WILD CODE SCHOOL

Scripting Powershell
Partie 1

Quelle est la différence entre un script et un programme ?

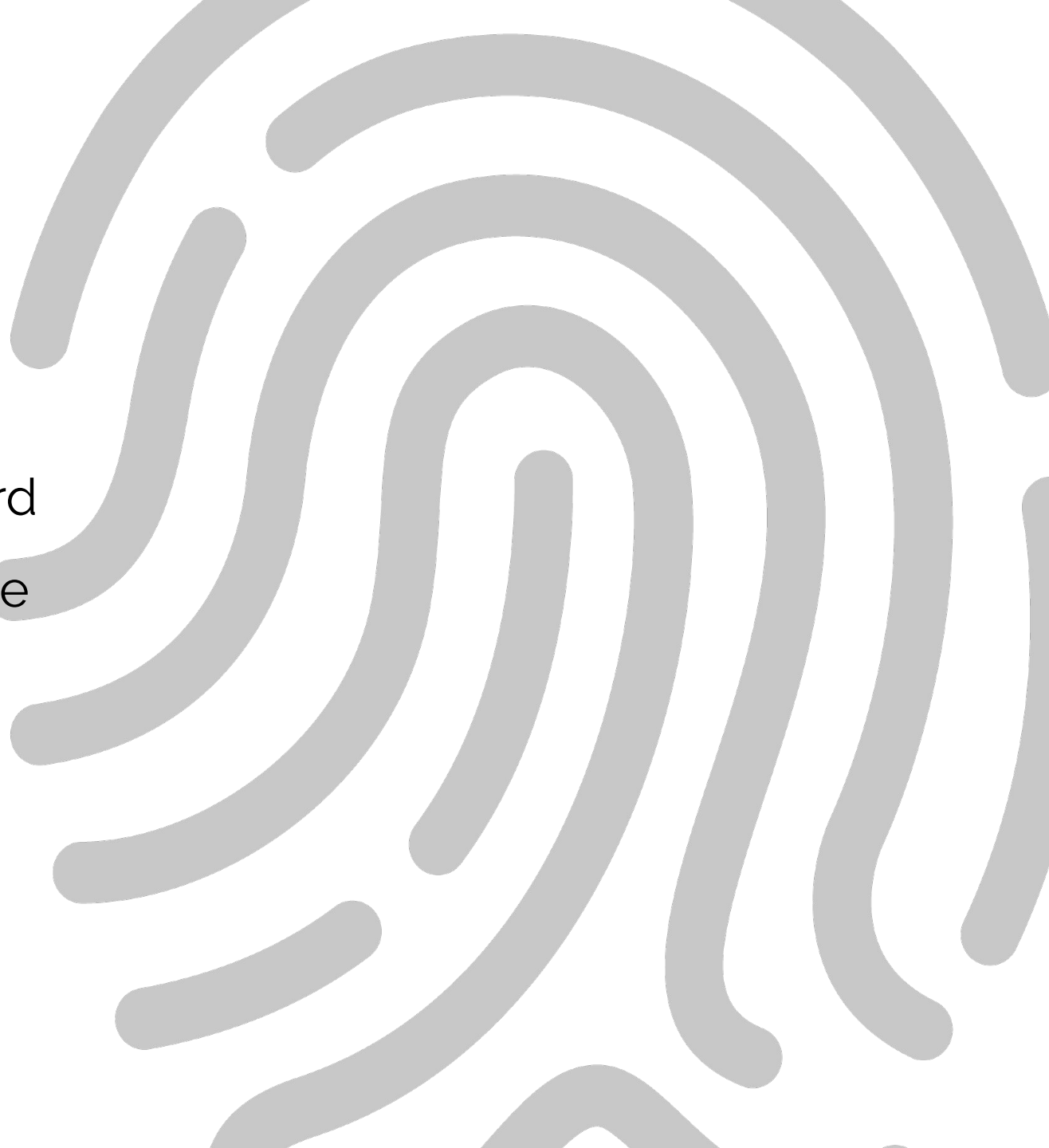
Plan

1 - Définition

2 - La base

3 - Les variables - utilisation standard

4 - Les variables - utilisation avancée



WILD CODE SCHOOL

Définition

Le PowerShell

Les scripts PowerShell sont des fichiers textes avec l'extension PS1.
On les exécute avec un interpréteur de commandes: **PowerShell.exe**.
En PowerShell on parlera de "**console**" au lieu de shell.
A la différence des autres interpréteurs de commandes qui n'acceptent et ne retournent que du texte, PowerShell accepte et retourne des objets .NET .

Le PowerShell - objet .NET

- .NET est un framework développé par Microsoft qui fournit un environnement pour construire et exécuter des applications.
- Dans .NET, les données sont représentées sous forme d'objets.
- Un objet est une instance d'une classe, qui combine des données (propriétés) et des actions (méthodes) qui peuvent être effectuées sur ces données.

Pour quoi faire ?

PowerShell sert à faire de l'automatisation de tâches et de la gestion de configuration de systèmes Microsoft.
Au quotidien :

- Ne pas répéter les même lignes de commandes tout le temps
- Gagner du temps
- Décomposer des tâches complexes en tâches simples

Comment ?

Un shell en ligne de commandes (la console)
Un langage de script associé (un script)
Donc en résumé :

- Avoir une connaissance du langage de script
- Avoir les logiciels adaptés (PowerShell ISE, Visual Studio Code, etc.)
- Avoir les droits d'accès pour l'écriture et l'exécution de script

Mon premier script

1. Ouvrir une console PowerShell
2. Ecrire le code suivant:

Write-Host “Hello World !”

3. Lancer l'exécution

```
PS C:\Lab> Write-Host “Hello World !”
Hello World !
```

Bonnes pratiques

- Les noms des scripts se terminent par .PS1
- Utiliser le symbole **#** pour mettre des commentaires dans vos scripts
- Nommage clair des variables

