Text 1
PowerShell objects, piping and filtering

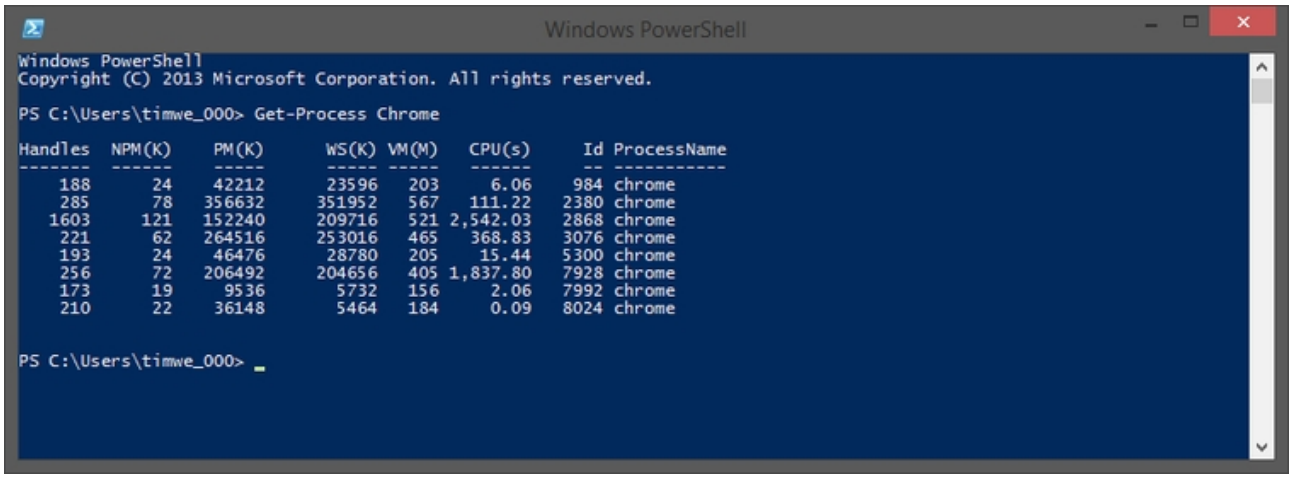
Quelle 1: Tom‘s IT PRO

Quelle: http://www.tomsitpro.com/articles/powershell-piping-filtering-objects,2-771.html

PowerShell objects, piping and filtering allow you to take PowerShell cmdlets to the next level. Here's how to use objects, piping and filtering along with some notes on best practices.

The key to PowerShell, in addition to the standardized naming and other features which make it so intuitive, is the fact that many of the cmdlets are object based. By working with objects, PowerShell allows you to easily make changes to multiple items with a single line of code, make changes to a specific subset of items within thousands, or use these objects to collect data or perform actions on other related objects.

