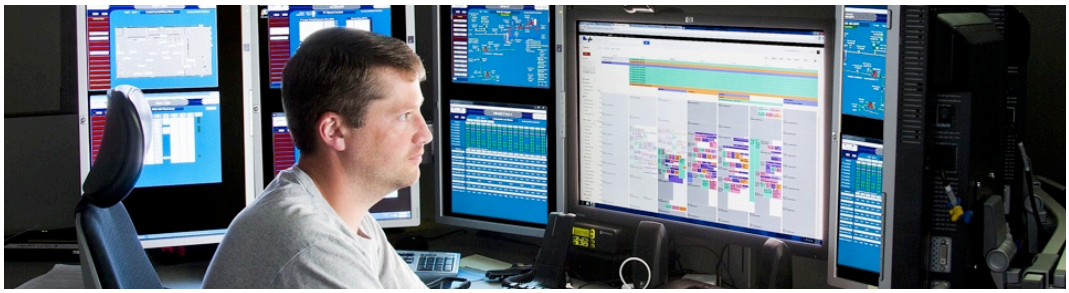
Administration Système — PowerShell



Année académique 2014/15

PowerShell
Introduction

- Windows PowerShell est une interface en ligne de commande et un langage de script pour Windows développé par Microsoft.
  - Successeur de command.com / cmd.exe.
  - Le premier shell très répandu utilisant les concepts orienté-objet et lambda-calcul.
  - Conçu par Jeffrey Snover (architecte principal), Bruce Payette et James Truher (LISA Outstanding Achievement Award 2012).
- PowerShell est intégré avec Windows, Windows Server, SQL Server, IIS, Hyper-V, Exchange, SharePoint, ...
- Versions
  - Version 4.0 incluse dans Windows 8.1 et Windows Server 2012 R2
  - Version 3.0 incluse dans Windows 8 et Windows Server 2012
  - Version 2.0 incluse dans Windows 7 et Windows Server 2008 R2
  - Version 1.0 publiée pour Windows XP SP2 et Windows Server 2003
- Le premier prototype s'appelait *Monad*, publié en 2002.

PowerShell

Pourquoi avoir créé PowerShell ?

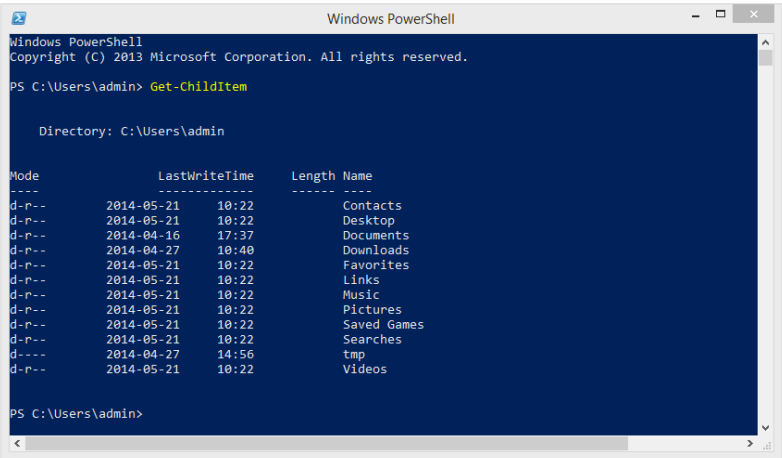
- L'architecte Jeff Snover sur le site StackOverflow répondant à la question si pour un utilisateur de Bash ça valait la peine d'apprendre PowerShell :
  - "My original intent was to include a set of Unix tools in Windows and be done with it (a number of us on the team have deep Unix backgrounds and a healthy dose of respect for that community.) What I found was that this didn't really help much. The reason for that is that awk/grep/sed don't work against COM, WMI, ADSI, the Registry, the cert store, etc, etc. In other words, UNIX is an entire ecosystem self-tuned around text files. As such, text processing tools are effectively management tools. Windows is a completely different ecosystem self-tuned around APIs and Objects. That's why we invented PowerShell."
  - "If/when you start to learn PowerShell, I think you'll be pretty happy. Much of the design is heavily influenced by our Unix backgrounds so while we are quite different, you'll pick it up very quickly (after you get over cussing that it isn't Unix :-)). We know that people have a very limited budget for learning - that is why we are super hard-core about consistency. You are going to learn something and then you'll use it over and over and over again."

