Les bases :

Documentation officielle de Powershell :
https://docs.microsoft.com/fr-fr/powershell/

MS Learn :
https://docs.microsoft.com/en-us/search/?terms=powershell&category=Learn

---

Admin

→ obj : automatiser
→ être + rapide
→ faire - d'erreurs

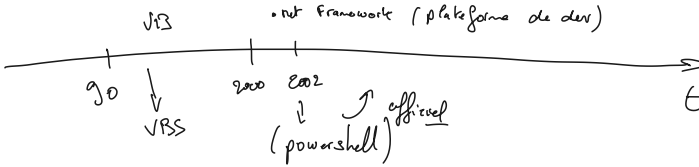
Automatisation
~~CMT~~ Scripts

NS années 90 → VBS
WSH (base de VB)
→ Batch (DOS, cmD)
Pas puissant du tout
Shell linux (equiv.)
BCP plus puissant, norme

Dev

Construire de Apps

→ Win form / C#
WPF
.net framework



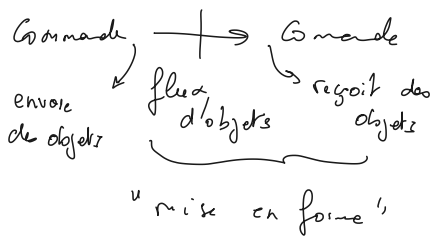
Objectif → scripting (- pas de compilation
- interactif
- exploite tout le .net framework
- orienté objet)

Bash Linux

Commande → Commande

Pipe au format flux de texte

Power shell



Programmation

→ Variables

→ commande → tests (if, switch)

boucles (for, foreach, while, until)

Variable → nom de variable ex : $v

→ type de variable / ex: String, Int32, Double, Boolean, DateTime } Types "simples"

Mais tout est objet dans le net

Catégorie 1

Catégorie 2 : Collections

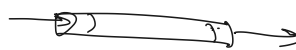
→ Tableaux } par index

→ Listes

→ Table de hachage → par hach

→ Piles → LIFO

→ FIFO



Cat 3 : Objets

→ Membres

Propriétés (valeurs en lecture ou lecture / écriture)

Méthodes (verbes, opérations, fonctions membres) ⇒ ce que l'objet peut faire

Variable → Type
↳ Valeur

Type Objet

Concept → classe (Valeur)

Instance → 1 objet issu d'une classe.

chaque instance a les mêmes types de caractéristiques mais avec des valeurs propres à elle.

Powershell

x, y } chargés → commandes accessibles.

modules

x
Commandes
y
z

_List Available

Par Machine

Get-Command → liste des commandes

Get-Module → " des modules

Get-command : liste des commandes
Get-Module [-ListAvailable] : liste des modules installés sur la machine locale.

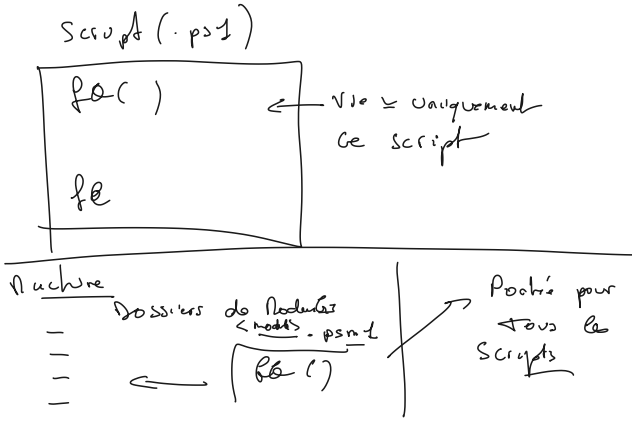
Gestionnaire de paquets NuGet : va chercher sur powershell gallery, un module spécifié en local (télécharge, installe)
https://www.powershellgallery.com/

