

Le PowerShell

PowerShell est la nouvelle interface en ligne de commande et le nouveau langage de scripts dédié à l'administration des systèmes Windows. PowerShell est orienté objet et utilise le framework Microsoft .NET pour l'exécution d'outils de ligne de commande appelés cmdlets ou commandlets. Ces commandes permettent d'administrer les systèmes Windows locaux ou distants. Le format ouvert de PowerShell permet le développement et l'ajout de cmdlets par des tiers sous forme de modules portables afin d'enrichir l'environnement standard. Microsoft met également à disposition de nombreux exemples de scripts PowerShell depuis le site : <https://code.msdn.microsoft.com>

Windows 10 intègre nativement la version 5.1 de Microsoft PowerShell.

De même que l'invite de commandes, PowerShell autorise la personnalisation de l'interface (clic droit sur la barre d'état de PowerShell, sous-menu **Propriétés**) et offre une fonctionnalité de zoom ([Ctrl]+molette de la souris).

1. Lancement de PowerShell

- Pour lancer l'hôte PowerShell, dans la zone **Rechercher** à droite du menu **Démarrer**, saisissez cette commande : `powershell`.
- Dans les résultats trouvés, cliquez sur **Windows PowerShell**. Vous remarquerez la présence de Windows PowerShell ISE ; cet environnement graphique, apparu avec Windows 7, permet l'écriture, l'exécution et le test de script PowerShell.

PowerShell assure la compatibilité avec les commandes DOS par l'intermédiaire des alias ; la liste des alias est disponible en exécutant la commande `Get-Alias`. Vous pouvez donc vous servir de l'interface PowerShell pour l'exécution de commandes DOS standards, comme par exemple la commande `DIR`.

Notez que l'utilisation des majuscules n'est pas obligatoire.

Vous pouvez identifier la version de PowerShell installée en affichant une variable système : `$PSVersionTable`.

PowerShell dispose de plusieurs modes d'exécution qui permettent de modifier son comportement :

- **Restricted** : il s'agit du mode par défaut. Aucun script ne peut être exécuté.
- **AllSigned** : seuls les scripts signés peuvent être exécutés, qu'ils soient locaux ou téléchargés.
- **Remote Signed** : les scripts présents sur votre ordinateur peuvent être exécutés même s'ils ne sont pas signés, ainsi que les scripts signés téléchargés depuis Internet.
- **Unrestricted** : il est possible d'exécuter tous les scripts, signés ou non. Une autorisation vous sera demandée.
- **Bypass** : aucun blocage ni avertissement ne sera présenté. Tous les scripts seront exécutés sans nécessité d'autorisation.
- **Undefined** : le mode d'exécution défini sera supprimé, excepté s'il s'agit d'une stratégie de groupe.

Il est possible de connaître le mode d'exécution avec la commande `Get-ExecutionPolicy`. Celui-ci est modifiable grâce à la commande `Set-ExecutionPolicy`.

2. Utilisation des cmdlets standards

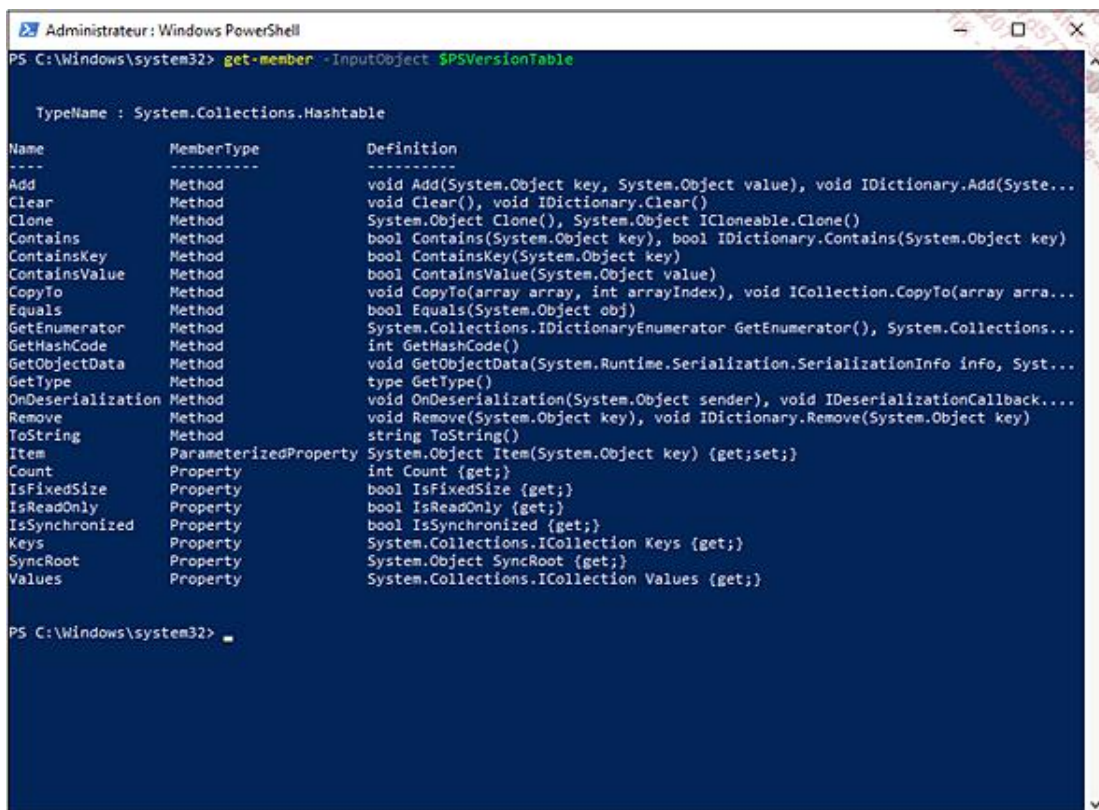
Une cmdlet, à prononcer "Commande-let", est une commande batch spécialisée dans l'exécution d'une fonctionnalité unique pour le contexte de l'interface de commande active. L'exécution d'une cmdlet ne crée pas de nouveaux processus sur le système hôte.

