

PowerShell

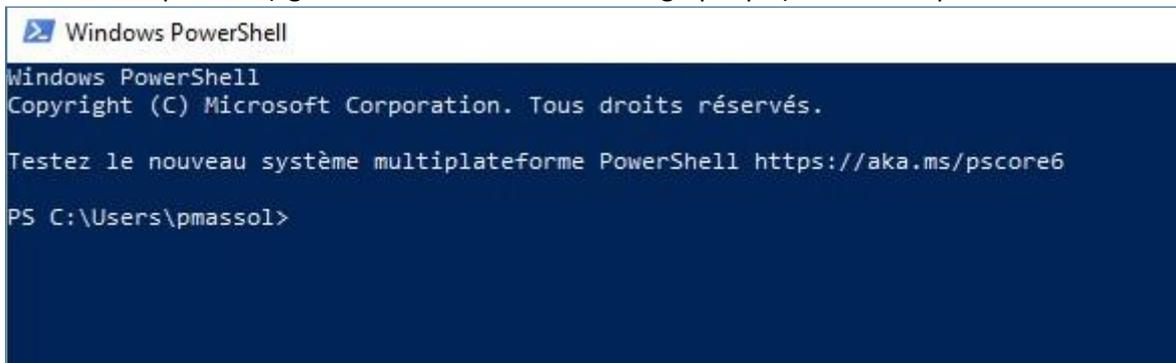
Contenu

1. Qu'est-ce que PowerShell ?	1
2. Petites astuces de la ligne de commandes	2
3. Comment obtenir de l'aide ?	4
4. Le pipeline et la variable \$_	4
5. Gestion des fichiers	5
6. Initiation aux variables, aux propriétés et aux méthodes des objets	5
7. Les scripts	6

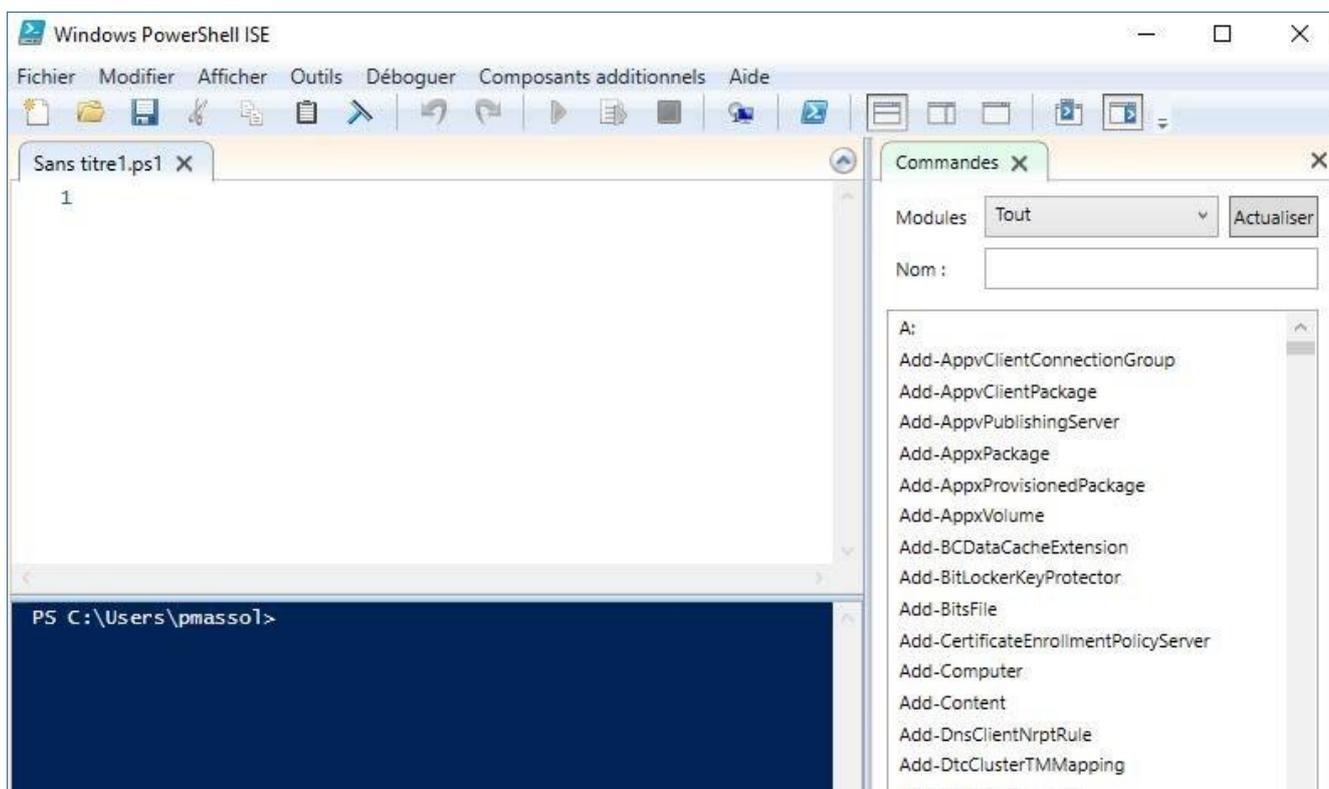
1. qu'est-ce que PowerShell ?