Source: Jeffrey Snover, http://stackoverflow.com/questions/573623/is-powershell-ready-to-replace-my-cygwin-shell-on-windows/573861#573861

PowerShell

Lancer la console

- Depuis l'écran **Démarrer** taper **PowerShell** et cliquer sur l'icône **PowerShell** qui apparaît.
- Depuis l'écran bureau ouvrir le menu de démarrage, cliquer sur **Chercher**, taper PowerShell et cliquer sur l'icône **PowerShell** qui apparaît.



Le terminal PowerShell. Le terminal est parfois appelé application hôte *(host application)*.

PowerShell
Introduction

- On peut donner au shell une expression, il l'évaluera et affichera le résultat.

```
PS> 2 + 3 * 4
14
PS>
```

- On peut donner au shell une commande, il l'exécutera.
- Il y a quatre types de commandes :
  - les commandes natives Windows (par exemple ipconfig)
  - les scripts (scripts cmd.exe ou scripts PowerShell)
  - les fonctions PowerShell
  - les cmdlets PowerShell

```
PS> ipconfig

Windows IP Configuration

Ethernet adapter Ethernet0:

   Connection-specific DNS Suffix  . : localdomain
   Link-local IPv6 Address . . . . . : fe80::308e:3
   IPv4 Address. . . . . . . . . . . : 192.168.246.
   Subnet Mask . . . . . . . . . . . : 255.255.255.
   Default Gateway . . . . . . . . . : 192.168.246.
[...]

PS>
```


PowerShell
Les scripts

- Les scripts sont des fichiers de texte qui sont interprétés par l'interpréteur de commandes PowerShell.
  - Ils ont une extension `.ps1`.
  - Ils peuvent être écrits avec n'importe quel éditeur de texte.
  - Il existe de nombreux environnements de développement fournis par Microsoft et autres :
    - PowerShell ISE (gratuit)
    - PowerShell Analyzer (gratuit)
    - PowerGUI (gratuit)
    - PowerShell Plus (commercial)
    - PrimalScript (commercial)
    - Admin Script Editor

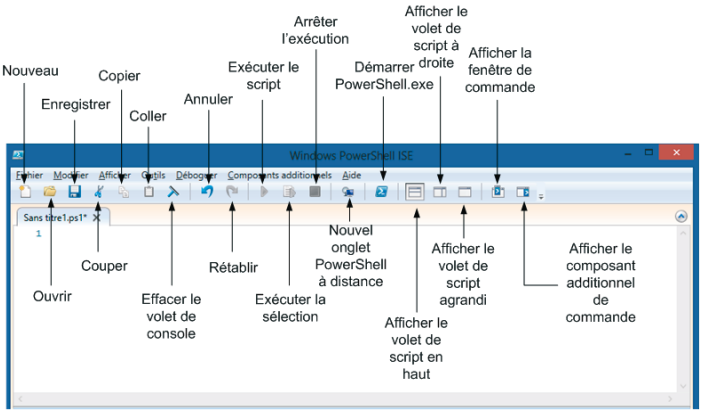
```
# ---------------------------------------------------------------------------
# NAME: CreateFileNameFromDate.ps1
# AUTHOR: ed wilson, Microsoft
# DATE:12/15/2008
#
# KEYWORDS: .NET framework, io.path, get-date
# file, new-item, Standard Date and Time Format Strings
# regular expression, fef, pass by reference
#
# COMMENTS: This script creates an empty text file
# based upon the date-time stamp. uses format string
# to specify a sortable date. uses getInvalidFileNameChars
# method to get all the invalid characters that are not allowed
# in a file name. It assumes there is a folder named fso off the
# c:\ drive. If the folder does not exist, the script will fail.
#
# ---------------------------------------------------------------------------
Function GetFileName([ref]$fileName)
{
 $invalidChars = [io.path]::GetInvalidFileNamechars()
 $date = Get-Date -format s
 $fileName.value = ($date.ToString() -replace "[$invalidChars]","-") + ".txt"
}

$fileName = $null
GetFileName([ref]$fileName)
new-item -path c:\fso -name $filename -itemtype file
```

