

第 6 章 管道：连接命令

在第 4 章中已经介绍过在 PowerShell 中运行命令的方式和其他 Shell 并无不同：输入一个命令名，传输给它一些参数，然后按“Enter”键。让 PowerShell 独树一帜的不是运行命令的方式，而是它提供了管道功能，通过管道功能，只需要在一个序列行中，多个命令就可以很好地彼此连接。

6.1 一个命令与另外一个命令连接：为你减负

PowerShell 通过管道（pipeline）把命令互相连接起来。管道通过传输一个命令，将其输出作为另外一个 Cmdlet 的输入，使得第二个命令可以通过第一个的结果作为输入并联合起来运行。

你已经见过如“Dir | More”命令的运行情况，它把“Dir”命令的输出以管道方式传输给“More”命令。“More”命令把目录每次展现到一个页中。PowerShell 把管道的概念有效延伸。实际上，PowerShell 的管道类似 Unix 和 Linux 的 Shell 中的管道功能。你将会在下面认识到，PowerShell 的管道功能是非常强大的。

6.2 输出结果到 CSV 或 XML 文件

下面尝试几个命令，比如：

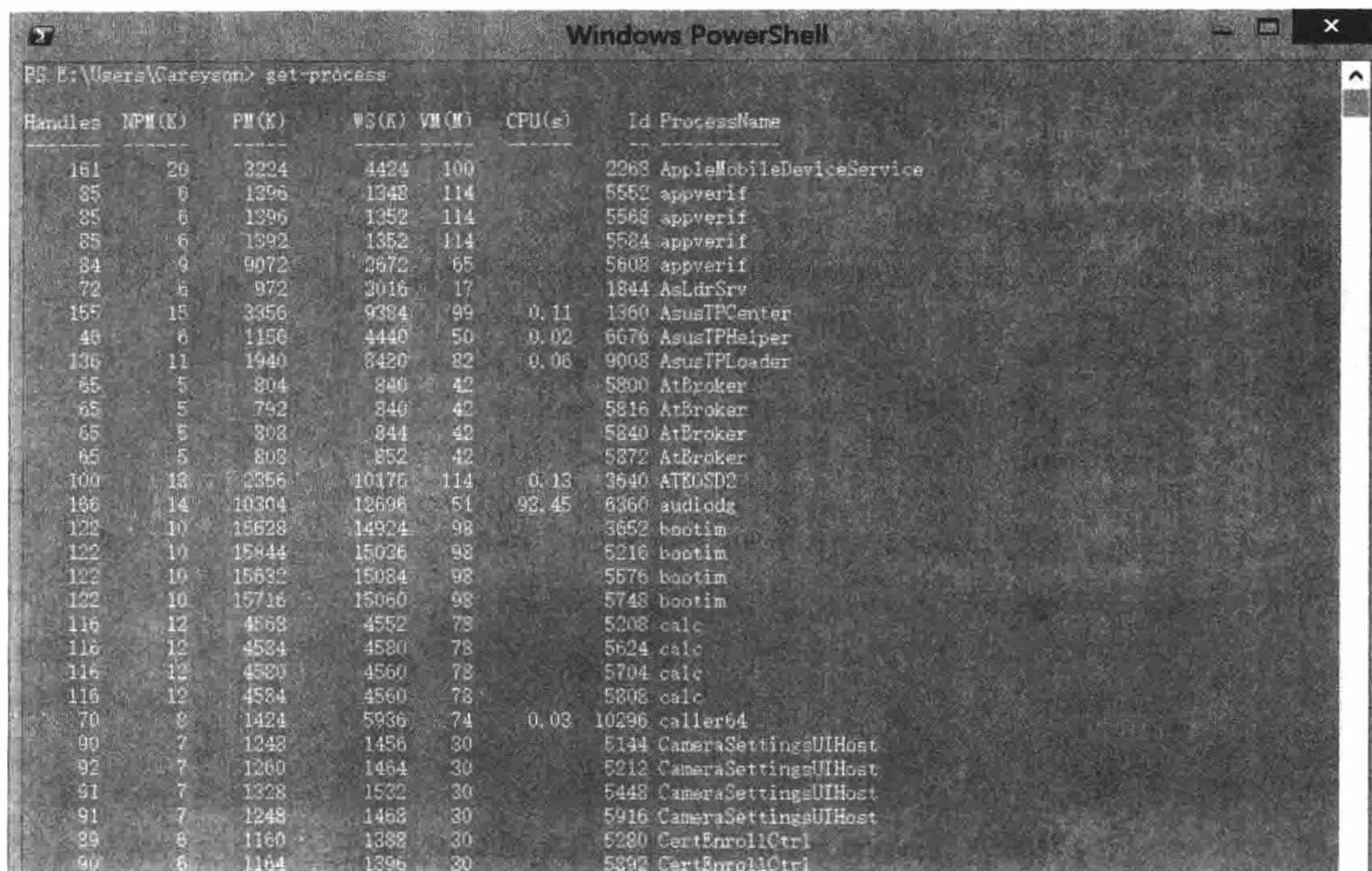
- Get-Process（或者 Ps）
- Get-Service（或者 Gsv）
- Get-EventLog Security-newest 100