PowerShell est un langage de script fondé sur la programmation orientée objet et intégré à Windows depuis la version 7.

Deux outils sont à disposition (ligne de commandes ou éditeur graphique) accessibles par le menu Windows :



PowerShell en mode « ligne de commandes »



PowerShell en mode graphique

Pour se familiariser avec le langage, nous ferons d'abord des manipulations en ligne de commandes.

2. Petites astuces de la ligne de commandes

Quelques touches intéressantes :

Touche	Description
[Flèche en haut] [Flèche en bas]	Permet de faire défiler l'historique des commandes déjà saisies
[F8]	Rechercher dans l'historique en tapant le début d'une commande
[Ctrl] C	Met fin à l'exécution de l'instruction courante.

Une fois la commande retrouvée dans l'historique, vous pouvez soit presser la touche [Entrée] pour la sélectionner et l'exécuter, soit presser la flèche droite (ou gauche) pour modifier la commande avant de l'exécuter.

La touche tabulation [tab] permet de compléter le nom des commandes, le nom des paramètres et les chemins d'accès aux fichiers et dossiers.

L'action successive de la touche tabulation [tab] liste les éléments commençant par les caractères spécifiés.

EXEMPLES

(ne saisir que la partie encadrée)

Saisir assez de caractères pour restreindre la liste des commandes listées :

```
Administrateur : Windows PowerShell
PS C:\WINDOWS\system32> get-ch
```

L'action de la touche [tab] propose la première commande commençant par get-c :

```
Sélection Administrateur : Windows PowerShell
PS C:\WINDOWS\system32> Get-ChildItem
```

La saisie d'un espace et l'action de la touche [tab] liste les éléments du dossier actif :

```
Administrateur : Windows PowerShell
PS C:\WINDOWS\system32> Get-ChildItem .\@AdvancedKeySettingsNotification.png
```



Remarque

Le point devant le \ représente le chemin du dossier actif (ici c:\windows\system32).

La saisie du début d'un paramètre -r

```
Administrateur : Windows PowerShell
PS C:\WINDOWS\system32> Get-ChildItem .\@AdvancedKeySettingsNotification.png -r
```

L'action de la touche [tab] complète le nom du paramètre :

```
Administrateur : Windows PowerShell
PS C:\WINDOWS\system32> Get-ChildItem .\@AdvancedKeySettingsNotification.png -Recurse
```



Remarque

La saisie seule du caractère (-) permet de lister tous les paramètres possibles.

Il est possible bien sûr de spécifier le début d'un chemin en partant d'une lettre de volume disque :

```
PS C:\WINDOWS\system32> Get-ChildItem c:\w
```

L'action de la touche [tab] complète le nom du dossier :

```
PS C:\WINDOWS\system32> Get-ChildItem C:\Windows\
```

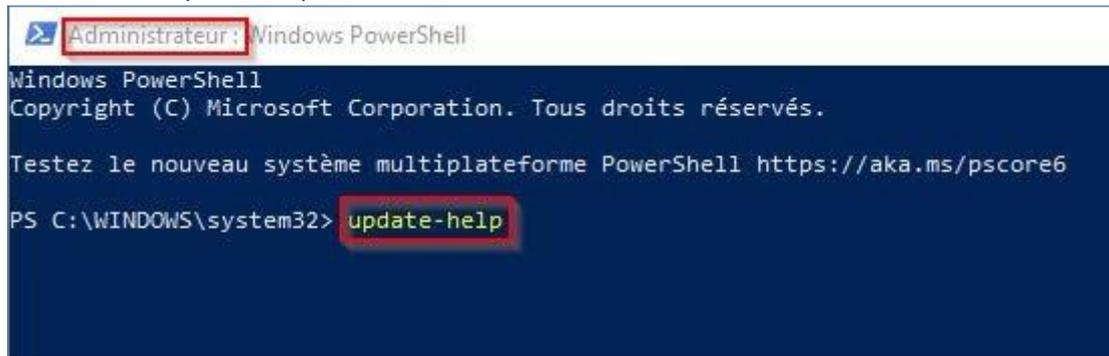


Remarque

Le chemin peut ainsi être entièrement complété à l'aide de l'action successive de [tab].