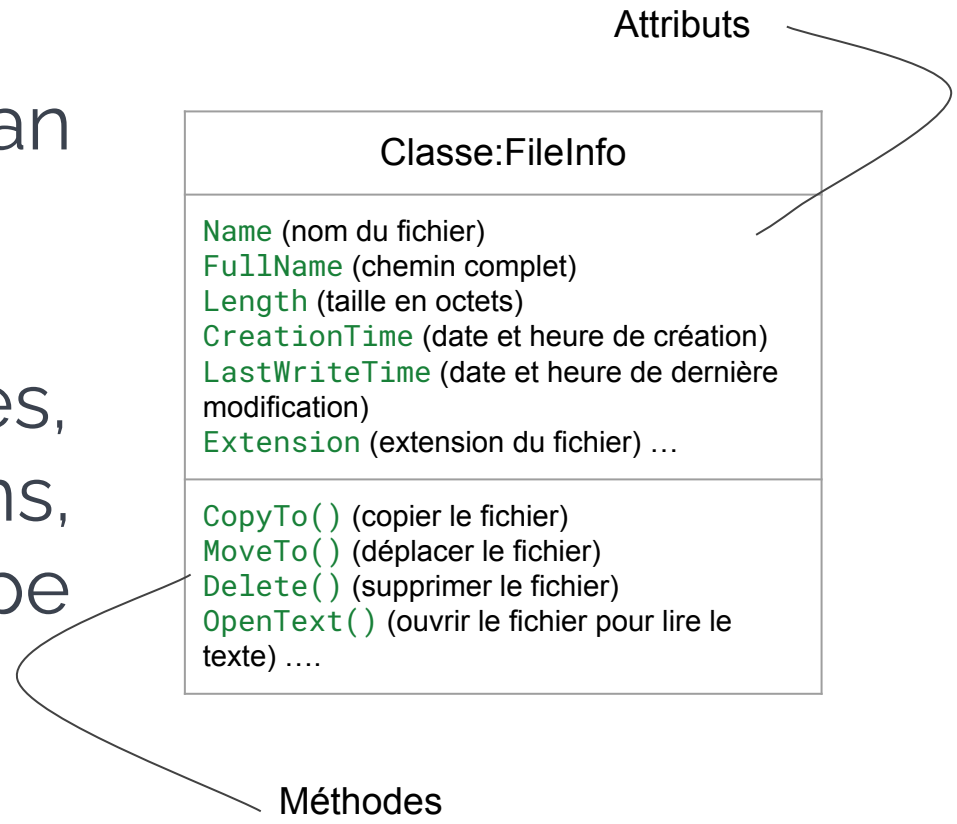
WILD CODE SCHOOL

La base

Introduction aux Classes (POO)

- Une classe est un modèle ou un plan pour créer des objets.

- Elle définit les propriétés (données, attributs) et les méthodes (actions, fonctions) que les objets de ce type auront.



Exemple de Classe

Get-ChildItem retourne des objets **FileInfo** (pour les fichiers) et **DirectoryInfo** (pour les dossiers).

Au lieu de manipuler du texte, on accède directement aux propriétés de ces objets.

```
PS C:\Lab> $file = Get-ChildItem monFichier.txt

PS C:\Lab> $file.Length
PS C:\Lab> $file.CreationTime

PS C:\Lab> $file.CopyTo("nouveauFichier.txt")
```

Les caractères d'échappement

Un **caractère d'échappement** commence avec ` (backtick).
Les séquences d'échappement ne sont interprétées que dans des chaînes de caractères avec " (double quote).
Quelques caractères d'échappement:

- `n → nouvelle ligne
- `t → tabulation

Les caractères d’échappement

- Ecrire le code PowerShell
- Voir son effet dans une console

```
PS C:\Lab> Write-Output "`nCeci est un saut de ligne`nEt ceci est une tabulation `tentre les mots"

Ceci est un saut de ligne
Et ceci est une tabulation      entre les mots
```

La lecture du flux

PowerShell **n'est pas sensible à la casse.** Ainsi les majuscules ou minuscules sont interprétées de la même façon.
PowerShell n'est pas sensible aux espace ou aux tabulations

```
PS C:\Lab> wRite-ouTput "Ceci est executé correctement`nDe même que la commande suivante";geT-chIldITeM -paTH *
```

Les commandes

PowerShell utilise un système d'alias prédéfini qui permet l'utilisation des commandes d'autres langages de script comme :

- Commandes batch (dos) : cd, dir, copy, etc.
- Commandes Linux : ls, cp, etc.

En réalité :
les alias prédéfini dans PowerShell pointent vers des cmdlet PowerShell.

Les commandes (suite)

Un cmdlet est sous la forme
<Verbe>-<Nom> -<nom_option> <valeur de l'option>

Exemples :
- Get-ChildItem
- Where-Object
- Set-Item

Quotes & Double quotes

PowerShell permet d'encapsuler des caractères :

- *Single quotes* (apostrophes) '
  - Aucun métacaractère sauf ' => fin de la chaîne
- *Double quotes* (guillemets doubles) "
  - Métacaractères : $ ` " et \
  - Attention à ne pas confondre ' et ` (caractère d'échappement)

```
PS C:\Lab> $i = 5
PS C:\Lab> Write-Output 'The value of $i is $i'
The value $i is $i
PS C:\Lab> Write-Output 'The value of $(2+3) is 5'
The value of $(2+3) is 5

PS C:\Lab> $j = 3
PS C:\Lab> Write-Output "The value of $j is $j"
The value of 3 is 3
PS C:\Lab> Write-Output "The value of `$j is $j"
The value of $j is 3

PS C:\Lab> Write-Output "The value of $(2+3) is 5"
The value of 5 is 5
```

WILD CODE SCHOOL

Les variables - Utilisation standard

Identifiant de variable

Une variable :