这里提到这些命令是因为它们相对简单、直观。其中括号部分是分别对应“Get-Process”和“Get-Service”的别名。对于“Get-EventLog”，我们强制使用了“-newest”

参数，避免命令运行太久。

动手实验：选择你想尝试的命令动手尝试。下面将使用“Get-Process”作为演示。当然，你可以选择其他命令，或者都尝试，以便查看它们的差异。

当运行“Get-Process”时，屏幕会显示出图 6.1 的结果。



Handles	NPM(K)	PM(K)	WS(K)	VM(M)	CPU(s)	Id	ProcessName
161	20	3224	4424	100		2268	AppleMobileDeviceService
85	6	1296	1348	114		5552	appverif
85	6	1296	1352	114		5568	appverif
85	6	1292	1352	114		5524	appverif
84	9	9072	2672	65		5608	appverif
72	5	972	3016	17		1844	AsLdrSrv
155	15	3356	9384	99	0.11	1360	AsusTPCenter
40	6	1156	4440	50	0.02	6676	AsusTPHelper
136	11	1940	8420	82	0.06	9008	AsusTPLoader
65	5	804	840	42		5800	AtBroker
65	5	792	840	42		5816	AtBroker
65	5	808	844	42		5840	AtBroker
65	5	808	852	42		5872	AtBroker
100	13	2356	10176	114	0.13	3640	ATEOSDE
166	14	10304	12696	51	92.45	6360	audiodg
122	10	15628	14924	98		3652	bootim
122	10	15444	15036	98		5216	bootim
122	10	15632	15084	98		5576	bootim
122	10	15716	15060	98		5748	bootim
116	12	4568	4552	78		5208	calc
116	12	4584	4580	78		5624	calc
116	12	4580	4560	78		5704	calc
116	12	4584	4560	78		5808	calc
70	8	1424	5936	74	0.03	10296	caller64
90	7	1248	1456	30		5144	CameraSettingsUIHost
92	7	1260	1464	30		5212	CameraSettingsUIHost
91	7	1328	1532	30		5448	CameraSettingsUIHost
91	7	1248	1468	30		5916	CameraSettingsUIHost
29	6	1160	1388	30		5280	CertEnrollCtrl
90	6	1164	1396	30		5392	CertEnrollCtrl

图 6.1 “Get-Process” 的输出是一个带有几列信息的表格

虽然屏幕上展示了结果，但是也许不是你想要的，比如如果你想把内存和 CPU 的利用率整理成一些图表，那么可能需要把数据导出到 CSV 文件中，比如微软的 Excel。

6.2.1 输出结果到 CSV

管道和另外一个命令可以在导出文件时派上用场：

```
Get-Process | Export-CSV procs.csv
```

类似于用管道把“Dir”连接到“More”，我们已经把进程信息传输到“Export-CSV”中。第二个 Cmdlet 有一个强制的位置参数，用于指定输出文件名。因为“Export-CSV”是一个内置的 PowerShell Cmdlet，它知道如何把通过“Get-Process”产生的常规表格转换到一个普通的 CSV 文件中。

