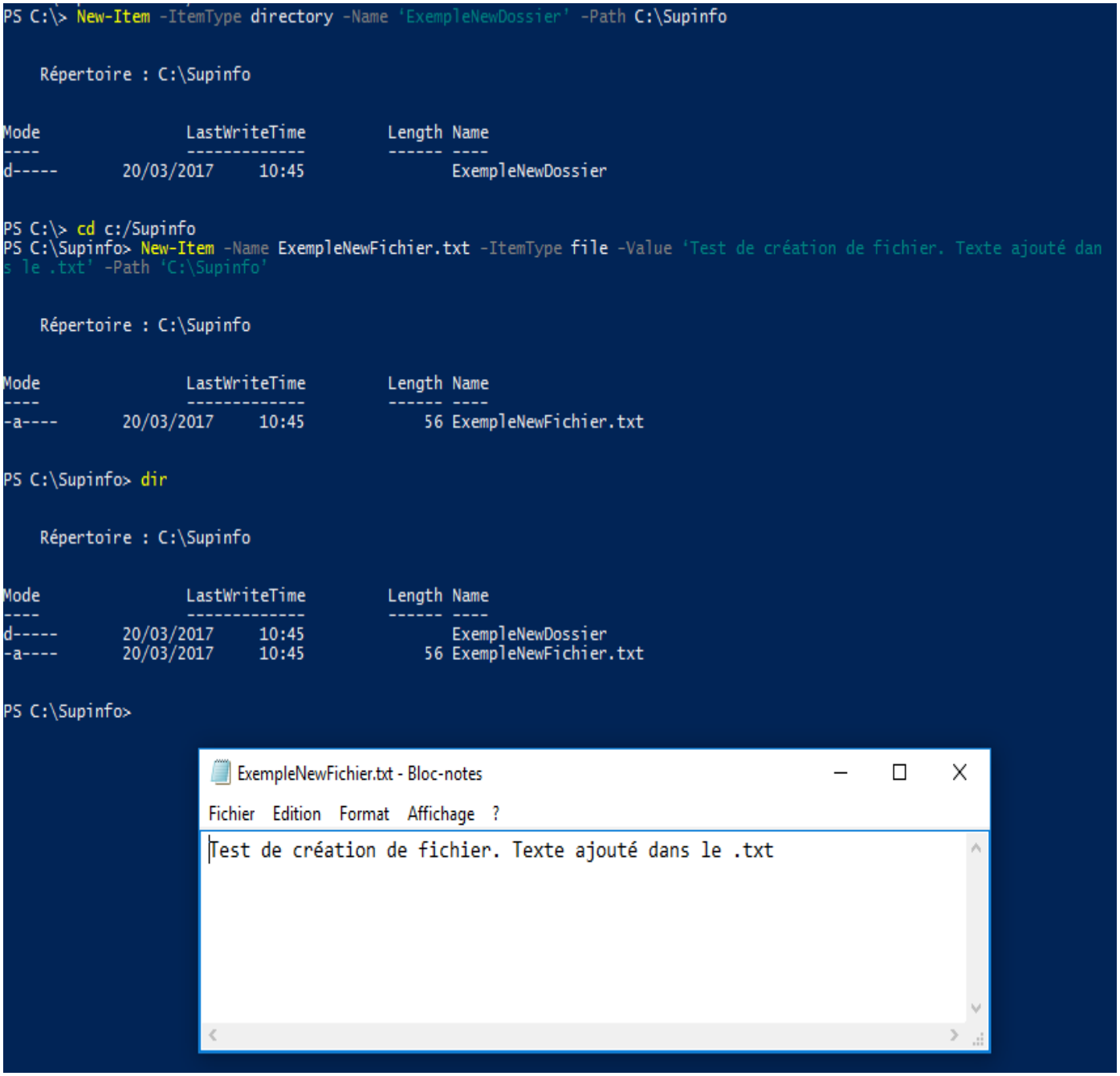
Permet de créer des fichiers et des dossiers.