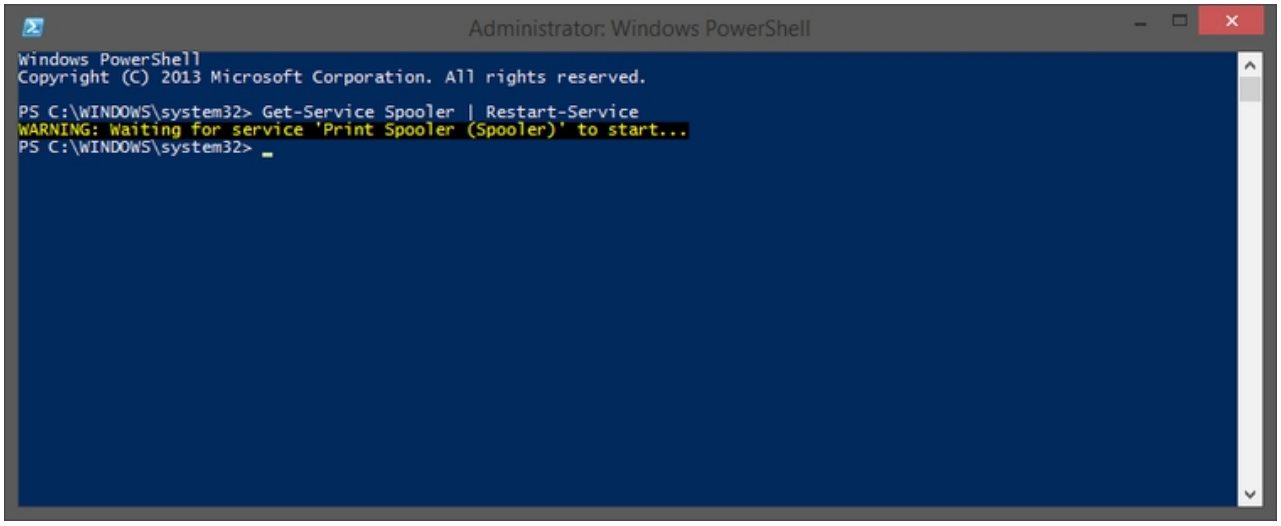
PowerShell Objects



Objects, for those not familiar with the terminology, refers to items which contain multiple attributes or properties; such as strings of characters, lists of information, and numerical values. A good example of an object is a Windows process, retrieved using the **Get-Process** cmdlet, which contains several properties indicating the executable name, priority, CPU utilization, and memory usage.

In the last PowerShell Basics article we discussed the **Get-Member** cmdlet, which can be used to explore objects and their member properties and methods. Not only will Get-Member show you the properties of an object and the data types they contain, but it will provide you with the object type as well, which can in turn be used to find other cmdlets which can accept a particular object type.

Piping



PowerShell allows you to leverage cmdlets and objects through a technique known as piping. Using the pipe character (|), you can quickly and easily select objects and then perform an action on them. A perfect example of piping is killing specific processes using **Get-Process java | Stop-Process**. Likewise you can restart services using a single line such as **Get-Service spooler | Restart-Service**. Often cmdlets with the same noun will be used when piping, but the technique is not limited to cmdlets with the same noun. Using the object type returned using Get-Member, you can find other cmdlets which can be used to receive a piped command. The Get-Command cmdlet with the object type specified using **-ParameterType** will return a list of cmdlets which can accept the designated object type.

Filtering

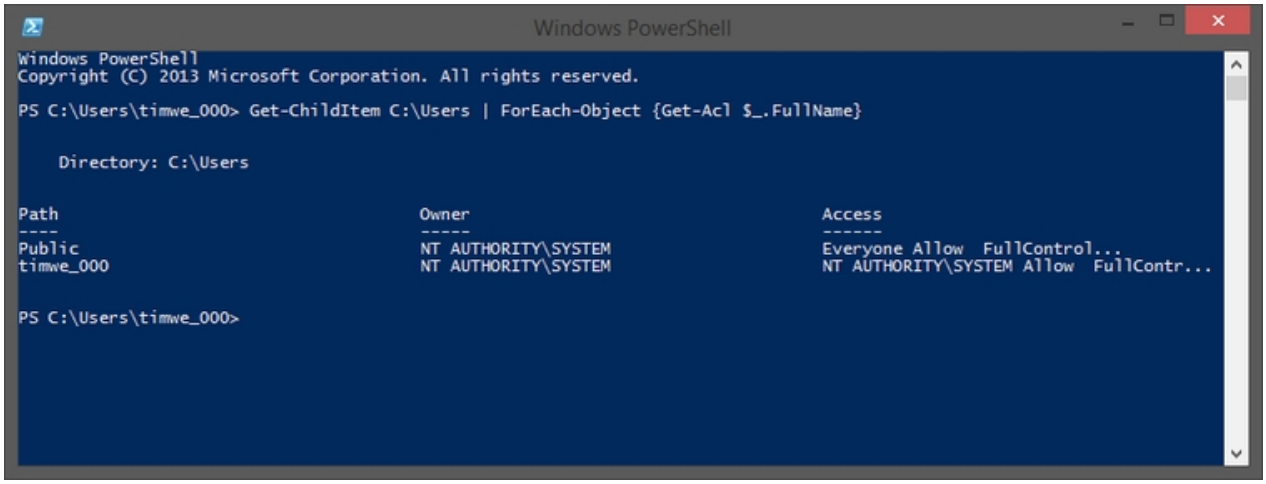
```
Windows PowerShell
Copyright (C) 2013 Microsoft Corporation. All rights reserved.

PS C:\Users\timwe_000> Get-Service | Where-Object {$_.DependentServices -ne $null}

Status   Name               DisplayName
------   ----               -----------
Running  AudioEndpointBu... Windows Audio Endpoint Builder
Running  BFE                Base Filtering Engine
Running  bthserv            Bluetooth Support Service
Running  CryptSvc           Cryptographic Services
Running  DeviceAssociati... Device Association Service
Running  Dhcp               DHCP Client
Running  Dnscache           DNS Client
Stopped  Eaphost            Extensible Authentication Protocol
Running  EventLog           Windows Event Log
Running  EventSystem        COM+ Event System
Running  fdPHost            Function Discovery Provider Host
Running  FDResPub           Function Discovery Resource Publica...
Running  iphlpsvc           IP Helper
Running  KeyIso             CNG Key Isolation
Running  LanmanServer       Server
Running  LanmanWorkstation  Workstation
Running  MMCSS              Multimedia Class Scheduler
Stopped  Netman             Network Connections
Running  netprofm           Network List Service
Running  NlaSvc             Network Location Awareness
Running  nsi                Network Store Interface Service
Running  p2pimsvc           Peer Networking Identity Manager
Running  PNRPsvc            Peer Name Resolution Protocol
Running  ProfSvc            User Profile Service
Stopped  RasMan             Remote Access Connection Manager
Running  SamSs              Security Accounts Manager
Running  SENS               System Event Notification Service
Running  Spooler            Print Spooler
Running  SSDPSRV            SSDP Discovery
Stopped  SstpSvc            Secure Socket Tunneling Protocol Se...
Stopped  TapiSrv            Telephony
Stopped  TermService        Remote Desktop Services
Running  VaultSvc           Credential Manager
Running  Wcmsvc             Windows Connection Manager
Running  WinHttpAutoProx... WinHTTP Web Proxy Auto-Discovery Se...
Running  Winmgmt            Windows Management Instrumentation
Running  WSearch            Windows Search
Running  wudfsvc            Windows Driver Foundation - User-mo...

PS C:\Users\timwe_000>
```

