NOTES DE COURS

Présentation de PowerShell

Automne 2019

Table des matières

1 Présentation de PowerShell

1.1 Introduction

PowerShell est un environnement de ligne de commande Windows conçu pour les administrateurs système. Il comprend une invite interactive et un environnement de script.

PowerShell est basé sur le Common Language Runtime (CLR) .NET et le .NET Framework, il manipule des objets .NET.

Les fournisseurs de PowerShell permettent d'accéder à plusieurs magasins de données, tels que le Registre, les certificats de signatures numériques, les variables, les alias…etc.

- PowerShell manipule des objets de la plateforme .NET.
- PowerShell est livré avec un important jeu de commandes intégrées, dotées d'une interface homogène.
- PowerShell supporte les outils Windows traditionnels (NET, SC et REG…).

1.2 Applets de commande PowerShell

Une applet de commande est une commande à fonctionnalité unique qui manipule des objets dans PowerShell. Elles sont identifiables au format de leur nom : un verbe et un substantif anglais, séparés par un tiret (-), comme **Get-Help**, **Get-Process** et **Start-Service**.

Dans les environnements traditionnels, les commandes sont des programmes exécutables allant du très simple au très complexe.

Dans PowerShell, la plupart des applets de commande sont très simples et conçues pour une utilisation en association avec d'autres applets de commande. Par exemple, les applets de commande « **get** » ne font que récupérer des données, les applets de commande « **set** » permettent uniquement de définir ou de modifier des données, les applets de commande « **format** » servent exclusivement à la mise en forme de données et les applets de commande « **out** » seulement à diriger la sortie vers une destination spécifiée.

Chaque applet de commande est assortie d'un fichier d'aide auquel on peut accéder en utilisant la commande Get-Help:

```
get-help <nom_applet_commande> -detailed
```

L'affichage détaillé du fichier d'aide de l'applet de commande comprend une description de l'applet de commande, la syntaxe de commande, la description des paramètres et un exemple qui illustre l'utilisation de l'applet de commande.

Les commandes PowerShell ne sont pas sensibles à la casse.

```
PS C:\> get-date

May 2, 2019 4:54:34 PM
```

```
PS C:\> GET-DATE

May 2, 2019 4:54:44 PM
```

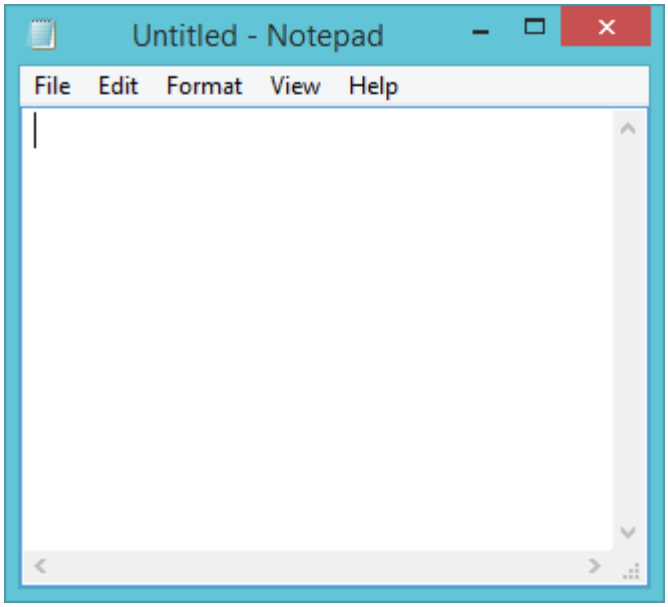
```
PS C:\> Get-Date

May 2, 2019 4:54:53 PM
```

1.3 Application Windows

Il est possible d'exécuter des applications et des utilitaires Windows dans PowerShell.

```
PS C:\> notepad
```




```
PS C:\> ping 127.0.0.1

Pinging 127.0.0.1 with 32 bytes of data:
Reply from 127.0.0.1: bytes=32 time<1ms TTL=128
Reply from 127.0.0.1: bytes=32 time<1ms TTL=128
Reply from 127.0.0.1: bytes=32 time<1ms TTL=128
Reply from 127.0.0.1: bytes=32 time<1ms TTL=128

Ping statistics for 127.0.0.1:
    Packets: Sent = 4, Received = 4, Lost = 0 (0%
loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 0ms, Average = 0ms
```

1.4 Orienté objet

PowerShell manipule des objets .NET.