3. Comment obtenir de l'aide ?

Tout d'abord, il faut installer ou mettre à jour l'aide en ligne. Dans un PowerShell lancé en tant qu'administrateur on tape la commande « update-help » :



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

Testez le nouveau système multiplateforme PowerShell https://aka.ms/pscore6

PS C:\WINDOWS\system32> update-help
```

Mise à jour de l'aide en ligne

Désormais, il est possible d'obtenir une documentation détaillée sur toutes les commandes :

Rôle	Syntaxe	Exemple
Afficher de l'aide sur une commande	Get-Help Commande	Get-Help Get-ChildItem
Afficher des exemples d'utilisation de la commande	Get-Help Commande -examples	Get-Help Get-ChildItem -examples
Afficher la liste des méthodes et des propriétés des objets	Commande Get-member	Get-Children Get-Member

4. Le pipeline et la variable \$_

Le pipeline, symbolisé par le caractère « | » ([AltGr] + [6]) permet de « chaîner » plusieurs commandes. Autrement dit, la sortie d'une commande est liée à l'entrée de la suivante. Si on reprend le dernier exemple :

```
Get-Children | Get-Member
```

La sortie de la commande Get-Children n'est pas affichée mais envoyée à la commande Get-Member. Voici un autre exemple, plus complexe :

```
Get-ChildItem -recurse | Where {$_.extension -eq ".txt"}
```

La première commande renvoie le contenu du dossier courant et de tous ses sous-dossiers. Chaque résultat, **un par un**, est passé à la commande where qui permet de filtrer. Ici, on ne veut que les fichiers dont l'extension est .txt. Le \$_ correspond donc à chaque résultat envoyé par la première commande. Enfin, on peut filtrer l'affichage, par exemple si on ne veut que les noms de fichiers :

```
Get-ChildItem -recurse | Where {$_.extension -eq ".txt"} | Select fullname
```

5. Gestion des fichiers

Voici les principales commandes les plus utiles pour gérer les fichiers et dossiers :

Rôle	Syntaxe
Se déplacer dans les dossiers	Set-Location chemin (ex. : Set-Location c:\windows)
Afficher le chemin du dossier courant	Get-Location

Afficher le contenu d'un dossier	<code>Get-ChildItem</code>
Afficher le contenu d'un dossier et tous les sous-dossiers	<code>Get-ChildItem -recurse</code>
Créer un dossier	<code>New-Item nomDossier -ItemType directory</code>
Créer un fichier avec du texte	<code>New-Item nomFichier.txt -ItemType file -Value "le texte"</code>
Supprimer un fichier ou un dossier	<code>Remove-Item nomFichier.txt</code>
Déplacer un fichier	<code>Move-Item nomFichier.txt -Destination chemin\nomFichier.txt</code>
Déplacer un dossier	<code>Move-Item nomDossier -Destination chemin\nomDossier</code>
Renommer un fichier ou dossier	<code>Rename-Item nomFichier.txt -NewName nomFichier2.txt</code>
Copier un fichier	<code>Copy-Item nomFichier.txt -Destination nomFichier2.txt</code>
Copier un dossier avec ses fichiers	<code>Copy-Item nomDossier -Destination nomDossier1 -Recurse</code>
Tester l'existence d'un fichier ou dossier	<code>Test-Path chemin/nomFichier.txt</code>

6. Initiation aux variables, aux propriétés et aux méthodes des objets

Le nom d'une variable commence toujours par \$, il peut inclure tout caractère alphanumérique ou le trait de soulignement.

Windows PowerShell permet de créer des variables qui sont pour l'essentiel des objets nommés. La sortie de toute commande Windows PowerShell valide peut être stockée dans une variable.

EXEMPLE

```
$loc = Get-Location
```

Il est possible d'utiliser `Get-Member` pour afficher des informations sur le contenu des variables.

EXEMPLE

