



PowerShell

S.A.É. R1.05

Olivier ECKLE



Tous les OS (Windows, MacOS ou Linux) disposent de « **shells** » natifs en ligne de commandes.

- Sous Windows : cmd.exe, PowerShell...
- Sous MacOS & Linux : Bash, Zsh...



« cmd.exe » qui se nomme aussi « Invite de commandes » est le « shell » historique de Windows. Il permet d'accéder aux commandes en ligne héritées des toutes premières versions de MS-DOS (avant l'apparition de Windows...)
Si ce « shell » permet de faire des opérations basiques, il est loin d'offrir la puissance de « PowerShell »...



En informatique, un « **shell** » désigne une interface qui permet aux utilisateurs d'interagir avec le système d'exploitation.

Le terme « **shell** » signifie littéralement « *coquille* » et fait référence à une couche externe qui entoure le noyau (ou *kernel*) du système d'exploitation, permettant aux utilisateurs d'envoyer des commandes et d'obtenir des réponses.



- **Exécution des commandes** : Permet de lancer des programmes, exécuter des scripts et gérer des processus.
- **Automatisation** : Grâce aux scripts, le « **shell** » permet d'automatiser des tâches répétitives.
- **Gestion des fichiers et processus** : Manipuler des fichiers (créer, copier, déplacer, supprimer) et contrôler les processus (lister, tuer, etc.).
- **Communication avec le noyau** : Transmet les instructions au noyau pour qu'elles soient exécutées.



« *shell* » CLI versus « *shell* » GUI

Tous les OS offrent également la possibilité d'interagir avec le noyau via une interface graphique : « *shell* » graphique (GUI)...

L'ensemble des administrateurs systèmes, développeurs et techniciens réseau préfèrent de loin l'utilisation d'un « *shell* » en ligne de commandes (CLI). Il permet d'avoir un contrôle fin et direct sur le système, souvent bien au-delà des capacités des GUI...



Concept de base de PS

Les commandes « *PowerShell* » sont formées de « *Cmdlets* » écrites en « *.NET* ». Elles ont un format standard :

Verbe-Substantif

« *PowerShell* » met à disposition des informaticiens des jeux de « *Verbes* » destinés chaque domaine de l'informatique pour pouvoir réaliser toutes les tâches nécessaires.




« *PowerShell* » dispose de plus de 700 « *Cmdlets* »...

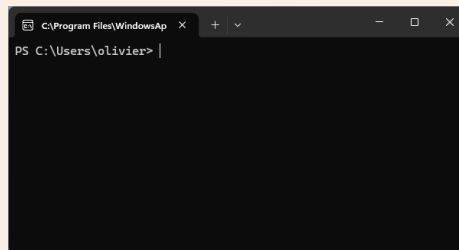


Accès au CLI de PowerShell

Depuis Windows 10, « *PowerShell* » est nativement présent dans l'OS avec une version 5.1 au minimum suivant les versions et MAJ de Windows...

Pour lancer la CLI de « *PowerShell* » :

Appuyer sur :  + R
Tapez : **pwsh**



Comment voir l'ensemble des Verbes PS ?

Lister tous les Verbes PS	
Get-Verb	Etablir la liste de tous les « Verbes » de PS.
<pre># Liste de tous les Verbes PS : Add, Clear, Close, Copy, Enter, Exit, Find, Format, Get, Hide, Join, Lock, Move, New, Open, Optimize, Push, Pop, Redo, Remove, Rename, Reset, Resize, Search, Select, Set, Show, Skip, Split, Step, Switch, Undo, Unlock, Watch, Connect, Disconnect, Read, Receive, Send, Write, Backup, Checkpoint, Compare, Compress, Convert, ConvertFrom, ConvertTo, Dismount, Edit, Expand, Export, Group, Import, Initialize, Limit, Merge, Mount, Out, Publish, Restore, Save, Sync, Unpublish, Update, Debug, Measure, Ping, Repair, Resolve, Test, Trace, Approve, Assert, Build, Complete, Confirm, Deny, Deploy, Disable, Enable, Install, Invoke, Register, Request, Restart, Resume, Start, Stop, Submit, Suspend, Uninstall, Unregister, Wait, Use, Block, Grant, Protect, Revoke, Unblock, Unprotect</pre>	



Evidemment, on ne va pas étudier tous ces « *verbes* »...



