

420-436-SH Développement de scripts

P3 – Scripts PowerShell

Partie 1

Plan

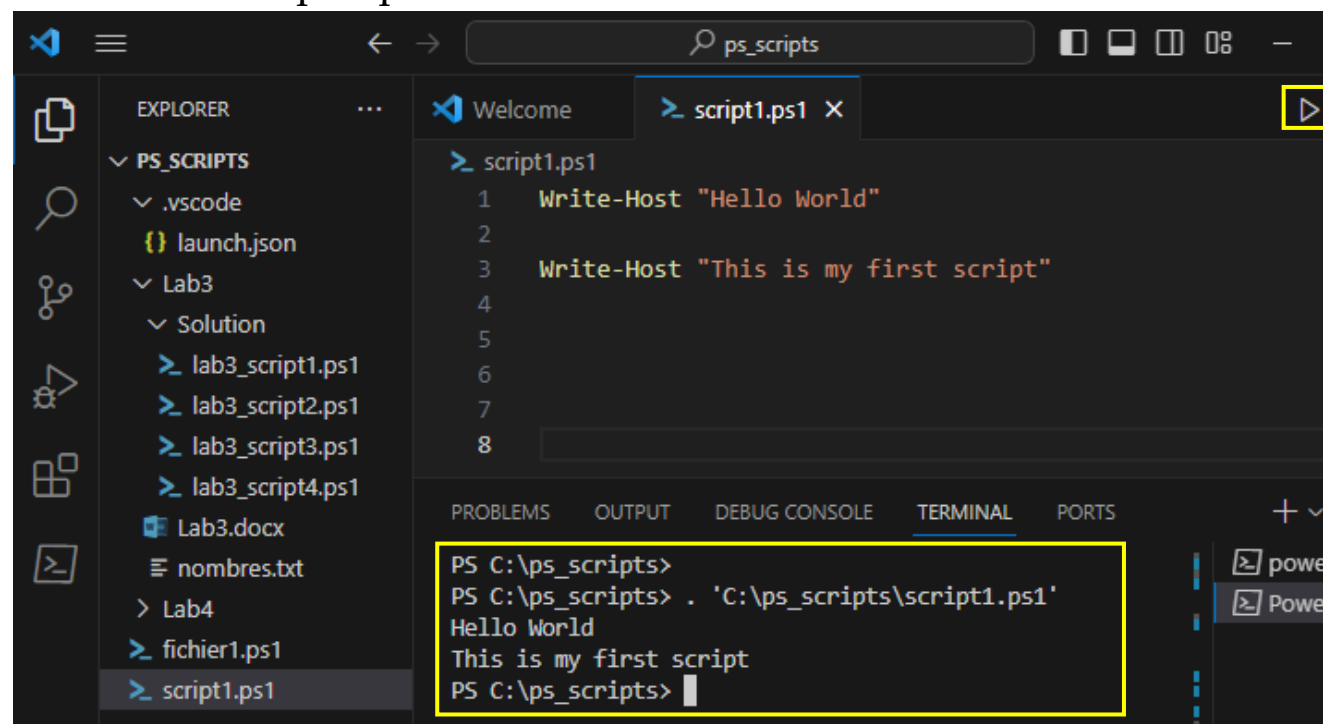
- Scripts PowerShell : généralités
- Scripts PowerShell : programmation
 - Niveaux d'*Execution Policy*
 - Options de débogage
 - Syntaxe
 - Variables
 - Substitution de commandes et de variables
 - Tableaux
 - Calculs mathématiques
 - Variables d'environnement
 - Arguments
 - Opérateurs
 - Structures de base

Scripts PowerShell (PS)

- PowerShell
 - C'est un outil d'**automatisation** et un langage de **script** utilisé principalement dans des environnements Windows
 - PowerShell est **orienté objet** et il est étroitement lié à la plateforme **.NET**
 - PowerShell est très évolué et permet d'**automatiser** des **tâches système** + des **applications** sur Windows
 - PowerShell permet la création d'**interfaces graphiques** : on peut créer nos propres apps de gestion pour les SysAdmin
 - PowerShell est **plus puissant** et **flexible** que **CMD**
 - PowerShell peut être utilisé également dans des machines **Linux**, mais avec plusieurs limitations
- Comment créer un script PowerShell ?
 - À l'aide d'un éditeur de texte (ex : **notepad++**, **notepad**, etc.)
 - À l'aide de **VSCode** + **extension PowerShell**
 - À l'aide de **PowerShell ISE** (*Integrated Scripting Environment*) : *Deprecated*