```
$loc | Get-Member (même résultat que Get-Location | Get-Member)
```

Le nom de la variable suivi du point permet d'accéder aux propriétés de l'objet référencé par la variable, exemple pour la propriété `Path` de la variable `$loc`.

EXEMPLE

```
$loc.Path
```



Remarque

L'usage de la touche tabulation [tab] permet de compléter le nom de la propriété. De même, l'exécution d'une méthode (action) d'un objet :

EXEMPLE

```
$fichier.Delete()
```



Remarque

Pour les méthodes, ne pas oublier les parenthèses avec ou sans paramètre.

7. Les scripts

7A. Autoriser l'exécution

Dans PowerShell, il existe quatre paramètres de stratégie d'exécution des scripts qui sont :

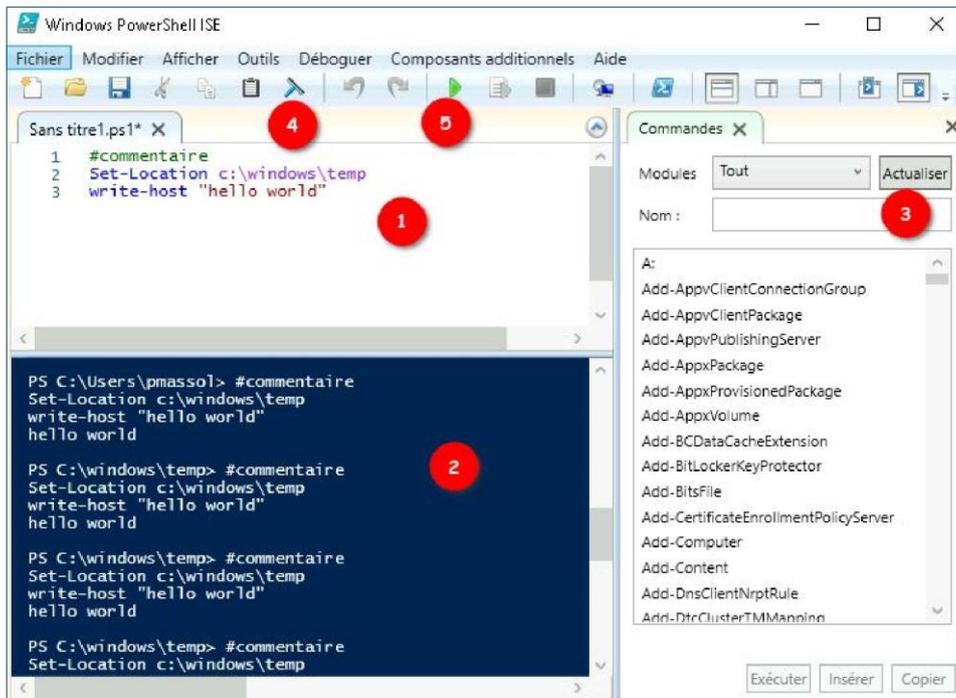
Restricted	paramètre par défaut , n'autorise pas l'exécution des scripts
AllSigned	n'exécute que les scripts de confiance, donc signés numériquement
RemoteSigned	exécute les scripts locaux sans obligation de confiance et les scripts de confiance issus d'Internet
Unrestricted	autorise l'exécution de tous les scripts

Démarrer Windows PowerShell en tant qu'administrateur, puis utiliser la commande suivante pour définir la stratégie :

- Commande pour connaître la stratégie en cours : `Get-ExecutionPolicy` ;
- Commande pour modifier la stratégie : `Set-ExecutionPolicy RemoteSigned`.

7B. Outil graphique (ISE)

L'environnement par défaut pour créer des scripts PowerShell est ISE (*Integrated Scripting Environment*) installé par défaut avec Windows. Il est cependant possible d'utiliser d'autres outils. Lancer depuis le menu Windows, PowerShell ISE :



- 1) Zone de saisie du script
- 2) Zone d'affichage des résultats du script
- 3) Référence et aide en ligne de toutes les commandes
- 4) Bouton pour effacer la zone 2
- 5) Bouton pour exécuter le script (raccourci : touche F5)

Avant l'exécution d'un script, il faut l'enregistrer. On lui donnera une extension .ps1

7C. Les structures de contrôle

7C1. Les conditions if else elseif

Une structure conditionnelle permet, via une évaluation de condition, d'orienter l'exécution vers un bloc d'instructions ou vers un autre. La syntaxe générale d'une structure conditionnelle est la suivante :

