

# Tutoriel de Windows PowerShell Scripting pour débutants

!

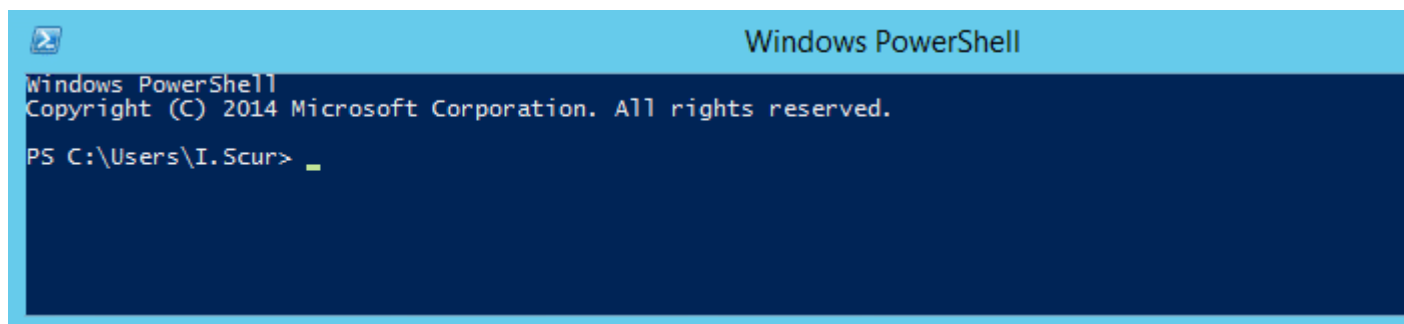
## Faites connaissance avec le tutoriel PowerShell

Windows PowerShell est un moteur d'automatisation orienté objet ainsi qu'un langage de script. Doté d'un interpréteur de commandes interactif, il est conçu pour aider les professionnels de l'informatique à configurer les systèmes et automatiser les tâches d'administration. Il est intégré à tous les systèmes d'exploitation Windows modernes depuis Windows 2008R2. L'apprentissage de Windows PowerShell peut-être comparé à l'apprentissage d'un outil polyvalent. Dans le présent billet, j'aborde les bases des scripts PowerShell dans le but de vous permettre d'effectuer plus facilement la plupart des tâches d'administration liées à votre environnement informatique Windows.