- Création d'un dossier :

```
New-Item -ItemType directory -Name "ExempleNewDossier" -Path C:\Supinfo
```

- Création d'un fichier :

```
New-Item -Name ExempleNewFichier.txt -ItemType file -Value "Test de création de
fichier. Texte ajouté dans le .txt" -Path "C:\Supinfo"
```



Write-Host & Read-Host

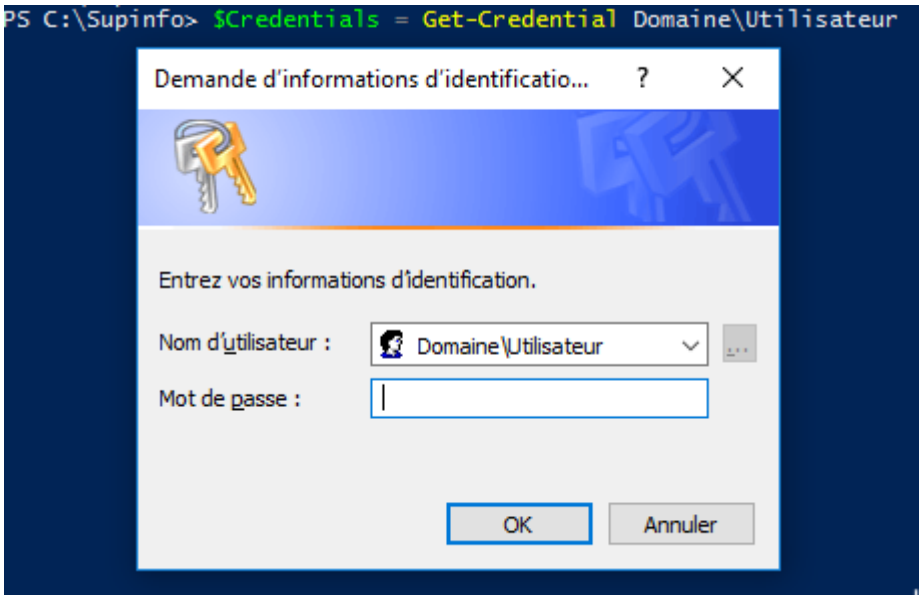
La commande Write-Host permet d'afficher quelque chose à l'écran (une string, le contenu d'une variable, son type, etc.) et la commande Read-Host permet à l'utilisateur de pouvoir entrer du texte. Pour bien comprendre, voici un exemple tout simple de l'utilisation de ces deux commandes :

```
Write-Host "Veuillez entrer le texte votre choix :"
$input = Read-Host #Cette ligne sert à assigner ce que l'utilisateur va écrire à une variable, ici nommée "input"
Write-Host "Vous avez écrit : " $input
```

```
Veuillez entrer le texte votre choix :
42 is everything.
Vous avez écrit :  42 is everything.
```

Get-Credential

Cette commande est extrêmement utile lorsqu'on a besoin de s'identifier à quelque chose par le biais de PowerShell.



Dans cet exemple, le nom d'utilisateur et le mot de passe sont enregistrés dans la variables "$Credentials" et sont accessibles via les attributs "$Credentials.Password" et "$Credentials.UserName" (le mot de passe qui est stocké dans "$Credentials.Password" est une Secure String). Comme vu ci-dessus, nous pouvons mettre un nom d'utilisateur de base par le biais d'une string ou d'une variable placée dans l'appel de la commande.