There is a whole list of cmdlets in PowerShell used for performing heavy lifting with objects, specifically those with the Object noun. Many of these cmdlets are among the most commonly used cmdlets, while others are used for more specialized tasks.

The **Where-Object** cmdlet allows you to limit or filter the object being passed to the pipeline. For example the command **Get-Service | Where-Object {$_.DependentServices -ne $null}** will return a list of services which have dependencies. The syntax used with Where-Object is worth noting, and applies to some of the other object cmdlets as well. The squiggly brackets are used to delineate a code block in PowerShell, and in this case indicate the condition being applied to the object in the pipeline. The automatic variable $_ is used to indicate the current instance of the object being evaluated. PowerShell comparison operators use hyphenated formatting, so -eq (equals) is used to find an exact match with the word "Stopped" in our example.

For interactive use within the PowerShell console, using aliases can save time and effort. The Where-Object cmdlet makes use of the question mark (?). PowerShell 3.0 even allows you to simplify your Where-Object syntax even more by removing the need for the script block and the automatic variable from the pipeline. In PowerShell 3.0 this command is equivalent to the one provided above: **Get-Service | ? DependentServices -ne $null**.

Acting On Objects



ForEach-Object is used to perform an action on each instance of an object. From a syntax perspective ForEach-Object is very similar to Where-Object, with both the script block and automatic variables being used with both cmdlets. Where ForEach-Object excels is being able to perform tasks against each object instance which are too complex for simple piping. For example you may need to list the file security for a file share, in which case you could use pipe the **Get-ChildItem** cmdlet to **ForEach-Object**, and then use **Get-ACL** against the fullname parameter ($_.FullName) to list the file security for the list of files.

As with Where-Object the ForEach-Object cmdlet can be simplified using an alias, indicated with the percent sign (%). Also the PowerShell 3.0 syntax is supported in order to provide even more intuitive use.

While filtering an object or performing an action on instances of an object are both common tasks, it's a good idea to avoid both Where-Object and ForEach-Object when possible. Many cmdlets offer a **-Filter** option or other parameters which can help limit the number of results without having to look at each instance of an object, typically resulting in a significant performance improvement. Likewise ForEach-Object performs an individual action on each instance of the piped object. When possible, objects should be piped directly to cmdlets which can perform the required action on the entire object, without having to enumerate each item within the object.