PowerShell

Table des matières

1. À propos de PowerShell

PowerShell représente la dernière génération des langages de scripts de Microsoft et est complémentaire à l'utilisation des interfaces graphiques pour les tâches d'administration.

L'interface graphique a tout son sens sur les postes de travail mais beaucoup moins sur les serveurs. C'est la raison pour laquelle il existe également une version de Windows Server sans interface graphique, Windows Server en mode Core.

PowerShell est maintenant open source et multiplateforme pour administrer des OS que ceux de Microsoft. Il s'agit de Powershell Core.

2. Des Scripts pour faire quoi ?

Les administrateurs système utilisent des scripts pour automatiser les tâches fastidieuses Cela permet de gagner du temps, temps qui pourra être consacré à d'autres choses.

Actuellement la place de l'automatisation est importante dans les solution de Cloud car sans automatisation, il n'y aurait tout simplement point de Cloud.

3. Rappel sur l'utilisation de à PowerShell

PowerShell est à la fois :

- un interpréteur de commandes
- et un langage de scripts.

Il tire sa puissance en grande partie grâce au Framework .NET bien connu des développeurs, mais beaucoup moins des administrateurs système et autres développeurs de scripts.

Le Framework .NET est une immense bibliothèque de classes à partir desquelles il est possible de créer des objets pour agir sur l'ensemble du système d'exploitation en un minimum d'effort.

Avec PowerShell vous ne manipulerez donc plus uniquement du texte, comme c'est le cas avec la plupart des autres shells, mais des **objets**, sans vraiment vous en rendre compte.

Exemple : lorsque vous utiliserez le pipe « | » pour passer des données à une commande, ce n'est du texte qui est transmis mais un objet avec tout ce qui le caractérise (ses propriétés et ses méthodes). Cela permet de réaliser des scripts PowerShell beaucoup plus concis que les autres langages.

PowerShell est fourni avec un jeu de commandes extrêmement riche qui ne cesse de croître à chaque nouvelle version. Les commandes CMD restent néanmoins toujours utilisables à partir de PowerShell, si besoin est.

3.1 Les versions de PowerShell

2006 : première version de PowerShell.

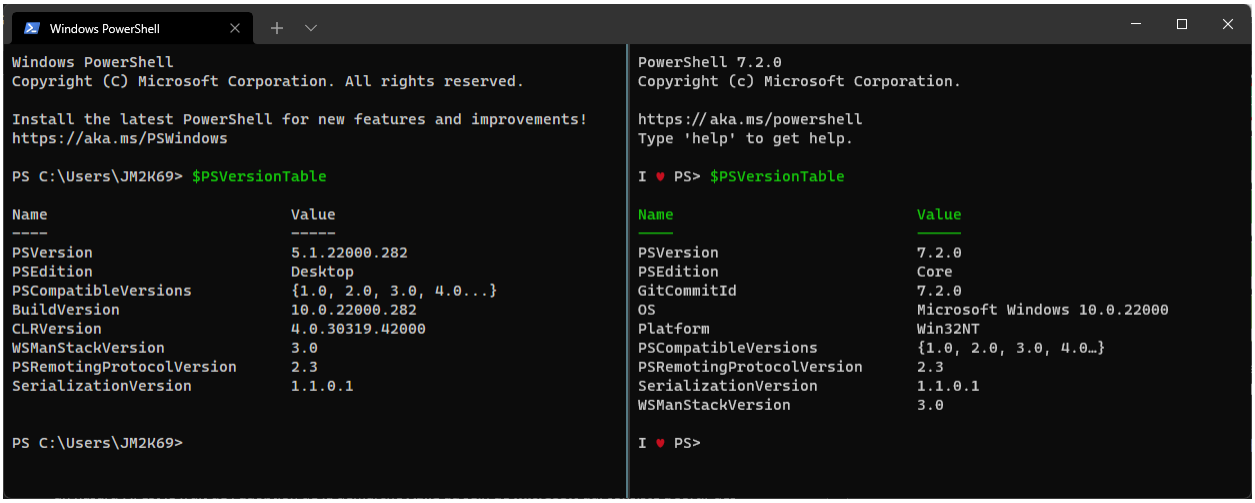
2016 : première sortie de PowerShell Core la première version Open Source et la dernière version de Windows PowerShell 5.1 qui la dernière version aujourd'hui même dans la dernière version de Windows Server 2022.

3.2 Différences entre Windows PowerShell et PowerShell Core

La principale différence est que PowerShell Core est multiplateforme, contrairement à Windows PowerShell qui ne fonctionne que sous Windows. On peut donc faire tourner PowerShell Core sur Mac OS ainsi que sur les principales distributions Linux telles que RedHat, Suse, Debian, etc.

PowerShell Core 6.0 ne couvre pas 100% de la surface fonctionnelle de Windows PowerShell 5.1 mais apporte d'autres fonctionnalités qui ne seront pas portées dans Windows PowerShell

Pour connaître la version disponible sur son ordinateur, il suffit d'afficher la variable d'environnement $PSVersionTable.