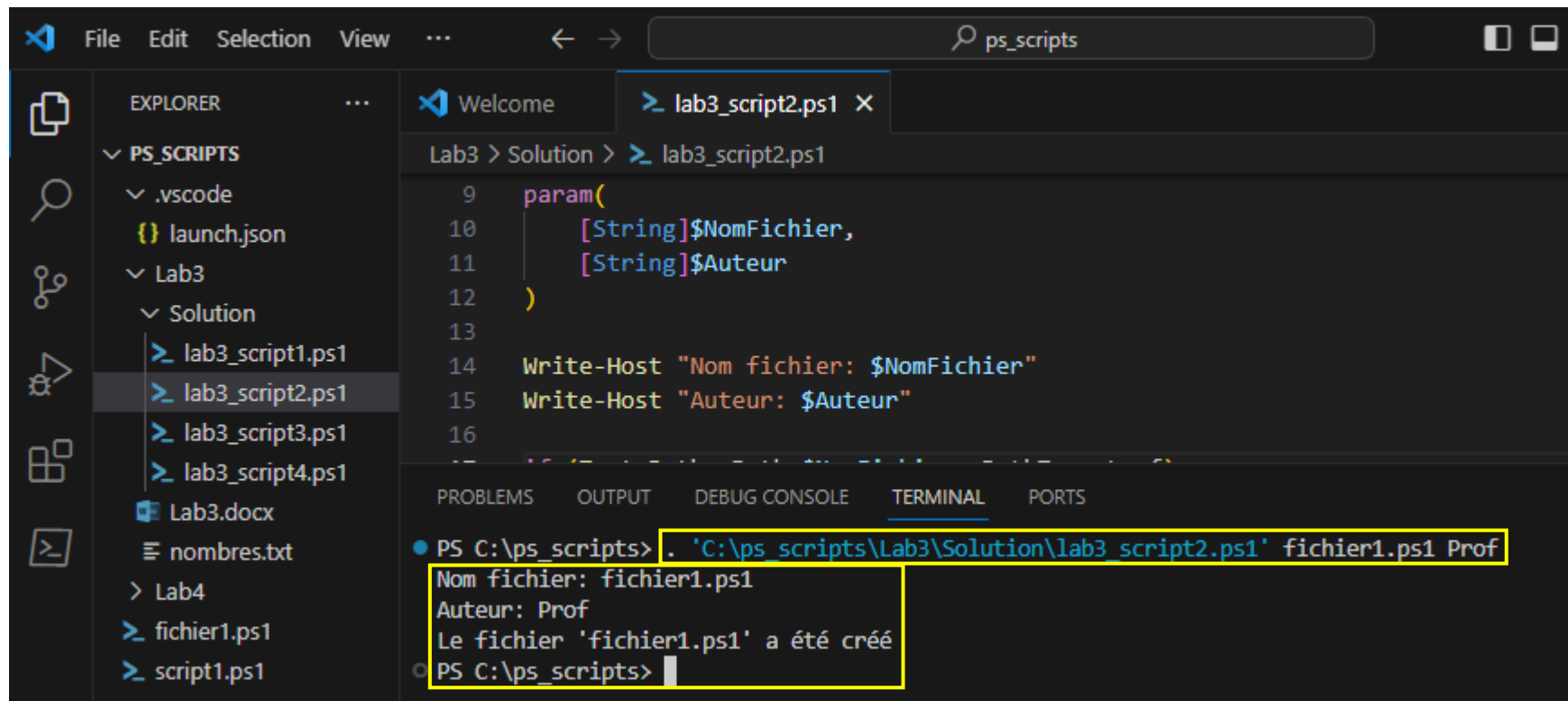
Scripts PowerShell (PS) avec VSCode

- Installer VS Code
- Installer l'extension PowerShell de Microsoft dans VSCode
- Créer le dossier **C:\ps_scripts**
- Ouvrir ce dossier dans VSCode
- Créer un script avec l'extension **.ps1**
- Exécuter le script à partir du bouton 'Exécuter'



Scripts PowerShell (PS) avec VSCode

- Exécuter un script à partir d'un terminal PowerShell de VSCode
 - Utile lorsqu'on doit passer des paramètres au script



The screenshot shows the Visual Studio Code interface with a dark theme. The Explorer sidebar on the left shows a project structure with a folder named 'PS_SCRIPTS' containing several PowerShell scripts. The file 'lab3_script2.ps1' is selected and open in the editor. The script content is as follows:

```
9 param(  
10     [String]$NomFichier,  
11     [String]$Auteur  
12 )  
13  
14 Write-Host "Nom fichier: $NomFichier"  
15 Write-Host "Auteur: $Auteur"  
16
```

At the bottom, the TERMINAL panel is active, showing the execution of the script. The command entered is:

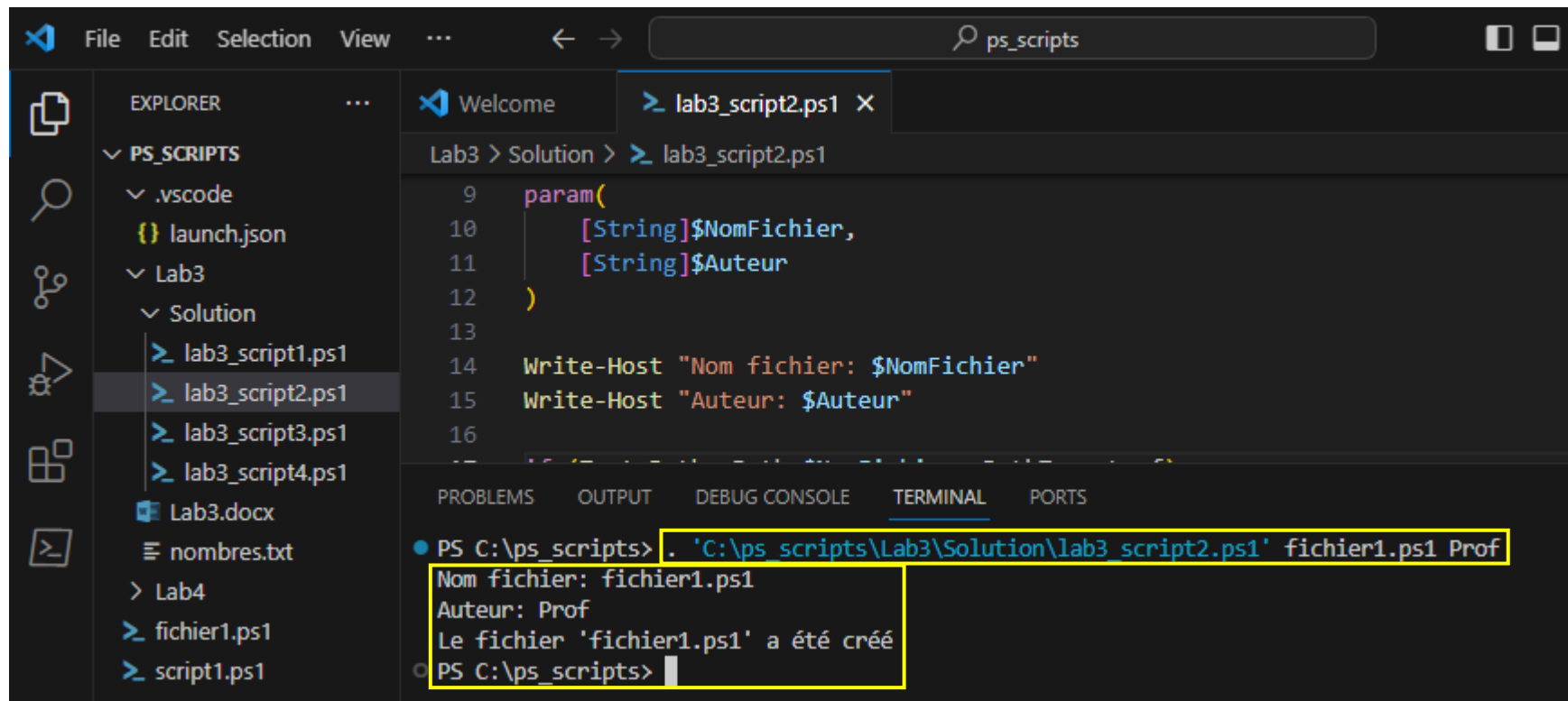
```
PS C:\ps_scripts> . 'C:\ps_scripts\Lab3\Solution\lab3_script2.ps1' fichier1.ps1 Prof
```