Récupérer un module :
Install-Module -Name Az.Accounts

Passer un module sur une machine offline :
1) récupérer le module
2) Save-Module -Name ModuleName -Path “FilePath”
3) copier sur une disquette
4) Copier sur module dans C:\Program Files\WindowsPowerShell\Modules sur la machine cible
5) Import-Module -Name ModuleName (ou utilisation de l'auto load)

help get-member -Full

help get-member -Full



Ne pas se faire avoir par get-member concernnat les collections : utilisez Get-Type() :

```
PS C:\Windows\system32> $nombres = 1, 2, 3
PS C:\Windows\system32> $nombres
1
2
3
PS C:\Windows\system32> $nombres | get-member


   TypeName : System.Int32

Name        MemberType Definition
----        ---------- ----------
CompareTo   Method     int CompareTo(System.Object value), int CompareTo(int
value), int IComparable.CompareTo(Syste...
Equals      Method     bool Equals(System.Object obj), bool Equals(int obj), bool
IEquatable[int].Equals(int other)
GetHashCode Method     int GetHashCode()
GetType     Method     type GetType()
GetTypeCode Method     System.TypeCode GetTypeCode(), System.TypeCode
IConvertible.GetTypeCode()
ToBoolean   Method     bool IConvertible.ToBoolean(System.IFormatProvider
provider)
ToByte      Method     byte IConvertible.ToByte(System.IFormatProvider provider)
ToChar      Method     char IConvertible.ToChar(System.IFormatProvider provider)
ToDateTime  Method     datetime IConvertible.ToDateTime(System.IFormatProvider
provider)
ToDecimal   Method     decimal IConvertible.ToDecimal(System.IFormatProvider
provider)
ToDouble    Method     double IConvertible.ToDouble(System.IFormatProvider
provider)
ToInt16     Method     int16 IConvertible.ToInt16(System.IFormatProvider provider)
ToInt32     Method     int IConvertible.ToInt32(System.IFormatProvider provider)
ToInt64     Method     long IConvertible.ToInt64(System.IFormatProvider provider)
ToSByte     Method     sbyte IConvertible.ToSByte(System.IFormatProvider provider)
ToSingle    Method     float IConvertible.ToSingle(System.IFormatProvider provider)
ToString    Method     string ToString(), string ToString(string format), string
ToString(System.IFormatProvider pro...
ToType      Method     System.Object IConvertible.ToType(type conversionType,
System.IFormatProvider provider)
ToUInt16    Method     uint16 IConvertible.ToUInt16(System.IFormatProvider
provider)
ToUInt32    Method     uint32 IConvertible.ToUInt32(System.IFormatProvider
provider)
ToUInt64    Method     uint64 IConvertible.ToUInt64(System.IFormatProvider
provider)


PS C:\Windows\system32> $nombres.GetType()

IsPublic IsSerial Name                                     BaseType
-------- -------- ----                                     --------
True     True     Object[]                                 System.Array
```

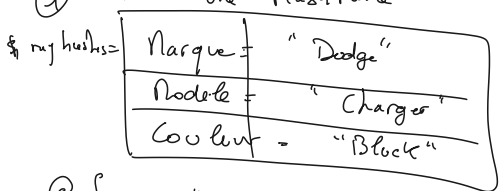
executer une opération depuis une collections sans passer par un fichier script :

$nombres | ForEach-Object { "Pour cette itération : $_" }
Pour cette itération : 1
Pour cette itération : 2
Pour cette itération : 3

Créer un objet Custom

Ex: Voiture (Marque, Modèle, Couleur)

① Creer une HashTable



@{ n = " ~~ " , e = " ~~ " ;}

$ v1 = New-Object -Type PSObject -Property $ my hashs .