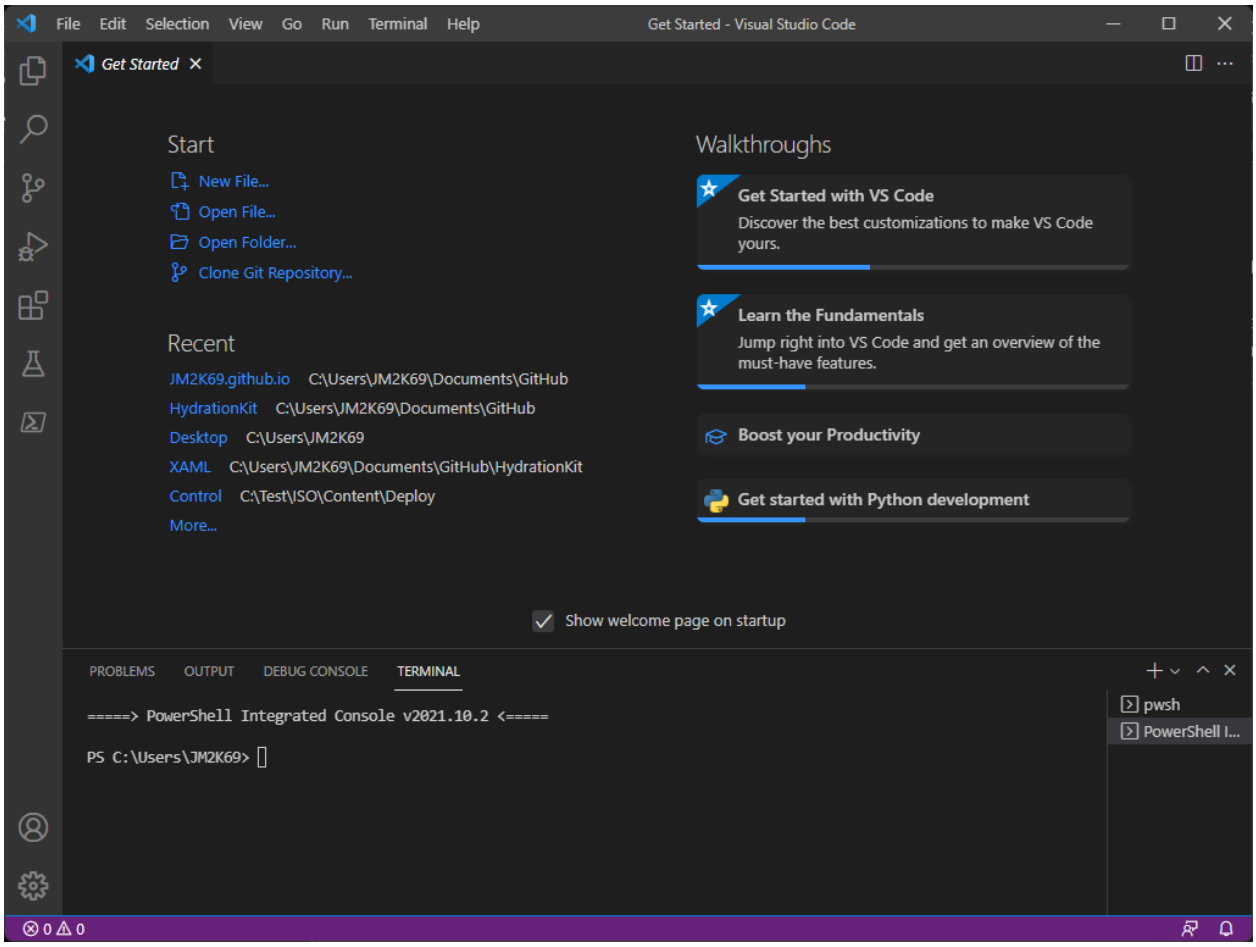
La valeur de la propriété **PSVersion** indique la version installée. La propriété **PSEdition** indique quant à elle l'édition de PowerShell. La valeur **Desktop** représente Windows PowerShell alors que la valeur **Core** représente PowerShell Core.

4. Visual Studio Code

Visual Studio Code, développé par Microsoft a la particularité de supporter PowerShell.

Visual Studio Code a une forte intégration avec le gestionnaire de codes sources **Git**. Sans être aussi puissant que **Visual Studio** classique, il suffit amplement pour la création de scripts PowerShell car il reprend l'ensemble des fonctionnalités de **PowerShell ISE**.

Visual Studio Code se base sur le framework **Electron**, projet open source permettant la génération d'interfaces graphiques multiplateformes (mettant en œuvre JavaScript, HTML et CSS). **Electron** repose sur les technologies bien connues que sont **Node.js** et **Chromium**. Il y a donc un lien de parenté évident entre l'éditeur **Atom** de GitHub et **Visual Studio Code**, ces éditeurs étant tous les deux basés sur Electron.



Si les couleurs sombres par défaut ne vous conviennent pas, sachez que vous pouvez changer de thème. Il en existe un grand nombre, dont un nommé *PowerShell ISE* qui reprend le même code couleur.

Un peu à la manière d'ISE, l'écran est divisé en deux parties horizontales avec, dans la partie supérieure, le code et, dans la partie inférieure, une console PowerShell interactive. Cette dernière permet de visualiser le résultat de l'exécution d'un script.

Pour tirer pleinement profit de Visual Studio Code avec PowerShell, il faudra installer l'extension PowerShell.

Pour ouvrir le menu des extensions, il faut soit se rendre dans le menu **Afficher -Extensions**

soit cliquer sur l'icône  dans la barre verticale située à gauche de l'écran.