- Commence toujours par un **$**
- Est constitué de lettres, chiffres, caractères spéciaux.
- Le nom ou une partie du nom d'une variable peut-être le contenu d'une autre variable.
- Doit être unique et ne pas être un mot clé du langage
- N'est pas sensible à la casse

Convention de nom

En plus des règles imposées par le langage :

- Le **PascalCase** est souvent utilisé.
- Prendre des noms de variables qui ont un sens
- En anglais ou en Français

Utiliser des variables

- Syntaxe : **$NomVariable = valeur**
- Détruire une variable : **Remove-Variable**

```
PS C:\Lab> $Var = "Coucou"
PS C:\Lab> Write-Host $Var
Coucou
PS C:\Lab> $Var
Coucou

PS C:\Lab> $Var1 = $Var
PS C:\Lab> $Var = "Hello"
PS C:\Lab> $Var
Hello
PS C:\Lab> $Var + " World !"
Hello World !
PS C:\Lab> $Var1 + ":`t `t" + $Var + " World !"
Coucou:             Hello World !

PS C:\Lab> Remove-Variable Var, Var1
PS C:\Lab> $Var1 + ":`t `t" + $Var + " World !"
:                   World !
```

Utiliser des variables (suite)

```
PS C:\Lab> $MyDirectory = "MonDossier"
PS C:\Lab> New-Item -ItemType Directory -Path $MyDirectory
PS C:\Lab> Get-ChildItem

    Répertoire : C:\Lab

Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----        30/05/2022     16:24                MonDossier
```

Interpolation de variable

Permet d'inclure la valeur d'une variable directement dans une chaîne de caractères en utilisant la syntaxe **$()**.

Affichage dans une chaîne

On utilise **$($Variable.Attribut)** dans une chaîne de caractères pour afficher un attribut.

```
PS C:\Lab> $Host = Get-Host
PS C:\Lab> Write-Host "La version est $($Host.Version)"
La version est 5.1.19041.4170
```

WILD CODE SCHOOL

Les variables - Utilisation avancée

Invoke-Expression

Cmdlet qui permet d'exécuter une chaîne de caractères en tant que commande PowerShell.
=> Exécution d'une construction dynamique de chaîne de caractères.

! Attention à la source

Exemple

1. Déclarer une variable **Commande** ayant pour valeur **“whoami”**
2. Afficher la valeur de la variable
3. Exécuter **whoami** en utilisant la variable comme argument dans la cmdlet **Invoke-Expression**

```
PS C:\Lab> $Commande = "whoami"
PS C:\Lab> Write-Host $Commande
whoami
PS C:\Lab> $Commande
whoami
PS C:\Lab> Invoke-Expression -Command $Commande
Computer1\Wilder
```

Les méthodes

Ce sont des fonctions associées à un objet qui effectuent une action spécifique sur cet objet.
On peut les appliquer aux variables.

Syntaxe :
<donnée>.<méthode(paramètre)>

```
PS C:\Lab> "hello"
hello
PS C:\Lab> "hello".ToUpper()
HELLO

PS C:\Lab> "          hello      hello    "
          hello      hello
PS C:\Lab> "          hello      hello    ".Trim()
hello      hello


PS C:\Lab> $String = "hello"
PS C:\Lab> $String.ToUpper()
HELLO
```

Typage

Le typage d'une variable désigne la nature du type de données qu'elle peut contenir.
Quelques exemples :

- **[int]**: Entier 32 bits
- **[string]**: Chaîne de caractères

La méthode **GetType()** permet de connaître le type.

```
PS C:\Lab> "Bonjour".GetType()

IsPublic       IsSerial       Name                     BaseType
--------       --------       ----                     --------
True           True           String                   System.Object

PS C:\Lab> $Var1 = "hello"
PS C:\Lab> $Var1.GetType()

IsPublic       IsSerial       Name                     BaseType
--------       --------       ----                     --------
True           True           String                   System.Object

PS C:\Lab> $Var2 = 10
PS C:\Lab> $var2.GetType()

IsPublic       IsSerial       Name                     BaseType
--------       --------       ----                     --------
True           True           Int32                    System.Object
```

Transtypage ou casting

Mécanisme de conversion explicite d'une valeur d'un type de données à un autre.