Send-MailMessage

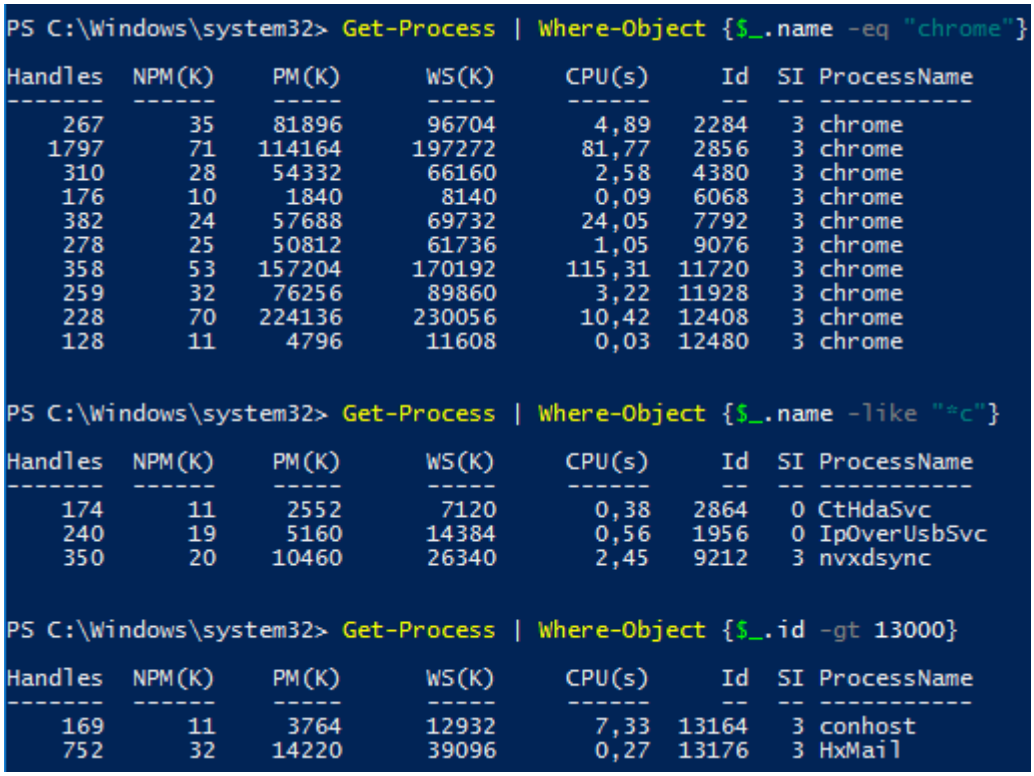
Indispensable si vous voulez implémenter une fonctionnalité d'envoi de mail dans un programme.

```
send-mailmessage -to "User01 <adresseDestinataire@gmail.com>" -from "User02
<adresseExpediteur@gmail.com>" -subject "Mail test" -smtpServer smtp.serveur.com
```

L'exemple ci-dessus précise seulement l'expéditeur, le destinataire et l'objet. En plus de ça, il est possible d'ajouter un corps au message, une pièce jointe, un accusé de réception ou encore d'envoyer le message à un groupe de destinataires.

Where-Object

Where-Object sert à filtrer les données retournées par une commande en précisant le filtre que nous voulons appliquer. Voici un exemple avec la commande Get-Process :



Dans tous ces exemples, le signe "$_" défini l'objet courant récupéré par la commande Get-Process (nous y reviendrons dans le chapitre sur le pipeline). Dans le premier exemple, le trie se fait par nom, et vérifie si l'attribut "name" de la commande Get-Process contient la string "chrome". Les résultats obtenus correspondent donc à tous les processus dont le nom est "chrome". Dans le deuxième, nous cherchons tous les processus finissant par la lettre "c" et dans le troisième, seuls les processus avec un Id supérieur à 13000 sont affichés. Les filtres peuvent se faire sur tous les attributs d'un objet.

Enter-PSSession

Cette commande permet de démarrer une session PowerShell sur un ordinateur distant. Durant cette session, toutes les commandes tapées seront exécutées sur l'ordinateur distant comme si elles étaient tapées directement sur celui-ci.

Syntaxe :

```
Enter-PSSession -computer ServerTest01 #Sert à démarrer la session distante
Exit-PSSession #Sert à stopper la session distante
```

Invoke-Command

Invoke-Command sert à lancer une commande ou un script sur plusieurs ordinateurs distants :

```
Invoke-Command -ComputerName ServerTest01, ServeurTest02 {Get-Process}
```

Le résultat de la commande sera affiché autant de fois qu'il y a de machines, avec le nom de celles-ci spécifié à chaque fois.

Dans le cas d'un Invoke-Command sur un fichier, le fichier qui se trouve dans le chemin spécifié sera exécuté sur tous les ordinateurs en même temps :

```
Invoke-Command -ComputerName ServerTest01, ServeurTest02 -FilePath
c:\Supinfo\TutoTest.ps1
```

Le pipeline : "|"

Le pipeline est une notion extrêmement importante du PowerShell qui va vous faire gagner pas mal de temps.

Un pipe est symbolisé par la caractère "|" (AlltGr + 6) et le principe de base de celui-ci est de "chaîner" la sortie d'une commande avec l'entrée d'une autre commande.

