Giới thiệu về PowerShell

Windows PowerShell là một trình tiện ích dòng lệnh và ngôn ngữ kịch bản mới do Microsoft cung cấp. Tạo sao lại phải nghiên cứu và quan tâm đến PowerShell? Vì nó là một trình tiện ích kiểu mới khác biệt? Tất nhiên, mọi tiện ích mới đều được khẳng định là "khác" so với những cái cũ, nhưng PowerShell có một số thành phần thực sự phân biệt được với các trình tiện ích khác. Ở bài này chúng ta sẽ xem xét một số thành phần ngôn ngữ kịch bản của PowerShell và tạo một kịch bản (script) PowerShell ví dụ từ những thứ dường như rất lộn xộn.

Lịch sử vắn tắt của tiện ích dòng lệnh Windows

Sau khi Windows NT ra đời, CMD.EXE trở thành tiện ích dòng lệnh cho Windows. Mặc dù CMD.EXE thừa hưởng một số thành phần của người tiền nhiệm DOS (COMMAN.COM), nhưng nó vẫn dựa trên một ngôn ngữ kịch bản khá "sơ khai": sử dụng các file Windows Command (.CMD và .BAT). Sự bổ sung của Windows Scripting Host và các ngôn ngữ VBScript, JScript đã nâng cao đáng kể tính năng kịch bản cho trình tiện ích.

Những công nghệ này là sự kết hợp khá cân bằng giữa tiện ích dòng lệnh nâng cao và môi trường kịch bản. Thực ra vấn đề có thể thao tác được với bao nhiêu file CMD.EXE, .CMD và Windows Scripting Host không phải là mối quan tâm thực sự. Thứ khiến người ta phàn nàn và bận tâm nhất là hoàn thành một số nhiệm vụ dường như đơn giản.

Sử dụng "khung làm việc" của các công cụ dòng lệnh và kịch bản, bất kỳ script ở mức tổng hợp vừa phải cũng đòi hỏi phải có sự kết hợp của cả các lệnh batch, Windows Scripting Host và các thực thi độc lập. Mỗi script đều sử dụng các quy ước khác nhau cho quá trình thực thi và yêu cầu, phân tích cú pháp, trả về dữ liệu.

Những biến yếu hỗ trợ trong CMD.EXE, các giao diện không nhất quán và khả năng truy cập giới hạn thiết lập Windows, kết hợp với một điểm yếu khác khiến kịch bản dòng lệnh trở nên khó khăn hơn trong triển khai và sử dụng. Chắc bạn sẽ thắc mắc ngay 'một điểm yếu khác' ở đây là gì? Xin thưa rằng đó là văn bản thuần tuý (text). Ở những công nghệ này, mọi thứ đều có dạng text. Dữ liệu đầu ra (output) của một lệnh hay kịch bản là text và phải được phân tích cú pháp cũng như định dạng lại để hoạt động như dữ liệu đầu vào (input) cho lệnh tiếp theo. Đây chính là điểm xuất phát cơ bản mà PowerShell lấy ra từ tất cả các trình tiện ích truyền thống.