现在用 Windows 记事本打开文件，如图 6.2 所示。

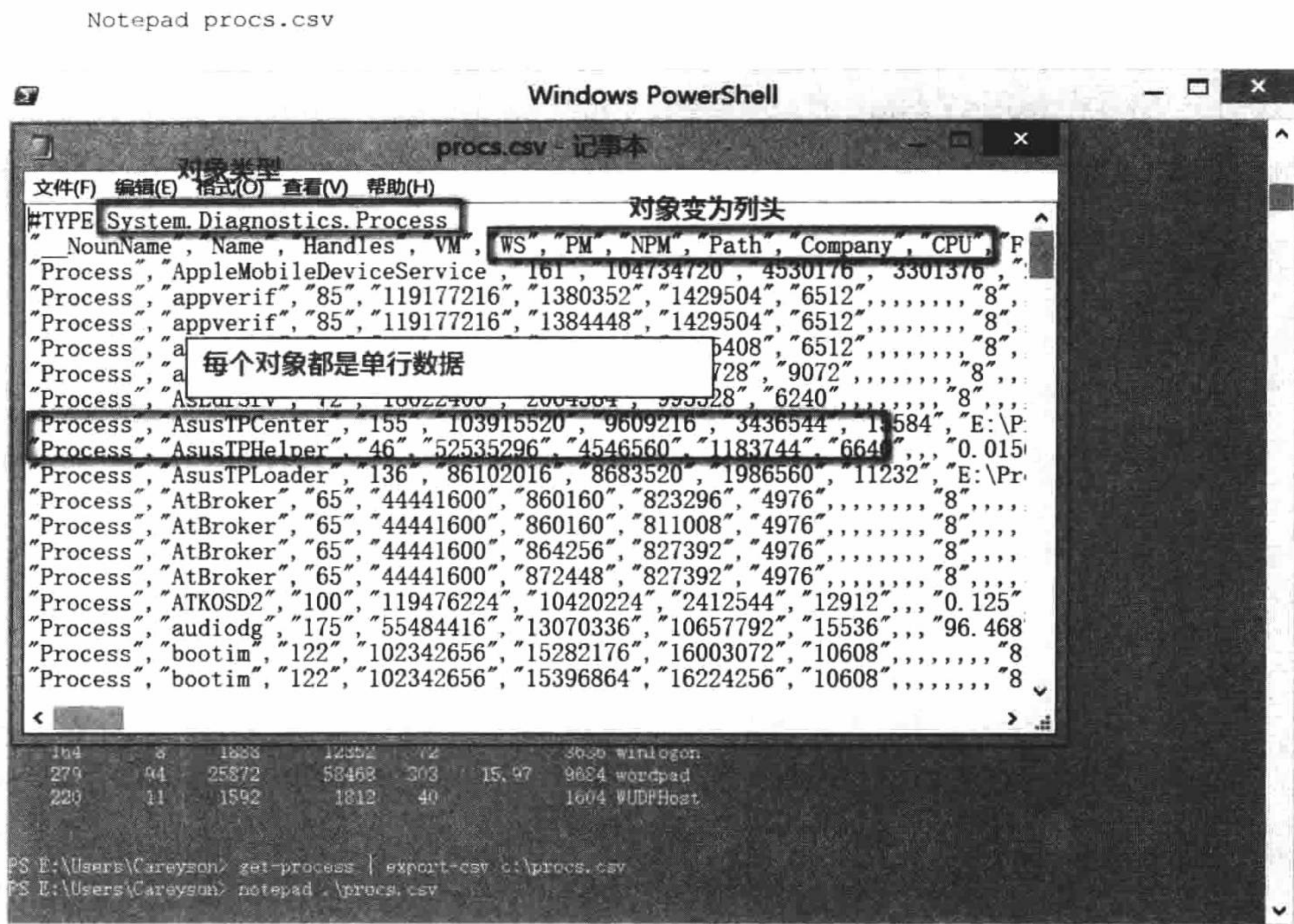


图 6.2 在 Windows 记事本中查看已导出的 CSV 文件

文件的第一行是以“#”开头的内容，代表着文件中包含的信息类型。以图 6.2 为例，“System.Diagnostics.Process”是 Windows 用于标识一个正在运行的进程相关的底层名字。文件的第二行是列名，接下来的每一行代表着每个正在计算机上运行的进程的信息。