PowerShell

Environnement intégré d'écriture de scripts, *Integrated Scripting Environment (ISE)*

- Le PowerShell ISE est une application graphique qui embarque un environnement d'exécution PowerShell et offre
  - un éditeur de script avec coloration de la syntaxe,
  - un explorateur de commandes,
  - un débogueur.
- Pour démarrer : lancer **powershell_ise.exe**.



PowerShell

Stratégies d'exécution *(execution policies)*

- PowerShell est un outil puissant et le détournement de cette puissance par des scripts malveillants présente un risque.
  - Mauvaises expériences avec des macros VisualBasic : épidémies de virus répétées
  - En conséquence Microsoft délivre PowerShell avec une configuration restrictive par défaut.
- Une *stratégie d'exécution (execution policy)* est une règle de base pour l'exécution des scripts. Il y en a six (voir page suivante).
  - PowerShell sait distinguer entre des scripts qui on été écrits sur la machine locale et des scripts qui ont été téléchargés ou copiés depuis d'autres machines ou Internet.
  - La plupart des outils de téléchargement marquent un fichier comme téléchargé à travers de métadonnées qu'ils ajoutent au fichier. Celles-ci sont examinées par PowerShell.

- La stratégie recommandée pour le développement est `RemoteSigned`. Pour la sélectionner, taper la commande
  - `Set-ExecutionPolicy RemoteSigned`

PowerShell

Stratégies d'exécution *(execution policies)*

Stratégie	Scripts écrits en local	Scripts téléchargés	
Restricted	Ne pas exécuter	Ne pas exécuter	Valeur par défaut
AllSigned	Exécuter si signé (authenticode)	Exécuter si signé	
RemoteSigned	Exécuter	Exécuter si signé	Recommandé pour le développement
Unrestricted	Exécuter	Dialogue d'avertissement puis exécuter	
Bypass	Exécuter	Exécuter	
Undefined	(Annulle la stratégie courante dans l'étendue courante.)		

PowerShell

Un shell orienté-objet

- Comme les shells Unix, PowerShell connaît le concept de pipeline. Au lieu d'envoyer du texte à travers le pipeline, PowerShell envoie des **objets**.

```
PS> Get-Process | Where-Object {$_.Handles -gt 750} | Sort-Object -Property PM -Descending

Handles  NPM(K)    PM(K)      WS(K) VM(M)   CPU(s)     Id ProcessName
-------  ------    -----      ----- -----   ------     -- -----------
    965      43   173992     107044   602   157.34   2460 MetroTwit
    784      21    88196      83588   290    19.84   5776 chrome
    952      44    39456      20100   287    29.27   2612 explorer
    784      34    34268       2836   109     4.56   3712 SearchIndexer
   1158      28    18868      14048   150     6.21    956 svchost
    779      14     3784       3900    36     4.46    580 lsass

PS>
```

PowerShell
Un shell orienté-objet

- Un objet est défini principalement en trois parties :
  - son type
  - ses méthodes
  - ses propriétés.
- Exemple : Objet service

Membre	Type du membre	Description
Nom du service	Propriété	Obtient le nom qui identifie le service.
Nom de l'ordinateur	Propriété	Obtient le nom de l'ordinateur sur lequel réside le service.
Type de service	Propriété	Obtient le type de service référencé par l'objet.
Statut	Propriété	Donne l'état du service.
Peut être arrêté ?	Propriété	Indique si le service peut être arrêté après avoir démarré.
Démarrer	Méthode	Démarre le service.
Arrêter	Méthode	Arrête le service
Interrompre	Méthode	Interrompt le fonctionnement du service.
Continuer	Méthode	Reprend le fonctionnement d'un service après qu'il a été suspendu.

