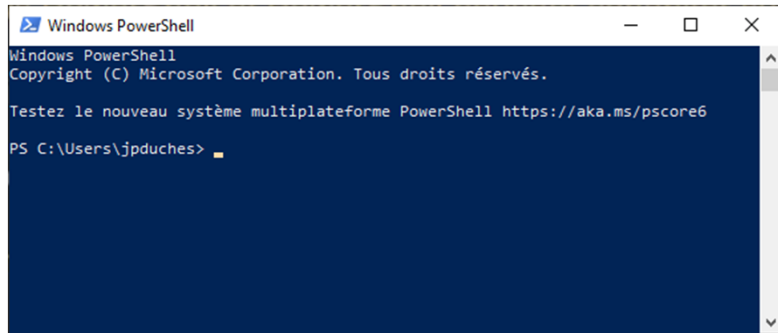


Infrastructure technologique et virtualisation

Module 6 : Utilisation des scripts

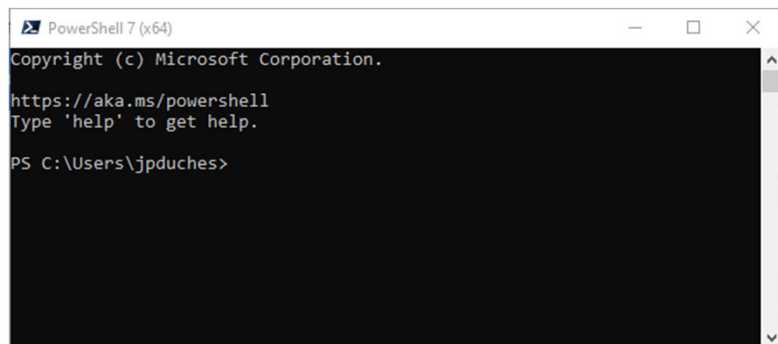
Power Shell



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

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

PS C:\Users\jpduches>
```



```
PowerShell 7 (x64)
Copyright (c) Microsoft Corporation.

https://aka.ms/powershell
Type 'help' to get help.