5. Commandes de base

Avant toute chose, PowerShell est un environnement en lignes de commandes au service du système d'exploitation mais aussi et surtout au service des utilisateurs. En tant que tel, il est livré avec un jeu de commandes qu'il est bon de connaître, ou tout du moins de savoir comment les trouver ou les retrouver

5.1 Leur construction

Les commandes PowerShell sont appelées **cmdlets** (pour command-applets). Pour notre part, comme il n'existe pas de traduction officielle, nous avons pris le parti de les nommer **commandelettes**. Cela étant dit, la plupart du temps, nous les appellerons plus simplement des **commandes**. Elles sont constituées de la manière suivante : un verbe suivi d'un nom séparé par un tiret (**-**) : **verbe-nom**.

Par exemple, **Get-Command**.

Le verbe (en anglais bien sûr) décrit l'action à appliquer sur le nom. Dans cet exemple on récupère (**Get**) les commandes (**Command**).

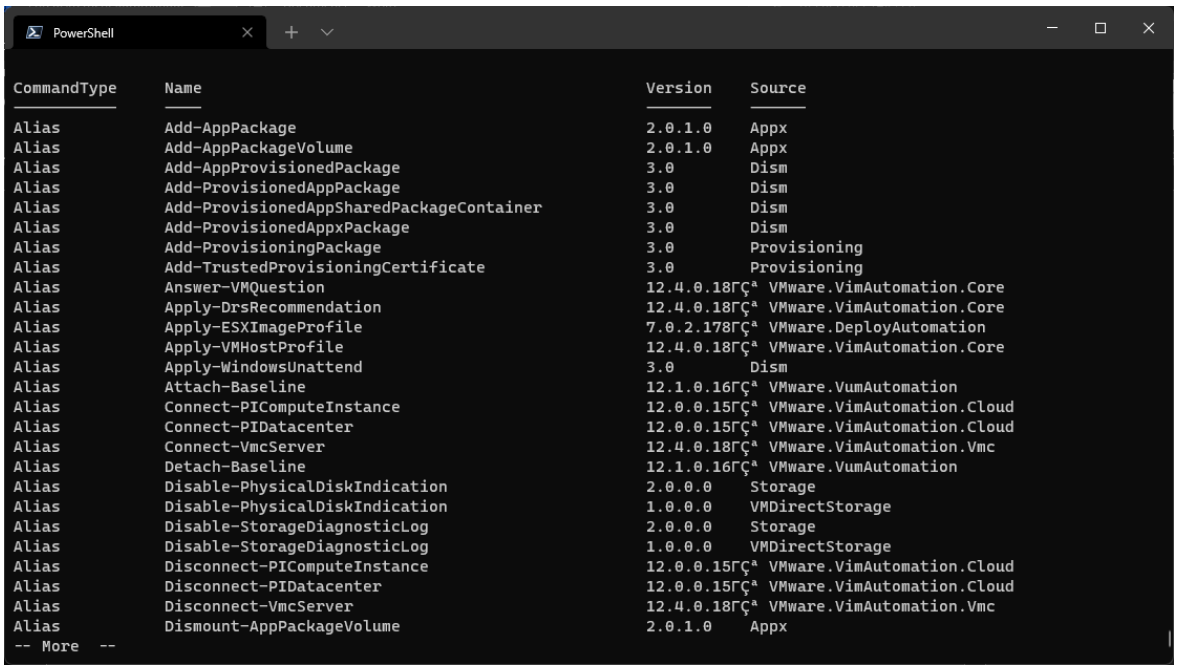
Avec PowerShell on trouve de nombreux verbes génériques tels que **Get**, **Set**, **Add**, **Remove**, etc. qui se combinent avec différents noms comme **Path**, **Variable**, **Item**, **Object**, **Computer**, etc.

Les noms constituant les commandes sont toujours au singulier ; et ceci vaut également pour les paramètres.

On peut donc, en croisant les verbes et les noms, se souvenir facilement de bon nombre de commandes. Notez que les commandes, ainsi que leurs paramètres associés, peuvent s'écrire indifféremment en majuscules ou en minuscules, l'analyseur de syntaxe PowerShell n'étant absolument pas sensible à la casse (sauf dans le cas de PowerShell 6 sous Mac OS ou Linux).