The output of the script is displayed below the command:

```
Nom fichier: fichier1.ps1  
Auteur: Prof  
Le fichier 'fichier1.ps1' a été créé
```

Scripts PowerShell (PS)

- Exécuter un script à partir d'un terminal PowerShell



The screenshot shows the Visual Studio Code interface with a PowerShell script named `lab3_script2.ps1` open in the editor. The script contains the following code:

```
9 param(  
10     [String]$NomFichier,  
11     [String]$Auteur  
12 )  
13  
14 Write-Host "Nom fichier: $NomFichier"  
15 Write-Host "Auteur: $Auteur"  
16
```

The Explorer sidebar on the left shows the file structure with `lab3_script2.ps1` selected. The Terminal panel at the bottom shows the execution of the script:

```
PS C:\ps_scripts> . 'C:\ps_scripts\Lab3\Solution\lab3_script2.ps1' fichier1.ps1 Prof  
Nom fichier: fichier1.ps1  
Auteur: Prof  
Le fichier 'fichier1.ps1' a été créé  
PS C:\ps_scripts>
```

The command and its output are highlighted with yellow boxes in the original image.

Scripts PowerShell (PS)

- Comment exécuter un script *PowerShell* ?
 - Exécution – méthode 1 : `.\script1.ps1` (Dans l'environnement *PowerShell*)
 - Exécution – méthode 1 (script avec paramètres) : `.\script2.ps1 param1 param2`
 - Exécution – méthode 2 : `powershell -file script1.ps1` (Dans l'environnement *CMD*)
- Permissions d'exécution de scripts
 - Les droits d'exécution sont ceux de l'utilisateur qui lance le script
 - Si on a besoin des droits *Administrateur*, on doit ouvrir l'environnement « en tant qu'Administrateur »
 - De plus, afin d'améliorer la sécurité du système, l'exécution se fait selon une stratégie d'exécution (*Execution Policy*)
- *Execution Policy (EP)*
 - Son but est de prévenir l'exécution, par mégarde, de commandes « dangereuses »
 - L'EP peut être modifiée ou contournée au besoin
 - Une EP est associée à la machine, une autre EP est associée à l'utilisateur et une autre EP à la session courante

Niveaux d'*Execution Policy* (EP)

- *AllSigned*
 - Toutes les scripts (locaux et téléchargés) doivent avoir une signature électronique vérifiée par une CA*
- *Bypass*
 - Aucune restriction d'exécution. La gestion des permissions est gérée par d'autres applications
- *RemoteSigned*
 - Les scripts téléchargés doivent avoir une signature électronique vérifiée par une CA*
- *Restricted*
 - Les scripts ne peuvent pas être exécutés
- *Undefined*
 - Pas d'EP assigné à l'environnement d'exécution actuel
- *Unrestricted*
 - Aucune restriction d'exécution. Non recommandé, des scripts malveillants peuvent être exécutés.

```
PS C:\Windows\system32> Set-ExecutionPolicy RemoteSigned

Execution Policy Change
The execution policy helps protect you from scripts that you do not trust. Changing the execution policy might expose
you to the security risks described in the about_Execution_Policies help topic at
https://go.microsoft.com/fwlink/?LinkID=135170. Do you want to change the execution policy?
[Y] Yes [A] Yes to All [N] No [L] No to All [S] Suspend [?] Help (default is "N"): A
PS C:\Windows\system32>
```


Programmation de scripts *PS*

Exemple 1

```
1 # Cette commande fait un clear screen
2 cls
3
4 # Cette commande retourne le chemin du dossier courant
5 Get-Location
6
7
8 # Cette commande affiche le contenu du dossier courant
9 Get-ChildItem
```

Indique le chemin et le nom du dossier courant

Retourne la liste d'objets qui se trouvent dans le dossier courant

Résultat de l'exécution du script

```
Path
----
C:\scripts

LastWriteTime : 2023-02-05 11:30:37 AM
Length        : 0
Name          : fichier.txt

LastWriteTime : 2023-02-05 9:49:42 AM
Length        : 60
Name          : script1.ps1
```

Programmation de scripts PS

Exemple 1

Même script de la page précédente mais on améliore l'affichage

```
1  # -----
2  # Script : script1.ps1
3  # Auteur : AlexJ
4  # Description: Affiche le dossier courant et son contenu
5  # Paramètres :
6  # Date : 2023-01-02
7  # -----
8
9  # Cette commande fait un clear screen
10 cls
11
12 # Cette commande retourne le chemin du dossier courant
13 Write-Host "*** Dossier courant : $(Get-Location)"
14
15 # Cette commande affiche le contenu du dossier courant
16 Write-Host "*** Contenu du dossier courant : "
17 (Get-ChildItem).BaseName
18
19
20 <#
21 Ceci est un bloc de commentaires. Ce bloc ne sera pas exécuté.
22 Très utile lorsqu'on veut mettre en commentaire plusieurs lignes
23 #>
```