Par exemple, en reprenant un exemple vu plus haut : si nous voulions récupérer uniquement les processus Windows qui se nomment "chrome", la méthode de base sans pipe serait la suivante :

1. Il faudrait récupérer la liste de tous les processus et les incorporer dans une variable :

```
$allProcess = Get-Process
```

2. Ensuite parcourir chaque itération de la liste pour trouver les processus recherchés :

```
3. $allProcess = Get-Process #Variable contenant tous les processus
4. $filterProcess = @()
5. foreach ($process in $allProcess) #Parcours de $allProcess par le biais d'un
   foreach
6. {
7.    if ($process.name -eq "chrome") #Test pour savoir si le nom du process courant
   est égal à "chrome"
8.       {
9.           $filterProcess += $process.Id #Si il est égal, on met l'Id du process
   dans la liste $filterProcess
10.       }
11.  }
   Write-Host $filterProcess
```

Alors que pour faire ça avec les pipeline, une seule ligne suffit :

```
Get-Process | Where-Object {$_.name -eq "chrome"} | Select -Property Id
```

Dans la première partie, on récupère la liste de tous les processus. Cette liste est ensuite passée vers la commande Where-Object, par le biais du pipe. Celle-ci va se charger de retourner toutes les occurences dont le nom correspond à "chrome" ("$_" est ici la variable de parcours, le processus courant dans notre cas) et cette liste triée va ensuite être dirigée vers la commande "Select" qui va nous permettre de récupérer uniquement la propriété "Id".

Les pipelines sont assez difficiles à appréhender au début mais deviennent vite indispensables quand on maitrise leur fonctionnement.

Les fonctions

Comme dans tous les langages, les fonctions sont des blocs de code nommés qu'on peut appeler à volonté, en leurs passant ou non des paramètres.

Voici la syntaxe des fonctions PowerShell :

- Fonction sans paramètres :

```
function NomDeLaFonction
{
    #Bloc de code de la fonction
}
```

- Fonction avec paramètres :

```
function NomDeLaFonction ($parametre1, $parametre2)
{
    #Bloc de code de la fonction
    #Les paramètres peuvent être directement utilisés ici :
    $resultat = $parametre1 + $parametre2
    Write-Host $resultat
}
```

Pour appeler une fonction il suffit simplement d'exécuter l'une des deux lignes suivantes :

```
NomDeLaFonction() #Appel d'une fonction sans paramètres
NomDeLaFonction($paramètre1, $paramètre2) #Appel d'une fonction avec paramètres
```

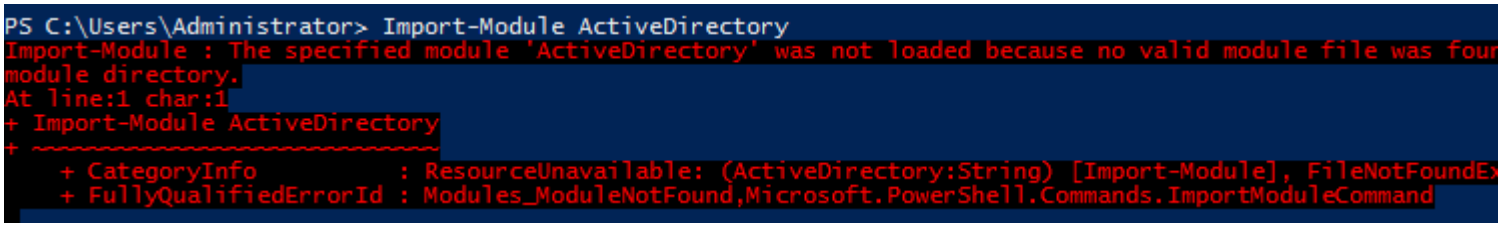
Quelques commandes d'administration

Active Directory

Il est possible d'administrer un Active Directory avec PowerShell. Pour utiliser les commandes propres à Active Directory, nous devons d'abord importer le module correspondant :

```
Import-Module ActiveDirectory
```

Il se peut que vous receviez un message de ce type lors de l'importation du module :



Cela peut venir du fait que les RSAT (Remote Server Administration Tools) ne sont pas installés. Pour ce faire, il suffit de taper ces commandes :

```
Import-Module ServerManager
Add-WindowsFeature RSAT-AD-PowerShell
```

Et ensuite vous devriez pouvoir importer le module Active Directory.