$ v1. Marque

Dodge

```
$champs1 = @{ Marque="Dodge";  Modele="Charger";  Couleur="Black" }
$champs1.Marque
# $champs1.Marque = "Ford"
$champs1.Modele

$champs2 = @{ Marque="Ford";  Modele="F-150";  Couleur="Grey" }
$champs2.Marque
$champs2.Modele

$v1 = New-Object -Type PSObject -Property $champs1
$v2 = New-Object -Type PSObject -Property $champs2

$v1  | get-member
$v1.Marque

$voitures = $v1, $v2
$voitures | ForEach-Object { $_.Marque }
```

```
function Get-CorpCompSysInfo {
<#
.SYNOPSIS
Retrieves computer information from one or more computers.
.DESCRIPTION
This command uses Common Information Model (CIM) commands to
retrieve information from one or more computers. Remote computers
must have Windows Management Framework (WMF) 3.0 or later, and
most have PowerShell Remoting enabled.
.PARAMETER ComputerName
One or more computer names. This parameter accepts pipeline input.
You cannot use IP addresses or non-canonical computer names.
.EXAMPLE
Get-CorpCompSystemInfo -ComputerName LON-DC1,LON-CL1
#>
    [CmdletBinding()]
    param(
        [Parameter(Mandatory=$True,
                   HelpMessage='Computer name to query',
                   ValueFromPipeline=$True,
                   ValueFromPipelineByPropertyName=$True)]
        [Alias('hostname')]
        [ValidatePattern('LON-\w{2,3}\d{1,2}')]
        [string[]]$ComputerName
    )
    PROCESS {
        foreach ($computer in $computername) {
            Write-Verbose "Now connecting to $computer"
            $compsys = Get-CimInstance -ClassName Win32_ComputerSystem -ComputerName $computer
            $bios = Get-CimInstance -ClassName Win32_BIOS -ComputerName $computer
            $properties = @{'ComputerName'=$Computer;
                            'BIOSSerial'=$bios.serialnumber;
                            'Manufacturer'=$compsys.Manufacturer;
                            'Model'=$compsys.Model}
            $output = New-Object -TypeName PSObject -Property $properties
            Write-Output $output
        }
    }
}
```

```
# Add support for ShouldProcess
# Save this as \Documents\WindowsPowerShell\Modules\DemoTools\DemoTools.psm1
# (overwrite the existing file)

function Set-CorpComputerState {
    [CmdletBinding(SupportsShouldProcess=$True,ConfirmImpact="High")]
    param(
        [Parameter(Mandatory=$True,
                   HelpMessage='Computer name to set state for',
                   ValueFromPipeline=$True,
                   ValueFromPipelineByPropertyName=$True)]
        [Alias('hostname')]
        [string[]]$ComputerName,

        [Parameter(Mandatory=$True)]
        [ValidateSet('PowerOff','Shutdown','Restart','Logoff')]
        [string]$State,

        [switch]$force
    )
    BEGIN {
        switch ($state) {
            'LogOff'   { $_action = 0 }
            'Shutdown' { $_action = 1 }
            'Restart'  { $_action = 2 }
            'PowerOff' { $_action = 8 }
        }
        if ($force) { $_action += 4 }
        Write-Verbose "Action value is $_action"
    }
    PROCESS {
        foreach ($computer in $computername) {
            if ($PSCmdlet.ShouldProcess("$computer - action is $_action")) {
                Write-Verbose "Contacting $computer"
                $os = Get-WmiObject -Class Win32_OperatingSystem -ComputerName $Computer -EnableAllPrivileges
                $os.win32shutdown($_action)
            }
        }
    }
}
# SIG # Begin signature block
```

Class = Concept (1..1 fois)

Objet, instance (0..n fois)

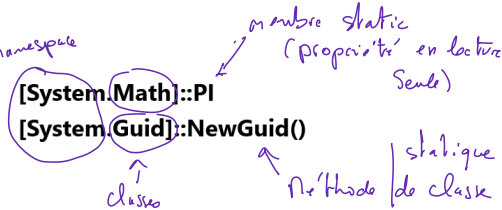
Membres (contenu - par classe ou - par instance)

Propriétés

Méthodes

Membres _ ou Membres d'instance
Sinon

Membres de classe ou membres statiques



Connaître un type en passant par les classes du Framework
[Type]::GetType([System.Security.Principal.WindowsBuiltinRole])

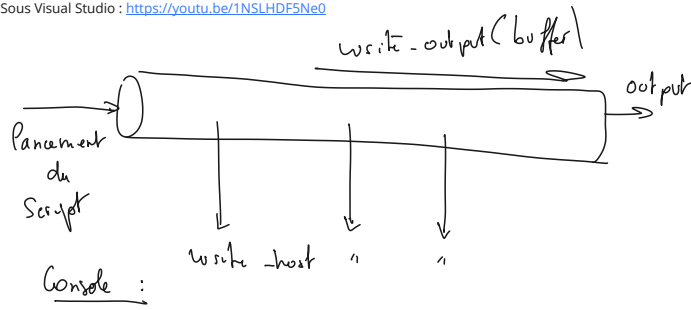
Liste d'APIS :
https://github.com/public-apis/public-apis

Client très puissant de tests d'APIs :
http://www.postman.com

Créer des formulaires Windows en Powershell : https://docs.microsoft.com/fr-fr/powershell/scripting/samples/creating-a-custom-input-box?view=powershell-7.2

Sous VSCode :
https://docs.poshtools.com/powershell-pro-tools-documentation/visual-studio-code/windows-forms-designer

