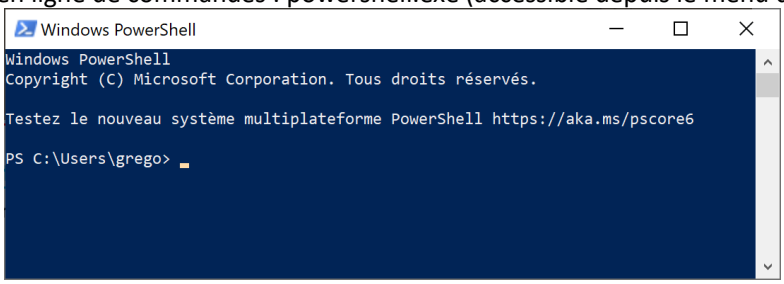
# Scripting avec PowerShell

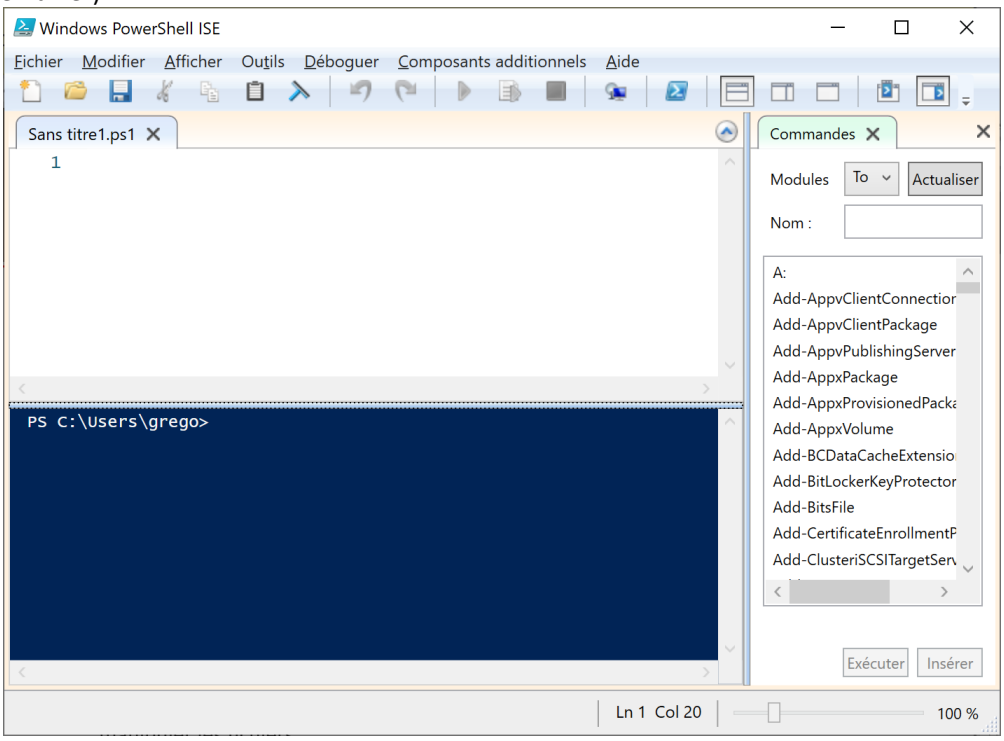
## Présentation

PowerShell est un langage de scripts développé par Microsoft en 2006. Il tend à remplacer l'ancienne invite de commandes MS-DOS. Il est présent dans toutes les versions de Windows.

Il comprend une interface en ligne de commandes : powershell.exe (accessible depuis le menu démarrer).




Il comprend aussi un éditeur de scripts plus évolué (avec des couleurs, l'auto-complétion…) : PowerShell ISE (accessible aussi depuis le menu démarrer).




Configuration préliminaire :

Dans PowerShell, il existe quatre paramètres de stratégie d'exécution des scripts qui sont :

| | |
|---|---|
| Restricted | : paramètre par défaut, n'autorise pas l'exécution des scripts, |
| AllSigned | : n'exécute que les scripts de confiance, donc signés, |
| RemoteSigned | : exécute les scripts locaux sans obligation de confiance et les scripts de confiance issus d'Internet, |
| Unrestricted | : autorise l'exécution de tous les scripts. |

Démarrer Windows PowerShell en tant qu'administrateur, puis utiliser la commande suivante pour définir la stratégie :

- Commande pour connaître la stratégie en cours : Get-ExecutionPolicy
- Commande pour modifier la stratégie : Set-ExecutionPolicy Unrestricted

## Principales utilisations

1. Interagir avec le système d'exploitation (sans interface graphique) : par exemple, lancer un processus (la calculatrice…), manipuler le système de fichiers (créer un dossier, déplacer un fichier…).
2. Utiliser les options avancées du système : toutes les fonctionnalités n'ont pas été implémentées en interface graphique ; ainsi, on a souvent des fonctionnalités supplémentaires en ligne de commandes.
3. Interagir avec les applications : par exemple, ajouter des utilisateurs à Active Directory, modifier le site Web d'un serveur de fichiers…
4. Automatiser des tâches : via les fondamentaux de la programmation (conditions, boucles, variables…), on pourra exécuter des tâches répétitives.