你可以把几乎所有的“Get-Cmdlet”用管道传输到“Export-CSV”，然后输出结果。同时，你应该意识到 CSV 文件包含了比显示到屏幕时更多的信息，因为 Shell 知道不可能把所有信息全部显示到屏幕中，所以它使用微软提供的配置文件，把最重要的部分显示到屏幕上。在本章的后面，我们会展示如何覆盖配置从而显示你期望的样子。

一旦信息保存到 CSV 文件，可以轻易地以附件形式发送给同事并让其在 PowerShell 中查看。只需要用下面的命令把文件导入即可：

```
Import-CSV procs.csv
```

Shell 会读取 CSV 文件然后展示，但是展示的结果并不是和原来格式一样，而是 CSV 创建时的快照。

6.2.2 输出结果到 XML

如果 CSV 文件不是你想要的, 怎么办? 没关系, PowerShell 还提供了“Export-CliXML” Cmdlet, 用于创建常规的命令行界面可扩展标记语言文件 (generic command-line interface (CLI) Extensible Markup Language(XML)). CliXML 是 PowerShell 专用的, 但是目前几乎所有程序都能兼容 XML。对应的还有“Import-CliXML” Cmdlet。所有的 import 和 export 的 Cmdlets (比如“Import-CSV”和“Export-CSV”) 都强制需要提供文件名作为参数。

动手实验: 尝试导出一些信息如服务、进程或者事件日志到 CliXML 文件中。确保导出的文件可以用于导入, 并且尝试使用记事本和 IE 浏览器查看这些信息。

除此之外, PowerShell 还提供了其他导入导出命令吗? 有, 可以用“Get-Command” Cmdlet 配合“-verb”参数来找到所有“Import”或“Export”的命令。

动手实验: 尝试找一下 PowerShell 是否自带其他导入导出的 Cmdlets。你可以在加载新命令到 Shell 之后反复尝试, 详情请见下一章。

6.2.3 对比文件

在展示、共享信息给别人及后续重新查看过程中, CSV 和 CliXML 文件都很有用。实际上, “Compare-Object”可以在此过程中发挥重要作用。我们会用到它的别名: Diff。

首先, 运行“help diff”并阅读相关帮助信息。注意三个参数: -ReferenceObject, -DifferenceObject 和 -Property。

Diff 用于把两个结果集组合一起并进行对比。比如, 你在两台不同的机器上运行“Get-Process”。可以把期望用于做匹配的计算机的配置信息放到左边 (称为参照计算机)。右边的计算机信息应该尽可能相似 (称为差异计算机)。在两边运行命令之后, 就可以开始对比两者的信息了, 你只需要从中找出它们的差异。

因为这些进程都是类似的, 比如你只需要检查类似 CPU、内存使用率的值的差异, 因此可以忽略一些列。如把注意力放到“Name”列, 用于查看是否包含了多于或少于参照计算机的处理器。如果使用 Diff, 可以减少你的人工匹配花销。

下面在参照计算机上运行:

```
Get-Process | Export-CliXML reference.xml
```

在这里, 我们选择 CliXML 而不用 CSV, 是因为 CliXML 包含了比 CSV 更多的信息。然后把 XML 文件传输到差异计算机, 运行:

```
Diff -reference (Import-CliXML reference.xml)
➡ -difference (Get-Process) -property Name
```


下面解释一下前面这个比较棘手的步骤：

- 在数学层面上，括号在 PowerShell 中用于控制执行的顺序。在前面的例子中，强制“Import-CliXML”和“Get-Process”先于“Diff”运行。接着从“Import-CLI”得到的结果被送到“-reference”参数中，而“Get-Process”的结果被送到“-difference”参数中。参数名实际上是“-referenceObject”和“-differenceObject”，在这里你可以提供足够 Shell 用于识别参数的缩写名即可。也就是本例中的“-reference”和“-difference”已经足够唯一标识这两个参数了。即使我们把这两个参数缩短到“-ref”和“-diff”，命令依旧能运行。
- 相对于匹配两个完整的表格，Diff 更加关注“Name”列，所以例子中使用了“-property”这个参数。如果我们不这样定义，结果将全部有差异，因为如“VM”“CPU”和“PM”这些列的值都不一样，结果集是被认为有差异的。
- 匹配结果将以表格形式展示，对于存在于参照结果集但是不存在于差异结果集的数据，会用“<=”标识符表示。对于存在于差异结果集但是不存在于参照结果集的数据，会用“=>”标识符表示。而两者均存在的，则不会出现在“Diff”输出的结果中。