PS C:\Users\jpduches>
```

# Histoire de PowerShell

---

- En 2020, PowerShell Core devient PowerShell
- Windows PowerShell et PowerShell peuvent cohabiter sur la même machine Windows
  - Windows PowerShell se lance avec l'exécutable powershell.exe
  - PowerShell se lance avec l'exécutable pwsh.exe
- Dans un futur proche : PowerShell sera intégré à Windows.

# Interpréteur de commandes :

---

## Trouvez l'aide

Utilisation de la Cmdlet `Get-Help`

Exemple :

- `Get-Help Get-Verb`

<code>Get-Help</code> dispose de plusieurs paramètres	:	
Pour obtenir de l'aide en ligne	:	<code>Get-Help Get-Verb -Online</code>
Pour obtenir de l'aide détaillée	:	<code>Get-Help Get-Verb -Detailed</code>
Pour obtenir des exemples d'utilisation	:	<code>Get-Help Get-Verb -Example</code>

L'aide sur des concepts, par exemple sur les structures de contrôle if , on utilise `Get-Help about_if`  
Pour voir tous les about, on utilise : `Get-Help about_*`

# Interpréteur de commandes :

---

## Structure des commandes PowerShell

Les commandes PS ont une structure cohérente et sont faciles à retenir

Structure commande : **Verbe-nom** **–Paramètre** **valeur**

Exemple :

**New-Item** **–itemType** **File** **–Path** **d:\test.txt**

↑      ↑                      ↑              ↑              ↑

Verbe   Nom                      paramètre   Valeur   Paramètre   Valeur

Les commandes PowerShell se nomment « Cmdlet » et se prononce Commandlette

Exécuter plusieurs Cmdlet : **Get-Date**; **Get-Item** **c:\Windows**

# Interpréteur de commandes :

---

## Structure des commandes PowerShell

Le préfixe de **cmdlet** est appelé **verbe**, car il détermine l'action à effectuer sur les entités désignées dans la phrase. Voyons-en quelques-uns des plus utiles :

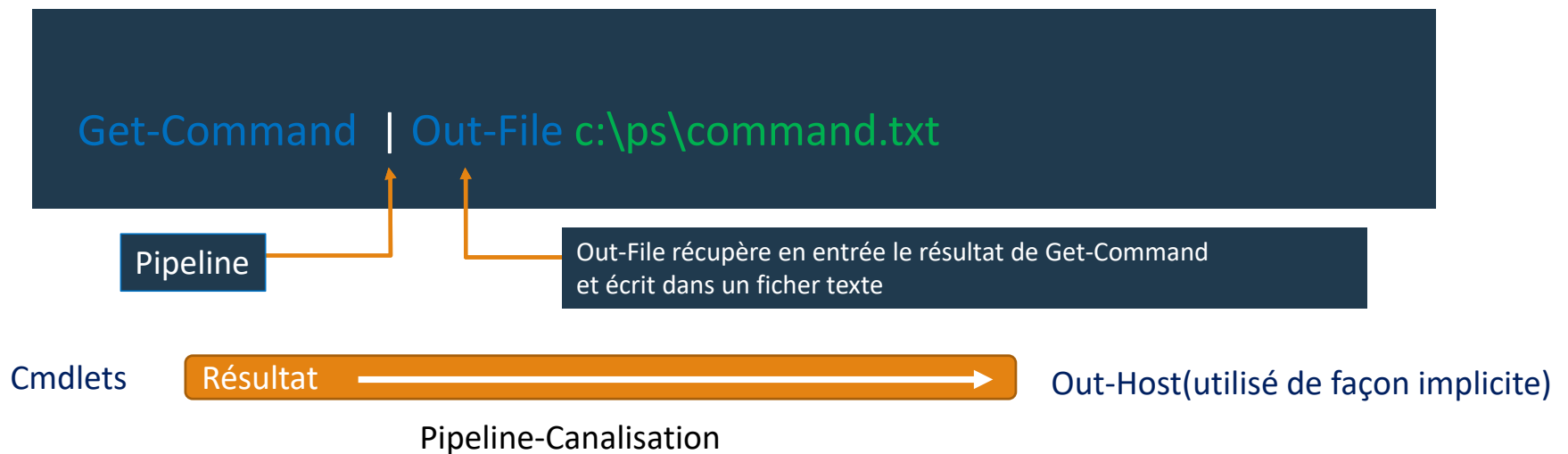
Verbe	Description
Add	Permet d' <b>ajouter</b> des données ou informations sur le nom qui suit;
Get	permet d' <b>obtenir</b> des données ou informations sur le nom qui le suit
Read	permet de <b>lire</b> des données ou informations sur le nom qui le suit ;
Clear	permet de <b>réinitialiser</b> l'affichage de l'interface ;
Import et Export	ermettent d' <b>importer/exporter</b> des fichiers de commande ou des <b>Alias</b> ;
New	permet de <b>créer</b> de nouveaux objets ou variables ;
Set	permet de <b>définir</b> des données ou informations sur le nom qui le suit ;
Write	permet de d' <b>écrire</b> des données ou informations sur le nom qui le suit et peut agir comme le compte rendu d'une commande.

# Interpréteur de commandes :

## Pipeline

Attention : connecter n'est pas exécuter à la suite comme avec ;  
à la [diapo 4](#)

Le Pipeline est symbolisé par le caractère | et il permet de **connecter** plusieurs Cmdlets entre elles.  
Une Cmdlet peut recevoir en entrée la sortie d'une autre Cmdlet.



# Interpréteur de commandes :

---

## Redirection des résultats (Sorties)

Par défaut, PS envoie la sortie de chaque Cmdlet à la console PowerShell (Sortie Standard)

### Opérateurs de redirection :

- > : Redirige la sortie vers un fichier texte et écrase le contenu déjà existant
- >> : Redirige la sortie vers un fichier texte sans écraser le contenu déjà existant
- 2> : Redirige les messages d'erreurs vers un fichier texte et écrase le contenu déjà existant
- 2>> : Redirige les messages d'erreurs vers un fichier texte et sans écraser le contenu déjà existant

[Liste complète des opérateurs de redirection](#) sur le site de Microsoft dans la documentation de PowerShell.

## Liste de commandes à maîtriser au plus vite :

---

Cmdlet	Description	Alias
Get-Command	Information de base sur les commandes	gcm
Get-Help	Aide de base (utiliser avec –full ou –example)	Help, man
Get-Member	Information sur les méthodes et propriété des objets	gm
Get-PSDrive	Information sur les « lecteurs' PowerShell	gdr
Get-Module	Liste les « modules' actuellement chargés	gmo



# Exemple de cmdlet à maîtriser

---

- Get-Help | Get-Date -Online
- Get-Help about\_if
- Get-Command
- Get-Member
- Notion alias (dir, ls, cp, etc.)

## One-liners