Pour récupérer la liste de l'ensemble des commandes PowerShell liées à Active Directory, il suffit d'utiliser cette commande :

```
Get-Command –Module ActiveDirectory
```

Voici une liste des commandes les plus utilisées pour administrer un AD :

- Obtenir la liste de tous les comptes :

  ```
  Get-ADUser –Filter * -Properties *
  ```

  Le paramètre filter sert à filtrer les résultats et le paramètre "Properties" sert à sélectionner les propriétés qu'on veut récupérer. Le symbole "*" signifie qu'on veut tous récupérer.

  Avec cette commande, on aura donc la liste de toutes les propriétés de tout les comptes. Bien sûr, il est conseillé d'utiliser les filtres pour récupérer juste le nécessaire.

- Changer le mot de passe d'un compte :

  Pour changer un mot de passe, il faut d'abord convertir celui-ci en "secure string". Nous allons donc pour cet exemple demander à l'utilisateur d'entrer un nouveau mot de passe, convertir celui-ci en secure-string et l'enregistrer dans une nouvelle variable :

  ```
  $nouveauMDP = Read-Host "Veuillez entrer le nouveau mot de passe" -AsSecureString
  ```

  Nous avons donc maintenant le nouveau mot de passe crypté dans une variable. Il suffit ensuite de rentrer la commande suivante pour assigner le nouveau mot de passe à l'utilisateur spécifié :

  ```
  Set-ADAccountPassword bfevrat -NewPassword $nouveauMDP
  ```

  Dans ce cas, "bfevrat" est la propriété "SamAccountName" du compte AD.

- Déverrouiller un compte :

  Pour déverrouiller un compte AD locké, il faut utiliser la commande suivante :

  ```
  Unlock-ADAccount bfevrat
  ```

- Activer ou désactiver un compte :

  Au-dessus, nous avons vu comment déverrouiller un compte. La différence entre un compte verrouillé et un compte désactivé est la suivante : un compte se verrouille au bout de plusieurs tentatives de mot de passe échouées alors qu'un compte désactivé a été sciemment désactivé par l'administrateur (période d'absence, période de maladie, départ de l'employé...).

  Voici respectivement les deux commandes qui servent à activer et désactiver un compte AD :

  ```
  Enable-ADAccount bfevrat #Activer un compte
  Disable-ADAccount bfevrat #Desactiver un compte
  ```

- Supprimer un compte :

  Pour supprimer un compte de l'AD, il suffit d'utiliser cette commande :

  ```
  Remove-ADUser bfevrat
  ```

Exchange

Comme pour active directory, il faut tout d'abord importer les commandes propres à Exchange. La commande est différente entre chaque version d'Exchange :

- Exchange 2007 :

  ```
  Add-PSSnapin Microsoft.Exchange.Management.PowerShell.Admin
  ```

- Exchange 2010 :

  ```
  Add-PSSnapin Microsoft.Exchange.Management.PowerShell.E2010
  ```

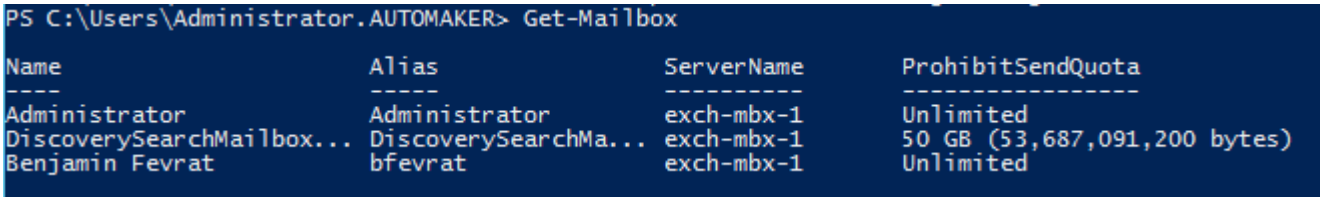
- Exchange 2013 et 2016 :

  ```
  Add-PSSnapin Microsoft.Exchange.Management.PowerShell.SnapIn
  ```

Voici une liste des commandes les plus utiles pour administrer un Exchange :