Entête

Exécution d'une commande et utilisation de son résultat

Afficher uniquement le nom de chaque élément dans le dossier courant

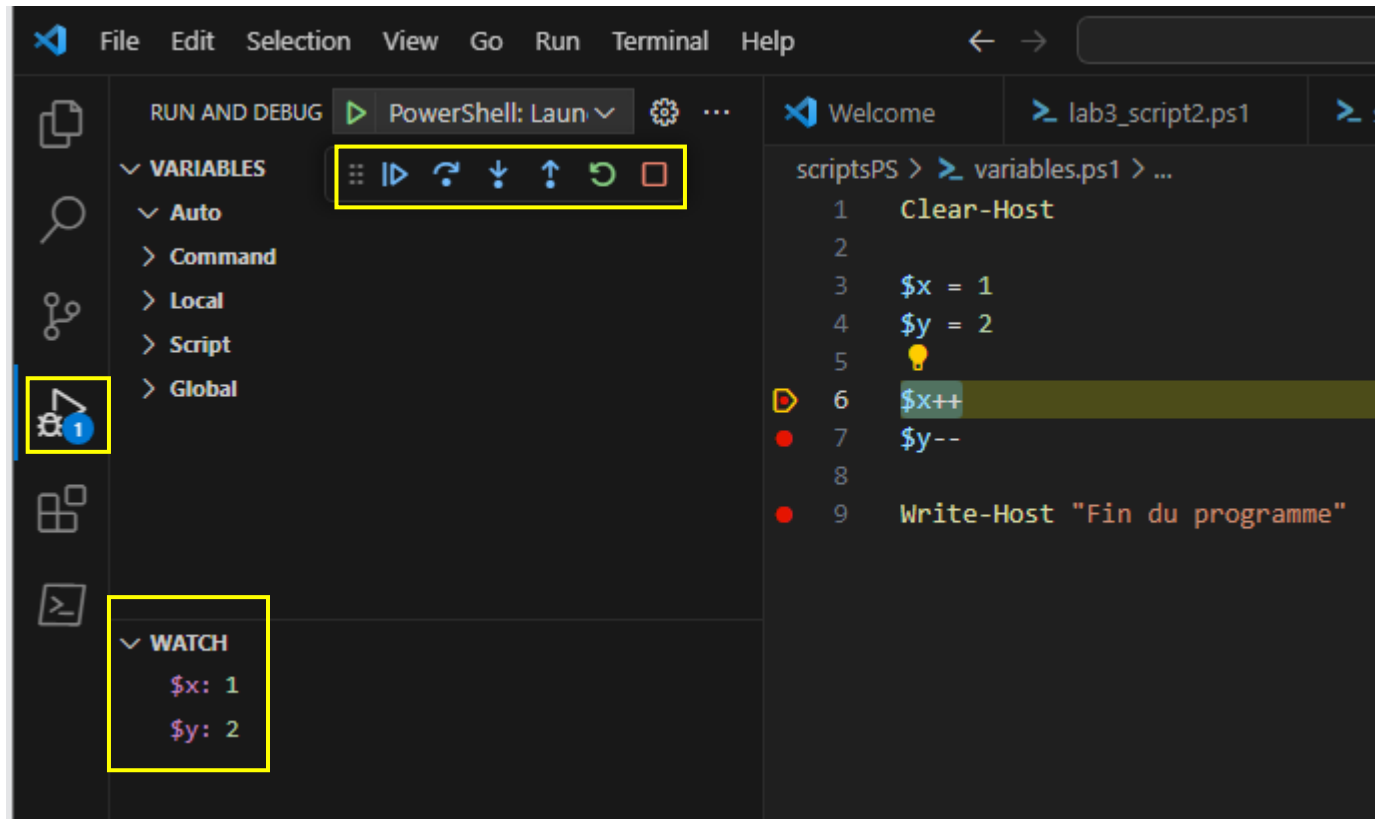
Commentaire sur plusieurs lignes

Résultat de l'exécution du script

```
*** Dossier courant : C:\scripts
*** Contenu du dossier courant :
fichier
script1
```

Programmation de scripts PS

- Débogage



Src: <https://code.visualstudio.com/docs/editor/debugging>

Debug actions

Once a debug session starts, the Debug toolbar will appear on the top of the editor.



Action	Explanation
Continue / Pause F5	Continue: Resume normal program/script execution (up to the next breakpoint). Pause: Inspect code executing at the current line and debug line-by-line.
Step Over F10	Execute the next method as a single command without inspecting or following its component steps.
Step Into F11	Enter the next method to follow its execution line-by-line.
Step Out Shift+F11	When inside a method or subroutine, return to the earlier execution context by completing remaining lines of the current method as though it were a single command.
Restart Ctrl+Shift+F5	Terminate the current program execution and start debugging again using the current run configuration.
Stop Shift+F5	Terminate the current program execution.

Programmation de scripts PS

- Création de variables
 - PS est un langage typé à la base, mais il fonctionne comme un langage **non typé** : il n'est pas obligatoire d'indiquer le type de variable lors de sa **déclaration** (création)
 - On peut indiquer le type de variable lors de sa déclaration
 - Si le type n'est pas déclaré, celui-ci est défini implicitement lors de la première utilisation de la variable

Déclaration typée en PS

```
[int]$a = 3  
[float]$b = 5.47  
[string]$c = "Bonjour"  
[bool]$d = $false
```

Déclaration non-typée en PS

```
$a = 3  
$b = 5.47  
$c = "Bonjour"  
$d = $false
```

- Utilisation des variables

```
Write-Host $a  
Write-Host "Voici la valeur de a : $a"  
Write-Host $b  
Write-Host $c  
Write-Host $d
```

Résultat

```
3  
Voici la valeur de a : 3  
5.47  
Bonjour  
False
```

Programmation de scripts PS

Substitution de commandes et de variables :

- Création d'un fichier dans un dossier

```
New-Item -Path C:\Users\jaramial\Documents -Name monFichier.txt -ItemType File
```

- \$()** : technique pour exécuter une commande et se servir du résultat dans une autre commande. Exemple :

```
New-Item -Path $(Get-Location) -Name monFichier.txt -ItemType File
```

La commande **Get-Location** retourne le dossier courant. Ce résultat est utilisé par la commande **New-Item** pour créer un fichier

- \$()** : utilisé également lorsqu'une variable doit être concaténée avec d'autres éléments. Exemple :

```
$prefix = "420"  
Write-Host "Le code du cours est $prefix_436_SH"  
Write-Host "Le code du cours est ($prefix)_436_SH"  
Write-Host "Le code du cours est $($prefix)_436_SH"
```

Résultat

Le code du cours est

Le code du cours est (420)_436_SH

Le code du cours est 420_436_SH

Programmation de scripts PS

- Tableaux : PS peut manipuler des tableaux facilement

```
$tableauFruits = "Pomme", "Orange", "Banane"
$tableauFruits = @("Pomme", "Orange", "Banane")
```

```
Write-Host $tableauFruits[0]
Write-Host $tableauFruits[1]
Write-Host $tableauFruits[2]
Write-Host $tableauFruits
```

Résultat

```
Pomme
Orange
Banane
Pomme Orange Banane
```

- Variables d'environnement

- Variables présentes dans le système qui peuvent être accédées à partir de la console ou à partir de scripts.