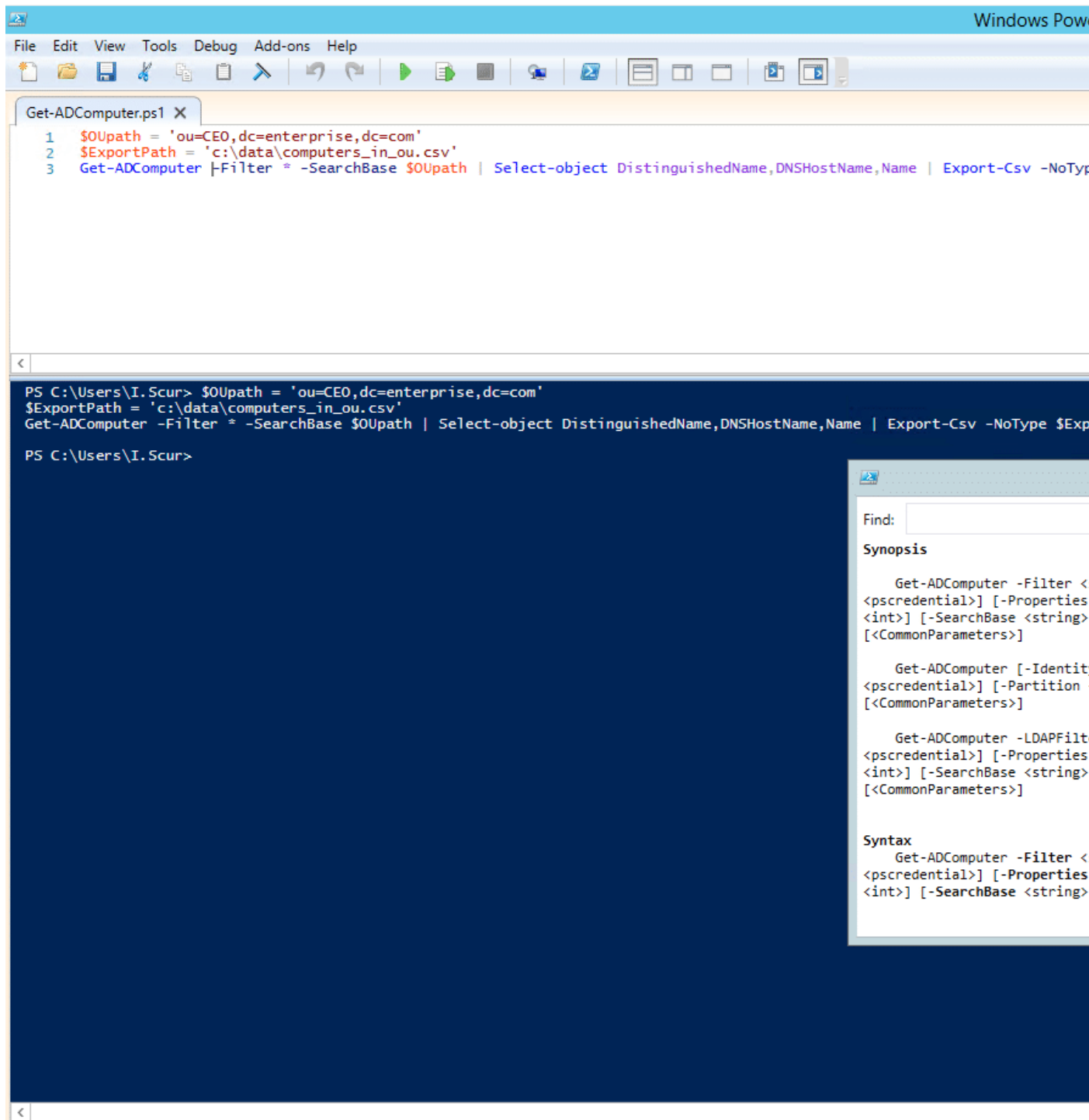
Commençons à apprendre PowerShell.

PowerShell comprend une option de ligne de commande et un environnement d'écriture de scripts intégré (ISE) :

- Pour ouvrir la ligne de commande PowerShell, tapez *exe* dans le menu Démarrer de Windows. Un écran comme celui-ci s'affiche :



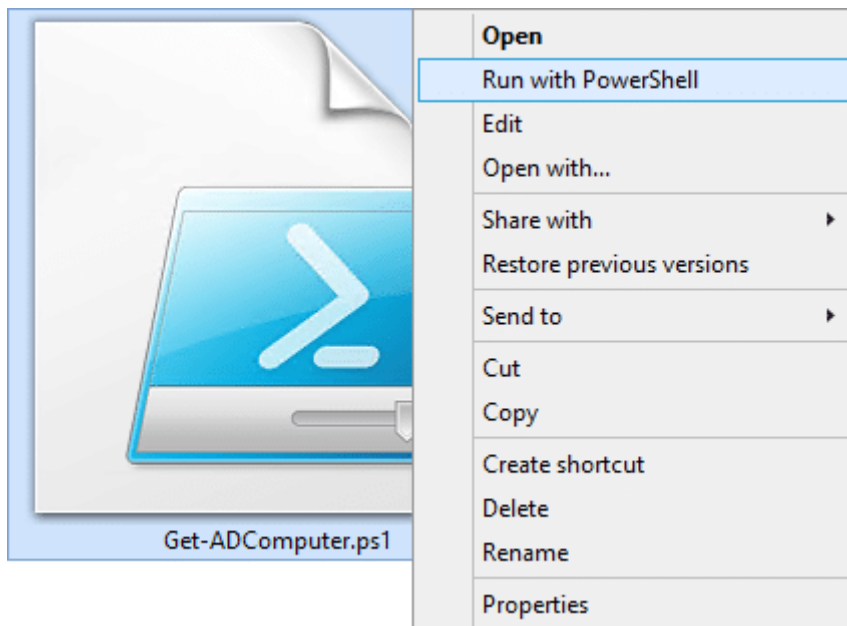
- Pour l'ISE PowerShell, tapez *powershell\_ise.exe* dans le menu Démarrer. L'utilisation de l'ISE PowerShell est la meilleure façon de travailler avec le langage de script, car cet environnement offre une surbrillance syntaxique, un remplissage automatique des commandes et d'autres fonctions d'automatisation qui simplifient la rédaction et le test des scripts.



