Commande → Données

Donnée → Simple (entier)
(chaines)
Complexe (objet, ses champ)
Tableau de données

Type Complexe peut être constitué de:
- entier, chaine
- Type Complexe
- Tableau, ...

Cmd 1 | cmd 2 | cmd 3 → resultat
Sortie
entrée

Commande | Get-Member

<namespace>.Type (renvoyé)

Champs
Type (Methode) (Prop)
Parametres

⚠ Si Pa Commande renvoie un Tableau, ce n'est pas indiqué !

```
PS C:\> get-childitem | get-member

   TypeName: System.IO.DirectoryInfo

Name                 MemberType    Definition
----                 ----------    ----------
LinkType             CodeProperty  System.String LinkType{get=GetLinkType;}
Mode                 CodeProperty  System.String Mode{get=Mode;}
```

get-member renvoie une Collection

Accès au n-ième élément : 0 → le premier, 1 → le second, etc...

```
PS C:\> (get-childitem)[0] | get-member

   TypeName: System.IO.DirectoryInfo

Name                 MemberType    Definition
----                 ----------    ----------
LinkType             CodeProperty  System.String LinkType{get=GetLinkType;}
Mode                 CodeProperty  System.String Mode{get=Mode;}
```

Je demande le 1er élément, c'est un DirectoryInfo

---

1) Select Object pour filtrer les Cmdlets :

Get-Process | Select-Object -First 3

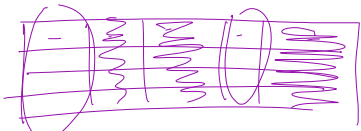
les 3 premiers de la liste

2) Select-Object pour choisir les colonnes

Commande | Select-Object -Property ... |

→ Toutes les colonnes dans le pipe

→ une Partie des colonnes dans le pipe



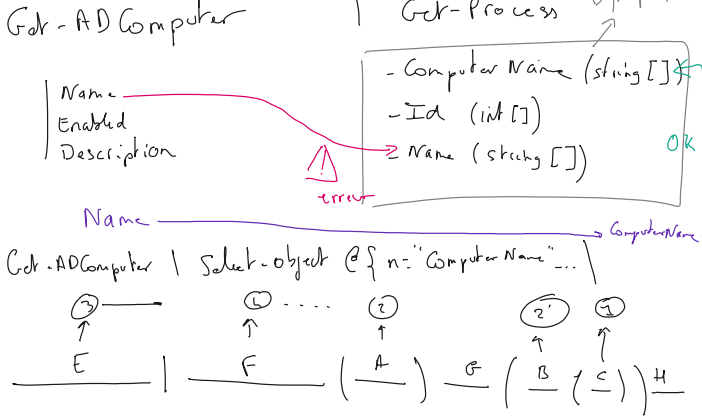
Colonnes calculées

Pipe $_ en "Ligne par Ligne, l'élément en cours"

"BITS" | Get-Service

Powershell cherche un paramètre byValue qui correspond au Type (String) et trouve -Name

---



Type du Pa sous-commande ne correspond Pas au type du Param

SS Commande → Tableau d'utilisateurs
- FullName (string)
- Address
- Service
- Email, ...

→ On doit avoir une liste de string

⇒ Il faut utiliser Select-object -ExpandProperty

Saisir au clavier :
Read-Host

Création d'une boite de dialogue de saisie depuis PowerShell (Graphique, et non textuelle)
PowerShell est prévu pour ne fonctionner que via la console.
Mais il est possible d'exécuter tout code .Net (dont WinForm, WPF, ...)
WinForm et WPF ne fournissent pas de classe toute faite pour saisir au clavier
Mais les bibliothèques de compatibilité VB6 en disposent.
Pour cela :

```
[System.Reflection.Assembly]::LoadWithPartialName('Microsoft.VisualBasic') | Out-Null
$User = [Microsoft.VisualBasic.Interaction]::InputBox('Enter a user name', 'User',
"$env:UserName")
```

ou : (nouvelle méthode :)

```
Add-Type -AssemblyName Microsoft.VisualBasic
$User = [Microsoft.VisualBasic.Interaction]::InputBox('Enter a user name', 'User',
"$env:UserName")
```

SecureString :

 # utiliser le backquote (AltGr + 7) pour échapper le $
$pwd = "Pa`$`$w0rd59" | ConvertTo-SecureString -AsPlainText -Force
obtenir un logon en passant le mot de passe obtenu : (prérenseigner) :
$cred = new-object -typename System.Management.Automation.PSCredential -ArgumentList "stage\administrateur", $pwd

Activer WSRemoting :
Enable-PSRemoting -Force (sur le poste destinataire)
Test-WsMan COMPUTER (tester le poste distant)
Invoke-Command -ComputerName COMPUTER -ScriptBlock { dir c:\ } -credential USERNAME
Enter-PSSession -ComputerName COMPUTER -Credential USER (-ConfigurationName
Microsoft.PowerShell32 pour une session 32 bits)