Un objet .NET est une instance d'une classe .NET ayant des propriétés et des méthodes spécifiques.

EXEMPLE

Un objet de type Service est caractérisé par les propriétés suivantes :

Propriété	Description
Name	Nom du service.
Status	Statut du service. **Started** pour **démarré** et **Stopped** pour **arrêté**.

Une méthode est une action qu'on peut appliquer sur l'objet. Dans le cas d'un objet de type Service, il est possible d'appliquer les actions suivantes :

Méthode	Description
Start()	Démarrer le service
Stop()	Arreter le service.

Pour déterminer le type d'objet obtenu par une applet de commande, on utilise la commande **Get-Member :**

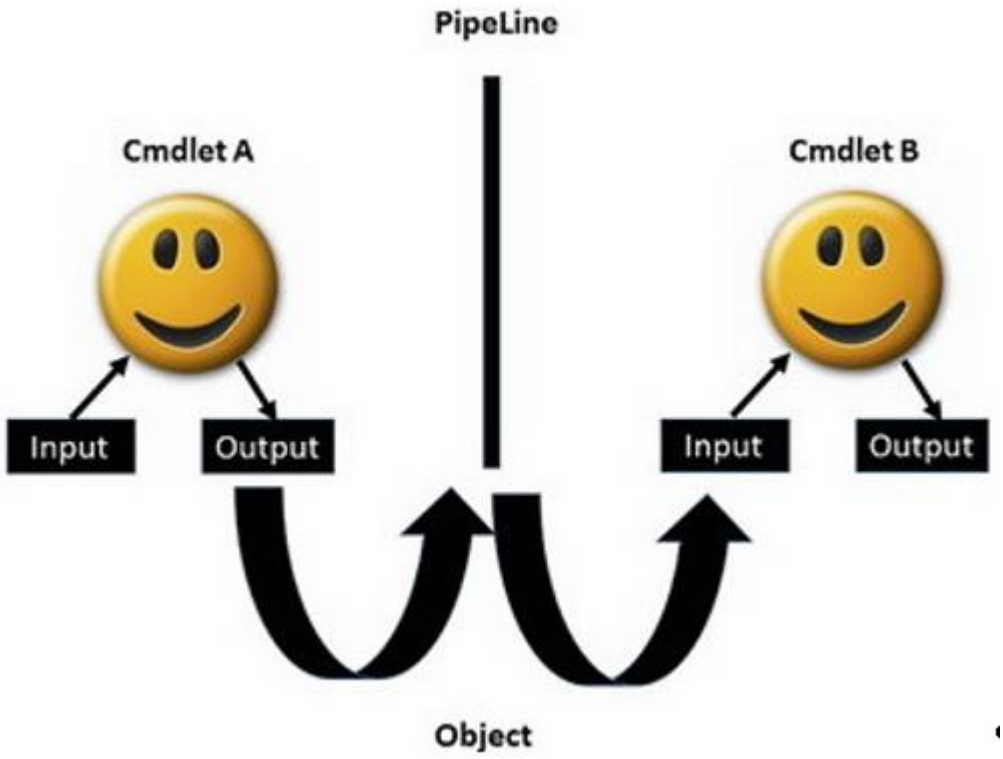
```
PS C:\> Get-Service | Get-Member

   TypeName: System.ServiceProcess.ServiceController

Name                      MemberType    Definition
----                      ----------    ----------
Name                      AliasProperty Name = ServiceName
add_Disposed              Method        System.Void
add_Disposed(EventHandler value)
Close                     Method        System.Void Close()
Continue                  Method        System.Void
Continue()
...
```

1.5 Enchainement (Pipeline)

L'enchainement permet de passer la sortie d'une commande à une autre commande, en tant qu'entrée.



EXEMPLE

Démarrer trois instances de Notepad.

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

Handles  NPM(K)  PM(K)  WS(K)  VM(M)  CPU(s)  Id ProcessName
-------  ------  -----  -----  -----  ------  -- -----------
    104   9      1456   6720   92     0.05    6300 notepad
    104   9      1460   6724   92     0.03    9196 notepad
    104   9      1460   6752   96     0.08   10992 notepad
```

Pour arrêter les trois processus de Notepad :

```
PS C:\> Get-Process notepad|Stop-Process
```