Kịch bản PowerShell = Các file Batch trên Steroids

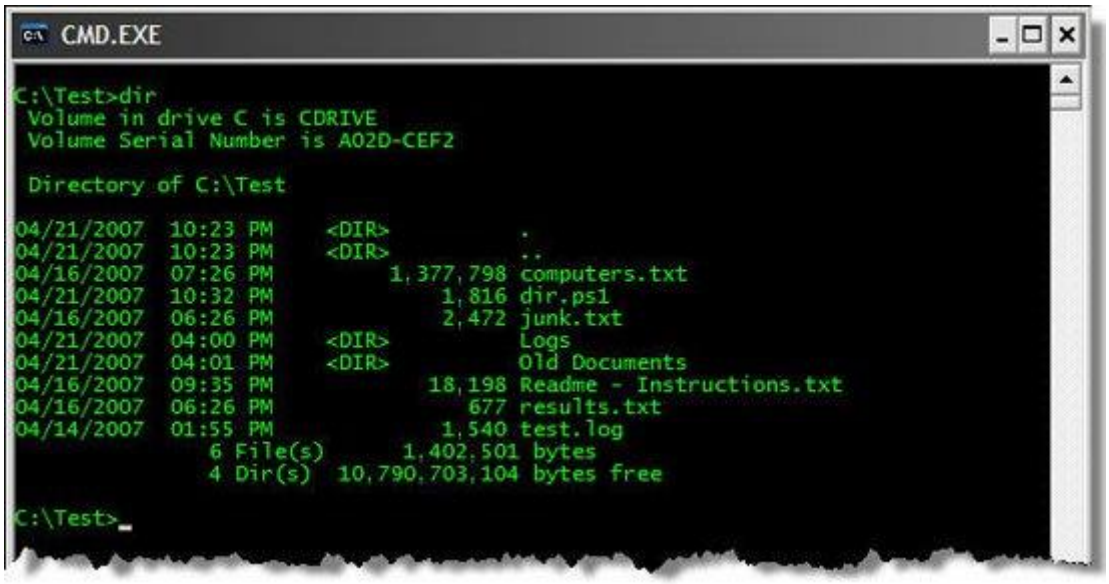
Bản thân PowerShell được viết trong ngôn ngữ .NET và dựa chủ yếu trên .NET Framework. Bởi vậy mà PowerShell được thiết kế như một trình tiện ích hướng đối tượng và ngôn ngữ kịch bản. Tất cả trong PowerShell đều được xem như một đối tượng với đầy đủ tính năng của .NET Framework. Một lệnh đưa ra tập hợp các đối tượng có thể được sử dụng bằng cách dùng thuộc tính và phương thức của kiểu đối tượng đó. Khi bạn muốn đưa dữ liệu đầu ra của một lệnh vào ống dẫn cho một lệnh khác, PowerShell thực tế sẽ cho đối tượng qua, không phải chỉ là dữ liệu đầu ra dạng text của dòng lệnh đầu tiên. Điều này tạo cho lệnh tiếp theo khả năng truy cập đầy đủ tất cả thuộc tính và phương thức của đối tượng trong đường ống dẫn.

Việc coi tất cả mọi thứ như là một đối tượng và khả năng chấp thuận đối tượng giữa các lệnh là một thay đổi lớn về mặt lý thuyết cho các tiện ích dòng lệnh. Điều đó nói lên rằng, PowerShell vẫn hoạt động giống như một trình tiện ích shell truyền thống. Lệnh, kịch bản, thực thi có thể được gõ và chạy từ dòng lệnh và kết quả được hiển thị ở dạng text. Các file windows .CMD và .BAT, VBScripts, JScripts và các thực thi hoạt động bên trong CMD.EXE, tất cả vẫn chạy trong PowerShell. Tuy nhiên, vì chúng không hướng đối tượng nên không có quyền truy cập đầy đủ tới những đối tượng được tạo và dùng trong PowerShell. Các kịch bản và thực thi di sản này vẫn sẽ coi mọi thứ dưới dạng text, nhưng bạn có thể kết hợp PowerShell với một số công nghệ khác. Đây là điểm rất quan trọng nếu bạn muốn bắt đầu sử dụng PowerShell bằng một tập tổng hợp các script đã tồn tại mà không thể chuyển đổi hết chúng trong một lần.

Một PowerShell Script

Đọc và hiểu cái tuyệt vời của công nghệ mới là một chuyện, còn xem xét và sử dụng nó lại là chuyện khác! Ở phần còn lại của bài này, chúng ta sẽ cùng phát triển một script PowerShell để minh chứng cho những khả năng cũng như cách sử dụng của nó.