Contenus connexes sélectionnés :

## Préparation à l'exécution de scripts PowerShell

Les scripts PowerShell sont stockés dans des fichiers .ps1. Vous ne pouvez pas exécuter un script en double-cliquant simplement sur son fichier, ceci afin d'éviter d'endommager accidentellement vos systèmes. Au lieu de cela, cliquez avec le bouton droit de la souris sur le fichier et sélectionnez « Exécuter avec PowerShell » :



De plus, une politique restreint l'exécution des scripts. Vous pouvez consulter cette politique en exécutant la commande [Get-ExecutionPolicy](#) dans PowerShell :

```
Windows PowerShell
Copyright (C) 2014 Microsoft Corporation. All rights reserved.

PS C:\Users\I.Scur> Get-ExecutionPolicy
Unrestricted
PS C:\Users\I.Scur> _
```

Vous obtiendrez l'une des valeurs suivantes :

- **Restricted** — Aucun script n'est autorisé. Il s'agit du paramètre par défaut, que vous verrez donc lors de votre première exécution de la commande.
- **AllSigned** — Vous pouvez exécuter les scripts signés par un développeur de confiance. Ce paramétrage vous demandera, avant l'exécution d'un script, de confirmer que vous souhaitez bien l'exécuter.
- **RemoteSigned** — Vous pouvez exécuter vos propres scripts ou les scripts signés par un développeur de confiance.
- **Unrestricted** — Vous pouvez exécuter tous les scripts que vous voulez.

Pour commencer à travailler avec PowerShell, changez le paramétrage de cette politique en la passant de Restricted à RemoteSigned, à l'aide de la commande [Set-ExecutionPolicy RemoteSigned](#) :

```
PS C:\Users\I.Scur> Set-ExecutionPolicy RemoteSigned

Execution Policy Change
The execution policy helps protect you from scripts that you do not trust. Changing the execution policy might expose
you to the security risks described in the about_Execution_Policies help topic at
http://go.microsoft.com/fwlink/?LinkID=135170. Do you want to change the execution policy?
[Y] Yes [N] No [S] Suspend [?] Help (default is "Y"): y
```

# PowerShell Cmdlets

## À propos des Cmdlets

Un cmdlet est une commande PowerShell qui a une fonction prédéfinie, comme un opérateur dans un langage de programmation. Voici quelques informations importantes concernant les cmdlets :

- Il existe des cmdlets système, utilisateur et personnalisés.
- Les cmdlets fournissent des résultats sous forme d'objet ou de tableau d'objets.
- Les cmdlets peuvent obtenir des données à analyser ou transférer des données vers un autre cmdlet par le biais de canaux (je reviendrai sur ces canaux – ou *pipes* – dans un instant).
- Les cmdlets sont « insensibles à la casse ». C'est-à-dire que les majuscules et minuscules n'ont aucune importance, vous pouvez tout aussi bien taper « Get-ADUser », « get-aduser » ou « gEt-AdUsEr ».
- Si vous voulez utiliser plusieurs cmdlets dans une même chaîne, vous devez les séparer par un point-virgule (;).

## Format de cmdlet

Un cmdlet se compose toujours d'un verbe (ou d'un mot qui fait office de verbe) et d'un nom, séparés par un trait d'union (c'est la règle « verbe-nom »). Voici quelques exemples de verbes :

- **Get** — pour obtenir quelque chose
- **Set** — pour définir quelque chose
- **Start** — pour exécuter quelque chose
- **Stop** — pour arrêter quelque chose en cours d'exécution
- **Out** — pour générer quelque chose
- **New** — pour créer quelque chose (« new » n'est pas un verbe, mais il fonctionne comme un verbe)

Pour vous exercer, essayez d'exécuter les cmdlets suivants :

- [Get-Process](#) — affiche les processus en cours d'exécution sur votre ordinateur :

```
PS C:\Users\I.Scur> Get-Process
```

