

## PowerShell – `Write-Host` : afficher un message à l'écran

`Write-Host` permet d'afficher un message directement dans la console PowerShell. Il est principalement utilisé pour informer l'utilisateur, afficher des messages de progression, des avertissements simples ou des résultats lisibles lors de l'exécution d'un script.

6TQ

 niveau

## Syntaxe de base

```
1 | Write-Host "Message à afficher"
```

Exemple :

```
1 | Write-Host "Script démarré"
```

## Afficher le contenu d'une variable

```
1 | $nom = "Alice"  
2 | Write-Host "Bonjour $nom"
```

## Afficher plusieurs informations

```
1 | $age = 17
2 | Write-Host "Nom : " $nom "Âge :" $age
```

PowerShell sépare automatiquement les éléments par des espaces.

## Affichage avec couleurs

```
1 | Write-Host "Succès" -ForegroundColor Green
2 | Write-Host "Erreur" -ForegroundColor Red
```

Couleurs courantes :

- Black, White, Gray
- Red, Yellow, Green
- Blue, Cyan, Magenta

## Message sur une seule ligne (sans retour à la ligne)

```
1 | Write-Host "Chargement..." -NoNewline
```

## Exemple concret dans un script

```
1 | Write-Host "Analyse en cours..."
2 |
3 | if ($fichier.Length -gt 1MB) {
4 |     Write-Host "Fichier trop volumineux" -ForegroundColor Red
```

```
5 }
6 else {
7     Write-Host "Fichier OK" -ForegroundColor Green
8 }
```

## Point important à retenir

`Write-Host` n'envoie rien dans le pipeline. Il sert uniquement à l'affichage visuel.

Cela signifie que le résultat affiché par `Write-Host` ne peut pas être récupéré par `Select-Object`, `Export-Csv` ou `Where-Object`.

## Bonne pratique pédagogique

- Utiliser `Write-Host` pour informer l'utilisateur
- Ne pas l'utiliser pour produire des données à traiter
- Pour produire des données exploitables, préférer :
  - `Write-Output`
  - `return`

## Résumé

Usage	Write-Host
Afficher un message	Oui
Couleurs	Oui
Données exploitables	Non
Pipeline	Non

# Pourquoi écrit-on `$( $fichiers.Count )` ?

```
1 | Write-Host "Nombre de fichiers : $( $fichiers.Count )"
```

## Problème à résoudre

Dans une chaîne de caractères, PowerShell peut remplacer :

- une variable simple → `$nom` , `$age`
- mais pas directement une expression

Ceci fonctionne :

```
1 | $nom = "Alice"
2 | Write-Host "Bonjour $nom"
```

Ceci ne fonctionne pas correctement :

```
1 | Write-Host "Nombre de fichiers : $fichiers.Count"
```

PowerShell interprète ici :

- `$fichiers` → variable
- `.Count` → texte normal

Résultat affiché (incorrect) :

Nombre de fichiers : System.Object[].Count

## Rôle de `$(...)`

`$(...)` signifie :

« ÉVALUE CETTE EXPRESSION AVANT DE L'INSÉRER DANS LA CHAÎNE »

C'est ce qu'on appelle une **sub-expression**.

```
1 | $(expression)
```

PowerShell calcule l'expression **avant** de construire la chaîne.

## Donc ici :

```
1 | $($fichiers.Count)
```

1. PowerShell calcule `$fichiers.Count`
2. Il obtient un nombre (ex: `12`)
3. Il insère ce nombre dans le texte

Résultat correct :

Nombre de fichiers : 12

## Autres exemples utiles

### Calcul dans une chaîne

```
1 | $a = 5
2 | $b = 7
3 | Write-Host "Somme : $($a + $b)"
```

### Accès à une propriété

```
1 | Write-Host "Nom du fichier : $($fichier.Name)"
```

### Méthode dans une chaîne

```
1 | Write-Host "Nom en majuscules : $($nom.ToUpper())"
```

## Règle simple à retenir

Dans une chaîne :

- `$variable` → OK
- `$variable.propriete` → utiliser `$()`
- toute **expression** → utiliser `$()`

## Variante sans `$()` (solution alternative)

On peut aussi séparer les éléments :

```
1 | Write-Host "Nombre de fichiers :" $fichiers.Count
```

Mais :

- moins lisible
- moins flexible

## Conclusion pédagogique

`$(...)` est indispensable dès qu'on veut insérer **un calcul, une propriété ou une méthode** dans une chaîne de caractères.

C'est un réflexe à acquérir pour écrire des scripts PowerShell propres et compréhensibles.

## Conclusion

`Write-Host` est idéal pour rendre un script **lisible et compréhensible** pour l'utilisateur. Il doit être utilisé comme un **outil d'affichage**, et non comme un moyen de transmettre des données.