## 1) Découverte de la console

Les touches les plus intéressantes en mode invite de commandes (console) :

| Touche | Description |
|---|---|
| [Flèche en haut]<br>[Flèche en bas] | Permet de faire défiler l'historique des commandes déjà frappées. |
| [F8] | Fait défiler l'historique sur la ligne de commande. |
| [Ctrl] C | Met fin à l'exécution de l'instruction courante. |

En mode ISE :

| Touche | Description |
|---|---|
| [F5] | Exécuter le script |
| [F8] | Exécuter la sélection uniquement (ou la ligne active) |

La touche tabulation [tab] permet de compléter le nom des commandes, le nom des paramètres et les chemins d'accès aux fichiers et dossiers.

L'action successive de la touche tabulation liste les éléments commençant par les caractères spécifiés.

## 2) Bases

### A) L'affichage

```
Write-Host "Texte à afficher"
```

Remarque : les guillemets ne sont pas toujours obligatoires ; la fonction d'affichage peut être remplacée par « echo » ou parfois il n'y a même pas besoin de l'écrire…

Pour écrire un commentaire, il faut mettre un hashtag en début de ligne.
Exemple :

```
#Ceci est un commentaire
```

### B) La saisie

```
$variableRecupereSaisie = Read-Host "Texte à afficher"
```

Remarque : la saisie sera récupérée dans la variable.

### C) Les variables

Syntaxe :

```
$nomVariable = valeur
```

Exemple :

```
$nb1 = 5
$nb2 = 10
$somme = $nb1 + $nb2
Write-Host "La somme est de $somme"
```

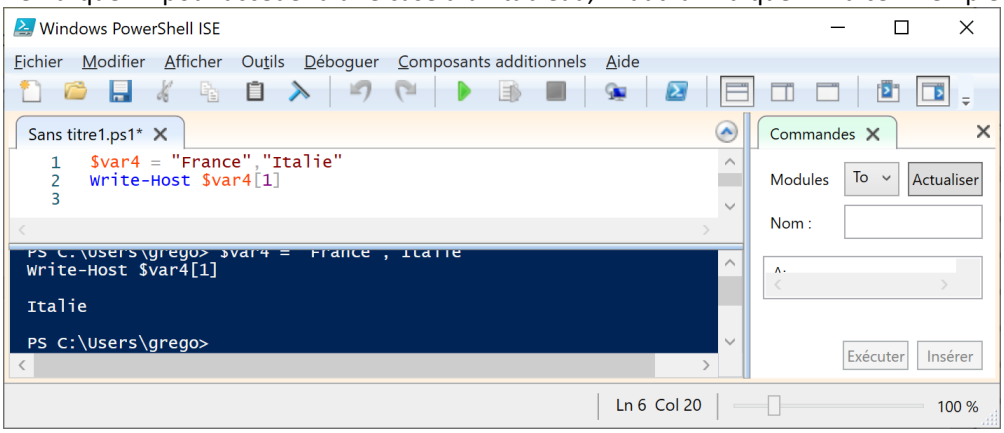
Types de variables :

| | | |
|---|---|---|
| [string] | Chaîne de caractères | $var1 = "Bonjour" |
| [char] | Caractères / Symboles | [char]$var1 = 0x 203 |
| [bool] | Booléen (vrai / faux) | $var1 = $true |
| [int] | Nombre entier | $var2= 123456 |
| [double] | Nombre à virgule flottante | [double]$var2 = 123. 45 |
| [date] | Date et heure | $var3 = get-date |
| [array] ou [object] | Tableau | $var4 = "France","Italie"<br>Write-Host $var4[1] |

Remarque : les variables sont automatiquement typées par PowerShell. Parfois, souvent lors d'une saisie d'un nombre au clavier par l'utilisateur, la variable stockant la saisie est mise en type « string » au lieu de « int ». Cela empêche ensuite de réaliser des calculs avec la valeur. Pour résoudre le problème, il faut indiquer à PowerShell le type de valeur attendu. Exemple :

```
[int]$saisie = Read-Host "Saisir le montant"
```