5.1.1 Get-Command

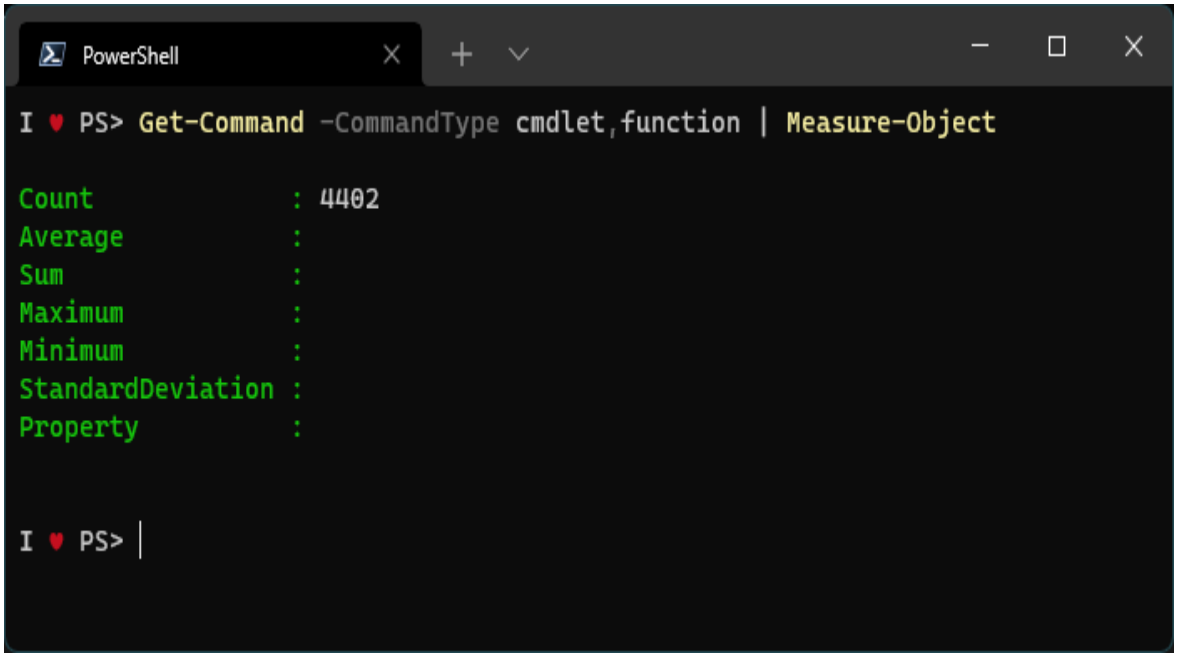
Si vous ne deviez n'en retenir qu'une seule, alors retenez au moins celle-là : **Get-Command**. Cette commande permet de découvrir toutes les commandes PowerShell. Sans préciser de paramètre, **Get-Command** retourne également les alias, et les fonctions de la session. Pour l'instant, intéressons-nous uniquement aux commandes, et pour cela, ajoutons le paramètre **-CommandType** suivi du type de commandes choisi, à savoir **cmdlet**.



```
PowerShell

CommandType     Name                                               Version    Source
-----------     ----                                               -------    ------
Alias           Add-AppPackage                                     2.0.1.0    Appx
Alias           Add-AppPackageVolume                               2.0.1.0    Appx
Alias           Add-AppProvisionedPackage                          3.0        Dism
Alias           Add-ProvisionedAppPackage                          3.0        Dism
Alias           Add-ProvisionedAppSharedPackageContainer           3.0        Dism
Alias           Add-ProvisionedAppxPackage                         3.0        Dism
Alias           Add-ProvisioningPackage                            3.0        Provisioning
Alias           Add-TrustedProvisioningCertificate                 3.0        Provisioning
Alias           Answer-VMQuestion                                  12.4.0.18ГÇª VMware.VimAutomation.Core
Alias           Apply-DrsRecommendation                            12.4.0.18ГÇª VMware.VimAutomation.Core
Alias           Apply-ESXImageProfile                              7.0.2.178ГÇª VMware.DeployAutomation
Alias           Apply-VMHostProfile                                12.4.0.18ГÇª VMware.VimAutomation.Core
Alias           Apply-WindowsUnattend                              3.0        Dism
Alias           Attach-Baseline                                    12.1.0.16ГÇª VMware.VumAutomation
Alias           Connect-PIComputeInstance                          12.0.0.15ГÇª VMware.VimAutomation.Cloud
Alias           Connect-PIDatacenter                               12.0.0.15ГÇª VMware.VimAutomation.Cloud
Alias           Connect-VmcServer                                  12.4.0.18ГÇª VMware.VimAutomation.Vmc
Alias           Detach-Baseline                                    12.1.0.16ГÇª VMware.VumAutomation
Alias           Disable-PhysicalDiskIndication                     2.0.0.0    Storage
Alias           Disable-PhysicalDiskIndication                     1.0.0.0    VMDirectStorage
Alias           Disable-StorageDiagnosticLog                       2.0.0.0    Storage
Alias           Disable-StorageDiagnosticLog                       1.0.0.0    VMDirectStorage
Alias           Disconnect-PIComputeInstance                       12.0.0.15ГÇª VMware.VimAutomation.Cloud
Alias           Disconnect-PIDatacenter                            12.0.0.15ГÇª VMware.VimAutomation.Cloud
Alias           Disconnect-VmcServer                               12.4.0.18ГÇª VMware.VimAutomation.Vmc
Alias           Dismount-AppPackageVolume                          2.0.1.0    Appx
-- More  --
```

Ici par exemple sur cette machine nous avons 4402 commandes, fonctions.



```
PowerShell
I ♥ PS> Get-Command -CommandType cmdlet,function | Measure-Object

Count             : 4402
Average           :
Sum               :
Maximum           :
Minimum           :
StandardDeviation :
Property          :


I ♥ PS>
```