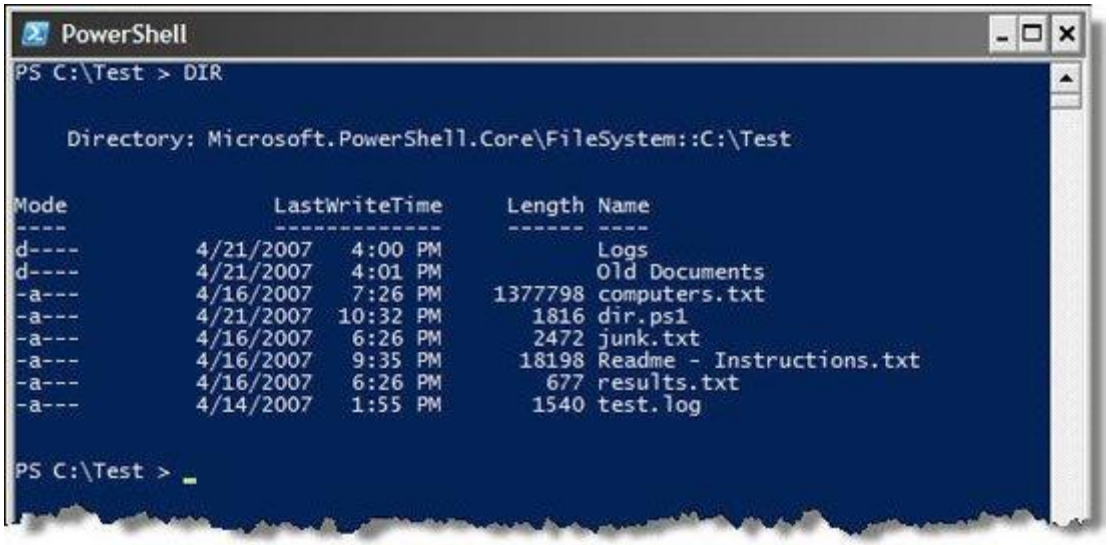
DIR là một trong những lệnh phổ biến nhất ở CMD.EXE. Lệnh này đưa ra tất cả file và thư mục con chứa trong một thư mục mẹ (như Hình 1). Cùng với tên của từng đối tượng, thông tin đưa ra còn có ngày giờ update mới nhất và kích thước của từng file. DIR còn hiển thị kích thước tổng hợp của tất cả các file trong thư mục, cũng như tổng số file và tổng thư mục con.



Hình 1

Chạy DIR trong PowerShell cũng đưa ra một danh sách thư mục như Hình 2, nhưng hơi khác một chút. PowerShell không có lệnh DIR mà thay vào đó là Get-ChildItem, cũng thực hiện chức năng tương tự. Trong PowerShell, DIR là một bí danh cho Get-ChildItem. Tôi không có ý định đi sâu vào những bí danh trong bài này. Bạn có thể tưởng tượng DIR trong PowerShell như là một tên viết tắt cho Get-ChildItem.

DIR trong PowerShell cung cấp nhiều thông tin giống như đã nói ở trên: một danh sách file và folder, ngày tháng thời gian update lần cuối và kích thước từng file. Tuy nhiên nó thiếu phần thông tin tóm tắt mà DIR trong CMD.EXE cung cấp: tổng kích thước của tất cả các file trong thư mục, tổng số file và tổng số thư mục con.



Hình 2

Với kịch bản ví dụ, bạn sẽ cần tạo một script PowerShell mô phỏng lệnh CMD.EXE DIR. Bên dưới tôi sẽ giải thích các phần cốt yếu nhất trong một script.

DIR.PS1: Tiêu đề (Header)
Một script PowerShell bao gồm các lệnh PowerShell trong một file văn bản thuần tuý với đuôi mở rộng .PS1. Thay thế DIR, bạn sẽ dùng một file text gọi là DIR.PS1.

Để chạy script, gõ lệnh như sau ở màn hình của PowerShell:

.DIR.PS1 X:Folder

Trong đó X là ký tự phân vùng ổ (như C, D, E) và Folder là tên thư mục.

Nếu muốn biết một số thông tin về phân vùng ổ, bạn sẽ phải dùng Windows Management Instrumentation (WMI). Chi tiết về WMI nằm ngoài phạm vi của bài này nên chúng ta sẽ không đề cập tới ở đây. Nhưng đoạn mã PowerShell bên dưới cũng khá dễ hiểu để không cần dùng đến trợ giúp của WMI. Bạn có thể tạo một biến "$filter" để dùng với lệnh Get-WmiObject. Biến lọc này (tức filter) nói với lệnh Get-WmiObject rằng bạn chỉ muốn có thông tin về một ổ đĩa cụ thể. Kết quả của lệnh Get-WmiObject được lưu trữ ở một biến gọi là $volInfo. Nhớ rằng, trong PowerShell mọi thứ đều là đối tượng; $volInfo bây giờ cũng là một đối tượng kết quả trả ra từ Get-WmiObject.

```
$filter = "DeviceID = '" + $drive + ":'"
$volInfo = Get-WmiObject -Class Win32_LogicalDisk -Filter $filter
```