```
If (condition)
{
    #bloc d'instructions
}
```

Prenons un exemple, imaginons que nous souhaitons déterminer si une valeur entrée par l'utilisateur est la lettre A. Pour cela, nous allons utiliser une structure conditionnelle avec une condition sur la valeur de la variable testée. En utilisant un opérateur de comparaison, la structure est la suivante :

```
$var = Read-Host "Entrez un caractère"

If ($var -eq 'A')
{
    "Le caractère saisi par l'utilisateur est un 'A'"
}
```

On peut aussi mettre un else :

```

$var = Read-Host "Entrez un caractère"

If ($var -eq 'A')
{
    "Le caractère saisi par l'utilisateur est un 'A'"
}
Else
{
    "Le caractère saisi par l'utilisateur n'est pas un 'A' !"
}

```

Et un ou plusieurs elseif :

```

$var = Read-Host "Entrez un caractère"

If ($var -eq 'A')
{
    "Le caractère saisi par l'utilisateur est un 'A'"
}
Elseif ($var -eq 'B')
{
    "Le caractère saisi par l'utilisateur est un 'B'"
}
Else
{
    "Le caractère saisi par l'utilisateur n'est pas un 'A' ni un 'B' !"
}

```

Il est important de préciser que l'instruction "Else" doit toujours être la dernière si vous désirez inclure une ou plusieurs instructions Elseif : question de logique en fait.

Les différents opérateurs de comparaison :

-eq	égal à
-lt	plus petit que
-gt	plus grand que
-ge	plus grand ou égal
-le	plus petit ou égal
-ne	différent

Et quelques autres opérateurs utiles :

-not	Not
!	Not
-and	And
-or	Or

On peut réaliser des conditions complexes :

```

If (($var1 -eq 15) -and -not ($var2 -eq 18))
{
    write-host "lkjlkj" }

```

Quelques variables automatiques :

\$null	Représente la valeur INDEFINIE
\$true	Représente la valeur VRAI
\$false	Représente la valeur FAUX

7C2. Les répétitions

PowerShell propose toutes les boucles utilisées dans les langages de programmation :

- While
- Do...While
- Do...Until
- For
- Foreach

Nous nous intéressons ici à cette dernière, car c'est la plus utilisée dans les scripts. Le Foreach est pratique lorsqu'il y a besoin de manipuler une collection de données. Elle va automatiquement traiter, un par un, tous les éléments de cette collection et il n'y a pas besoin de connaître à l'avance le nombre !

Contrairement à un simple for. La

syntaxe est la suivante :

```
Foreach(<élément> in <collection>)  
{  
    # bloc d'instructions qui traite un élément  
}
```

Et voici un exemple qui utilise la commande Get-Childitem déjà présentée :

```
$collection = get-childitem  
  
Foreach ($element in $collection)  
{  
    if(Test-Path -Path $element -PathType Container)  
    {  
        write-host($element.Name + " est un dossier")  
    }  
else  
    {  
        write-host($element.Name + " est un fichier")  
    }  
}
```

On obtient tous les éléments (dossiers et fichiers) contenus dans le dossier courant. On parcourt cette collection et on teste si c'est un dossier ou un fichier.

7D. Les paramètres de script

Une script peut recevoir des paramètres en utilisant le bloc param :

```
param(
[string]$p1,
[int]$p2
)

Write-Host($p1 + ' a ' + $p2 + ' ans')
```

On l'exécute de la façon suivante :

```
PS C:\Users\pmasso1\Documents\powershell> .\ex1.ps1 toto 10
toto a 10 ans

PS C:\Users\pmasso1\Documents\powershell> .\ex1.ps1 10 toto
C:\Users\pmasso1\Documents\powershell\ex1.ps1 : Impossible de traiter la transformation d'argument sur le
paramètre «p2». Impossible de convertir la valeur «toto» en type «System.Int32». Erreur: «Le format de la chaîne
d'entrée est incorrect.»
Au caractère Ligne:1 : 14
+ .\ex1.ps1 10 toto
+ ~~~~~
+ CategoryInfo          : InvalidData : (:) [ex1.ps1], ParameterBindingArgumentTransformationException
+ FullyQualifiedErrorId : ParameterArgumentTransformationError,ex1.ps1

PS C:\Users\pmasso1\Documents\powershell> |
```

Un contrôle est réalisé automatiquement par PowerShell des types de paramètres. Ici, une erreur indique que « toto » n'a pas pu être converti en nombre.