# ICT122 – INTRODUCTION À POWERSHELL

# Historique

- L'interpréteur "*cmd.exe*" et les scripts VBscripts sont devenus désuets, pas prévus pour l'administration
- Support en natif pour tous les nouveaux OS MS depuis Vista, server 2008 etc..
- Uniformisation de l'administration des systèmes d'exploitation Windows
- Offrir un outil digne de la concurrence Linux
- Syntaxe facilitant l'apprentissage et l'utilisation
- Hérite du Framework.Net => orienté objets

# Les 3 commandes de base

- **`Get-Help`**: affiche l'aide, rubrique ou commande
- **`Get-Command`** : permet de rechercher les commandes disponibles de multiples façons
- **`Get-Member`** : comme PowerShell est orienté objets, permet d'obtenir les informations sur un objet, variable, etc..

Ce sont les 3 commandes indispensables à maîtriser!

# Alias

- Les alias permettent l'assignation de noms raccourcis sur les *cmdlets*. Ceci permet de personnaliser la syntaxe des *cmdlets*.

- **Exemple**: assignation de l'alias **`gh`** sur la *cmdlet* **`get-help`**:
  1. S'assurer qu'aucun alias du même nom n'existe: **`get-alias g*`**
  2. Appeler l'aide de la commande **`set-alias`**: **`get-help set-alias`**
  3. Appeler la *cmdlet* **`set-alias`** avec le nom du nouvel alias: **`set alias gh get-help`**
  4. Utiliser `get-alias` pour vérifier que l'alias est assigné: **`get-alias gh`**

# Le Pipe

- Le tube permet de transmettre le résultat d'une commande à une autre commande.

- **Exemple**: `gci . | sort`

- Liste les éléments du répertoire courant puis les transmet à la commande `sort` qui va les classer par ordre alphabétique.

# Format

- Quatre *cmdlets* permettent le formatage de l'affichage dans PowerShell:

| | |
|---|---|
| `Format-list` | Affichage par liste |
| `Format-wide` | Affichage par colonnes |
| `Format-table` | Affichage sous forme de tableau |
| `Format-custom` | Affichage personnalisé |

# Naviguer dans le système de fichier

| PowerShell | Cmdlet | |
|---|---|---|
| `DIR` | `Get-ChildItem` | Lister le contenu d'un répertoire |
| `CD` | `Set-Location` | Changer de répertoire courant |
| `MD` | `New-Item` | Créer un fichier/répertoire |
| `RD` | `Remove-Item` | Supprimer un fichier/répertoire |
| `MOVE` | `Move-Item` | Déplacer un fichier/répertoire |
| `REN` | `Rename-Item` | Renommer un fichier/répertoire |
| `COPY` | `Copy-Item` | Copier un fichier/répertoire |

# Les chaînes de caractère

- Comme dans la plupart des langages scripts, les chaînes de caractères sont délimitées par des guillemets simples ou doubles.
  - `$a = 'Hello'`
  - `$b = 'world'`
- Entre guillemets simples, les variables ne sont pas substituées:
  - `Write-Host '$a $b' => $a $b`
- Entre guillemets doubles, les variables sont substituées :
  - `Write-Host "$a $b" => Hello world`

# Concaténation de chaînes de caractère

- A compléter https://blogs.technet.microsoft.com/heyscriptingguy/2014/07/15/keep-your-hands-clean-use-powershell-to-glue-strings-together/

# Remplacement de chaîne de caractère

- https://www.safaribooksonline.com/library/view/windows-powershell-cookbook/9780596528492/ch05s09.html