PowerShell
Les cmdlets

- Une ***cmdlet*** (prononcer *command-let*) est une commande intégrée au shell.
  - Implémentée comme classe .NET et compilée dans une DLL.
  - Chargée au démarrage d'une session shell.
  - Consomme en entrée des objets .NET et produit en sortie des objets .NET.

```
PS> Get-ChildItem


    Directory: Z:\marcel.graf\tech\powershell\cours_ads\examples


Mode                LastWriteTime     Length Name
----                -------------     ------ ----
d----         2014-05-31     10:24            play
d----         2014-05-31     10:24            work
-----         2014-05-31     10:23     352608 ch1.txt
-----         2014-05-31     10:24       7136 ch2.txt


PS>
```

PowerShell

Cmdlets — Règles de nommage

- Les cmdlets respectent des règles de nommage bien précises.
  - Format : `<Verbe>-<Nom>`
  - Le verbe décrit le genre d'action à mener.
  - Le nom décrit l'objet sur lequel accomplir l'action.
- Quelques verbes courants
  - `Get / Set`
  - `New`
  - `Add / Remove`
  - `Start / Stop`
  - `Suspend / Resume`
  - `Enable / Disable`
  - `Import / Export`

Exemple: Commandes pour les processus
Get-Process
Start-Process
Stop-Process
Wait-Process
Debug-Process

PowerShell

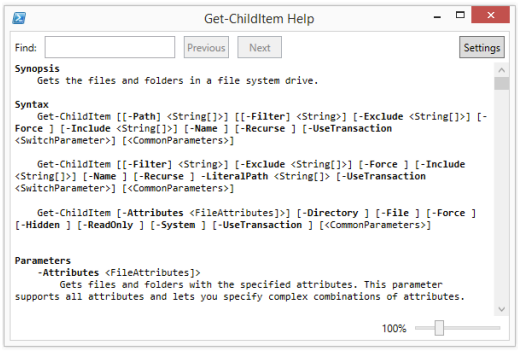
Obtenir de l'aide — Découvrir les commandes disponibles

- Il est très utile de découvrir les commandes disponibles à travers la commande `Get-Command`.
  - Chercher des commandes par motif
  - Recherche structurée, par nom, par verbe
- Exemples
  - Obtenir la liste de toutes les commandes :
    `Get-Command`
  - Obtenir la liste de toutes les commandes dont le nom contient `where` :
    `Get-Command *where*`
  - Obtenir toutes les commandes avec le nom `Process` :
    `Get-Command -Noun Process`
  - Obtenir toutes les commandes avec le verbe Set :
    `Get-Command -Verb Set`

PowerShell

Obtenir de l'aide — Afficher la documentation d'une commande

- PowerShell a un manuel intégré au shell à travers la commande `Get-Help`.
- Avant d'utiliser l'aide pour la première fois il faut télécharger les fichiers d'aide avec la commande `Update-Help`.
- Exemples
  - Obtenir le manuel d'une commande :
    `Get-Help <commande>`
  - Obtenir des exemples d'utilisation :
    `Get-Help <commande> -Examples`
  - Afficher l'aide complète :
    `Get-Help <commande> -Full`
  - Afficher l'aide complête dans une fenêtre séparée :
    `Get-Help <commande> -ShowWindow`
  - Afficher tous les concepts traités dans le manuel :
    `Get-Help about_*`



PowerShell

Obtenir de l'aide — Afficher la documentation d'une commande — La syntaxe

```
SYNTAX
    Get-ChildItem [[-Path] <String[]>] [[-Filter] <String>] [-Exclude
    <String[]>] [-Force] [-Include <String[]>] [-Name] [-Recurse]
    [-UseTransaction [<SwitchParameter>]] [<CommonParameters>]

    Get-ChildItem [[-Filter] <String>] [-Exclude <String[]>] [-Force]
    [-Include <String[]>] [-Name] [-Recurse] -LiteralPath <String[]>
    [-UseTransaction [<SwitchParameter>]] [<CommonParameters>]

    Get-ChildItem [-Attributes <FileAttributes]>] [-Directory] [-File]
    [-Force] [-Hidden] [-ReadOnly] [-System] [-UseTransaction]
    [<CommonParameters>]
```