Bây giờ bạn đã có thể truy cập tất cả các đối tượng và phương thức gắn với đối tượng. Dãy số của phân vùng ổ có thể truy cập qua thuộc tính VolumeSerialNumber. Thông tin trả về là một dãy số dạng xâu 8 ký tự. Nhưng thường thì bạn muốn định dạng nó theo kiểu bốn số tách riêng một, cách nhau bởi một dấu nối ngang. có thể thực hiện tương tự như ở dòng bên dưới. Dấu nối ở cuối dòng thứ nhất là ký tự nối tiếp dòng trong PowerShell. Về cơ bản, nó chỉ nói cho PowerShell biết rằng dòng không bị ngắt quãng mà bao gồm cả dòng tiếp theo. Khi viết mã không cần phân tách dòng, nhưng để giảm bề rộng và để cho đoạn mã dễ đọc, bạn nên thực hiện điều này.

```
$serial = $volInfo.VolumeSerialNumber.SubString(0, 4) + "-" + `
        $volInfo.VolumeSerialNumber.SubString(4, 4)
```

Bây giờ đã có đối tượng $volInfo, bạn có thể viết thông tin header DIR cho màn hình. Nếu ổ đĩa không có tên, đoạn text viết cho màn hình sẽ khác một chút so với ổ đĩa có tên. Một lệnh If-Else đơn giản được dùng để kiểm tra xem thuộc tính VolumeName có là xâu rỗng hay không. Lệnh Write-Host được dùng để viết từng dòng lệnh cho màn hình.

If ($volInfo.VolumeName -eq "") { Write-Host (" Volume in drive " + $drive + " has no label") } Else { Write-Host (" Volume in drive " + $drive + " is " + $volInfo.VolumeName) } Write-Host (" Volume Serial Number is " + $serial) Write-Host ("`n Directory of " + $args[0] + "`n")

Ký tự " `n " ở đầu và cuối lệnh Write-Host được dùng để chèn dòng mới vào trước và sau văn bản. Lệnh Write-Host thêm một dòng mới vào cuối mỗi dòng. Vì thế tác động của " `n " là tạo dòng trống trước và sau dòng văn bản.

Bạn có chú ý đến cụm "-eq" trong lệnh If? Đó là một toán tử so sánh bằng. Bảng bên dưới sẽ cho bạn biết tất cả các toán tử so sánh:

-eq, -ieq	So sánh bằng
-ne, -ine	So sánh không bằng
-gt, -igt	So sánh lớn hơn
-ge, -ige	So sánh lớn hơn hoặc bằng
-lt, -ilt	So sánh nhỏ hơn
-le, -ile	So sánh nhỏ hơn hoặc bằng

Ký tự -i ở trước các toán tử so sánh là để chỉ toán tử đó không phân biệt chữ hoa, chữ thường.

Hình 3: Dữ liệu đầu ra (output) của script hiện bạn đang có

DIR.PS1: Danh sách file/thư mục

Bây giờ, bạn đã sẵn sàng hiển thị nội dung và đặc tính của thư mục này. Điều đầu tiên cần làm là gọi lệnh PowerShell Get-ChildItem để đưa ra tập hợp các file và đưa vào script như một tham số. Lệnh Get-ChildItem sẽ lấy ra tập hợp đối tượng file và folder, không chỉ tên mà còn dẫn ống các đối tượng này trực tiếp vào lệnh Sort-Object để sắp xếp chúng. Mặc định lệnh Sort-Object sẽ sắp xếp đối tượng dựa theo thuộc tính Name. Vì thế bạn không cần phải mô tả bất kỳ tham số khác nào. Tập hợp các đối tượng đã sắp xếp sau đó sẽ được lưu trữ trong một biến có tên $items.