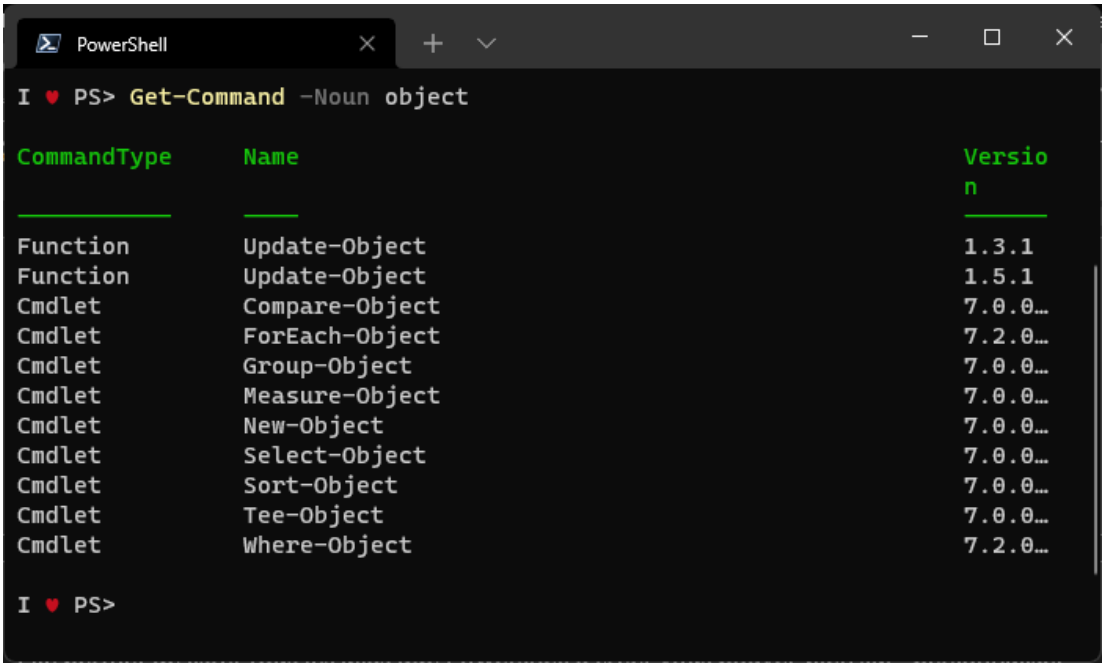
- Quelles commandes utiles :

```
Get-Command -Commandtype alias
```

Elle permet de lister tous les alias que Powershell a créé. Vous pouvez bien sur personnaliser et créer les vôtres.

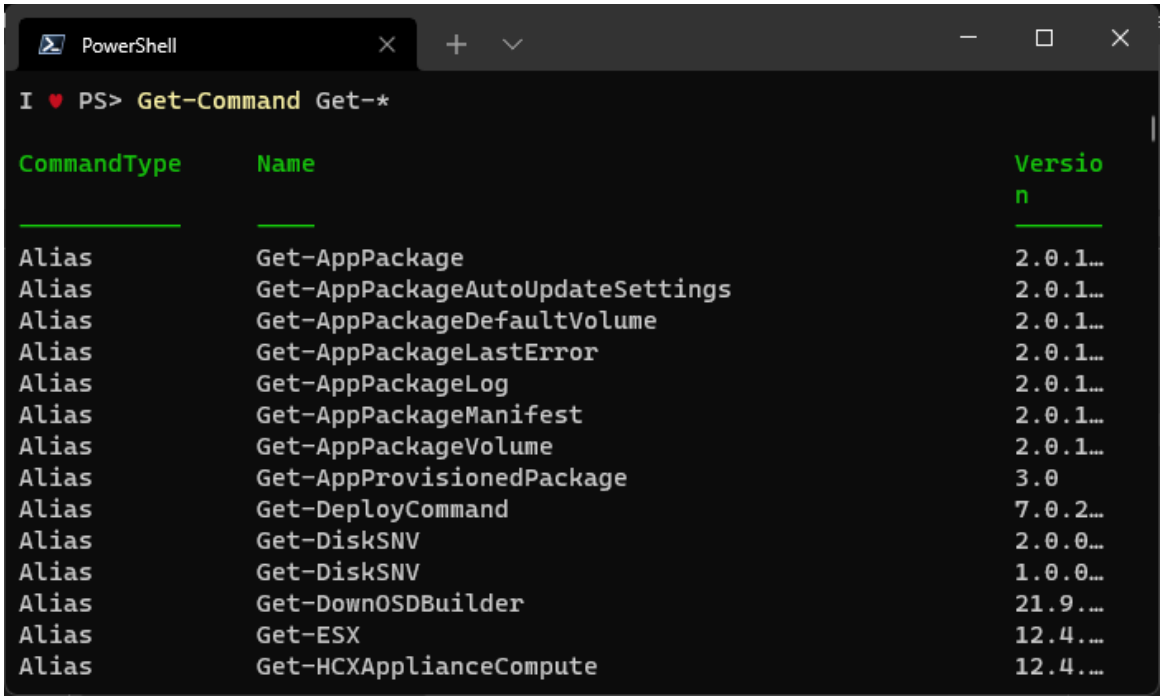
```
Get-Command -Noun object
```

Comme avec Powershell tout est Objet nous avons des commandes pour lister les cmdlets pour manipuler ces objects.