Signification de la syntaxe

`-` indique un paramètre	`type[]` l'argument accepte plusieurs valeurs	`[Argument]` est positionnel
`< >` indique un argument		`[-Param argument]` est optionnel

PowerShell

Obtenir de l'aide — Afficher la structure orientée-objet des données produites par une commande

- Tous les cmdlets produisent des objets .NET.
  - PowerShell réutilise généralement des classes existantes de la bibliothèque .NET standard.
  - PowerShell rajoute généralement des propriétés aux objets pour uniformiser l'accès aux données.
- Quand on construit un pipeline on veut souvent savoir quels sont les propriétés des objets que l'on peut utiliser pour faire un traitement. On peut à tout moment inspecter le type, les propriétés et méthodes des objets en les pipant dans la commande `Get-Member`.
- Exemples :
  - Afficher le type et les propriétés des objets fournis par `Get-ChildItem` :
    `Get-ChildItem | Get-Member`
  - Seulement afficher les propriétés :
    `Get-ChildItem | Get-Member -MemberType Properties`
- Pour avoir la documentation complète sur une classe .NET consulter sa documentation dans le .NET Framework sur MSDN.

PowerShell

Obtenir de l'aide — Afficher la structure orientée-objet des données produites par une commande

Type utilisé pour décrire les répertoires

Type utilisé pour décrire les fichiers

```
PS> Get-ChildItem | Get-Member

   TypeName: System.IO.DirectoryInfo

Name                      MemberType     Definition
----                      ----------     ----------
Mode                      CodeProperty   System.String Mode{get=Mode;}
Create                    Method         void Create(), void Create(System.S...
CreateObjRef              Method         System.Runtime.Remoting.ObjRef Crea...
[...]
PSChildName               NoteProperty   System.String PSChildName=play
PSDrive                   NoteProperty   System.Management.Automation.PSDriv...
[...]
Attributes                Property       System.IO.FileAttributes Attributes...
CreationTime              Property       datetime CreationTime {get;set;}
[...]
BaseName                  ScriptProperty System.Object BaseName {get=$this.N...

   TypeName: System.IO.FileInfo

Name                      MemberType     Definition
----                      ----------     ----------
Mode                      CodeProperty   System.String Mode{get=Mode;}
AppendText                Method         System.IO.StreamWriter AppendText()
CopyTo                    Method         System.IO.FileInfo CopyTo(string de...
[...]
PSChildName               NoteProperty   System.String PSChildName=ch1.txt
PSDrive                   NoteProperty   System.Management.Automation.PSDriv...
[...]
Attributes                Property       System.IO.FileAttributes Attributes...
CreationTime              Property       datetime CreationTime {get;set;}
[...]
BaseName                  ScriptProperty System.Object BaseName {get=if ($th...
VersionInfo               ScriptProperty System.Object VersionInfo {get=[Sys...
```

PowerShell
Les alias

- Un *alias* est un surnom que l'on donne à une commande existante.
  - Par exemple `dir` est un alias pour le cmdlet `Get-ChildItem`.
- PowerShell vient avec un grand nombre d'alias prédéfinis. On peut distinguer trois classes d'alias qui ont des finalités différentes :
  - Économiser des frappes : les alias canoniques.
  - Rendre PowerShell plus familier aux utilisateurs de cmd.exe : les alias transitionnels cmd.exe.
  - Rendre PowerShell plus familier aux utilisateurs des shells Unix : les alias transitionnels Unix.