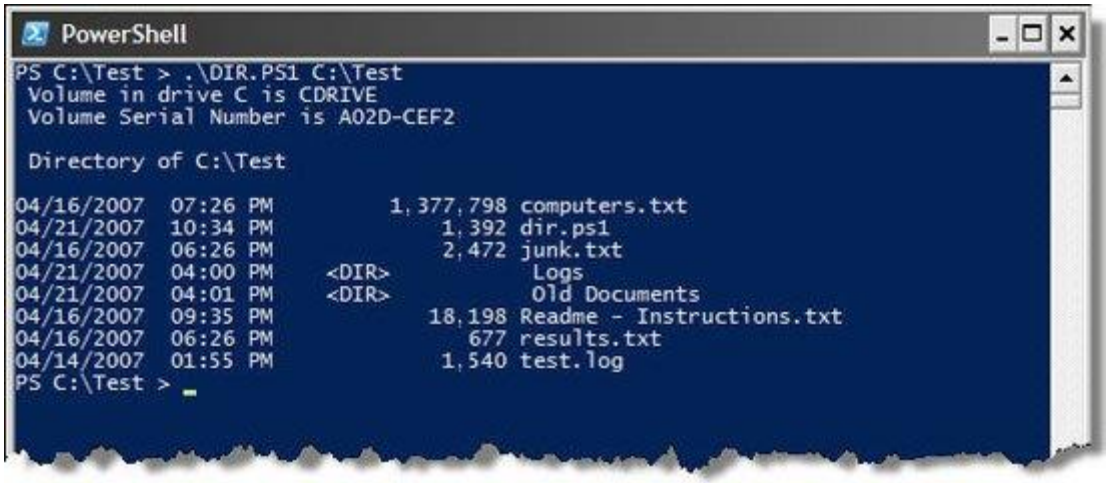
$items = Get-ChildItem $args[0] | Sort-Object

Sau khi có được tập hợp đối tượng file và folder, bạn cần lặp vòng lại chúng và hiển thị những đặc trưng phù hợp. Lệnh dùng cho việc này là ForEach. Với từng file hay folder, đặc trưng được hiển thị ra sẽ là ngày giờ update cuối cùng, tên, chiều dài hay kích thước file. Thứ lạ trông như các xâu ký tự bên trong ngoặc tròn là mã định dạng xâu .NET. Chúng được dùng để căn lề trái/phải cho các trường và định dạng ngày tháng, thời gian, số. Việc hiểu các mã định dạng xâu này không quan trọng lắm, vì chúng không phải là thứ cốt yếu phản ánh bản chất của script này.

Lệnh If là nơi bạn xác định xem liệu có đối tượng nào là thư mục hay không. Nếu ký tự đầu tiên của thuộc tính Mode là “d”, đối tượng là một thư mục. Bạn cần kiểm tra lại vì mã viết cho thư mục thường khác với mã viết cho file.

Chú ý dòng $totalDirs++ bên trong lệnh If. Đây là bộ đếm chịu trách nhiệm theo dõi số thư mục. Tương tự, có một biến $totalFiles được dùng để theo dõi tổng kích thước của tất cả các file. Những giá trị này luôn được tính toán trong quá trình thực thi. Nhưng chúng chỉ được hiển thị khi quá trình lập danh sách file kết thúc.

```
ForEach ($i In $items)
{
 $date = "{0, -20:MM/dd/yyyy hh:mm tt}" -f $i.LastWriteTime
 $file = $i.Name
 If ($i.Mode.SubString(0, 1) -eq "d")
 {
   $totalDirs++
   $list = $date + "   {0, -15}" -f "
     " + " " + $file
 }
 Else
 {
   $totalFiles++
   $size = "{0, 18:N0}" -f $i.Length
   $list = $date + $size + " " + $file
 }
 $totalSize += $i.Length
 Write-Host $list
}
```