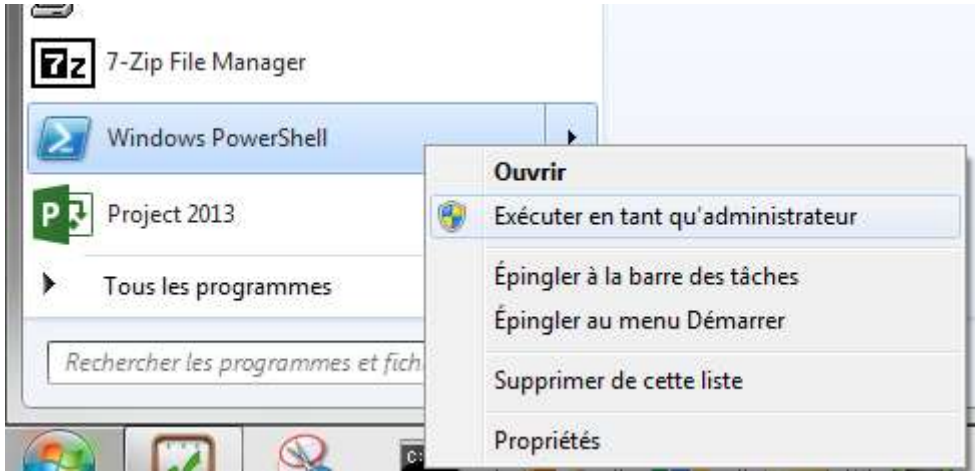
# Substitution des variables

- Syntaxe d'accès à une propriété d'objet:
  - `$($objet.propriété)`
- PowerShell substitue la variable.
  Exemple:
  - `$a = Get-ChildItem c:\config.sys`
  - `Write-Host "Taille fichier = $a.Length octets"`
  - `=> Taille du fichier = c:\config.sys.Length octets`
- Il faut donc écrire:
  - `Write-Host "Taille fichier = $($a.Length) octets"`
  - `=> Taille du fichier = 10 octets`

# Démarrage de la console

- La console démarre avec les droits de simple utilisateur

- Les droits sont donc limités
- Pour ouvrir la console classique ou graphique (ISE) avec les privilèges Administrateur
  - Clic droit – **Exécuter en tant qu'administrateur**



# Stratégie de sécurité

- Pour des raisons de sécurité, les scripts ne sont pas activés par défaut dans PowerShell, il faut donc modifier la stratégie de sécurité PowerShell,
- **`Get-ExecutionPolicy`** retourne la stratégie actuelle

| Niveau | Signification |
|---|---|
| `Restricted` | Scripts et fichiers de configuration sont bloqués |
| `AllSigned` | Scripts et fichiers de config doivent être signés, source autorisée |
| `RemoteSigned` | Scripts et fichiers de config téléchargés d'Internet doivent être signés, source autorisée |
| `Unrestricted` | Scripts et fichiers de config téléchargés d'Internet demanderons une confirmation avant exécution |

# Stratégie de sécurité

**Pour pouvoir exécuter des scripts**:

- Démarrer la console en Administrateur
- Modifier la stratégie de sécurité
  par exemple: `Set-ExecutionPolicy RemoteSigned`
- Les scripts doivent avoir l'extension *.ps1*
- Double-clique pas possible (améliore la sécurité)
  Seulement depuis PowerShell si la stratégie est OK
- Depuis la fenêtre exécuter avec le chemin complet
  par exemple:
  `powershell -noexit -command "c:\122\HelloWorld.ps1"`

# Les commentaires

- Commentaires en ligne

```
Write-Host «Hello World" # ici un commentaire

# ici aussi
Write-Host "Bonjour"
```

- Bloc de commentaires

```
<# du commentaire ici ...
... et ici aussi
#>
```

# Les variables

- Langage non typé
- Type défini lors de chaque affectation
- Le nom de variable débute par un $

```
$fltCarLength = 4.56
```

- Pour connaître son type exact

```
$fltCarLength.GetType()
```

- Le typage reste possible

```
[int] $intCptAge = 2
$intCptAge = 'A' => IMPOSSIBLE
```

- Récupération de valeur

```
[int] $intCptAge = Read-Host "Entrez l'age
du capitaine: "
```

# Les variables

- PowerShell contient un certain nombre de variables spéciales, en voici quelques unes:

| Nom | Utilisation |
|---|---|
| `$_` | Objet courant, where-object, foreach-object, switch, filtres |
| `$?` | Booléen pour savoir si la dernière opération a réussi |
| `$Args` | Tableau des arguments passés à une fonction ou un script |
| `$Error` | Tableau des erreurs de la dernière session |
| `$Home` | Répertoire de base de l'utilisateur |
| `$Host` | Information sur l'hôte qui exécute PowerShell |
| `$PWD` | Indique le chemin complet du répertoire actif |

# Les constantes

- Dans PowerShell une constante est déclarée lorsqu'elle reçoit une valeur qui ne pourra pas changer ni être effacée:

```
Set-Variable -name INTCARLENGTH -value 4.56
-option constant
```

# Les opérateurs arithmétiques

| Opérateur | Signification |
|---|---|
| + | Addition |
| - | Soustraction |
| * | Multiplication |
| / | Division |
| & | Reste de la division entière (modulo) |