Obtenir les variables d'environnement :

```
PS C:\scripts> dir env:

Name                           Value
----                           -
ALLUSERSPROFILE                C:\ProgramData
APPDATA                        C:\Users\jaramial.CSH\AppData\Roaming
CDROMDrive                     NoCDROM
CommonProgramFiles             C:\Program Files\Common Files
CommonProgramFiles(x86)       C:\Program Files (x86)\Common Files
CommonProgramW6432            C:\Program Files\Common Files
COMPUTERNAME                   PCS027
ComSpec                        C:\Windows\system32\cmd.exe
Devmgr_Show_NonPresent_Devices 1
DriverData                    C:\Windows\System32\Drivers\DriverData
HOMEDRIVE                      C:
HOMEPATH                      \Users\jaramial.CSH
IntelliJ IDEA Community Edi... C:\Program Files\JetBrains\IntelliJ IDEA Community Edi...
LOCALAPPDATA                   C:\Users\jaramial.CSH\AppData\Local
LOGONSERVER                    \\CD1
MOZILLAUSERLOCAL               fr
NUMBER_OF_PROCESSORS           12
OneDrive                       C:\Users\jaramial.CSH\OneDrive - Cégep de Sherbrooke
OneDriveCommercial             C:\Users\jaramial.CSH\OneDrive - Cégep de Sherbrooke
OneDriveConsumer               C:\Users\jaramial.CSH\OneDrive
OS                              Windows_NT
Path                           C:\Program Files (x86)\VMware\VMware Workstation\bin\;
```

Créer une variable d'environnement :

```
$env:NAME = VALUE
```

```
PS C:\scripts> $env:MA_VARIABLE = 'Ceci est la valeur de la variable'
PS C:\scripts> Write-Host $env:MA_VARIABLE
Ceci est la valeur de la variable
```

Ajouter un dossier à la variable d'environnement PATH:

```
PS C:\scripts> $env:Path = 'C:\scripts;' + $env:Path
```

Programmation de scripts *PS*

Calculs mathématiques

```
# calcul mathématique simple
$a = 3
$b = 2
$sum = $a + $b

Write-Host "La somme de $a et $b est $sum"
```

Résultat

La somme de 3 et 2 est 5

```
$puissance = [Math]::Pow($a, $b)
Write-Host "$a ^ $b = $puissance"
```

Résultat

 $3 \wedge 2 = 9$

```
$racine_carree = [math]::Round([Math]::Sqrt($a), 2)
Write-Host "La racine carrée de $a est $racine_carree"
```

Résultat

La racine carrée de 3 est 1.73

Programmation de scripts *PS*

Opérateurs arithmétiques

Operator	Description	Example	a=10 b=20
+ (Addition)	Adds values on either side of the operator.	A + B will give 30	
- (Subtraction)	Subtracts right-hand operand from left-hand operand.	A - B will give -10	
* (Multiplication)	Multiplies values on either side of the operator.	A * B will give 200	
/ (Division)	Divides left-hand operand by right-hand operand.	B / A will give 2	
% (Modulus)	Divides left-hand operand by right-hand operand and returns remainder.	B % A will give 0	

Programmation de scripts *PS*

Opérateurs booléens

Operator	Description	Example
eq (equals)	Compares two values to be equal or not.	A -eq B will give false
ne (not equals)	Compares two values to be not equal.	A -ne B will give true
gt (greater than)	Compares first value to be greater than second one.	B -gt A will give true
ge (greater than or equals to)	Compares first value to be greater than or equals to second one.	B -ge A will give true
lt (less than)	Compares first value to be less than second one.	B -lt A will give false
le (less than or equals to)	Compares first value to be less than or equals to second one.	B -le A will give false

a=10
b=20

Programmation de scripts PS

Opérateurs booléens

Operator	Description	Example
AND (logical and)	Called Logical AND operator. If both the operands are non-zero, then the condition becomes true.	(A -AND B) is false
OR (logical or)	Called Logical OR Operator. If any of the two operands are non-zero, then the condition becomes true.	(A -OR B) is true
NOT (logical not)	Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false.	-NOT(A -AND B) is true

a=10
b=20

Autres comparateurs

-like	wildcard comparison
-notlike	wildcard comparison
-match	Regular expression comparison
-notmatch	Regular expression comparison
-replace	Replace operator
-contains	Containment operator
-notcontains	Containment operator
-in	Like -contains, but with the operands reversed.(PowerShell 3.0)
-notin	Like -notcontains, but with the operands reversed.(PowerShell 3.0)

Src : https://www.tutorialspoint.com/powershell/powershell_operators.htm

Src : <https://ss64.com/ps/syntax-compare.html>

Programmation de scripts PS

Paramètres (arguments) d'un Script

- **Paramètre** : une donnée que l'on fournit à un script
- Il est possible de fournir plusieurs paramètres à un script
- Les paramètres sont fournis au moment où le script est exécuté

```
PS C:\scripts> .\script3.ps1 param1 param2 param3
```

Appel d'un script **script3.ps1** en indiquant des paramètres

- Code du fichier script3.ps1

```
Write-Host "***** Paramètre du script"  
Write-Host "Nombre de paramètres en entrée : $($args.count)"  
Write-Host "La valeur du premier paramètre (s'il y en a un) : $($args[0])"
```

Pour connaître le nombre de paramètres en entrée

Pour connaître la valeur du premier paramètre

- Résultat de l'exécution du script

```
***** Paramètre du script  
Nombre de paramètres en entrée : 3  
La valeur du premier paramètre (s'il y en a un) : param1
```

Programmation de scripts PS

Paramètres (arguments) d'un Script

- On peut gérer les paramètres en tant que variables individuelles dans un script
- Code du fichier script5.ps1

```
1 param(  
2     [String]$Nom,  
3     [int]$Age  
4 )  
5  
6 Write-Host "Bonjour $($Nom)"  
7 Write-Host "Vous avez $($Age) ans"
```

- Appel du script:

```
PS C:\scripts> .\script5.ps1 Julie 25
```

- Résultat de l'exécution du script

```
Bonjour Julie  
Vous avez 25 ans
```

Programmation de scripts PS

Quelques variables réservées et commandes utiles

\$MyInvocation.MyCommand.Name : pour obtenir le nom du fichier du script en exécution