Syntaxe :
[<Type de donnée>] <donnée>

```
PS C:\Lab> [Int]$Var="10"
PS C:\Lab> $var.GetType()

IsPublic       IsSerial       Name                       BaseType
--------       --------       ----                       --------
True           True           Int32                      System.Object

PS C:\Lab> $Var + 5
15
```

Substitution de commandes

Récupérer le résultat d'une commande (au lieu de l'afficher) :

- Syntaxe : **$(commande)**

Utilisation :

- Stocker dans une variable
- Utiliser dans une autre commande

```
PS C:\Lab> Get-Host | Select-Object Version

Version
-------
5.1.19041.1237


PS C:\Lab> $HostVersion = $(Get-Host | Select-Object Version)
PS C:\Lab> $HostVersion

Version
-------
5.1.19041.1237

PS C:\Lab> $Name = "Var"
PS C:\Lab> $Var = "Hello"
PS C:\Lab> Write-Host (Get-Variable -Name $Name -ValueOnly)
Hello
```

Substitution arithmétique

Effectuer un calcul

- Syntaxe : **<operation>**
  ou **$(<operation>)**

```
PS C:\Lab> 12 * 6
72

PS C:\Lab> Write-Host 12 * 6
12 * 6

PS C:\Lab> Write-Host $(12 * 6)
72

PS C:\Lab> $Total1 = 10+2
PS C:\Lab> $Total1
12

PS C:\Lab> $Total2 = $(7+3)
PS C:\Lab> $Total2
10

PS C:\Lab> Write-Host $($Total2*2 + 1)
21
```

Portée des variables

La portée protège les variables. Les niveaux de portée protègent les éléments qui ne doivent pas être modifiés.

- **Global** : Dans une console PowerShell, une nouvelle instance d'exécution, ou une nouvelle session.

Les variables sont présentes et disponibles dans la portée globale. L'ensemble des variables, alias et fonctions définis dans votre profil PowerShell sont également disponibles dans la portée globale.

- **Script** : Lors de l'exécution d'un script.

Les variables définies dans le script sont uniquement disponibles pour la portée du script et non pour la portée globale ou parente.

- **Local** : Lors de l'exécution d'une commande ou d'un script.

Les variables définies dans la portée script sont considérées comme sa portée locale.

- **Private** : Permet d'empêcher la visibilité d'une variable en dehors de la portée où la variable est définie.

Portée locale

Écrire le script suivant et l'enregistrer.

- Afficher le contenu de $Greeting
- Exécuter le script
- Afficher de nouveau $Greeting

$Greeting a une **portée locale** (au niveau du script).
Elle n'est pas disponible pour la **portée parent** (la console)

```
#script.ps1

$Greeting = "Hello, World!"
$Greeting
```

```
PS C:\Lab> $Greeting

PS C:\Lab> .\script.ps1
Hello, World !

PS C:\Lab> $Greeting

PS C:\Lab>
```

Portée globale

Écrire le script suivant et l'enregistrer.

- Exécuter le script
- Modifier le contenu de $Greeting
- Exécuter le script
- Afficher le contenu de $Greeting

Le contenu de $Greeting a été hérité de la console en tant que portée parente par le fichier script en tant que portée enfant.

```
#script.ps1

$Greeting
```

```
PS C:\Lab> .\script.ps1

PS C:\Lab> $Greeting = “Hello from the global scope !”

PS C:\Lab> .\script.ps1
Hello from the global scope !

PS C:\Lab> $Greeting
Hello from the global scope !

PS C:\Lab>
```

Modificateur de portée

Global : La variable existe dans la portée globale
Local : La variable existe dans la portée locale
Private : La variable est seulement visible dans la portée actuelle
Script : La variable existe dans la portée script, qui est la portée la plus proche du fichier script, ou dans la portée globale s'il n'y en a pas

Modifier la portée d'une variable

Reprendre le 1er script (portée locale) et le modifier comme ceci.