Remarque 2 : pour accéder à une case d'un tableau, il faut lui indiquer l'indice. Exemple :




## D) Opérateurs disponibles

| | | |
|---|---|---|
| * | Répète une chaine | "-" * 20 |
| + | Concatène deux chaînes | $nom= "Damico"<br>$prenom = "Greg"<br>$nomComplet = $nom + " "+ $prenom<br>➔ La variable $nomComplet contient : Damico Greg |
| -replace | Remplace une chaîne de caractère (insensible à la casse) | "Damico Greg" -replace "greg", "Gregory"<br>⇨ Affiche : Damico Gregory |
| -creplace | Idem mais sensible à la casse | "Damico Greg" -creplace "greg", "Gregory"<br>⇨ Affiche : Damico Greg |
| - eq<br>-ceq | Vérifie l'égalité | "Damico" -eq "Dami"<br>⇨ Renvoie False |
| -like<br>-clike | Vérifie si une chaîne est contenue dans une autre | "Damico" -like "*ami*"<br>⇨ Renvoie True |

| | | |
|---|---|---|
| -nolike<br>-cnolike | Vérifie si une chaîne n'est pas contenue dans une autre | "Damico" -nolike "greg"<br>⇨ Renvoie True |
| -match<br>-cmatch | Vérifie qu'une chaîne en contient une autre | "Damico" -match "ami"<br>⇨ Renvoie True |
| -notmatch<br>-cnotmatch | Vérifie qu'une chaîne n'en contient pas une autre | "Damico" - notmatch "greg"<br>⇨ Renvoie True |

### E) Opérateurs de comparaison

| | |
|---|---|
| -eq | Egal |
| -lt | Plus petit que |
| -gt | Plus grand que |
| -le | Egal ou plus petit |
| -ge | Egal ou plus grand |
| -ne | Pas égal / différent de |
| -not | Inverse |
| ! | Inverse |
| -and | ET |
| -or | OU |

### F) Condition if

Syntaxe :

```
if (condition) {
        actions si condition est vraie
} else {
        actions si condition est fausse
}
```

Exemple :

```
$nb1 = 15
$nb2 = 20
if ($nb1 -eq $nb2) {
        Write-Host "Les nombres sont identiques"
}
else {
        Write-Host "Les nombres sont différents"
}
```

### G) Boucles

#### 1. Boucle Do Until (Faire jusqu'à)

Syntaxe :

```
do {
      actions
} until (condition)
```

Exemple :

```
#Pour afficher 11 fois Bonjour
$i = 0
do {
     Write-Host "Bonjour"
     $i++
} until ($i -eq 10)
```

#### 2. Boucle Do While (Faire tant que)

Syntaxe :

```
do {
      actions
} while (condition)
```

Exemple :

```
#Pour afficher 11 fois Bonjour
$i = 0
do {
    Write-Host "Bonjour"
    $i++
} while ($i -le 10)
```

3. **Boucle foreach (Pour chaque)**

Permet de boucler sans connaître le nombre de fois. Idéal pour lire un fichier (ligne par ligne) ou un tableau.

Syntaxe :

```
foreach ($valeur in $valeurs)
{
   actions
}
```

Exemple :

```
$liste = "France","Italie","Espagne"
foreach ($pays in $liste){
    Write-Host $pays
}
```

## 3) Principales commandes

### A) Compatibilité avec les commandes MS-DOS

Certaines commandes d'autres systèmes peuvent être utilisées. Par exemple : dir, ls…