Pour lister toutes les commandes qui commence par Get-* vous devez utilisez la commande suivante :

```
Get-Command Get-*
```

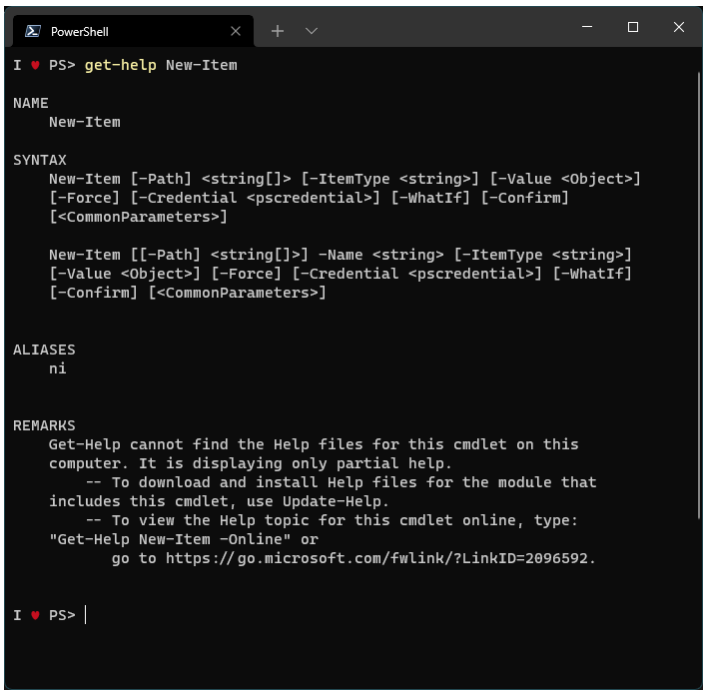


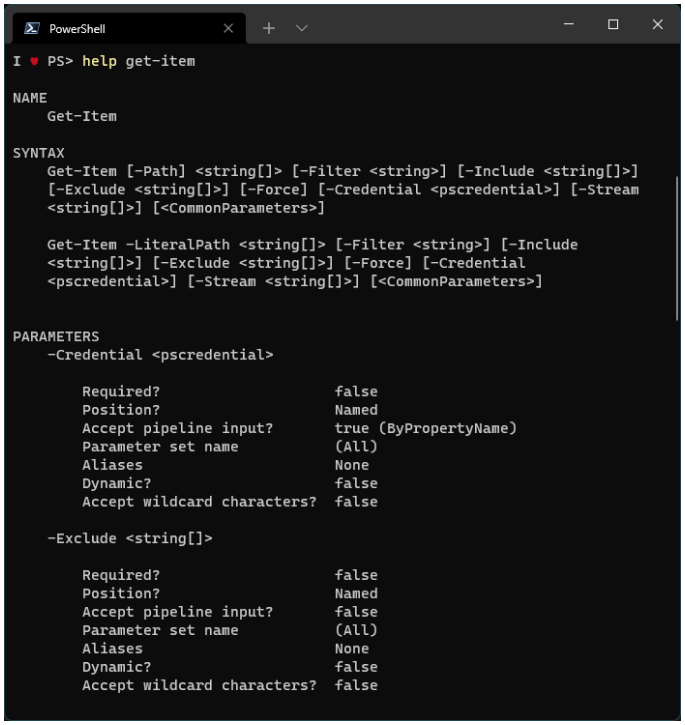
5.1.2 Get-Help

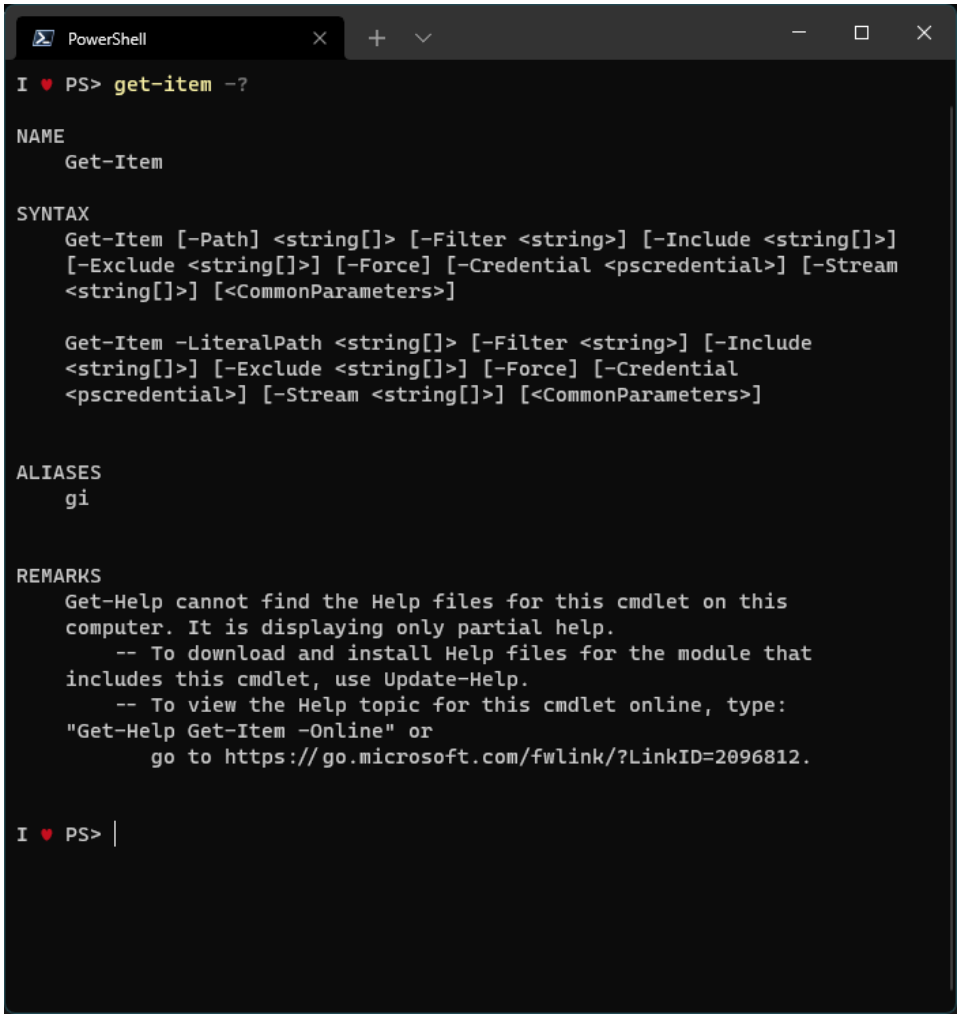
Cette commande de base permet, comme son nom l'indique, d'obtenir de l'aide sur n'importe quelle commande, voire davantage. Elle est très simple à utiliser et nous vous recommandons d'en user et d'en abuser !