\$args : un tableau contenant les paramètres reçus

\$PID : numéro du processus en cours d'exécution

\$? : état d'exécution de la dernière commande (« True » si la commande s'est bien exécuté)

\$_ : contient l'objet courant présent dans le pipeline

\$error : un tableau contenant les dernières erreurs d'exécution

\$this : la propriété ou la fonction du script qui est traitée actuellement

\$null : représente la valeur NULL

Liste complète : https://learn.microsoft.com/en-us/powershell/module/microsoft.powershell.core/about/about_automatic_variables?view=powershell-7.3

Programmation de scripts PS

Quelques variables réservées et commandes

```
PS C:\scripts> .\script_reserved_variables.ps1 param1 param2 param3
```

Appel d'un script en indiquant des paramètres

```
Write-Host "Nom du script : $($MyInvocation.MyCommand.Name)"
Write-Host "Nombre de paramètres envoyé à ce script : $($args.count)"
Write-Host "Liste de paramètres envoyé à ce script : $($args)"
Write-Host "Premier paramètre envoyé à ce script : $($args[0])"
Write-Host "Deuxième paramètre envoyé à ce script : $($args[1])"
Write-Host "Numéro du processus sous lequel ce script est exécuté : $($PID)"
```

Code source du script

```
ls
$?

ls fichier_non_present
$?
```

```
Nom du script : script_reserved_variables.ps1
Nombre de paramètres envoyé à ce script : 3
Liste de paramètres envoyé à ce script : param1 param2 param3
Premier paramètre envoyé à ce script : param1
Deuxième paramètre envoyé à ce script : param2
Numéro du processus sous lequel ce script est exécuté : 54848
```

Résultat

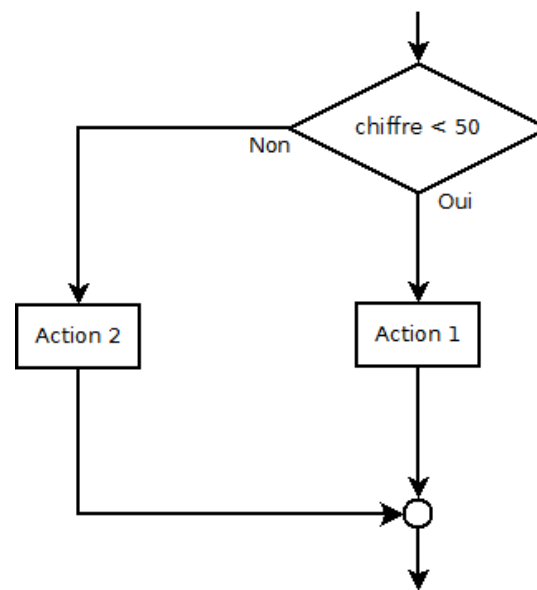
```
Directory: C:\scripts

Mode                LastWriteTime         Length Name
----                -
-a----         2023-02-05 11:30 AM              0 fichier.txt
-a----         2023-02-05  1:34 PM              0 monFichier.txt
-a----         2023-02-05  9:49 AM             60 script1.ps1
-a----         2023-02-06  8:37 AM            224 script3.ps1
-a----         2023-02-06  9:26 AM            857 script_reserved_variables.ps1
True
ls : Cannot find path 'C:\scripts\fichier_non_present' because it does not exist.
At C:\scripts\script_reserved_variables.ps1:16 char:1
+ ls fichier_non_present
+ ~~~~~
+ CategoryInfo          : ObjectNotFound: (C:\scripts\fichier_non_present:String) [Get-ChildItem], It
emNotFoundException
+ FullyQualifiedErrorId : PathNotFound,Microsoft.PowerShell.Commands.GetChildItemCommand

False
```

Programmation de scripts PS

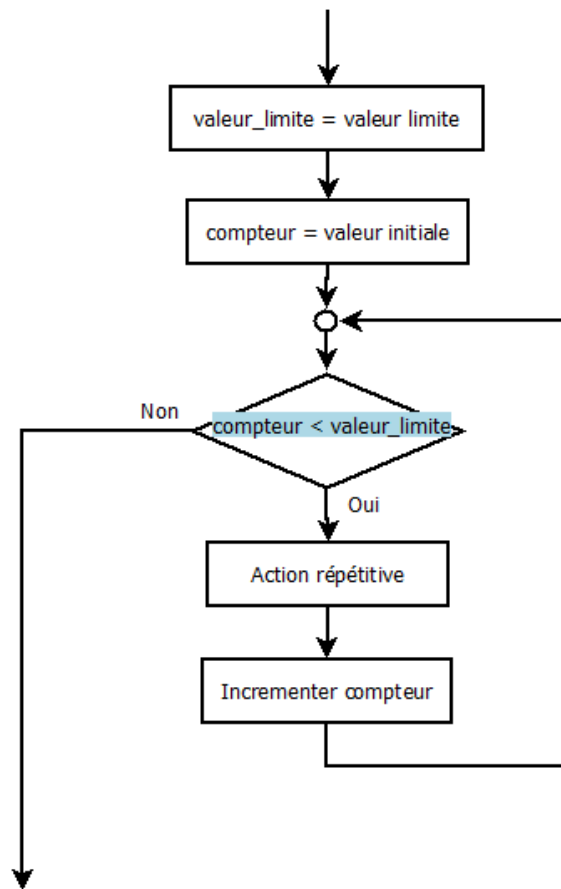
Codage des structures logiques : **IF...ELSE**