La cmdlet `get-command` permet de retourner la liste des cmdlets standards. Vous ne pourrez pas retenir l'ensemble des cmdlets disponibles pour l'environnement d'exécution PowerShell, celles-ci étant trop nombreuses. Par contre, vous devez pouvoir les retrouver et déterminer la syntaxe principale de celles-ci.

Les cmdlets importantes à connaître sont donc les suivantes :

- `get-command` : cette commande liste l'ensemble des cmdlets disponibles dans l'environnement d'exécution de PowerShell. La commande `get-command -noun process` permet de lister les cmdlets contenant la chaîne "process", cmdlets relatives à la gestion des processus.
- `get-help` : cette commande permet d'obtenir des informations détaillées sur l'utilisation des cmdlets, comme le nom et la description de la syntaxe de la cmdlet. La commande `get-help ls` permet de lister les informations détaillées sur la cmdlet qui fait référence à l'alias "ls".
- `get-member` : cette commande permet d'obtenir des informations sur les objets et listes d'objets des cmdlets. La liste des commandes suivantes permet de répertorier les objets attachés à un objet du système d'exploitation, ici le répertoire Windows, et d'afficher le contenu d'un objet de type propriété.

Exemple de l'utilisation de la commande `get-member` :



```
Administrateur: Windows PowerShell
PS C:\Windows\system32> get-member -InputObject $PSVersionTable

TypeName : System.Collections.Hashtable

Name      MemberType Definition
-----
Add        Method      void Add(System.Object key, System.Object value), void IDictionary.Add(Syste...
Clear      Method      void Clear(), void IDictionary.Clear()
Clone      Method      System.Object Clone(), System.Object ICloneable.Clone()
Contains   Method      bool Contains(System.Object key), bool IDictionary.Contains(System.Object key)
ContainsKey Method      bool ContainsKey(System.Object key)
ContainsValue Method      bool ContainsValue(System.Object value)
CopyTo     Method      void CopyTo(array array, int arrayIndex), void ICollection.CopyTo(array arra...
Equals     Method      bool Equals(System.Object obj)
GetEnumerator Method      System.Collections.IDictionaryEnumerator GetEnumerator(), System.Collections...
GetHashCode Method      int GetHashCode()
GetObjectData Method      void GetObjectData(System.Runtime.Serialization.SerializationInfo info, Syst...
GetType    Method      type GetType()
OnDeserialization Method      void OnDeserialization(System.Object sender), void IDeserializationCallback....
Remove     Method      void Remove(System.Object key), void IDictionary.Remove(System.Object key)
ToString   Method      string ToString()
Item       ParameterizedProperty System.Object Item(System.Object key) {get;set;}
Count      Property     int Count {get;}
IsFixedSize Property     bool IsFixedSize {get;}
IsReadOnly Property     bool IsReadOnly {get;}
IsSynchronized Property     bool IsSynchronized {get;}
Keys       Property     System.Collections.ICollection Keys {get;}
SyncRoot   Property     System.Object SyncRoot {get;}
Values     Property     System.Collections.ICollection Values {get;}
```

3. Windows Remote Management (WinRM)

Ce service implémente le protocole WS-Management utilisé par PowerShell pour l'administration distante des systèmes Windows dans un mode de connexion client-serveur. Le protocole WS-Management est un Web Service standard de type SOAP. Le service WinRM agit comme un listener (service d'écoute) côté serveur. L'outil de ligne de commande Winrs.exe permet de lancer des commandes distantes côté client réceptionnées par le service WinRM. PowerShell utilise le service WinRM pour l'exécution de scripts distants.

Avant de pouvoir lancer des commandes distantes et utiliser le service WinRM, vous devez configurer ce service.

- À l'aide de la commande `services.msc`, ouvrez le gestionnaire de services. Vérifiez la configuration du service **Gestion à distance de Windows (Gestion WSM)**. Démarrez ce service s'il n'est pas en cours de fonctionnement. Configurez le service pour qu'il démarre automatiquement à l'ouverture de session.