Exemples d'alias prédéfinis

Commande	Alias canonique	Alias transitionnel cmd.exe	Alias transitionnel Unix
`Get-ChildItem`	`gci`	`dir`	`ls`
`Get-Content`	`gc`	`type`	`cat`
`Copy-Item`	`ci`	`copy`	`cp`

PowerShell
Les alias — Manipuler les alias

But	Commande
Obtenir les alias de la session en cours	`Get-Alias`
Créer un alias	`New-Alias`
Modifier un alias	`Set-Alias`
Exporter les alias actuellement définis dans un fichier	`Export-Alias`
Importer une liste d'alias à partir d'un fichier	`Import-Alias`

- Recommandation
  - Utiliser les alias en mode interactif si désiré. (auteur = lecteur)
  - **Ne pas utiliser les alias dans les scripts.** Penser aux futurs lecteurs du script. (auteur ≠ lecteur)

PowerShell

Pipelines — Obtenir des données — `Get-*`

- Généralement un pipeline contient des objets .NET
  - Les cmdlets commençant avec le verbe `Get` permettent d'interroger diverses ressources et produisent une séquence d'objets .NET :
    - Obtenir les fichiers dans un répertoire : `Get-ChildItem`,
    - Obtenir les processus en mémoire : `Get-Process`,
    - Obtenir les services : `Get-Service`,
    - Obtenir les comptes utilisateur : `Get-ADUser`, etc.
- Quand on veut lire un fichier et traiter son contenu dans un pipeline deux cas se présentent :
  - Traiter les données comme chaînes de caractère
    - Par exemple en lisant le contenu d'un fichier texte avec `Get-Content` : `Get-Content ch1.txt`
    - `Get-Content` émet pour chaque ligne du fichier un objet du type `System.String`.
  - Traiter les données comme un tableau
    - Par exemple en lisant le contenu d'un fichier CSV avec `Import-Csv` : `Import-Csv accounts.csv`
    - `Import-Csv` émet pour chaque ligne du fichier un objet. Les propriétés des objets correspondent aux colonnes du fichier CSV.

PowerShell

Pipelines — Trier les objets — `Sort-Object`

- On peut trier les objets d'un pipeline grâce à la commande `Sort-Object`.
  - On spécifie sur quel propriété on veut trier avec le paramètre `-Property`.
  - On invertit l'ordre de tri avec l'option `-Descending`.
  - On peut éliminer les doublons après le tri avec l'option `-Unique` (On peut aussi utiliser la commande `Get-Unique` pour cela.)
- Exemples :
  - Trier les processus courants par consommation de temps processeur :

```
PS> Get-Process | Sort-Object -Property CPU -Descending

Handles  NPM(K)    PM(K)      WS(K) VM(M)   CPU(s)     Id ProcessName
-------  ------    -----      ----- -----   ------     -- -----------
   2043     119    90628      77912   723   558.47   1484 explorer
    142      12     6480       7188   115    99.55   2160 TPAutoConnect
    382      30    17144      15528   185    98.06   2432 vmtoolsd
    335      23    11020      10984   142    95.09   1756 Taskmgr
    825      53   232760     245512   777    50.44   2720 powershell
[...]
```

PowerShell

Pipelines — Sélectionner certaines propriétés d'objets — `Select-Object`

- Pour éliminer des propriétés dans un pipeline et laisser passer seulement quelques-unes, utiliser la commande `Select-Object`.
- Exemples
  - Lister les processus en cours, mais sélectionner uniquement les propriétés ProcessName et CPU :

```
PS> Get-Process | Select-Object -Property ProcessName,CPU

ProcessName    CPU
-----------    ---
BvSshServer
conhost        0.046875
conhost        7.640625
csrss
csrss
dwm
explorer       558.5
[...]
```


PowerShell

Pipelines — Grouper les objets en s'appuyant sur une valeur de propriété — `Group-Object`

- Pour organiser les objets en groupes en fonction d'une valeur de propriété utiliser la commande `Group-Object`.
- Exemple
  - Un service peut être dans un de deux états : `Running` ou `Stopped`. Grouper les services selon leur état :

```
PS> Get-Service | Group-Object -Property Status

Count Name    Group
----- ----    -----
  116 Stopped {AeLookupSvc, ALG, AppIDSvc, AppMgmt...}
   59 Running {Appinfo, AudioEndpointBuilder, Audiosrv, BFE...}
```