Ajout d'un host qui n'est pas dans le même domaine, sur chaque poste ! (client et server) :
winrm set winrm/config/client @{TrustedHosts="RemoteComputerName"} - ou Set-Item
wsman:\localhost\client\trustedhosts *

WMI :
$s = get-wmiobject win32_service
Accès a une instance : $s[0].StartName
gwmi win32_service | where {$_.StartMode -eq "Disabled"}
gwmi win32_service | where {$_.StartMode -ne "Disabled"} | select name,startname
invoke-wmimethod -path win32_process -name create -argumentlist notepad.exe
ou : (Get-WmiObject -Class "Win32_TerminalServiceSetting" -Namespace
root\cimv2\terminalservices).SetAllowTsConnection(1)
Appels à distance :
$service = Get-WmiObject -ComputerName DC1 -Class Win32_Service ` -Filter "Name='wuauserv'"
$service.stopservice()
Sinon :
Invoke-WmiMethod -Path "Win32_Service.Name='wuauserv'" ` -Name StopService -Computername DC1

Executer un script en parallèle à distance :
#execution de commande asynchrone et // à distance :
$servers = "w2K16", "trainer"
$jobs = invoke-command -ScriptBlock { ping 8.8.8.8 } -ComputerName $servers -AsJob

Documenter un script : mettre en entête :

<#
.SYNOPSIS
Retrieves network adapter information from a computer.
.DESCRIPTION
Uses CIM to retrieve information about physical adapters only.
.PARAMETER ComputerName
The name of the computer to query.
.EXAMPLE
.\Get-NetAdapterInfo.ps1 -ComputerName LON-DC1 –Verbose
#>

Paramétrer un script :

```
[CmdletBinding()]
Param(
   [Parameter(Mandatory=$True)]
   [string]$ComputerName,
   [int]$EventID = 4624
)
Get-EventLog -LogName Security -ComputerName $ComputerName |
Where EventID -eq $EventID |
Select -First 50
```

Signer un script : (ici en auto-signé, sans PKI)

Créer un certificat racine :
makecert -n "CN=PowerShell Local Certificate Root" -a sha1 -eku 1.3.6.1.5.5.7.3.3 -r -sv root.pvk root.cer -ss Root -sr localMachine

Ensuite, créer un certificat basé sur cette racine :
makecert -pe -n "CN=PowerShell User" -ss MY -a sha1 -eku 1.3.6.1.5.5.7.3.3 -iv root.pvk -ic root.cer

Signer un script :
Set-AuthenticodeSignature c:\foo.ps1 @(Get-ChildItem cert:\CurrentUser\My -codesign)[0]
ou : New-SelfSignedCertificate -CertStoreLocation "cert:\CurrentUser\My" -DnsName "localhost" -TestRoot -Type CodeSigningCert -Subject "Objet du certificat"

Et déployer le certificat de signature dans la liste des éditeurs approuvés du magasin

Exporter le certificat racine : (depuis trusted root\certificates) :
Le selectionner, l'exporter, choisir DER (.cer)
Puis importer dans la destination :
Importer dans Trusted root cert authorities\Certificates

Executer en tant qu'admin (PowerShell v4)

Auto Elevation des droits :

```
# Get the ID and security principal of the current user account
$myWindowsID=[System.Security.Principal.WindowsIdentity]::GetCurrent()
$myWindowsPrincipal=new-object
System.Security.Principal.WindowsPrincipal($myWindowsID)

# Get the security principal for the Administrator role
$adminRole=[System.Security.Principal.WindowsBuiltInRole]::Administrator

# Check to see if we are currently running "as Administrator"
if ($myWindowsPrincipal.IsInRole($adminRole))
  {
  # We are running "as Administrator" - so change the title and background color to indicate
this
  $Host.UI.RawUI.WindowTitle = $myInvocation.MyCommand.Definition + "(Elevated)"
  $Host.UI.RawUI.BackgroundColor = "DarkBlue"
  clear-host
  }
else
  {
  # We are not running "as Administrator" - so relaunch as administrator

  # Create a new process object that starts PowerShell
  $newProcess = new-object System.Diagnostics.ProcessStartInfo "PowerShell";

  # Specify the current script path and name as a parameter
  $newProcess.Arguments = $myInvocation.MyCommand.Definition;

  # Indicate that the process should be elevated
  $newProcess.Verb = "runas";

  # Start the new process
  [System.Diagnostics.Process]::Start($newProcess);

  # Exit from the current, unelevated, process
  exit
  }

# Run your code that needs to be elevated here
Write-Host -NoNewLine "Press any key to continue..."
$null = $Host.UI.RawUI.ReadKey("NoEcho,IncludeKeyDown")
```

DSC :
1) Configurer le LCM sur les nœuds :

```
Configuration LCMConfigs
{
   Node "W2k16"
   {
      LocalConfigurationManager
      {
         ConfigurationMode = "ApplyAndAutoCorrect"
         RefreshMode = "PUSH"
         RefreshFrequencyMins = 30
         RebootNodeIfNeeded = $true
      }
   }
}
LCMConfigs -verbose
```