```
Get-Service |  
Where-Object CanPauseAndContinue -eq $true |  
Select-Object -Property *
```

PowerShell : Entrée  
PowerShell ISE : Maj+Entrée

# Travailler avec les fichiers et les dossiers

---

Les principales commandes à connaître pour travailler avec les fichiers et les dossiers :

Get-Location, Set-Location

New-Item -Path [Chemin] -Name [NomDeL'objet] -ItemType[Directory or File]

-Value [Texte qu'on veut écrire]

```
PS H:\> New-Item -ItemType Directory -Path h:\ -Name DossierTest
```

Directory: H:\

Mode	LastWriteTime	Length	Name
d----	13/04/2020 17:07		DossierTest

```
PS H:\> Remove-Item -Path H:\test.txt
```

```
PS H:\> Remove-Item -Path H:\DossierTest\
```

```
PS H:\> Remove-Item -Path H:\res\ -force
```

Confirm

The item at H:\res\ has children and the Recurse parameter was not specified. If you continue, all children will be removed with the item. Are you sure you want to continue?

[Y] Yes [A] Yes to All [N] No [L] No to All [S] Suspend [?] Help (default is "Y"): y

```
PS H:\> Copy-Item -Path H:\ressources\word.docx -Destination h:\word.docx
```

```
PS H:\> Copy-Item -Path H:\ressources\ -Recurse -Destination h:\res
```

```
PS H:\> Move-Item -Path H:\ressources\test\ -Destination h:\dossier1
```

```
PS H:\> Move-Item -Path H:\ressources\ppt.pptx -Destination h:\ppt.pptx
```

```
PS H:\> Rename-Item -Path H:\ppt.pptx
```

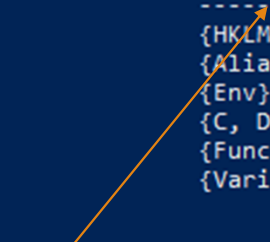
# Interpréteur de commandes :

## On gère quoi sur les Windows ? **Les providers**

- Les providers vous permettent d'accéder aux données de votre PC :
  - Système de fichier, Registre, Variable d'environnement, Service, etc.

```
PS C:\WINDOWS\system32> Get-PSProvider
```

Name	Capabilities	Drives
----	-----	-----
Registry	ShouldProcess, Transactions	{HKLM, HKCU}
Alias	ShouldProcess	{Alias}
Environment	ShouldProcess	{Env}
FileSystem	Filter, ShouldProcess, Credentials	{C, D, E}
Function	ShouldProcess	{Function}
Variable	ShouldProcess	{Variable}



La colonne « Drives » indique le chemin d'accès

Pour accéder au registre l'emplacement HKEY\_LOCAL\_MACHINE, vous taper : [Set-Location HKLM](#):

Pour accéder aux variables d'environnement : [Set-Location Env](#):

# Interpréteur de commandes :

---

## Les providers

### Avantage des providers :

Les Cmdlets que vous utiliserez pour créer, supprimer ou renommer un fichier seront identiques pour toutes les données que vous manipulerez (Registre, Variable d'environnement, Alias, etc.)

### Exemple :

**New-Item** sera utilisée pour créer un nouveau fichier, variable d'environnement, nouvelle valeur du registre, etc.

#### Cmdlets pour les providers:

- Get-PSProvider** : lister les providers disponibles
- Get-PSDrive** : Lister les lecteurs d'accès aux données
- Remove-PSDrive** : Supprimer un lecteurs.

# Interpréteur de commandes :

---

## Manipuler les résultats, il existe plusieurs Cmdlets:

**Where-object** : filtrer les résultats

Exemple – sélectionner les services arrêtés : `Get-Service | Where-Object -Property Status -eq "Stopped"`

**Group-Object** : Grouper les résultats en fonction d'une propriété

Exemple – sélectionner les services arrêtés : `Get-Service | Group-Object -Property Status`