### B) Obtenir de l'aide sur une commande

Syntaxe :

```
Get-Help nomCommande
```

Exemple :

```
Get-Help ls
```

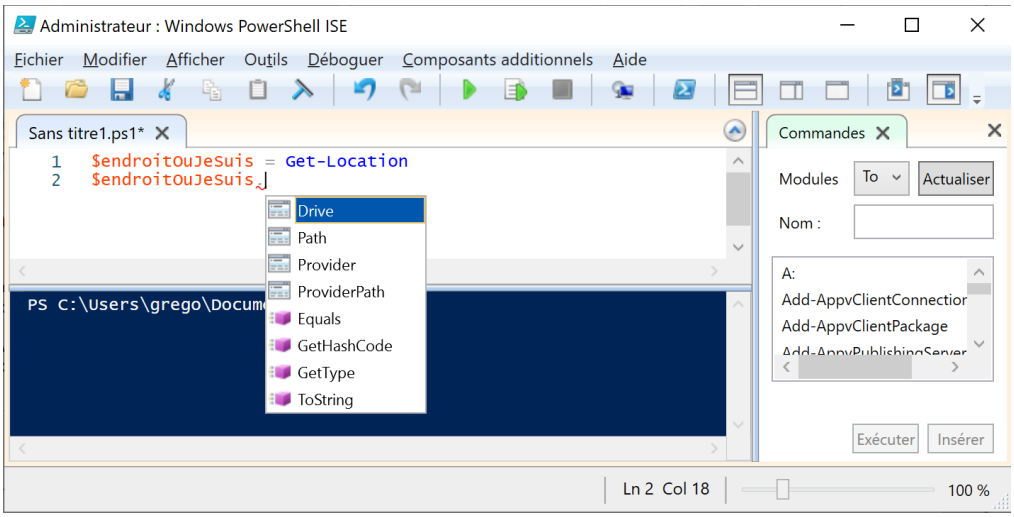
### C) Gérer les fichiers et les dossiers

| Action | Syntaxe | Exemple |
|---|---|---|
| Se déplacer | Set-Location chemin | `Set-Location C:\Users\grego\Documents` |
| Afficher le chemin du dossier courant | Get-Location | `Get-Location` |
| Afficher le contenu d'un dossier | Get-ChildItem (paramètre -recurse pour inclure les sous-dossiers) | `Get-ChildItem -Recurse` |
| Créer un dossier | New-Item nomDossier -ItemType directory | `New-Item "Contrôles 2020" -ItemType directory` |
| Créer un fichier avec du texte | New-Item nomFichier.txt -ItemType file -Value "Texte à écrire" | `New-Item notes.txt -ItemType file -Value "Notes du 1er DS"` |
| Supprimer un fichier ou un dossier | Remove-Item nomFichier.txt | `Remove-Item notes2018.txt` |
| Déplacer un fichier | Move-Item nomFichier.txt -Destination chemin\nomFichier.txt | `Move-Item notes2019.txt -Destination \archives` |
| Déplacer un dossier | Move-Item nomDossier -Destination chemin\nomDossier | `Move-Item TP -Destination \mesTP` |
| Renommer un fichier ou dossier | Rename-Item nomFichier.txt -NewName nomFichier2.txt | `Rename-Item notes.txt -NewName notesJanvier2020.txt` |
| Copier un fichier | Copy-Item nomFichier.txt -destination nomFichier2.txt | `Copy-Item sujet.txt -destination sujetBis.txt` |
| Copier un dossier avec ses fichiers | Copy-Item nomDossier -Destination nomDossier1 -Recurse | `Copy-Item TP -Destination \sauvegarde -Recurse` |
| Tester l'existence d'un fichier ou dossier | Test-Path chemin\nomFichier.txt | `Test-Path mesTP\TP1.doc` |
| Récupérer le contenu d'un fichier | Get-Content chemin\nomFichier.txt | `$fichier= Get-Content C:\2019\CA.txt` |

## D) Accès aux propriétés et méthodes des objets

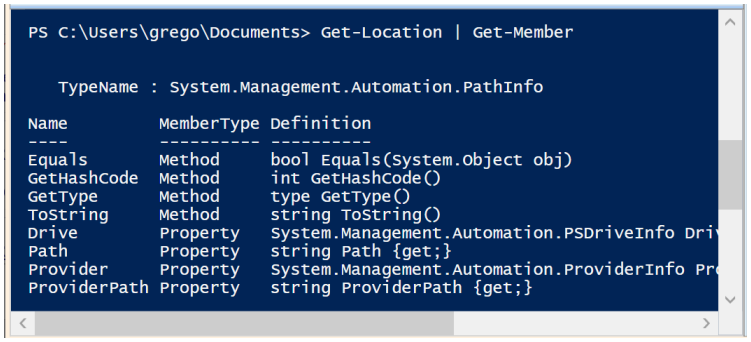
Certains éléments sont des objets. Cela signifie qu'ils possèdent d'autres informations en eux… Pour y accéder, il suffit de compléter le nom de l'élément par un point (.) et d'écrire le nom de l'information souhaitée. Remarque : souvent, juste après le point, PowerShell ISE propose la liste des propriétés et méthodes possibles de l'objet. Sinon on peut connaître la liste en appelant la commande Get-Member.

Exemple :




→ Dans une variable, on a récupéré l'endroit où on se situe. On a récupéré un objet qui possède pleins d'informations : le lecteur, le chemin complet, le fournisseur (système de fichiers), etc.