Handles	NPM(K)	PM(K)	WS(K)	VM(M)	CPU(s)	Id	ProcessName
-----	-----	-----	-----	-----	-----	--	-----
865	59	79260	75612	662		2620	ALESERVICE
206	33	38272	71784	219	1,602.80	8768	AuditIntelligence
367	55	59460	109840	347	329.28	8808	AuditIntelligence
48	5	712	3076	30		2004	conhost
49	5	704	3108	30		2388	conhost
516	15	2396	4488	52		348	csrss
94	8	1184	3480	43		424	csrss
234	12	1800	40952	82		3164	csrss
198	13	3376	10812	49		4900	dllhost
181	14	14908	25212	91		724	dwm
201	21	12996	69268	158		8728	dwm
1457	89	84340	144068	577	42.38	8428	explorer

- [Get-Service](#) — affiche la liste des services et leur état
- [Get-Content](#) — affiche le contenu du fichier spécifié (par exemple, [Get-Content C:\Windows\System32\drivers\etc\hosts](#))

## Cmdlets disponibles

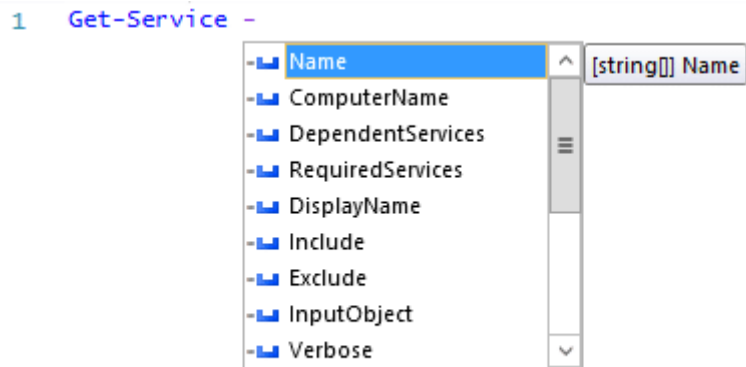
Bonne nouvelle : vous n'avez pas à mémoriser tous les cmdlets. Vous pouvez afficher la liste de tous les cmdlets en exécutant le cmdlet [Get-Help-Category](#), qui renvoie ce qui suit :

Get-Command	Cmdlet	Microsoft.PowerShell.Core ...
Export-ModuleMember	Cmdlet	Microsoft.PowerShell.Core ...
Get-Module	Cmdlet	Microsoft.PowerShell.Core ...
Import-Module	Cmdlet	Microsoft.PowerShell.Core ...
New-Module	Cmdlet	Microsoft.PowerShell.Core ...
New-ModuleManifest	Cmdlet	Microsoft.PowerShell.Core ...
Remove-Module	Cmdlet	Microsoft.PowerShell.Core ...
Test-ModuleManifest	Cmdlet	Microsoft.PowerShell.Core ...
Get-Help	Cmdlet	Microsoft.PowerShell.Core ...
Update-Help	Cmdlet	Microsoft.PowerShell.Core ...
Save-Help	Cmdlet	Microsoft.PowerShell.Core ...
Get-History	Cmdlet	Microsoft.PowerShell.Core ...
Invoke-History	Cmdlet	Microsoft.PowerShell.Core ...
Add-History	Cmdlet	Microsoft.PowerShell.Core ...
Clear-History	Cmdlet	Microsoft.PowerShell.Core ...
Register-PSSessionConfiguration	Cmdlet	Microsoft.PowerShell.Core ...
Unregister-PSSessionConfiguration	Cmdlet	Microsoft.PowerShell.Core ...
Get-PSSessionConfiguration	Cmdlet	Microsoft.PowerShell.Core ...
Set-PSSessionConfiguration	Cmdlet	Microsoft.PowerShell.Core ...
Enable-PSSessionConfiguration	Cmdlet	Microsoft.PowerShell.Core ...
Disable-PSSessionConfiguration	Cmdlet	Microsoft.PowerShell.Core ...
Enable-PSRemoting	Cmdlet	Microsoft.PowerShell.Core ...
Disable-PSRemoting	Cmdlet	Microsoft.PowerShell.Core ...
Invoke-Command	Cmdlet	Microsoft.PowerShell.Core ...
New-PSSession	Cmdlet	Microsoft.PowerShell.Core ...
Disconnect-PSSession	Cmdlet	Microsoft.PowerShell.Core ...
Connect-PSSession	Cmdlet	Microsoft.PowerShell.Core ...
Receive-PSSession	Cmdlet	Microsoft.PowerShell.Core ...
Get-PSSession	Cmdlet	Microsoft.PowerShell.Core ...
Remove-PSSession	Cmdlet	Microsoft.PowerShell.Core ...
Start-Job	Cmdlet	Microsoft.PowerShell.Core ...
Get-Job	Cmdlet	Microsoft.PowerShell.Core ...
Receive-Job	Cmdlet	Microsoft.PowerShell.Core ...
Stop-Job	Cmdlet	Microsoft.PowerShell.Core ...
Wait-Job	Cmdlet	Microsoft.PowerShell.Core ...
Remove-Job	Cmdlet	Microsoft.PowerShell.Core ...
Suspend-Job	Cmdlet	Microsoft.PowerShell.Core ...
Resume-Job	Cmdlet	Microsoft.PowerShell.Core ...
Enter-PSSession	Cmdlet	Microsoft.PowerShell.Core ...
Exit-PSSession	Cmdlet	Microsoft.PowerShell.Core ...