Sous Visual Studio : https://youtu.be/1NSLHDF5Ne0

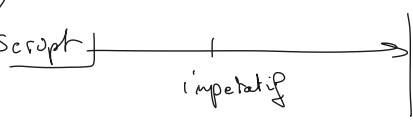


Gestion des erreurs "A l'ancienne" :
```
$ErrorActionPreference = 'SilentlyContinue'
$Error.Clear()
Get-ChildITem "/mnt"
if( $Error.Count -gt 0 )
{
   Write-host "il y a eu une erreur"
}
Write-Host "ok"
```

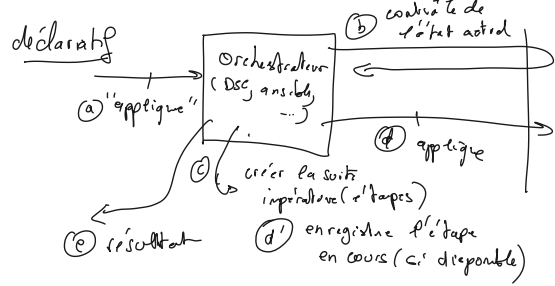
Conaître l'exception exacte suite à une erreur (dans un catch) :
```
[DBG]: PS C:\Users\PhilippePecqueur>> ($Error[0]).Exception.GetType()

IsPublic IsSerial Name                                     BaseType
-------- -------- ----                                     --------
True     True     RuntimeException                         System.SystemException
```

```
$Error.Clear()
try {
   write-host "try"
   Get-ChildlTem "/mnt" -ErrorAction Stop
   $res = 10 / 0
   write-host "fin de la section try"
}
catch [System.Management.Automation.ItemNotFoundException] {
   write-host "Element non trouvé ( Get-childitem )"
}
catch [System.Management.Automation.RuntimeException] {
   write-host "Erreur d'execution"
}
catch {
   write-host "catch non sélectif"
}
finally {
   write-host "ok finally"
}
$Error.Count
```

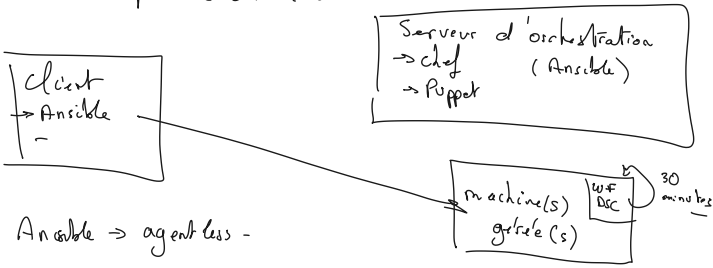
(1)



(2)



- position de l'orchestrateur



Ansible → agentless -

DSC :

1. Déclarer une conf
2. Appliquer la conf (la compile dans un dossier local)
3. Exécuter la conf
4. Vérifier la/les confs appliquées sur la machine cible : Get-DscConfiguration

Développer une interface graphique en XAML WPF avec Visual Studio Community Edition
VS -> Vous éditez l'interface, copiez le XML généré.
Vous collez et pluggez les évènenemts en powershell :
https://4sysops.com/archives/create-a-gui-for-your-powershell-script-with-wpf/

Vidéo Youtube d'implémentation : https://youtu.be/L76LhhgCzUY

Enregistrer la session, l'exécution d'un script powershell, dans un fichier :
https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.host/start-transcript?view=powershell-7.2