```
PS C:\> Get-Service | Group-Object -Property Status

Count Name          Group
-----
154 Stopped {AarSvc_bf602, AJRouter, ALG, AppIDSvc...}
136 Running {AdobeARMservice, AdobeUpdateService, AGMSservice, AGSService...}

PS C:\>
```

# Interpréteur de commandes :

---

## Manipuler les résultats, il existe plusieurs Cmdlets:

**Foreach-Object** : effectuer une action pour chaque résultat reçu via le pipeline

Exemple – Action sur chaque service : `Get-Service | Foreach-Object {Action sur chaque service}`

### **Get-Service | ForEach-Object**

```
{ if ( $_.Status -eq "Stopped ") {"Service arrêtes : " + $_.Name} }
```

### **Get-Service | ForEach-Object**

```
{ if ( $_.Status -eq "Stopped ")  
    {"Service arrêtes : " + $_.Name}  
else  
    {"Service en marche : " + $_.Name}  
}  
> d:\EtatDesServices.txt
```

# Personnalisez la sortie d'une cmdlet simple

---

- Windows PowerShell fournit plusieurs applets de commande qui vous permettent de contrôler directement la sortie de données.
- Chaque applet de commande de sortie est conçue pour rediriger la sortie vers un emplacement différent:

<b>Out-Printer</b>	: Permet d'imprimer des données. Si vous ne fournissez pas de nom d'imprimante, l'applet de commande Out-Printer utilise votre imprimante par défaut.
<b>Out-File</b>	: Une sortie vers un fichier Par défaut, l'applet de commande Out-File crée un fichier Unicode
<b>Out-Host</b>	: Envoie les données à la fenêtre hôte
<b>Out-GridView</b>	: Ouvre une fenêtre avec les propriétés et les méthodes
<b>Out-Null</b>	: Ignorance de la sortie sauf si erreur.

# Get-Member

La technique la plus simple pour analyser les objets qu'une commande retourne consiste à diriger sa sortie vers l'applet de commande **Get-Member**.

L'applet de commande **Get-Member** affiche le nom formel du type d'objet et la liste complète de ses membres. Le nombre d'éléments retournés est parfois écrasant.