```
if($chiffre1 -lt 50) {  
    Write-Host "Le chiffre est plus petit que 50"  
}else {  
    Write-Host "Le chiffre n'est pas plus petit que 50"  
}
```

Programmation de scripts PS

Codage des structures logiques : Boucle **WHILE**



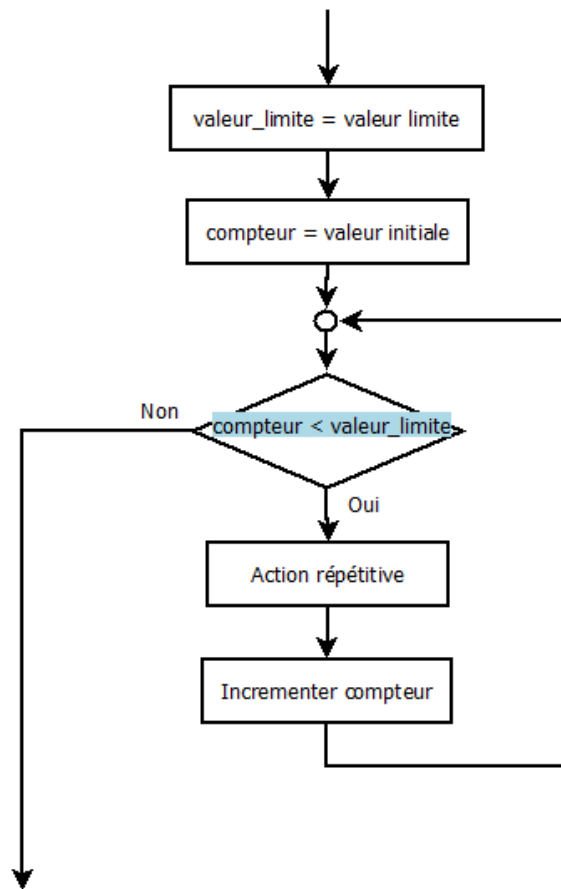
```
$valeurLimite = 50
$compteur = 0

while($compteur -lt $valeurLimite)
{
    write-Host "Le compteur est rendu à $($compteur)"
    $compteur++
}
```

```
Le compteur est rendu à 0
Le compteur est rendu à 1
Le compteur est rendu à 2
Le compteur est rendu à 3
Le compteur est rendu à 4
Le compteur est rendu à 5
Le compteur est rendu à 6
Le compteur est rendu à 7
Le compteur est rendu à 8
Le compteur est rendu à 9
Le compteur est rendu à 10
...
Le compteur est rendu à 48
Le compteur est rendu à 49
Merci !
```


Programmation de scripts PS

Codage des structures logiques : Boucle **FOR**

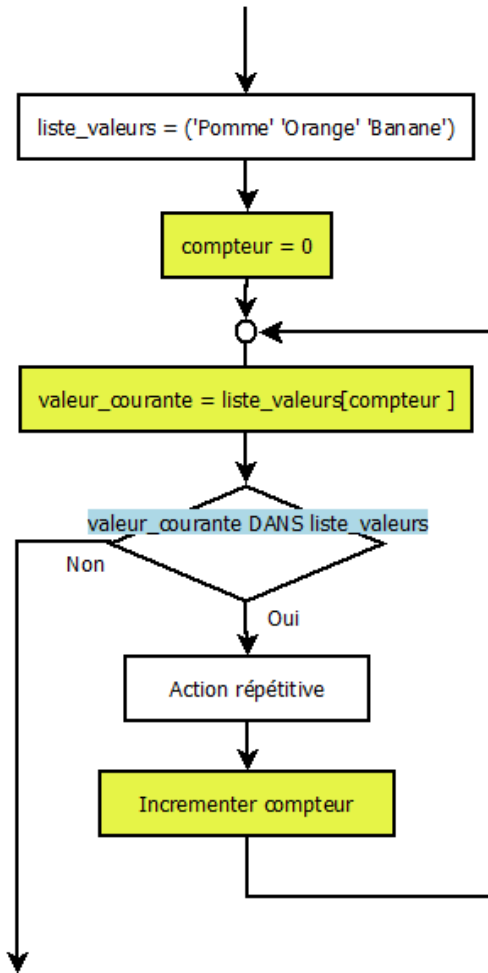


```
$valeurLimite = 50  
  
for($compteur = 0; $compteur -lt $valeurLimite; $compteur++)  
{  
    write-Host "Le compteur est rendu à $($compteur)"  
}
```

```
Le compteur est rendu à 0  
Le compteur est rendu à 1  
Le compteur est rendu à 2  
Le compteur est rendu à 3  
Le compteur est rendu à 4  
Le compteur est rendu à 5  
Le compteur est rendu à 6  
Le compteur est rendu à 7  
Le compteur est rendu à 8  
Le compteur est rendu à 9  
Le compteur est rendu à 10  
...  
Le compteur est rendu à 48  
Le compteur est rendu à 49  
Merci !
```

Programmation de scripts PS

Codage des structures logiques : Boucle **FOR**



```
$liste_valeurs = "Pomme", "Orange", "Banane"
foreach ($valeur_courante in $liste_valeurs)
{
    Write-Host "Fruit = $($valeur_courante)"
}
```

```
Fruit = Pomme
Fruit = Orange
Fruit = Banane
```

Programmation de scripts Bash

Codage des structures logiques : Boucle **FOR** / **FOREACH**

```
foreach ($i in (1..10))  
{  
    Write-Host $i  
}
```

```
i = 1  
i = 2  
i = 3  
i = 4  
i = 5  
i = 6  
i = 7  
i = 8  
i = 9  
i = 10
```

```
Write-Host "Compte à rebours..."  
  
for ($i = 30; $i -ge 0; $i -= 3)  
{  
    Write-Host "i = $i"  
}
```

```
foreach ($i in 30..0)  
{  
    if ($i % 3 -eq 0) {  
        Write-Host "i = $i"  
    }  
}
```

```
i = 30  
i = 27  
i = 24  
i = 21  
i = 18  
i = 15  
i = 12  
i = 9  
i = 6  
i = 3  
i = 0
```

```
$unMot = "Bonjour"  
  