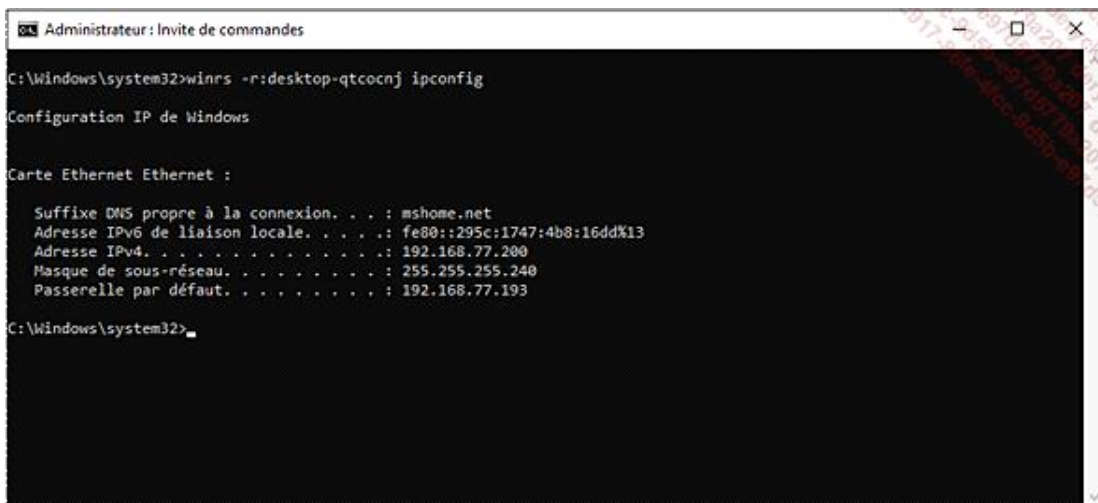
Côté serveur

- Ouvrez une invite de commandes en mode administrateur puis exécutez la commande `WinRM quickconfig` pour lancer la configuration du service **WinRM**.
- Tapez **y** pour confirmer la configuration du service en tant que listener (service d'écoute).
- Tapez **y** pour confirmer la configuration du pare-feu Windows.

Le service WinRM est configuré pour l'exécution de commandes distantes.

Côté client

- Ouvrez une invite de commandes en mode administrateur.
- Exécutez une commande conforme à la syntaxe suivante : `winrs -r:%servername% remote_command`
par exemple `winrs -r:desktop-qtcoenj ipconfig` dans l'exemple suivant.



```
Administrateur : Invite de commandes

C:\Windows\system32>winrs -r:desktop-qtcoenj ipconfig

Configuration IP de Windows

Carte Ethernet Ethernet :

    Suffixe DNS propre à la connexion. . . . : mshome.net
    Adresse IPv6 de liaison locale. . . . . : fe80::295c:1747:4b8:16dd%13
    Adresse IPv4. . . . . : 192.168.77.200
    Masque de sous-réseau. . . . . : 255.255.255.240
    Passerelle par défaut. . . . . : 192.168.77.193

C:\Windows\system32>
```

4. Exemples de commandes pour le dépannage avec PowerShell

Si vous maîtrisez les nombreuses commandes PowerShell, vous gagnerez beaucoup de temps pour les tâches de maintenance courantes qui restent souvent répétitives.

Vous pouvez, par exemple, visualiser les entrées en erreur du journal d'événements **Application** à l'aide de la commande suivante :

```
Get-EventLog -LogName "application" -Newest 20 -EntryType error |  
fl EntryType,Category,CategoryNumber,Source,Message
```

Vous pouvez également redéployer les applications téléchargées depuis le magasin Windows Store à l'aide de la

commande suivante :

```
Get-AppXPackage -AllUsers | Foreach {Add-AppxPackage  
-DisableDevelopmentMode -Register  
"$($_.InstallLocation)\AppXManifest.xml"}
```

Cette commande est utile si une ou plusieurs applications téléchargées depuis le magasin Windows Store rencontrent des problèmes de fonctionnement ou de démarrage sur votre environnement.

Une autre commande intéressante est la commande `repair-volume`, équivalente à la commande `chkdsk`.

Vous pouvez, par exemple, exécuter la commande suivante pour scanner le disque c:\ :

```
Repair-Volume c -Scan
```

Notez que Windows 10 intègre de nouvelles commandes PowerShell pour la gestion des packages d'applications ou pour gérer Windows Defender par exemple.

Le site TechNet de Microsoft répertorie les centaines de commandes PowerShell pour la gestion et l'administration des systèmes Windows. Cette liste est accessible à l'adresse suivante : [https://docs.microsoft.com/en-us/previous-versions/windows/powershell-scripting/mt156946\(v=technet.10\)?redirectedfrom=MSDN](https://docs.microsoft.com/en-us/previous-versions/windows/powershell-scripting/mt156946(v=technet.10)?redirectedfrom=MSDN)