Vous pouvez également créer vos propres cmdlets personnalisés.

## Paramètres

Chaque cmdlet est assorti de plusieurs paramètres qui permettent de personnaliser son action. Dès que vous avez tapé un cmdlet et un trait d'union (-), l'ISE PowerShell propose automatiquement tous les paramètres valides et leurs types:



Par exemple, le cmdlet suivant affiche tous les services dont le nom commence par « W » :

`Get-Service -Name W*`

Si vous oubliez les paramètres d'un cmdlet, utilisez un script comme ci-dessous, qui affiche les paramètres du cmdlet `Get-Process` :

[Get-Process](#) | [Get-Member](#)



```

1 Get-Process | Get-Member
2 # | sign is a pipe, allowing you to pass data from one cmdlet to another

```

```

PS C:\Users\I.Scur> Get-Process | Get-Member

TypeName: System.Diagnostics.Process

Name      MemberType Definition
----      -
Handles   AliasProperty Handles = Handlecount
Name       AliasProperty Name = ProcessName
NPM        AliasProperty NPM = NonpagedSystemMemorySize
PM         AliasProperty PM = PagedMemorySize
VM         AliasProperty VM = VirtualMemorySize
WS         AliasProperty WS = WorkingSet
Disposed   Event        System.EventHandler Disposed(System.Object, System.EventArgs)
ErrorDataReceived Event        System.Diagnostics.DataReceivedEventHandler ErrorDataReceived(Sys
OutputDataReceived Event        System.Diagnostics.DataReceivedEventHandler OutputDataReceived(Sys
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 System.String __NounName=Process
BasePriority Property    int BasePriority {get;}
Container  Property    System.ComponentModel.IContainer Container {get;}
EnableRaisingEvents Property    bool EnableRaisingEvents {get;set;}
ExitCode   Property    int ExitCode {get;}
ExitTime   Property    datetime ExitTime {get;}
Handle     Property    System.IntPtr Handle {get;}
HandleCount Property    int HandleCount {get;}
HasExited  Property    bool HasExited {get;}

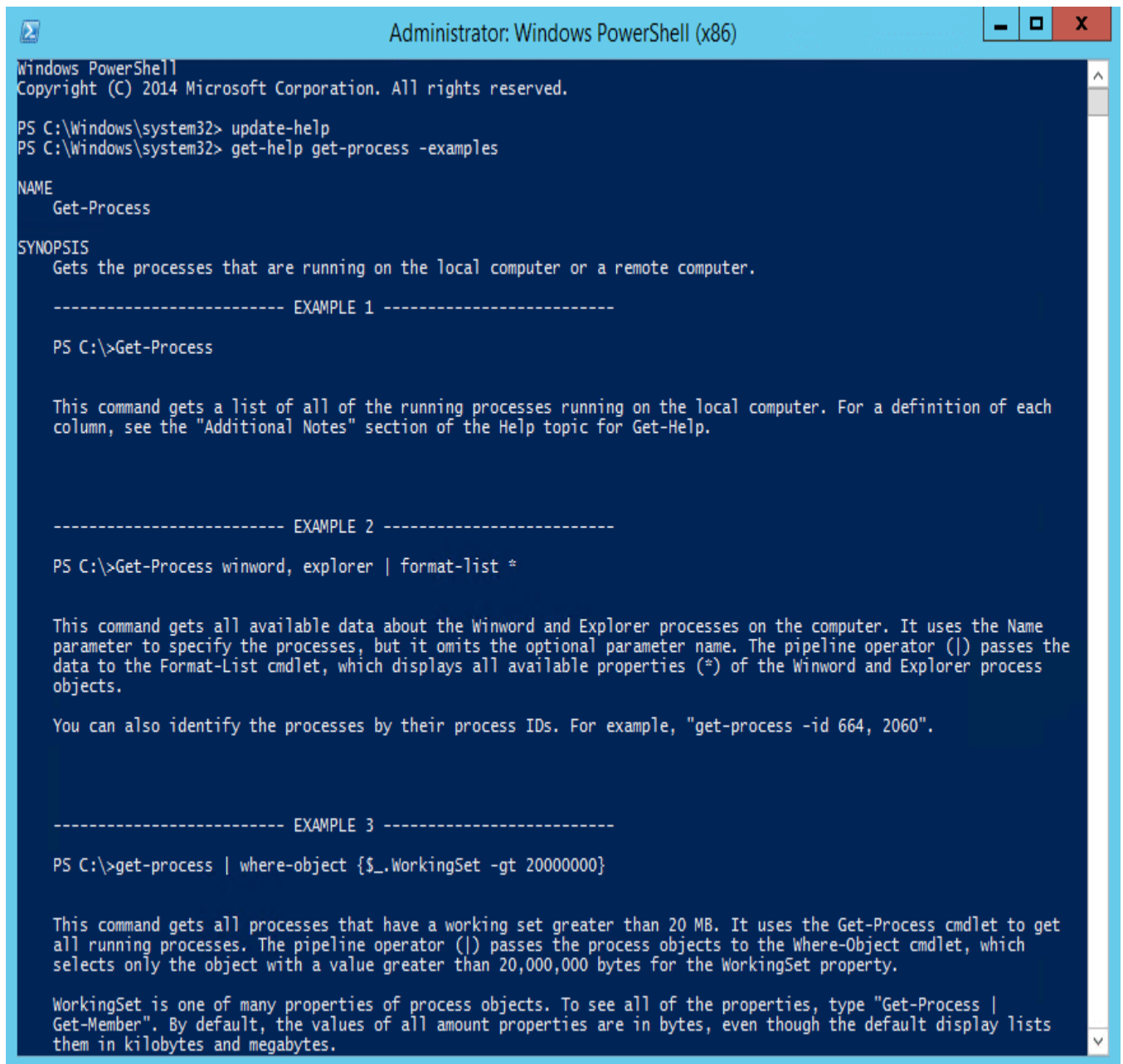
```

Si vous ne trouvez toujours pas le cmdlet dont vous avez besoin, vérifiez que votre aide est à jour et obtenez des exemples relatifs au cmdlet qui vous intéresse (par exemple Get-Process) en utilisant un script comme celui-ci :

```

Update-Help #to update the help data
Get-Help Get-Process -Examples