Verbes de PS pour : « Gérer des objets »

Gestion des objets			Gestion des objets		
New	Créer un nouvel objet	New-Item	Remove	Supprime un objet	Remove-Item
Set	Modifie des propriétés d'un objet existant	Set-Content	Get	Récupère des données ou des objets	Get-Service

```
# Créer un nouveau dossier
New-Item -Path C:\Temp\NewFolder -ItemType Directory

# Remplacer le contenu d'un fichier
Set-Content -Path C:\Temp\exemple.txt -Value "Nouveau contenu"

# Supprimer un fichier
Remove-Item -Path C:\Temp\NewFile.txt

# Récupérer un service spécifique, Exemple: "wuauserv" (Windows Update)
Get-Service -Name wuauserv
```



Verbes de PS pour : « Gérer les E/S »

Gestion des E/S			Gestion des E/S		
Read	Lire une entrée clavier	Read-Host	Write	Ecrire une sortie	Write-Host
Get	Lire le contenu d'un fichier	Get-Content	>	Redirection vers un fichier	>

```
# Lire la saisie du clavier et placer le résultat dans la variable $resultat
$resultat = Read-Host "Veuillez taper un message"

# Lire le contenu d'un fichier (texte) et placer le résultat dans la variable $contenu
$contenu = Get-Content -Path "C:\fichier.txt"

# Afficher un message dans la console pwsh
Write-Host "Message à donner..."

# Effectuer une redirection pour récupérer le résultat d'une commande dans un fichier
Get-ChildItem -Path "C:\Dossier" -Recurse > fichiers.txt
```



Verbes de PS pour : « Gérer des données »

Gestion des données			Gestion des données		
Add	Ajouter un élément ou un fichier	Add-Content	Copy	Copier un élément ou un fichier	Copy-Item
Clear	Supprime le contenu d'un élément	Clear-Host	Move	Déplace un objet ou un fichier	Move-Item

```
# Ajouter une ligne de texte à un fichier
Add-Content -Path C:\Temp\exemple.txt -Value "Ligne ajoutée au fichier"

# Effacer l'écran de la console
Clear-Host

# Copier un dossier (avec son contenu)
Copy-Item -Path C:\Temp\Folder1 -Destination C:\Temp\Folder2 -Recurse

# Déplacer un fichier
Move-Item -Path C:\Temp\source.txt -Destination C:\Temp\archive\source.txt
```



Verbes de PS de « Config Réseau »

Gestion des cartes réseaux	
Test-Connection	Tester une connexion réseau (équivalent en mieux d'un ping).
Resolve-DnsName	Résolution d'un nom de domaine (obtention de l'@IP correspondante).
Get-NetAdapter	Liste toutes les interfaces réseau disponibles.
Disable-NetAdapter Enable-NetAdapter	Désactive / Active une interface réseau donnée.
New-NetIPAddress	Configure l'@IP et la passerelle d'une interface réseau donnée.
Get-NetIPConfiguration	Affiche la configuration complète d'une interface réseau donnée.
New-SmbShare Remove-SmbShare	Créer / Supprime un partage réseau avec le protocole SAMBA.
New-NetRoute Remove-NetRoute	Créer / Supprime une route statique.



Obtenir de l'aide sur une « cmdlets » ?

Les différentes aides sur une cmdlets...

Get-Help <i>nom_de_la_cmdlets</i>	Affiche l'aide de base de la « cmdlets » indiquée...
Get-Help <i>nom_de_la_cmdlets -Detailed</i>	Affiche une aide avec plus de détails...
Get-Help <i>nom_de_la_cmdlets -Examples</i>	Affiche Uniquement des exemples d'utilisation.
Get-Help <i>nom_de_la_cmdlets -Full</i>	Affiche l'aide complète de la « cmdlets » indiquée...
Get-Help <i>nom_de_la_cmdlets -Online</i>	Affiche l'aide officielle dans un navigateur...



L'aide en « Online » est généralement mieux faite et en Français...



Les variables dans PS

Les variables en « **PowerShell** » se déclarent et s'initialisent avec la syntaxe suivante :

Le typage est facultatif...

[type] **\$***nom_variable* = valeur

Le « \$ » est OBLIGATOIRE...

Les types en « **PowerShell** » sont les suivants :

Liste.

Dictionnaire.

Types	[int]	[float]	[bool]	[string]	[array]	[hashtable]
-------	-------	---------	--------	----------	---------	-------------

Par défaut, les variables ne sont « visibles » uniquement dans le script PS...



Comment trouver un « cmdlets » PS

Pour l'immense majorité des actions que tout informaticien a besoin de faire dans le cadre d'un « **shell** » CLI, une « **cmdlets** » existe...

Il reste qu'à la trouver...

Faites pour cela, appel à votre meilleur ami virtuel !