Pour retrouver l'aide d'une commande vous avez différentes manières :

- Get-Help *maCommande*
- Help *maCommande*
- *maCommande* -?




PowerShell
I ♥ PS> help get-item
NAME
Get-Item
SYNTAX
Get-Item [-Path] <string[]> [-Filter <string>] [-Include <string[]>] [-Exclude <string[]>] [-Force] [-Credential <pscredential>] [-Stream <string[]>] [<CommonParameters>]
Get-Item -LiteralPath <string[]> [-Filter <string>] [-Include <string[]>] [-Exclude <string[]>] [-Force] [-Credential <pscredential>] [-Stream <string[]>] [<CommonParameters>]
PARAMETERS
-Credential <pscredential>
Required? false
Position? Named
Accept pipeline input? true (ByPropertyName)
Parameter set name (All)
Aliases None
Dynamic? false
Accept wildcard characters? false
-Exclude <string[]>
Required? false
Position? Named
Accept pipeline input? false
Parameter set name (All)
Aliases None
Dynamic? false
Accept wildcard characters? false


PowerShell
I ♥ PS> get-item -?
NAME
Get-Item
SYNTAX
Get-Item [-Path] <string[]> [-Filter <string>] [-Include <string[]>] [-Exclude <string[]>] [-Force] [-Credential <pscredential>] [-Stream <string[]>] [<CommonParameters>]
Get-Item -LiteralPath <string[]> [-Filter <string>] [-Include <string[]>] [-Exclude <string[]>] [-Force] [-Credential <pscredential>] [-Stream <string[]>] [<CommonParameters>]
ALIASES
gi
REMARKS
Get-Help cannot find the Help files for this cmdlet on this computer. It is displaying only partial help.
-- To download and install Help files for the module that includes this cmdlet, use Update-Help.
-- To view the Help topic for this cmdlet online, type: "Get-Help Get-Item -Online" or
go to https://go.microsoft.com/fwlink/?LinkID=2096812.
I ♥ PS>

Nous vous recommandons d'afficher systématiquement l'aide détaillée des commandes car l'aide standard n'affiche aucun exemple, ce qui est regrettable compte tenu de la richesse de compréhension qu'ils apportent. Il serait donc dommage de s'en priver ! L'aide détaillée apporte en plus des explications sur le fonctionnement de chaque paramètre, ce qui est appréciable, pour cela utiliser le paramètre ***-Detailed***

Quant à l'aide complète, celle-ci est très bien, mais elle peut s'avérer trop riche en détails et elle peut par conséquent noyer un scripteur débutant. C'est pourquoi nous pensons que l'aide détaillée est le niveau d'aide que Microsoft aurait dû proposer par défaut.