```



```
Administrator: Windows PowerShell (x86)
Windows PowerShell
Copyright (C) 2014 Microsoft Corporation. All rights reserved.

PS C:\Windows\system32> update-help
PS C:\Windows\system32> get-help get-process -examples

NAME
    Get-Process

SYNOPSIS
    Gets the processes that are running on the local computer or a remote computer.

    ----- EXAMPLE 1 -----

    PS C:\>Get-Process

    This command gets a list of all of the running processes running on the local computer. For a definition of each
    column, see the "Additional Notes" section of the Help topic for Get-Help.

    ----- EXAMPLE 2 -----

    PS C:\>Get-Process winword, explorer | format-list *

    This command gets all available data about the Winword and Explorer processes on the computer. It uses the Name
    parameter to specify the processes, but it omits the optional parameter name. The pipeline operator (|) passes the
    data to the Format-List cmdlet, which displays all available properties (*) of the Winword and Explorer process
    objects.

    You can also identify the processes by their process IDs. For example, "get-process -id 664, 2060".

    ----- EXAMPLE 3 -----

    PS C:\>get-process | where-object {$_.WorkingSet -gt 20000000}

    This command gets all processes that have a working set greater than 20 MB. It uses the Get-Process cmdlet to get
    all running processes. The pipeline operator (|) passes the process objects to the Where-Object cmdlet, which
    selects only the object with a value greater than 20,000,000 bytes for the WorkingSet property.

    WorkingSet is one of many properties of process objects. To see all of the properties, type "Get-Process |
    Get-Member". By default, the values of all amount properties are in bytes, even though the default display lists
    them in kilobytes and megabytes.
```

## Pseudonyms

Vous pouvez également utiliser des alias, qui sont des noms de cmdlet abrégés. Par exemple, au lieu de saisir « Get-Help », vous pouvez vous contenter de « Help ». Essayez d'exécuter les deux commandes suivantes et voyez si vous obtenez le même résultat :

- `Start-Process notepad`
- `start notepad`

De même, pour arrêter le processus, vous pouvez utiliser l'une des commandes suivantes :

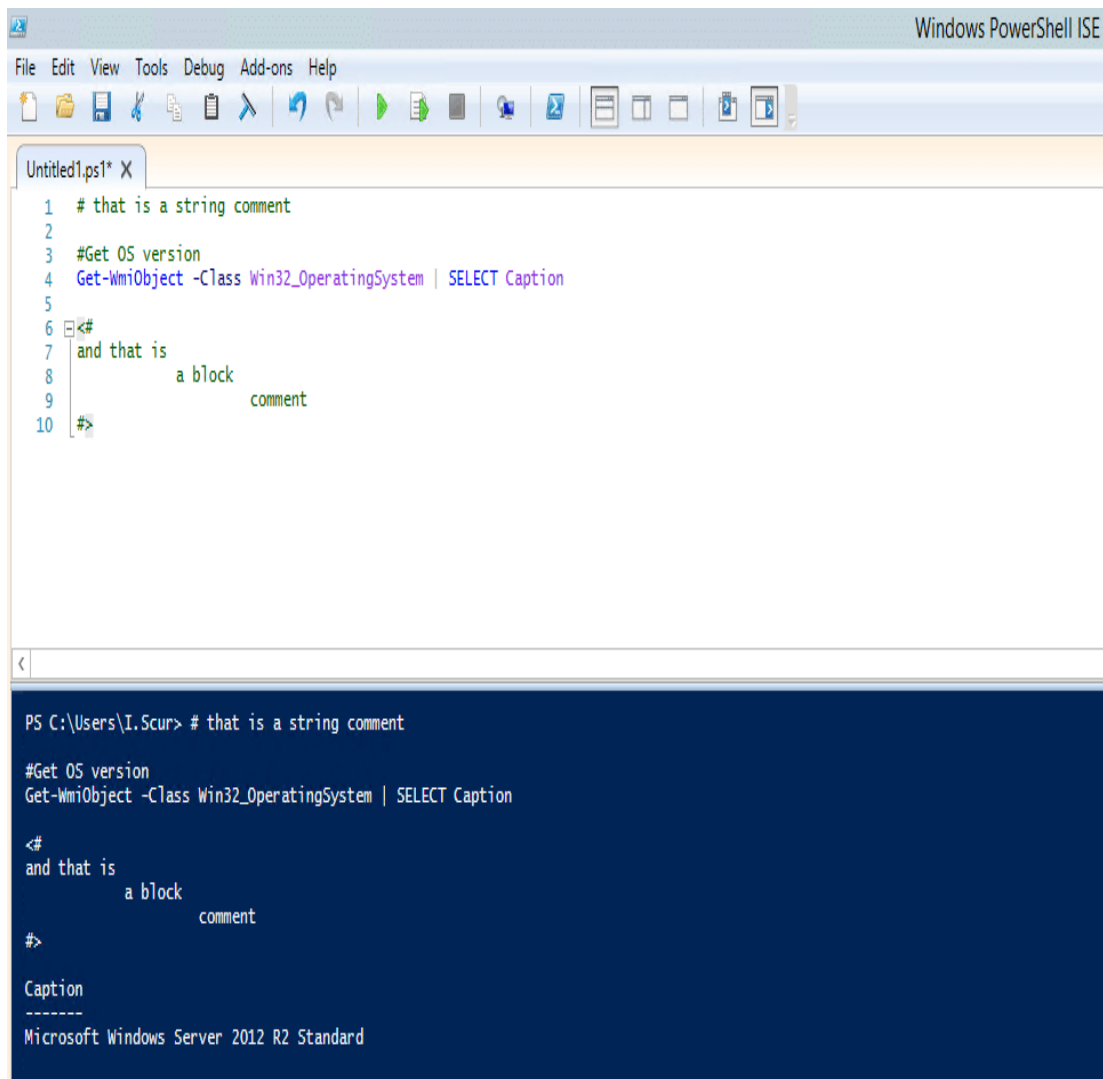
- `Stop-Process -Name notepad`
- `spps -Name notepad`