Exemples :

```
$valeur = 57
$nom = "Groot"
$prix = 21.50
[float]$prix_reduit = 18
```

```
$liste = @(13, 5, 22, 53)
$liste[1] --> 5
$liste ----->
```

13
5
22
53

```
$dico = @{
    prenom = "Charlotte"
    age = 23
}
$dico["age"] --> 23
$dico["prenom"] --> Charlotte
$dico ----->
```

Name	Value
----	-----
age	23
prenom	Charlotte



Les structures conditionnelles dans PS

Les tests en « *PowerShell* » ont pour syntaxes :

```
if (condition) {
    .....
}
```

```
if (condition) {
    .....
} else {
    .....
}
```

```
if (condition_a) {
    .....
} elseif (condition_b) {
    .....
}
```

```
elseif (condition_c) {
    .....
} else {
    .....
}
```



Exemples :

```
$nom_complet_fichier = "C:\Temp\fichier.txt »

if (Test-Path -Path $nom_complet_fichier) {
    Write-Host "Le fichier $nom_complet_fichier existe..."
} else {
    Write-Host "Le fichier $nom_complet_fichier n'existe pas..."
}
```

On peut inclure une variable dans une string...

```
Write-Host "Attention : cette suppression est définitivement ..."

$reponse = Read-Host "Veuillez confirmer la suppression de ce fichier : (OUI)"
if ($reponse -eq "OUI") {
    Remove-Item -Path "C:\Temp\fichier.txt" -Force
    Write-Host "Fichier supprimé..."
} else {
    Write-Host "Opération annulée..."
}
```



Les structures conditionnelles dans PS

Les conditions qu'offre « *PowerShell* » sont :

Conditions			Conditions		
-eq	Est égal à	(\$a -eq \$b)	-ne	Est différent de	(\$a -ne \$b)
-lt	Est < à	(\$a -lt \$b)	-le	Est ≤ à	(\$a -le \$b)
-gt	Est > à	(\$a -gt \$b)	-ge	Est ≥ à	(\$a -ge \$b)
Conditions					
-like	Correspondance de chaîne avec joker (« * »)	(\$str -like "*test*")			
-notlike	Non correspondance de chaîne avec joker	(\$str -notlike "*toto")			
-match	Correspondance « Regex »	(\$str -match "^Test.*\$")			
-notmatch	Non correspondance « Regex »	(\$str -notmatch "^Test.*\$")			
-contains	Élément présent dans la liste	(\$lst -contains 51)			



Les boucles dans PS

Les CINQ boucles en « *PowerShell* » ont pour syntaxes :

```
for (init; cond_fin; inc) {
    .....
}
```

```
while(condition) {
    .....
}
```

```
foreach ($var in start..stop) {
    .....
}
```

Comme un « range » simplifié...

```
foreach ($item in liste) {
    .....
}
```

```
do {
    .....
} while(condition)
```

Comme la boucle « while » mais avec une exécution au minimum...



Exemples :

```
for ($i = 0; $i -lt 5; $i++) {  
    Write-Host "Valeur : $i" ----->  
}  
  
do {  
    $reponse = Read-Host "Veuillez taper : OUI"  
} while ($reponse -ne "OUI")  
  
$inventeurs = @("Akasaki", "Amano", "Nakamura") # inventeurs des LED bleues  
foreach $physicien in inventeurs {  
    Write-Host "Bravo $physicien..." ----->  
}  
  
foreach ($val in 0..3) {  
    Write-Host "Valeur : $val" ----->  
}
```

Valeur : 0
Valeur : 1
Valeur : 2
Valeur : 3
Valeur : 4

Bravo Akasaki
Bravo Amano
Bravo Nakamura

Valeur : 0
Valeur : 1
Valeur : 2
Valeur : 3



Bilan

Ce CM ne présente que les bases de « *PowerShell* ».


Maîtriser ce « *shell* » moderne et multi-plateforme est indéniablement un atout à votre CV.

Libre à vous d'aller plus loin...



Les scripts PS

Pour coder un script « *PowerShell* », l'utilisation du plus simple des éditeurs suffit. L'usage veut que l'on utilise le suffixe « **.ps1** » dans le nom du script.

Il suffit alors de lancer « *shell* » CLI ( +R) et taper : **pwsh**

Dans l'interface PS, déplacez-vous dans le répertoire où se trouve de script « *nom_du_script.ps1* » et taper ensuite ce même nom...

Il est possible que, par défaut, l'option d'exécution des scripts PS soit désactivée.
Vous pouvez la réactiver avec :

```
Set-ExecutionPolicy -Scope CurrentUser -ExecutionPolicy RemoteSigned
```



En librairies...

- Jérôme BEZET-TORRES, « *Débuter avec PowerShell* ». **ENI**
- Robin LEMESLE, « *PowerShell Core et Windows PowerShell* ». **ENI**
- Etienne LADENT, « *Cybersécurité & PowerShell* ». **ENI**
- Robin LEMESLE, « *Windows PowerShell (Administration Systèmes)* ». **ENI**