Introduction à PowerShell et aux scripts `.ps1`

Voici un **mini-cours clair, progressif et directement exploitable** sur la syntaxe PowerShell (`.ps1`) 👇

PowerShell est un shell moderne orienté **objets**, conçu par Microsoft. Contrairement à un terminal classique (bash, cmd), il manipule des **objets .NET** plutôt que du simple texte, ce qui le rend très puissant pour l'administration système et l'automatisation.

Un fichier `.ps1` est simplement un **script PowerShell**.

Structure de base d'un script `.ps1`

Un script PowerShell est une suite d'instructions exécutées de haut en bas :

```
## Ceci est un commentaire

Write-Output "Bonjour le monde"
```

Affichage

```
Write-Output "Texte"
Write-Host "Texte en couleur"
```

Exemple :

```
Write-Host "Hello" -ForegroundColor Green
```

Variables

Déclaration :

```
$nom = "Julien"
$age = 30
```

Utilisation :

```
Write-Output "Bonjour $nom"
```

⚠️ Toujours avec `$`

Types de données

PowerShell est **typé dynamiquement**, mais tu peux préciser :

```
[string]$nom = "Julien"
[int]$age = 30
[bool]$ok = $true
```

Opérateurs

```
#+   # addition
#-   # soustraction
#*   # multiplication
#/   # division
```

Comparaisons :

```
#-eq  # égal
#-ne  # différent
#-gt  # >
#-lt  # <
#-ge  # >=
#-le  # <=
```

Exemple :

```
if ($age -gt 18) {
    Write-Output "Majeur"
}
```

Conditions

```
if ($note -ge 50) {
    Write-Output "Réussi"
} elseif ($note -ge 40) {
    Write-Output "Échec léger"
} else {
    Write-Output "Échec"
}
```

Boucles

While

```powershell
$i = 0

while ($i -lt 5) {
    Write-Output $i
    $i++
}
```

For

```powershell
for ($i = 0; $i -lt 5; $i++) {
    Write-Output $i
}
```

Foreach (très important en PowerShell)

```powershell
$liste = 1,2,3,4

foreach ($element in $liste) {
    Write-Output $element
}
```

Tableaux

```powershell
$fruits = @("Pomme", "Banane", "Orange")

Write-Output $fruits[0]
```

Ajouter :

```powershell
$fruits += "Poire"
```

Cmdlets (le cœur de PowerShell)

PowerShell utilise des commandes appelées **cmdlets** :

Format :

`Verbe-Nom`

Exemples :

```
Get-Process
Get-Service
Get-ChildItem
```

👉 Exemple concret :

```
Get-ChildItem
```

= liste les fichiers (équivalent `dir`)

Pipeline (super pouvoir de PowerShell)

```
Get-Process | Where-Object {$_.CPU -gt 100}
```

- `|` = envoie le résultat à la commande suivante
- `$_` = objet courant

Fonctions

```
function Dire-Bonjour {
    param($nom)
    Write-Output "Bonjour $nom"
}

Dire-Bonjour "Julien"
```

Lecture utilisateur

```
$nom = Read-Host "Ton nom ?"
Write-Output "Bonjour $nom"
```

Manipulation de fichiers

Créer un fichier :

```
New-Item "test.txt"
```

Lire :

```powershell
Get-Content "test.txt"
```

Écrire :

```powershell
"Hello" | Out-File "test.txt"
```

Exécution d'un script `.ps1`

Étape importante (sécurité)

Par défaut, les scripts sont bloqués.

Autoriser :

```powershell
Set-ExecutionPolicy RemoteSigned
```

Lancer un script

```powershell
.\mon_script.ps1
```

Exemple complet

```powershell
## Demande le nom
$nom = Read-Host "Comment tu t'appelles ?"

## Condition
if ($nom -eq "") {
    Write-Output "Nom vide"
} else {
    Write-Output "Bonjour $nom"
}

## Boucle
for ($i = 1; $i -le 3; $i++) {
    Write-Output "Tour $i"
}
```

Points clés à retenir

- `$` → variables
- `Verbe-Nom` → cmdlets

- `|` → pipeline
- `$_` → élément courant
- PowerShell manipule **des objets, pas du texte**

Très bonne question — c'est **fondamental en PowerShell**, surtout dès que tu touches au pipeline ou à l'automatisation.

Différence entre `Write-Host` et `Write-Output`

Principe général

- `Write-Output` → envoie des **données dans le pipeline**
- `Write-Host` → écrit directement dans la **console (affichage uniquement)**

👉 Autrement dit :

Cmdlet	Destination	Utilisable dans pipeline
Write-Output	Pipeline (objet)	✅ Oui
Write-Host	Écran (console)	❌ Non

`Write-Output` (à privilégier)

Fonctionnement

```powershell
Write-Output "Hello"
```

➡️ Envoie une **valeur exploitable**

Exemple puissant

```powershell
Write-Output "Hello" | Get-Member
```

👉 Ça fonctionne, car `"Hello"` est passé dans le pipeline

Cas réel

```powershell
function Get-Nom {
    Write-Output "Julien"
}

$nom = Get-Nom
```

✔ `$nom` contient `"Julien"`

`Write-Host` (affichage uniquement)

Fonctionnement

```powershell
Write-Host "Hello"
```

➡️ Affiche dans la console, mais **ne retourne rien**

Exemple révélateur

```powershell
Write-Host "Hello" | Get-Member
```

❌ Ne fonctionne pas → rien ne passe dans le pipeline

Cas réel

```powershell
function Get-Nom {
    Write-Host "Julien"
}

$nom = Get-Nom
```

❌ `$nom` est vide

Différence visuelle

```powershell
Write-Host "Hello" -ForegroundColor Green
```

✔ Couleur possible ❌ Non exploitable

```powershell
Write-Output "Hello"
```

✔ Exploitable ❌ Pas de couleur

Règle simple (à enseigner)

👉 **Write-Output = données** 👉 **Write-Host = affichage**

Bonne pratique (très importante)

❌ Mauvais réflexe (débutant)

```powershell
Write-Host "Résultat : $result"
```

✅ **Bonne pratique**

```
Write-Output $result
```

👉 et éventuellement :

```
Write-Host "Résultat : $result"
```

pour du debug / UI

Quand utiliser `Write-Host` ?

Uniquement pour :

- messages utilisateur
- debug visuel
- affichage coloré

Quand utiliser `Write-Output` ?

Toujours pour :

- scripts réutilisables
- fonctions
- pipeline
- automatisation

Résumé clair

- `Write-Output` → **logique / données / pipeline**
- `Write-Host` → **UI / affichage uniquement**