Par exemple, un objet de processus peut avoir plus de 100 membres.

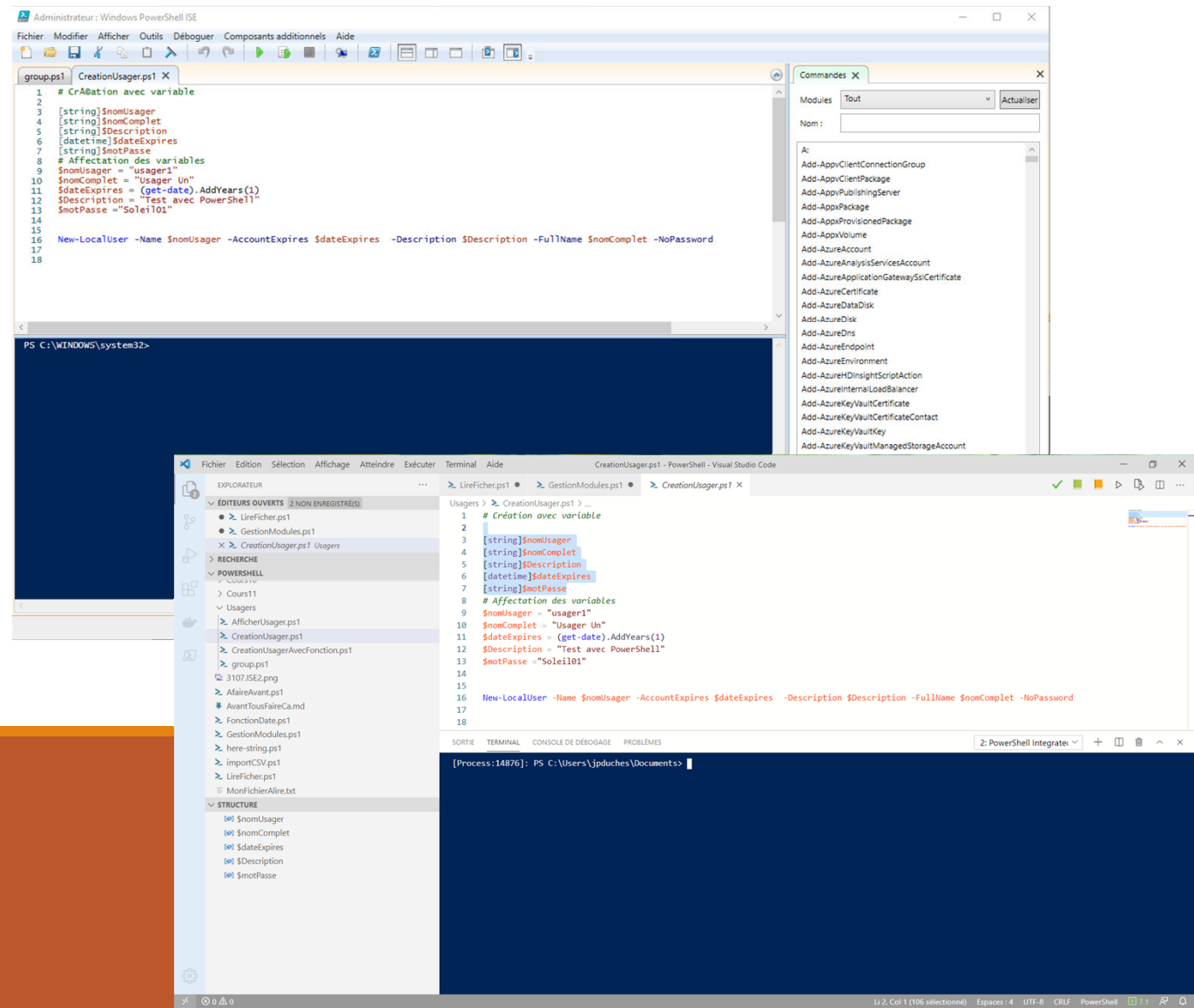
**Get-Process | Get-Member | Out-Host -Paging**

Output

TypeName: System.Diagnostics.Process

Name	MemberType	Definition
----	-----	-----
Handles	AliasProperty	Handles = Handlecount
Name	AliasProperty	Name = ProcessName
NPM	AliasProperty	NPM = NonpagedSystemMemorySize
PM	AliasProperty	PM = PagedMemorySize
VM	AliasProperty	VM = VirtualMemorySize
WS	AliasProperty	WS = WorkingSet
add_Disposed	Method	System.Void add_Disposed(Event...
...		





Visual Studio Code

# Les variables

---

## Déclaration et Manipulation des variables

- Pour rappel : Une variable est une donnée de votre script stockée en mémoire vive (RAM)
- Peut être modifié à tout moment pendant l'exécution de votre programme
- Une variable commence toujours par le signe \$ suivi du nom de la variable

### Exemple :

**\$NomVariable**

Pour affecter une valeur à une variable, on utilise le signe =

### Exemple

**\$NomVariable = Valeur**

Pour obtenir la valeur d'une variable, on utilise le nom de la variable : **\$NomVariable**

# Les variables

---

## Incrémentation et décrémentation des variables

- L'incrémentation permet d'augmenter la valeur d'une variable à chaque passage.
- La décrémentation permet de diminuer la valeur d'une variable à chaque passage.

### Incrémentation :

`$NomVariable +=1 (on donne le pas d'incrément)`

`$NomVariable++`

### Décrémentation :

`$NomVariable -=1 (on donne le pas d'incrément)`

`$NomVariable--`

# Fonction simple

---

- Une fonction permet de regrouper un ensemble d'instruction en un bloc nommé
- Elle permet de lancer des instructions sans avoir à réécrire tout le code
- Peut avoir des paramètre ou sans paramètre
- Avec PowerShell, vous avez la possibilité de créer 2 types de fonctions
  - Fonctions simples
  - Fonctions avancées, permet de reproduire le comportement d'une Cmdlet.