# Les opérateurs de comparaison

- Les opérateurs retourne un booléen

| Opérateur | Signification |
|---|---|
| `-eq` | Egal à |
| `-ne` | Différent de |
| `-gt` | Supérieur à |
| `-ge` | Supérieur ou égal à |
| `-lt` | Inférieur à |
| `-le` | Inférieur ou égal à |
| `-like` | Correspondance à l'aide du caractère * ou ? |
| `-notlike` | Pas de correspondance à l'aide du caractère * ou ? |

# Les opérateurs RegEx

| Opérateur | Signification |
|---|---|
| `-match` | Correspondance dans une RegEx |
| `-nomatch` | Pas de correspondance dans une RegEx |

# Les opérateurs

- L'opérateur de remplacement

| Opérateur | Signification |
|---|---|
| `-replace` | Permet de remplacer tout ou une partie d'une valeur |

- Les opérateurs logiques

| Opérateur | Signification |
|---|---|
| `-and` | Et logique |
| `-or` | Ou logique |
| `-not ou !` | Non logique |
| `-xor` | Ou exclusif |

# Les opérateurs de redirection

| Opérateur | Signification |
|---|---|
| `>` | Redirection du flux vers un fichier (remplacement) |
| `>>` | Redirection du flux vers un fichier (ajour à la fin) |
| `2>&1` | Redirige les messages d'erreurs vers la sortie standard |
| `2>` | Redirection des erreurs vers un fichier (remplacement) |
| `2>>` | Redirection des erreurs vers un fichier (ajout à la fin) |

# Le pipeline

- La sortie d'une commande est redirigée vers l'entrée de la suivante sous forme d'objet

- `Get-Command | Out-File -FilePath 'd:\temp\file.txt\`

- Filtre Where-Object
  Liste de tous les services **stoppés**

```
Get-Service |Where-Object {$_.Status -eq 'Stopped'}
```

# Tableau à une dimension

- Déclaration – initialisation

```
$tab_intVar = 1, 5, 10, 15, 20
$tab_intVar = 1..10
```

- Déclaration avec type forcé

```
[int[]]$tab_intVar = 1, 2, 3
```

- Accès aux valeurs

```
$tab_intVar[0] => 1
$tab_intVar[0,2] => 1 3
$tab_intVar[0..2] => 1 2 3
```

- Taille du tableau

```
$tab_intVar.Length
```

# Tableau à une dimension

- Concaténer 2 tableaux avec l'opérateur +

```
$tab_carDebut='s', 'a', 'l`
$tab_fin='u', 't`
$tab_carDebut+$tab_fin=>'s' 'a' 'l' 'u' 't'
```

- Ajout d'éléments avec l'opérateur +=

```
$tab_carHello='s', 'a', 'l`
$tab_carHello+='u','t'=>'s' 'a' 'l' 'u' 't'
```

- Suppression impossible mais… enlevons l'élément numéro 4 par copie

```
$tab_intVar = $tab_intVar[0..3 + 5]
```

# Les tableaux à plusieurs dimensions

- Idem que les tableaux à une dimension, on rajoute les indices en fonction des dimensions

```
$tab_intVar = (11, 12, 13),(21,22,23)
$tab_intVar[0] ->= 11 12 13
$tab_intVar[0] [1] ->= 12
```

# Les tableaux associatifs

- L'indice est une clé

```
$NomTableau=@{ cle1 = elem1; cle2=elem2;…}
$tab_intAge=@{Bob = 12; Al= 16; Luc = 14}
$tab_intAge['Bob']=>12
```

# Structure conditionnelle

```
if (expression booléenne)
{
     instructions
}
elseif(expression booléenne)
{
     instructions
}
else
{
     instructions
}
```

# Le switch

Le switch correspond à une suite de `if.. elseif` avec la possibilité d'avoir un bloc par défaut qui sera exécuté si il n'y a pas eu de correspondance avec la condition.

```
switch (expression){
  Valeur1 {
       instructions
  }
  Valeur2 {
       instructions
  }
  Default {
       instructions
  }
}
```

# Les boucles

- **While**

```
while (expression booléenne)
{
instructions
}
```

- **Do-While**

```
do
{
instructions
}
while(expression booléenne)
```

# Les boucles

- **For**

```
for (expr. initiale; expr. booléenne; expr. finale)
{
instructions
}
```

- En général:
  L'expression initiale contient **l'initialisation** du compteur
  L'expression finale contient **l'incrémentation** du compteur

# Les boucles

## Foreach-Object

- *Cmdlet* qui permet de parcourir les valeurs d'une collection

```
foreach ($Element in $Collection)
{
    instructions
}
```

# Sources

- Très largement inspiré du support du même cours donné à l'ETML de Patrick Chenaux
- Windows PowerShell (Version 1&2), Eni Editions