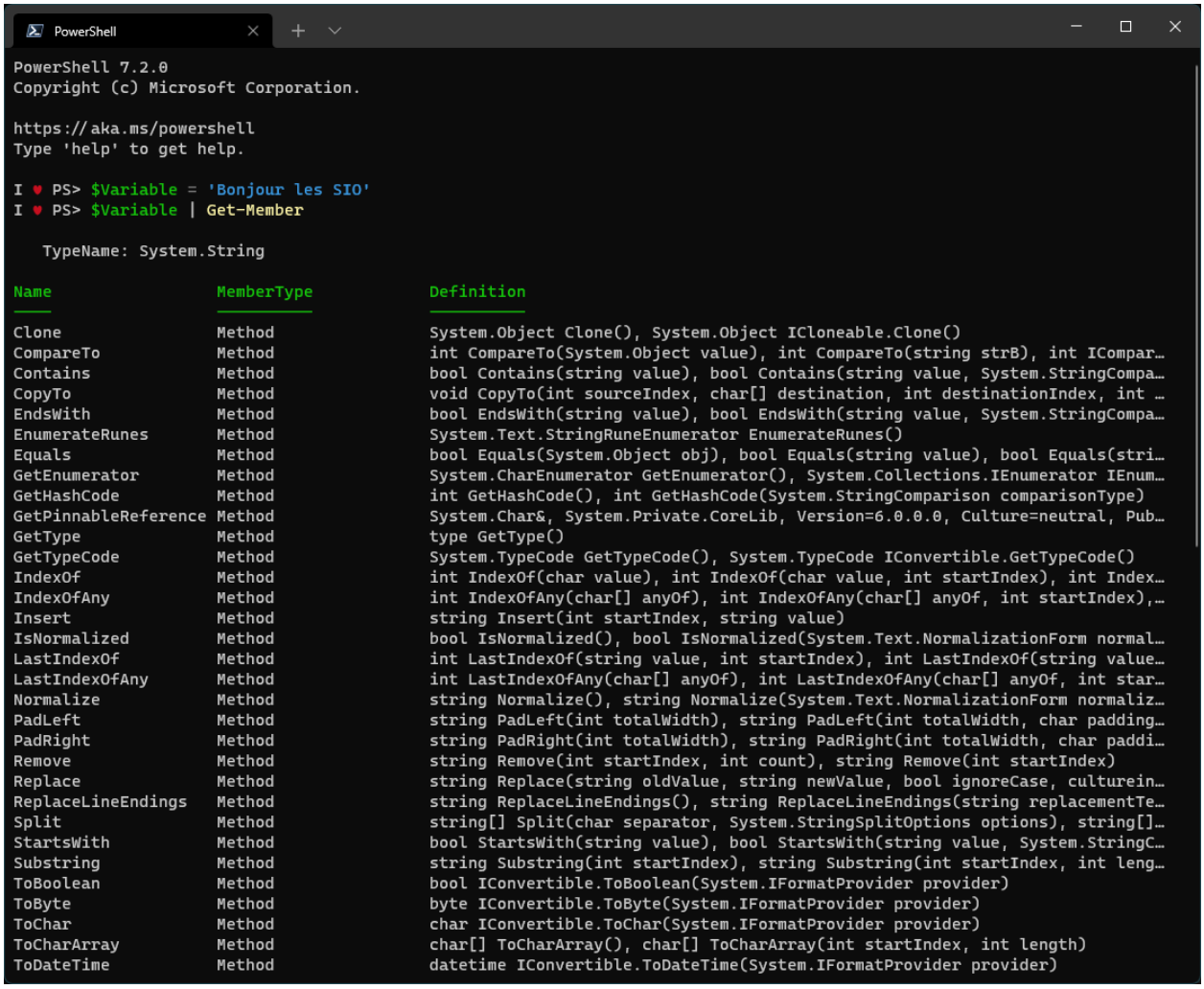
5.1.3 Get-Member

Get-Member est probablement la commande la plus intéressante de toutes car elle retourne toutes les propriétés et méthodes d'un objet ainsi que son type.

Par exemple si nous créons une variable $variable et lui avons affecté une valeur de type chaîne (String). Vous remarquerez que nous n'avons pas eu besoin de la déclarer car PowerShell affecte automatiquement un type en fonction du contenu. Une variable commence toujours par le caractère dollar. Nous discuterons en détail des variables dans le chapitre Variables et types de données.

Maintenant, imaginons que nous voulions faire des actions dessus, comme par exemple la convertir en majuscules ou bien compter le nombre de caractères qu'elle contient.

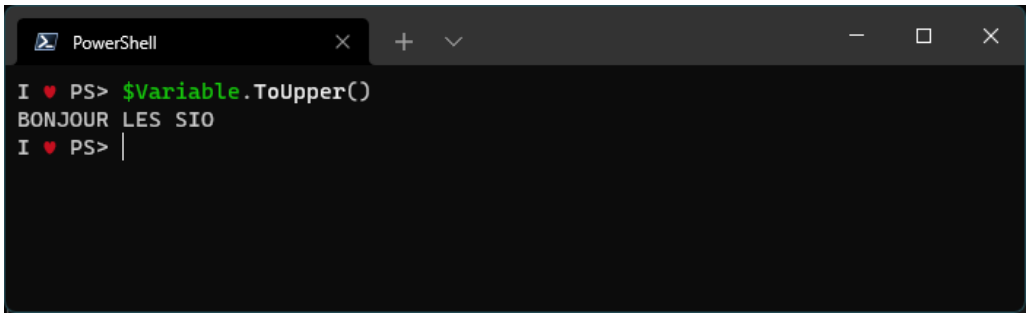
Pour faire cela habituellement dans tout langage de scripts ou de programmation, nous devons nous référer à la documentation pour connaître les commandes qui permettent la manipulation des chaînes de caractères. Bien sûr en PowerShell nous pouvons faire de même, mais c'est maintenant que la commande **Get-Member** prend tout son sens et vous allez comprendre pourquoi…



Nous voyons apparaître plusieurs éléments particulièrement intéressants :

- Le champ **TypeName** indique le type de la variable. Soit ici, comme on le supposait, un type String.
- Une liste de noms de méthodes, de propriétés, et leur définition associée.

Sans gros effort nous pouvons donc imaginer que la méthode **ToUpper** va nous permettre de retourner la chaîne contenue dans **$maVariable** en majuscules. Essayons-la pour voir ce qu'elle nous retourne :



Pour avoir le nombre de caractères nous utiliserons la propriété Lenght.