PowerShell

Pipelines — Filtrer les objets — `Where-Object`

- Pour éliminer certains objets de la pipeline et laisser passer d'autres utiliser la commande `Where-Object`.
  - Cette commande connaît deux syntaxes, une simplifiée pour des comparaisons simples, et une plus puissante s'appuyant sur des bouts de code passés dans un *scriptblock*.
- Exemple de la syntaxe simplifiée
  - Afficher tous les services qui sont dans l'état `Stopped` :

```
PS> Get-Service | Where-Object -Property Status -eq -Value Stopped

Status   Name               DisplayName
------   ----               -----------
Stopped  AeLookupSvc        Application Experience
Stopped  ALG                Application Layer Gateway Service
Stopped  AppIDSvc           Application Identity
[...]
```


PowerShell

Pipelines — Filtrer les objets avec un *scriptblock*

- Un *scriptblock* est une fonction qui n'a pas de nom.
  - Le code est entouré d'accolades `{ }`.
  - La dernière expression dans le *scriptblock* est utilisée comme valeur de retour.
- Quand on utilise `Where-Object` avec un *scriptblock*, ce dernier décide pour chaque objet s'il passe à l'étape suivante du pipeline ou non.
  - `Where-Object` appelle le *scriptblock* pour chaque objet.
  - Si le *scriptblock* retourne `true`, l'objet passe, si il retourne `false`, l'objet ne passe pas.
  - Le *scriptblock* reçoit l'objet dans la variable spéciale `$_`.
- Reprise de l'exemple précédent, afficher tous les services qui sont dans l'état `Stopped` :

```
PS> Get-Service | Where-Object { $_.Status -eq "Stopped" }

Status   Name               DisplayName
------   ----               -----------
Stopped  AeLookupSvc        Application Experience
Stopped  ALG                Application Layer Gateway Service
Stopped  AppIDSvc           Application Identity
[...]
```

PowerShell

Pipelines — Effectuer des opérations avec chaque objet — `ForEach-Object`

- La commande `ForEach-Object` sert à effectuer des actions sur chaque objet dans le pipeline.
  - Comme `Where-Object`, elle utilise des *scriptblocks*.
  - Elle en propose trois alors que Where-Object n'en propose qu'un :
    - Les actions à exécuter sur chaque objet (paramètre `-Process`).
    - Les actions à exécuter avant le traitement du premier objet (paramètre `-Begin`).
    - Les actions à exécuter après le traitement du dernier objet (paramètre `-End`).
- Exemple : Calculer la somme des tailles des fichiers .txt dans le répertoire courant :

```
PS> Get-ChildItem *.txt | ForEach-Object `
-Begin { $total = 0 } -Process { $total += $_.Length } -End { $total }
359744
```


PowerShell

Pipelines — Mettre en forme les objets — `Format-*`

- Il y a quatre cmdlets pour la mise en forme des objets d'un pipeline :
  - Afficher les objets sous forme de tableau : `Format-Table`
  - Afficher les objets sous forme de liste : `Format-List`
  - Afficher une seule propriété des objets efficacement : `Format-Wide`
  - Afficher les objets en préservant leur structure objet : `Format-Custom`
- Quand un pipeline ne se termine pas par une commande de mise en forme, PowerShell termine le pipeline par défaut avec `Format-Table`.
- Par défaut ces cmdlets montrent seulement les propriétés les plus importantes !
  - Passer les propriétés à afficher comme argument :
    `Format-Table -Property ProcessName,CPU,PeakWorkingSet`
  - Afficher toutes les propriétés : `Format-Table *`
- Attention ! La mise en forme ne préserve pas la structure des objets dans le pipeline. Prévoir la mise en forme **après** d'éventuelles manipulations d'objets.