foreach ($lettre in $unMot.ToCharArray())  
{  
    Write-Host $lettre  
}
```

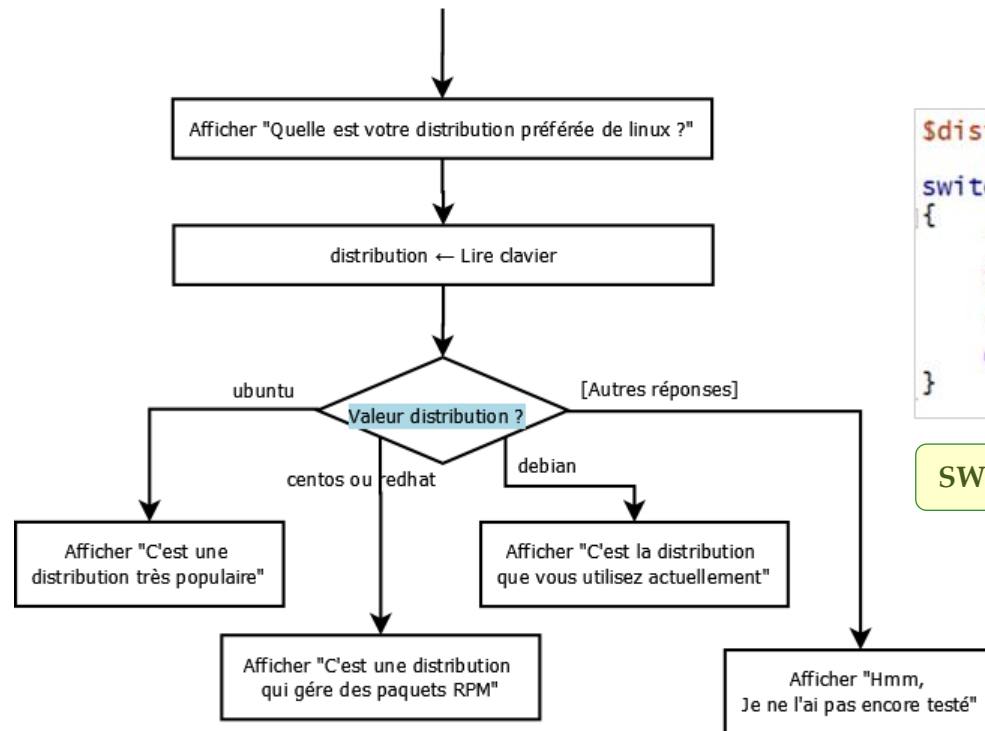
```
B  
o  
n  
j  
o  
u  
r
```

```
$i = 1  
  
foreach ($day in @('Lundi', 'Mardi', 'Mercredi', 'Jeudi', 'Vendredi'))  
{  
    Write-Host "JourSem $i = $day"  
    $i++  
}
```

```
JourSem 1 : Lundi  
JourSem 2 : Mardi  
JourSem 3 : Mercredi  
JourSem 4 : Jeudi  
JourSem 5 : Vendredi
```

Programmation de scripts PS

Codage des structures logiques : **SWITCH...CASE**



```
$distribution = "kali"
switch ($distribution)
{
    'Ubuntu' {"C'est une distribution très populaire"; break}
    {'Centos', 'Red Hat' -eq $_} {"C'est une distribution qui gère des paquets RPM"; break}
    'Debian' {"C'est une excellente distribution"; break}
    'Kali' {"C'est une distribution pour les test en sécurité"; break}
    default {"Hmm. Je ne l'ai pas encore testée"}
}
```

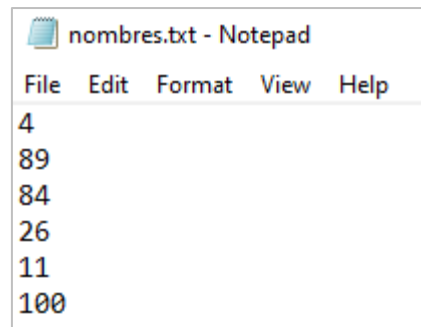
SWITCH...CASE n'est pas *case sensitif* par défaut en PowerShell

Programmation de scripts PS

- Manipulation de fichiers

```
1  # ----- Lecture d'une fichier et placement de son contenu dans un tableau -----  
2  
3  
4  # Le contenu du fichier 'nombres.txt' est chargé dans le tableau '$contenuFichier'  
5  $contenuFichier = Get-Content nombres.txt  
6  
7  # Affichage du contenu du tableau sur écran  
8  foreach ($valeur in $contenuFichier)  
9  {  
10     Write-Host $valeur  
11 }  
12
```

Contenu fichier :

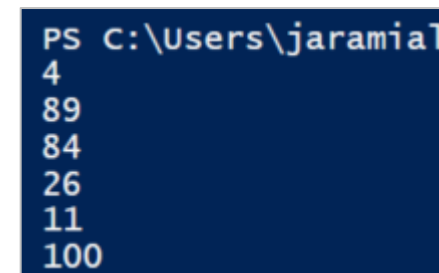


nombres.txt - Notepad

File Edit Format View Help

4
89
84
26
11
100

Affichage :



```
PS C:\Users\jaramial...  
4  
89  
84  
26  
11  
100
```

Programmation de scripts PS

- Manipulation de fichiers

```
14 # ----- Écriture d'une valeur dans un fichier -----
15
16 $nom = "Diane"
17
18 # Écrire le contenu de la variable '$nom' dans le fichier 'noms.txt'.
19 # Si le fichier n'existe pas, il est créé automatiquement
20 $nom | Out-File -FilePath noms.txt
21
22 $nom = "Charles"
23
24 # Le nouveau contenu envoyé vers le fichier 'noms.txt' écrase l'ancien contenu de ce fichier
25 $nom | Out-File -FilePath noms.txt
26
27 $nom = "Mike"
28
29 # Le nouveau contenu envoyé vers le fichier 'noms.txt' s'ajoute à l'ancien contenu de ce fichier
30 $nom | Out-File -FilePath noms.txt -Append
31
```

noms.txt - Notepad

File Edit Format View

Diane

noms.txt - Notepad

File Edit Format View

Charles

noms.txt - Notepad

File Edit Format View

Charles
Mike

Programmation de scripts *PS*

- Manipulation de fichiers

```
33 # ----- Ecriture d'un tableau dans un fichier -----  
34  
35 $fruits1 = "Pomme", "Orange", "Banane"  
36  
37 $fruits1 | Out-File fruits.txt  
38  
39 $fruits2 = "Kiwi", "Coconut", "Goyave"  
40  
41 $fruits2 | Out-File fruits.txt -Append  
42
```

fruits.txt - Notepad

File	Edit	Format	View
Pomme			
Orange			
Banane			

fruits.txt - Notepad

File	Edit	Format	View
Pomme			
Orange			
Banane			
Kiwi			
Coconut			
Goyave			

Références intéressantes

- Windows PowerShell Reference

<https://learn.microsoft.com/en-us/powershell/scripting/developer/windows-powershell-reference?view=powershell-7.3>

- *SS64 – PowerShell Commands*

<https://ss64.com/ps/>

- *Powershell Tutorial*

<https://www.tutorialspoint.com/powershell/index.htm>