En mode console, on aurait pu saisir :



```
PS C:\Users\grego\Documents> Get-Location | Get-Member


   TypeName : System.Management.Automation.PathInfo

Name         MemberType Definition
----         ---------- ----------
Equals       Method     bool Equals(System.Object obj)
GetHashCode  Method     int GetHashCode()
GetType      Method     type GetType()
ToString     Method     string ToString()
Drive        Property   System.Management.Automation.PSDriveInfo Dri
Path         Property   string Path {get;}
Provider     Property   System.Management.Automation.ProviderInfo Pr
ProviderPath Property   string ProviderPath {get;}
```

## E) Accès aux ressources du systèmes d'exploitation Windows

Les classes WMI (Windows Management Instrumentation) décrivent les ressources qui peuvent être gérées. Il existe des centaines de classes WMI, certaines d'entre elles contenant des dizaines de propriétés.

La commande principale est Get-WmiObject, elle permet de lire ces ressources.

Exemple pour consulter les informations suivantes :

Graphiques : Get-WmiOject win32_videocontroller

Système : Get-WmiObject win32_operatingsystem

Disques : Get-WmiObject win32_logicaldisk

Il est toujours possible d'affecter le résultat de la commande Get-WmiObject à une variable, et de consulter les propriétés et les méthodes de l'objet à l'aide de la commande Get-Member.

## F) Méthodes de traitements d'objets

| Fonction | Description | Exemple |
|---|---|---|
| CompareTo() | Compare une chaine avec une autre<br>Renvoie 0 ou 1 | `("Hello").CompareTo("Hello")` |
| Contains() | Test si la chaîne de comparaison spécifiée est présente dans une chaine ou si la chaine de comparaison est vide<br>Renvoie True ou False | `("Hello").Contains("Hel")` |
| EndsWith() | Test si la chaine termine avec la chaine spécifiée<br>Renvoie True ou False | `("Hello").EndsWith("lo")` |
| Equals() | Test si une chaine est identique à une autre chaine<br>Renvoie True ou False | `$var="Hello"`<br>`("Hello").Equals($var)` |
| IndexOf() | Retourne l'index de la première occurrence de la chaine de comparaison | `("Hello").IndexOf("o")` |
| IndexOfAny() | Retourne l'index de la première occurrence de la chaine de comparaison<br>Renvoie 1 ou -1 | `("Hello").IndexOfAny("loe")` |
| Insert() | Insert une chaine à l'index spécifié d'une autre chaine | `("Hello World").Insert(6, "brave ")` |
| GetEnumerator() | Récupère un objet qui peut énumérer tous les caractères d'une chaîne | `("Hello").GetEnumerator()` |
| LastIndexOf() | Recherche l'index de la dernière occurrence d'un caractère spécifié | `("Hello").LastIndexOf("l")` |
| LastIndexOfAny() | Recherche l'index de la dernière occurrence d'un caractère d'une chaîne spécifiée | `("Hello").LastIndexOfAny("loe")` |
| PadLeft() | Remplissage d'une chaîne à une longueur déterminée et qui ajoute des caractères blancs à gauche (résultat aligné à droite) | `("Hello").PadLeft(10)` |
| PadRight() | Remplissage d'une chaîne à une longueur déterminée et qui ajoute des caractères blancs à droite (aligné à gauche de chaîne) | `("Hello").PadRight(10) + "World!"` |
| Remove() | Supprime le nombre requis de caractères à partir d'une position spécifiée | `("Hello World").Remove(5,6)` |
| Replace() | Remplace un caractère par un autre caractère | `("Hello World").Replace("l", "x")` |
| Split() | Convertit une chaîne avec des points de séparation spécifiés dans un tableau | `("Hello-World").Split("-")` |
| StartsWith() | Test si une chaîne commence par un caractère spécifié<br>Renvoie True ou False | `("Hello World").StartsWith("He")` |
| Substring() | Extrait les caractères d'une chaîne | `("Hello World").Substring(4, 3)` |
| ToCharArray() | Convertit une chaîne en un tableau de caractères | `("Hello World").toCharArray()` |
| ToLower() | Convertit une chaîne en minuscules | `("Hello World").toLower()` |
| ToUpper() | Convertit une chaîne en majuscules | `("Hello World").toUpper()` |
| Trim() | Supprime les caractères en blanc à droite et à gauche | `(" Hello ").Trim() + "World"` |
| TrimEnd() | Supprime les caractères vide à droite | `(" Hello   ").TrimEnd() + "World"` |
| TrimStart() | Supprime les caractères en blanc à la gauche | `(" Hello ").TrimStart() + "World"` |
| Chars() | Fournit un caractère à la position spécifiée | `("Hello").Chars(0)` |