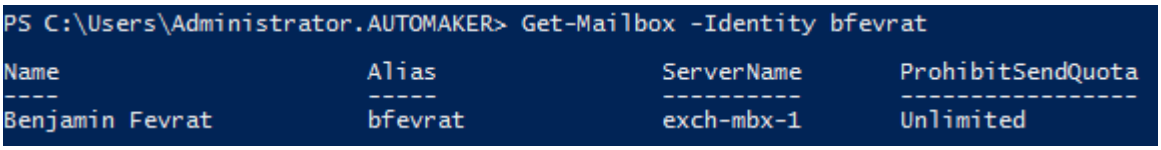
- Lister l'ensemble des boites mail Exchange :

  ```
  Get-Mailbox
  ```

- Récupérer une boite mail en particulier :

  ```
  Get-Mailbox -Identity bfevrat
  ```

- Créer une nouvelle boite mail :

  Pour créer une nouvelle mailbox, nous avons besoin de passer le mot de passe en secure string. Pour cela, nous utiliserons la même méthode que dans le chapitre Active Directory :

  ```
  $nouveauMDP = Read-Host "Veuillez entrer le nouveau mot de passe" -AsSecureString
  ```

  Après la récupération des données que l'utilisateur aura entré, voici la commande utilisée pour créer une nouvelle boite mail :

  ```
  New-Mailbox -UserPrincipalName Test@supinfo.com -Alias testsup -Name TestSupinfo -
  OrganizationalUnit Users -Password $nouveauMDP -Firstname John -LastName Doe -
  DisplayName "John Doe"
  ```

- Supprimer une boite mail :

Lorsqu'une boite mail est supprimée avec la ligne de code suivante :

```
Remove-Mailbox -Identity "bfevrat"
```

celle-ci ne sera pas supprimée immédiatement. En effet elle va rester dans la base de données durant la période de rétention de boite aux lettres qui est configurée pour cette base de donnée (propriété "MailboxRetention"). Si nous voulons que la boite soit supprimée instantanément, il faut écrire la commande suivante :

```
Remove-Mailbox -Identity "bfevrat" -Permanent $true
```

Avec cette commande, la boite mail est supprimée immédiatement de la base de données.

ATTENTION : la suppression d'une boite mail entraîne la suppression de l'utilisateur Active Directory lié à cette boite mail. Pour éviter que l'utilisateur soit supprimé, il faut désactiver la boite mail, et non pas la supprimer.

- Activer ou désactiver une boite mail :

  Comme dit auparavant, l'activation ou la désactivation d'une boite mail n'influera que sur celle-ci, et non sur l'utilisateur en lui-même et comme pour la suppression, la mailbox sera soumise à la période de rétention précédemment configurée.

  Voici la commande utilisée pour désactiver une boite mail :

  ```
  Disable-Mailbox <identity>
  ```

  Le paramètre "identity" peut être spécifié sous différentes formes :

  ```
  Disable-Mailbox bfevrat # Avec le login
  Disable-Mailbox "Salle de conference 1" # Avec une string contenant le nom
  Disable-Mailbox benjamin.fevrat@supinfo.com # Avec une adresse mail
  ```

  Voici la commande utilisée pour activer un compte désactivé :

  ```
  Enable-Mailbox -Identity "bfevrat"
  ```

- Exporter une boite mail :

  Le contenu d'une boite mail Exchange peut être exporté vers un fichier ".pst". Ce fichier contiendra tous les messages, le calendrier, les contacts, les listes de distribution, les tâches, les notes ainsi que les documents de l'utilisateur.

  Pour exporter une boite mail vers un fichier ".pst", il suffit d'utiliser cette commande en spécifiant le chemin de destination et l'utilisateur :

  ```
  Export-Mailbox -Identity "bfevrat" -PSTFolderPath C:\Pst\Ben.pst
  ```

- Importer une boite mail :

On peut importer simplement une boite mail (fichier ".pst") en l'attribuant à un utilisateur avec la commande suivante :

```
Import-Mailbox -Identity "bfevrat" -PSTFolderPath C:\Pst\Ben.pst
```

Conclusion

Cet article présente les bases de PowerShell et permet de découvrir les nombreuses possibilités offertes par ce langage. La connaissance de PowerShell est chaudement recommandée pour tout administrateur système qui souhaite automatiser et simplifier l'administration de ses serveurs et de ses postes clients. Encore une fois, ici sont exposées quelques bases de PowerShell qui sont destinées à présenter dans les grandes lignes le champ d'action et la puissance de ce langage, donc si vous êtes intéressés, n'hésitez pas à vous renseigner davantage sur le sujet.