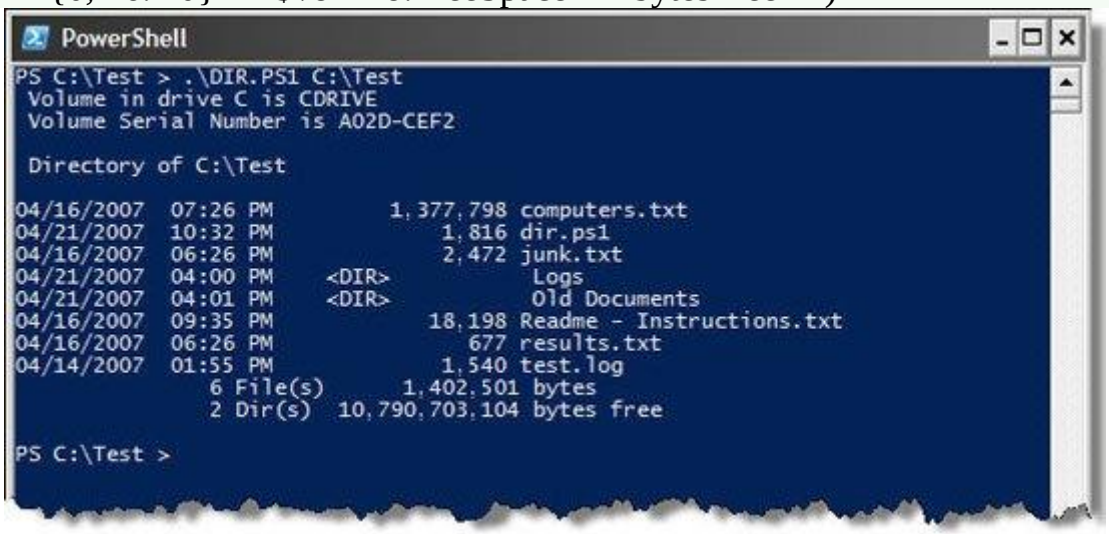


Hình 4: Hiển thị dữ liệu xuất của script được update.

DIR.PS1: Hậu đề (Footer)

Chỉ còn một việc còn lại là viết lên màn hình tổng số file, thư mục, kích thước tổng cộng của tất cả các file và không gian trống trên phân vùng ổ này. Để thực hiện bạn sẽ cần sử dụng các biến bộ đếm ($totalFiles, $totalDirs, $totalSize) đã tạo ở phần trước. Bạn có thể biết được lượng không gian trống từ biến $volInfo đã tạo từ lúc bắt đầu viết script.

```
Write-Host ("{0, 16:N0}" -f $totalFiles + " File(s)" + `
        "{0, 15:N0}" -f $totalSize + " bytes")
Write-Host ("{0, 16:N0}" -f $totalDirs + " Dir(s)" + `
        "{0, 16:N0}" -f $volInfo.FreeSpace + " bytes free`n")
```



Hình 5: Hiển thị hoàn toàn dữ liệu đầu ra (output) của script.

Những dự báo và khả năng nâng cao có thể

Mặc dù script mà bạn tạo đưa ra dữ liệu đầu ra gần giống hệt của lệnh CMD.EXE DIR nhưng cũng có một số dự báo bạn cần biết đến và một số nâng cao có thể thực hiện được.

- Bản script này không thực hiện bất kỳ kiểm tra lỗi nào.
- Nếu một đường dẫn hợp lệ không được đưa vào script, script sẽ hỏng với một thông báo lỗi PowerShell.
- Tổng số thư mục đưa ra trong script ít hơn 2 so với kết quả từ lệnh CMD.EXE DIR vì lệnh Get-ChildItem không tính hai thư mục "." và ".." như ở CMD.EXE.
- Script của bạn chỉ sắp xếp thứ tự theo tên file, tên thư mục và không cung cấp thêm bất kỳ cách sắp xếp theo thuộc tính nào khác.
- Script của bạn không có khả năng hiển thị nội dung thư mục và tất cả thư mục con.

Kết luận

Mặc dù PowerShell là một trình tiện ích và ngôn ngữ scripting mạnh nhưng bạn chỉ cần bỏ ra ít thời gian là đã có thể nắm bắt và sử dụng nó, nhất là khi chưa quen thuộc với môi trường .NET Framework. Tôi hy vọng bài báo này cùng với script ví dụ sẽ hữu ích cho bất kỳ ai muốn hiểu về PowerShell. Nhưng script được tạo ví dụ trong bài khá đơn giản. Tin rằng nó có thể được xây dựng, phát triển hoàn thiện hơn để phục vụ tốt cho nhiều trình ứng dụng phức tạp hơn.