PowerShell

Pipelines — Mettre en forme les objets — `Format-*`

`Format-Table`

```
Handles NPM(K)  PM(K)  WS(K) VM(M) CPU(s)   Id ProcessName
------- ------  -----  ----- ----- ------   -- -----------
      4      3    212   1048     7        1960 aitagent
    199     21   3748   2824    91        1236 BvSshServer
      3      2    204   1168     8   0.00 3944 cleanmgr
     41      5    604   2712    27         660 conhost
[...]
```

`Format-Wide`

```
audiodg             BvSshServer         conhost
conhost             csrss               csrss
dwm                 explorer            Idle
iexplore            iexplore            lsass
[...]
```

`Format-List`

```
Id      : 1236
Handles : 198
CPU     :
Name    : BvSshServer

Id      : 2204
Handles : 42
CPU     : 0.046875
Name    : conhost

Id      : 3612
Handles : 117
CPU     : 8.734375
Name    : conhost

Id      : 380
Handles : 432
CPU     :
Name    : csrss

[...]
```

`Format-Custom`

```
class Process
{
  Id = 1236
  Handles = 198
  CPU =
  Name = BvSshServer
}

class Process
{
  Id = 2204
  Handles = 42
  CPU = 0.046875
  Name = conhost
}

class Process
{
  Id = 3612
  Handles = 117
  CPU = 12.515625
  Name = conhost
}

[...]
```


PowerShell

Pipelines — Rediriger la sortie d'un pipeline — `Out-*`

- Par défaut PowerShell ajoute à la fin d'un pipeline la commande `Out-Default`.
  - Celle-ci met en forme la sortie si ce n'est pas déjà fait avec `Format-Table`...
  - ... et l'envoie ensuite vers la sortie par défaut, qui est `Out-Host`, qui est le terminal.
- Les autres cmdlets de sortie (`Out-*`) servent à
  - Écrire la sortie dans un fichier : `Out-File`,
  - Afficher la sortie dans explorateur de données interactif : `Out-GridView`,
  - Imprimer la sortie sur une imprimante : `Out-Printer`,
  - Écrire la sortie dans un objet chaîne de caractères : `Out-String`,
  - Détruire la sortie : `Out-Null`.
- On peut aussi utiliser les opérateurs de redirection :
  - `Get-ChildItem > files.txt` est équivalent à `Get-ChildItem | Out-File files.txt`.
  - `Get-ChildItem > $null` est équivalent à `Get-ChildItem | Out-Null`.

PowerShell

Système de fichiers et autres providers

- PowerShell offre un nombre de cmdlets pour naviguer et manipuler le système de fichiers
- Les mêmes commandes fonctionnent aussi pour
  - la registry
  - les variables d'environnement
- PowerShell généralise le concept d'un système de fichiers au concept d'un espace de nommage hiérarchique

PowerShell

Système de fichiers et autres providers — Cmdlets, alias et équivalences

	Cmdlet	**Canonical alias**	**Cmd.exe equivalent**	**UNIX sh equivalent**
Get the current directory	Get-Location	gl	cd	pwd
Change the current directory	Set-Location	sl	cd, chdir	cd, chdir
Copy files	Copy-Item	cpi	copy	cp
Remove a file or directory	Remove-Item	ri	del, rd	rm, rmdir
Move a file	Move-Item	mi	move	mv
Rename a file	Rename-Item	rni	ren	mv
Set the contents of a file	Set-Item	si		
Clear the content of a file	Clear-Item	cli		
Create a new empty file or directory	New-Item	ni		
Create a new empty directory (convenience)	MkDir		md	mkdir
Get the contents of a file	Get-Content	gc	type	cat
Set the contents of a file	Set-Content	sc	> file	> file

PowerShell
Opérateurs

- Les principaux groupes d'opérateurs avec quelques exemples

Opérateurs arithmétiques	Opérateurs d'affectation
`+ - * / %`	`= += -= *= /= %=`
Opérateurs de comparaison	**Opérateurs de relation**
`-eq -ne -gt -ge -lt -le`	`-contains -notcontains`
Opérateurs de recherche de motifs et de texte	**Opérateurs de manipulation binaire**
`-like -notlike -match -notmach -replace -split -join`	`-and -or -not -xor -band -bor -bnot -bxor`