- Définir le contenu de $Greeting
- Afficher le contenu de $Greeting
- Exécuter le script
- Afficher de nouveau $Greeting

$Greeting est définie dans la **portée globale de la console**.
En exécutant le script, $Greeting prend une nouvelle valeur grâce au **modificateur de portée globale**.
La valeur de $greeting a été modifiée dans la **portée globale de la console** pour devenir la valeur du script.

```
#script.ps1

$global:Greeting = "Hello, World !"
```

```
PS C:\Lab> $Greeting = "Hello, Toto !"
PS C:\Lab> $Greeting
Hello, Toto !

PS C:\Lab> .\script.ps1

PS C:\Lab> $Greeting
"Hello, World !"

PS C:\Lab>
```

Variables spéciales

Autres variables

$? : Représente l'état d'exécution de la dernière opération (True ou False)
$_ : Contient l'objet actuel dans l'objet pipeline. Vous pouvez utiliser cette variable dans des commandes qui exécutent une action sur chaque objet ou sur des objets sélectionnés dans un pipeline.
$ARGS : Représente un tableau des paramètres non déclarés et/ou des valeurs de paramètre qui sont passés à une fonction, un script ou un bloc de script.
$Null : Variable automatique qui contient une valeur NULL ou vide. Vous pouvez utiliser cette variable pour représenter une valeur absente ou indéfinie dans les commandes et les scripts.
$True / **$False** : Représente True ou False. Peut être utilisé dans les commandes et les scripts.

Un exemple avec $_

- **$_** s'utilise derrière un **|** ou la boucle **foreach**
- On peut y ajouter des attributs
- **Get-Member** permet de connaître les attributs possible

```
PS C:\Lab> $MyDirectory = "MonDossier2"
PS C:\Lab> New-Item -ItemType Directory -Path $MyDirectory

    Répertoire : C:\Lab
Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----        30/05/2022     16:24                MonDossier2

PS C:\Lab> Get-ChildItem | Get-Member -MemberType Property

Name            MemberType      Definition
----            ----------      ----------
[...]
Name            Property        string Name {get;}
[...]

PS C:\Lab> Get-ChildItem | Where-Object {$_.Name -like "*mon*"}

    Répertoire : C:\Lab

Mode            LastWriteTime            Length      Name
----            -------------            ------      ----
d-----          30/05/2022     16:24                MonDossier2
```

Variables d'environnement

Certaines variables sont dites **d'environnement**.
Ce sont des variables dynamiques et globales au sein d'un système d'exploitation.
Les différents processus de la machine peuvent accéder à ces variables pour obtenir des informations sur la configuration actuelle du système.
Elles sont toutes sous la forme **$env:xxx**

Pour aller plus loin

On peut avoir la liste de toutes les variables du système avec la commande **Get-ChildItem env:**

Un exemple

Créer un script qui affiche:

- Le chemin complet du répertoire personnel de l'utilisateur courant
- Le chemin complet du profil PowerShell pour l'utilisateur actuel et l'application hôte actuelle
- La langue utilisée
- Le type d'architecture processeur

```
#script.ps1

Write-Host "Chemin du répertoire personnel de l'utilisateur : $Home"
Write-Host "Chemin complet du profil PS pour l'utilisateur actuel :  $Profile"
Write-Host "Langue du système : $PSCulture"
Write-Host "Type d'architecture : $($env:PROCESSOR_ARCHITECTURE)"
```

Shell et variables

Chaque console PowerShell est indépendante.
Les variables déclarées dans une console ne sont pas accessibles dans les autres.

En conclusion

- Détails sur les variables