2) Définir les ressources :

```
Import-Module -Name PSDesiredStateConfiguration

Configuration WebsiteConfig
{
   Import-DSCResource -ModuleName PSDesiredStateConfiguration

   Node w2k16 {
      WindowsFeature WebServer1
      {
         Name = "Web-Server"
         DependsOn = "[WindowsFeature]ASP"
      }

      WindowsFeature ASP
      {
         Name = "Web-Asp-Net45"
      }
   }
}
WebSiteConfig -OutputPath \\trainer\share\dsc
```

3) Executer les scripts pour créer les fichiers MOF

4) mettre à jour les conf :
Start-DscConfiguration -Path <racine des MOF>

5) Contrôler une conf :
Test-DscConfiguration

Créer une boite de dialogue de saisie utilisateur :
https://jdhitsolutions.com/blog/powershell/5816/a-powershell-input-tool/

```
Function Invoke-InputBox {

    [cmdletbinding(DefaultParameterSetName="plain")]
    [OutputType([system.string],ParameterSetName='plain')]
    [OutputType([system.security.securestring],ParameterSetName='secure')]

    Param(
        [Parameter(ParameterSetName="secure")]
        [Parameter(HelpMessage = "Enter the title for the input box. No more than 25
characters.",
        ParameterSetName="plain")]

        [ValidateNotNullorEmpty()]
        [ValidateScript({$_.length -le 25})]
        [string]$Title = "User Input",

        [Parameter(ParameterSetName="secure")]
        [Parameter(HelpMessage = "Enter a prompt. No more than 50
characters.",ParameterSetName="plain")]
        [ValidateNotNullorEmpty()]
        [ValidateScript({$_.length -le 50})]
        [string]$Prompt = "Please enter a value:",

        [Parameter(HelpMessage = "Use to mask the entry and return a secure string.",
        ParameterSetName="secure")]
        [switch]$AsSecureString
    )

    if ($PSEdition -eq 'Core') {
        Write-Warning "Sorry. This command will not run on PowerShell Core."
        #bail out
        Return
    }

    Add-Type -AssemblyName PresentationFramework
    Add-Type –assemblyName PresentationCore
    Add-Type –assemblyName WindowsBase

    #remove the variable because it might get cached in the ISE or VS Code
    Remove-Variable -Name myInput -Scope script -ErrorAction SilentlyContinue

    $form = New-Object System.Windows.Window
    $stack = New-object System.Windows.Controls.StackPanel

    #define what it looks like
    $form.Title = $title
    $form.Height = 150
    $form.Width = 350
```

```
    $label = New-Object System.Windows.Controls.Label
    $label.Content = "    $Prompt"
    $label.HorizontalAlignment = "left"
    $stack.AddChild($label)

    if ($AsSecureString) {
        $inputbox = New-Object System.Windows.Controls.PasswordBox
    }
    else {
        $inputbox = New-Object System.Windows.Controls.TextBox
    }

    $inputbox.Width = 300
    $inputbox.HorizontalAlignment = "center"

    $stack.AddChild($inputbox)

    $space = new-object System.Windows.Controls.Label
    $space.Height = 10
    $stack.AddChild($space)

    $btn = New-Object System.Windows.Controls.Button
    $btn.Content = "_OK"

    $btn.Width = 65
    $btn.HorizontalAlignment = "center"
    $btn.VerticalAlignment = "bottom"

    #add an event handler
    $btn.Add_click( {
            if ($AsSecureString) {
                $script:myInput = $inputbox.SecurePassword
            }
            else {
                $script:myInput = $inputbox.text
            }
            $form.Close()
        })

    $stack.AddChild($btn)
    $space2 = new-object System.Windows.Controls.Label
    $space2.Height = 10
    $stack.AddChild($space2)

    $btn2 = New-Object System.Windows.Controls.Button
    $btn2.Content = "_Cancel"

    $btn2.Width = 65
    $btn2.HorizontalAlignment = "center"
    $btn2.VerticalAlignment = "bottom"

    #add an event handler
    $btn2.Add_click( {
            $form.Close()
        })

    $stack.AddChild($btn2)

    #add the stack to the form
    $form.AddChild($stack)

    #show the form
    $inputbox.Focus() | Out-Null
    $form.WindowStartupLocation =
[System.Windows.WindowStartupLocation]::CenterScreen

    $form.ShowDialog() | out-null

    #write the result from the input box back to the pipeline
    $script:myInput

}
```

Paramétrer les scripts : https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.core/about/about_functions_advanced_parameters?view=powershell-6

```
<#
.SYNOPSIS
Fais un test
.DESCRIPTION
fais des choses
.PARAMETER Message
Un message a afficher
.PARAMETER Nombre
Un nombre pour faire des choses
.EXAMPLE
.\hello.ps1 -Nombre 60
#>

Param(
   [Parameter(Mandatory=$true,ValueFromPipeline=$true,)] [string]$Message = "toto" ,
   [int]$Nombre
)

"hello "+$Message
```