```
Function EnvoiProccess-VersFichier
{
param([string]$chemin)
    Get-Process | Out-File $Chemin
}
```

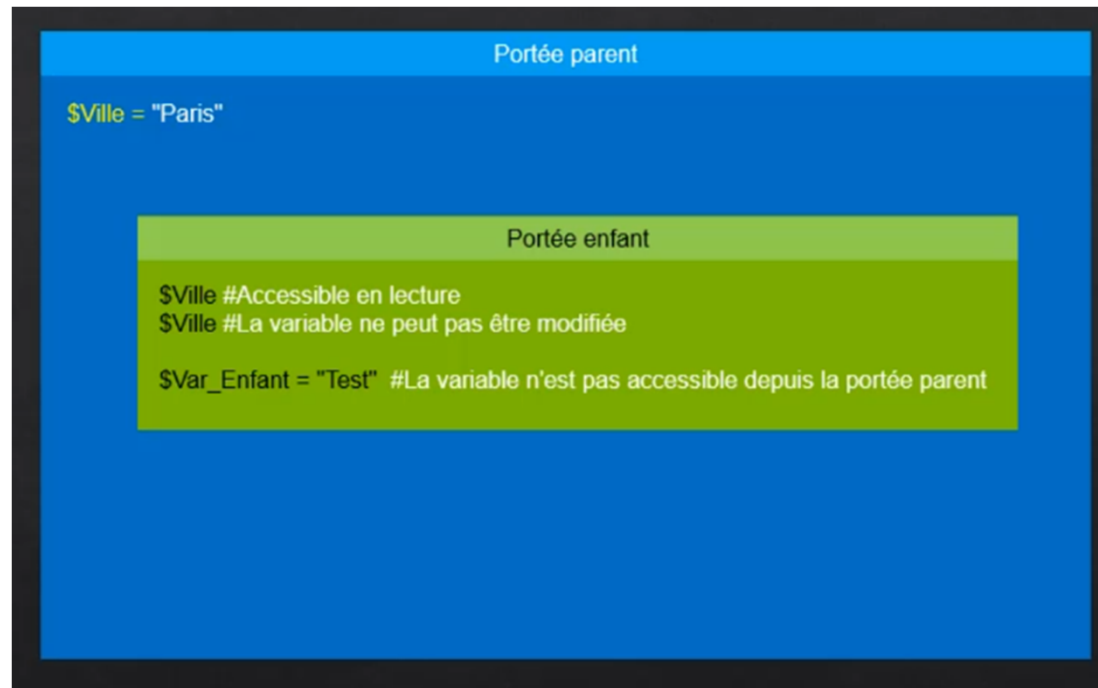
Pour utiliser la fonction simple avec paramètre on utilise la syntaxe :

```
EnvoiProccess-VersFichier -Chemin d:\Proccess.txt
```

# Les variables

---

## Portée des variables



# Écrire dans un fichier

---

```
3  # Voici un exemple pour rediriger le résultat d'une commande, ici Dir, dans un fichier
4  Dir | Out-File "C:\MonFichierDir.txt"
5
6  #Exporter un résultat dans un fichier csv
7  Get-Process | Export-csv -path D:\Export.csv -NoTypeInfoation
8
9  #Pour écrire dans le fichier on utilise ADD-content
10 ADD-content -path "C:\Fichier_de_test.txt" -value "Test d'écriture"
```

# Lire un fichier

---

```
5  Clear-Host
6
7  $EmplacementFichier = [string]
8
9  $EmplacementFichier = "D:\PowerShell\MonFichierALire.txt"
10 $MonFichier = Get-Content $emplacementFichier
11
12 foreach ($UneLigne in $MonFichier){
13     Write-Host $UneLigne
14 }
```

Ligne 5: Nettoyer la fenêtre  
Ligne 7 : Déclare et type la variable  
Ligne 9 : peupler la variable  
Ligne 10 : Déclare et peuple la variable.

Ligne 12 : Pour chaque \$UneLigne  
dans \$Monfichier.  
Affiche \$UneLigne.

# Boucle For

---

Lorsque l'on utilise une boucle for, on suit la logique suivante :

- on indique une valeur de départ (état initial),
- une valeur cible dans la condition de répétition (par exemple la valeur 10)
- et on incrémente la valeur à chaque tour de boucle (à chaque itération)  
on peut incrémenter de 1, de 2, de 10, etc... au choix.

```
For(<état initial>;<condition de répétition>;<incrémentations>)  
{  
  <Si la condition est vraie, on exécute ce bloc d'instructions>  
}  
  
<Si la condition est fausse, la boucle for se termine et le script continue...>
```