CSV :

```
PS C:\Users\PhilippePecqueur> get-childitem -File | Select-Object -Property Name, Length | ConvertTo-Csv
#TYPE Selected.System.IO.FileInfo
"Name","Length"
"test.ps1","113"
"toto.txt","919"
PS C:\Users\PhilippePecqueur> get-childitem -File | Select-Object -Property Name, Length | ConvertTo-Csv -NoTypeInformation
"Name","Length"
"test.ps1","113"
"toto.txt","919"
```

```
PS C:\Users\PhilippePecqueur> get-childitem -File | Select-Object -Property Name, Length |
ConvertTo-Csv -NoTypeInformation | ConvertFrom-Csv | get-member


   TypeName : System.Management.Automation.PSCustomObject

Name       MemberType   Definition
----       ----------   ----------
Equals     Method       bool Equals(System.Object obj)
GetHashCode Method      int GetHashCode()
GetType    Method       type GetType()
ToString   Method       string ToString()
Length     NoteProperty string Length=113
Name       NoteProperty string Name=test.ps1


PS C:\Users\PhilippePecqueur> get-childitem -File | Select-Object -Property Name, Length |
ConvertTo-Csv -NoTypeInformation | ConvertFrom-Csv

Name     Length
----     ------
test.ps1 113
toto.txt 919

get-childitem -File | Select-Object -Property Name, Length | ConvertTo-Csv -NoTypeInformation |
Out-File -FilePath "C:\Users\PhilippePecqueur\Documents\objs.csv"

PS C:\Users\PhilippePecqueur> get-command -noun csv

CommandType     Name                                               Version    Source
-----------     ----                                               -------    ------
Cmdlet          ConvertFrom-Csv                                    3.1.0.0    Microsoft.PowerShell.Utility
Cmdlet          ConvertTo-Csv                                      3.1.0.0    Microsoft.PowerShell.Utility
Cmdlet          Export-Csv                                         3.1.0.0    Microsoft.PowerShell.Utility
Cmdlet          Import-Csv                                         3.1.0.0    Microsoft.PowerShell.Utility

Import-Csv -Path "C:\Users\PhilippePecqueur\Documents\comptoirs\produits.csv" -Delimiter ";" | ft

Nom_Produit                No_fournisseur Code_categorie Quantite_par_unite       Prix_unitaire
Unites_en_stock Unites_commandees Niveau_de_reapprovisionnement Indisponible Ref_Produit
-----------                ------------- -------------- -----------------       -------------
--------------- ---------------- ------------------------------ ------------ ----------
Tarte au sucre             29            3              48 tartes               49,3          17
0               0                FAUX         62
Outback Lager              7             1              24 bouteilles (355 ml)  15            15
10              30               FAUX         70
```

```
PS C:\Users\PhilippePecqueur> Import-Csv -Path "C:\Users\PhilippePecqueur\Documents\comptoirs\produits.csv" -Delimiter ";" | ForEach-Object -Begin { Write-Host "Affichage de la liste d'objets" } -Process { write-host $_.Nom_Produit } -End { Write-Host "Fin d'affichage"  }
```

Exercices avec les CSV :

Lire le contenu
Filtrer pour afficher ce qui vous interresse (Select-Object)
Exporter le contenu filtré vers un autre CSV

Piloter Excel Sous Powershell :
Méthode native, puissante, complexe, (approche identique en VBA) via l'Objet COM Excel :

```
$Myexcel = New-Object -ComObject excel.application
$Myexcel.visible = $true
$Myworkbook = $Myexcel.workbooks.add()
$Sheet1 = $Myworkbook.worksheets.item(1)
$Sheet1.name = "Power level"
$Sheet1.cells.item(1,1) = 'NAME'
$Sheet1.cells.item(1,2) = 'POWER'
$Sheet1.cells.item(1,3) = 'TRANSFORMATION'
$Sheet1.Range("A1:C1").font.size = 18
$Sheet1.Range("A1:C1").font.bold = $true
$Sheet1.Range("A1:C1").font.ColorIndex = 2
$Sheet1.Range("A1:C1").interior.colorindex = 1
$Sheet1.cells.item(2,1) = 'Goku'
$Sheet1.cells.item(2,2) = '9500'
$Sheet1.cells.item(2,3) = 'SSJ3'

$Sheet1.cells.item(3,1) = 'Vegeta'
$Sheet1.cells.item(3,2) = '10000'
$Sheet1.cells.item(3,3) = 'SSJ2'

$Sheet1.cells.item(4,1) = 'Gohan'
$Sheet1.cells.item(4,2) = '5000'
$Sheet1.cells.item(4,3) = 'SSJ2'
$Myfile = 'C:\tmp\example.xlsx'
$Myexcel.displayalerts = $false
$Myworkbook.Saveas($Myfile)
$Myexcel.displayalerts = $true
```

https://techexpert.tips/fr/powershell-fr/powershell-creation-dune-feuille-de-calcul-excel/

Utilisation d'un module ImportExcel reconnu
https://www.it-connect.fr/comment-manipuler-des-fichiers-excel-avec-powershell/
https://github.com/dfinke/ImportExcel

```
Install-Module -Name ImportExcel
Get-Command -Module ImportExcel
Get-ADUser -Filter * | Export-Excel -Path './AD-utilisateurs.xlsx'

Import-Excel -Path '.\AD-utilisateursBis.xlsx'
```