Pour afficher tous les alias, exécutez le cmdlet [Get-Alias](#).

## Commentaires

Le fait de laisser des commentaires dans un script vous aidera – ainsi que vos collègues – à mieux comprendre ce que fait le script concerné. Un commentaire de type chaîne est une ligne simple qui commence par le signe dièse (#), alors que les commentaires de type bloc commencent et finissent par des signes dièse et des crochets et occupent plusieurs lignes.



The screenshot shows the Windows PowerShell ISE interface. The top pane displays a script file named 'Untitled1.ps1' with the following content:

```
1 # that is a string comment
2
3 #Get OS version
4 Get-WmiObject -Class Win32_OperatingSystem | SELECT Caption
5
6 <#
7 and that is
8     a block
9     comment
10 #>
```

The bottom pane shows the output of the script execution:

```
PS C:\Users\I.Scur> # that is a string comment

#Get OS version
Get-WmiObject -Class Win32_OperatingSystem | SELECT Caption

<#
and that is
    a block
    comment
#>

Caption
-----
Microsoft Windows Server 2012 R2 Standard
```

## Tuyaux

Un canal permet de transférer des données d'un cmdlet à un autre. Précédemment, j'ai utilisé un canal pour obtenir toutes les propriétés d'un objet.

Si, par exemple, vous exécutez le script suivant, vous obtiendrez tous les services triés selon leur état :

```
Get-Service | Sort-Object -property Status
```

Vous pouvez également utiliser un canal pour envoyer du texte dans un fichier à l'aide d'un script comme le suivant :

```
"Hello, World!" | Out-File C:\ps\test.txt
```

Vous pouvez utiliser plusieurs canaux. Par exemple, le script suivant dresse la liste de tous les services, le premier canal excluant les services arrêtés et le second limitant la liste aux noms d'affichage :

```
Get-Service | WHERE {$_.status -eq "Running"} | SELECT displayname  
# "$_." defines current element in the pipe
```

## Résumé

Résumons rapidement les points clés de ce didacticiel sur Windows PowerShell. Vous savez à présent comment exécuter PowerShell, comment changer la politique d'exécution, ce qu'est un cmdlet, comment transmettre des données en utilisant un canal et comment obtenir les propriétés des objets. Gardez à l'esprit que si vous oubliez quelque chose, vous pouvez toujours utiliser le cmdlet Get-Help.

J'espère que vous avez trouvé utile ce didacticiel PowerShell pour débutants !

Dans les parties suivantes de ce didacticiel gratuit consacré à PowerShell, j'aborderai les variables, les tableaux et les cycles, qui sont utilisés dans les scripts PowerShell pour simplifier l'administration des serveurs Windows.

En attendant, vous pouvez consulter les scripts PowerShell suivants, qui permettent aux professionnels de l'informatique d'effectuer des tâches spécifiques :

Si vous venez de commencer d'apprendre PowerShell, nous vous conseillons de jeter un coup d'œil sur les ressources suivantes :