动手实验：请动手尝试一下，如果手上没有两台电脑，可以把当前信息导出到一个 CliXML 文件中。然后开启一个新程序，比如记事本、Windows 游戏等。再导出数据作为差异结果集，就可以看到效果了。

这是本机的测试结果：

```
PS C:\> diff -reference (import-clixml reference.xml) -difference (get-  
-process) -property name
```

name	SideIndicator
----	-----
calc	=>
mspaint	=>
notepad	=>
conhost	<=
powerShell_ise	<=

这是一个不错的运维方法，特别是已经建立配置基线，可以对比现有计算机然后找出它们的差异。通过学习这本书，你可以发现很多 Cmdlets 都能用于运维方面，并且都可以通过管道导出到 CliXML 文件中以便建立基线。这些基线一般包括服务、进程、操作系统配置、用户及群组等，并且可用于任何时候对比现有系统的差异。

动手实验：作为尝试，再次执行“Diff”命令，但是不要使用“-property”参数。然后看结果，你会看到每个单独的进程都被列出来，因为如诸如 PM/VM 等的值都被更改，即使它们是相同的进程。这些输出看上去用处不大，因为它们只显示进程类型名和进程名而已。

顺便一提,“Diff”命令在对比文本文件时并不表现得很好。虽然有些操作系统或者 Shell 有专门用于匹配文本文件的“Diff”命令,但是 PowerShell 的“Diff”命令却不一样。你可以在本章的总结实验中体会得到。

注意: 我们希望你多用“Get-Process”“Get-Service”和“Get-EventLog”。这些命令是 PowerShell 内置的,并且不像 Exchange 或者 SharePoint 需要额外的插件才能使用。也就是说,这些技能可以用于以前你学过的所有 Cmdlet 中,包括 Exchange、SharePoint、SQL Server 和其他服务器产品。第 26 章将详细介绍它们。但是目前,请把注意力集中在“如何”使用这些 Cmdlets 上,而不要过多关注它们的工作原理。我们会在适当的时候加以解释。

6.3 管道传输到文件或打印机

每当你通过“Get-Service”或者“Get-Process”创建一些美观的输出时,你可能想把它们保存到一个文件中甚至纸上。通常来说,Cmdlet 是直接输出到 PowerShell 所在的本地机器的屏幕上,但是你可以修改输出位置。实际上,我们前面已经演示了其中一种方式:

```
Dir > DirectoryList.txt
```

其中“>”符是 PowerShell 向后兼容旧版本 cmd.exe 命令的一个快捷方式。而实际上,当你运行这个命令时,PowerShell 底层会以下面的方式实现:

```
Dir | Out-File DirectoryList.txt
```

可以自己尝试运行类似的命令,用这种方式替代“>”符号。“Out-File”提供了一些参数让你定制替代的字符编码(如 UTF8 或 Unicode)、追加内容到现有文件等功能。默认情况下,用“Out-File”创建的文件有 80 列,意味着有时候使用 PowerShell 需要修改命令的输出,以便适应这 80 列的限制。这种修改可能导致存到文件的内容格式与使用同样命令显示到屏幕上的不一致。仔细阅读“Out-File”的帮助文档,看看你是否能找到把默认值修改成大于 80 列的参数。

动手实验: 先别看下面的内容,请打开帮助文档看看能否找到答案。我保证你能很快找到。

PowerShell 有很多“Out-Cmdlets”,其中一个叫“Out-Default”。它是其中一个不需要额外指定的“Out-Cmdlets”,为什么?请看下面:

当你运行“Dir”时,实际上是在运行“Dir | Out-Default”。“Out-Default”只是把内容指向“Out-Host”,意味着你在无意中运行了:

```
Dir | Out-Default | Out-Host
```

而“Out-Host”是显示结果到显示器中。除此之外,你还找到其他什么“Out-Cmdlets”了吗?

动手实验: 是时候研究其他“Out-Cmdlets”了。我们从使用“Help”命令开始,使用如“Help Out*”这样的通配符来获取帮助。这种方式也可以用于“Get-Command”命令,如“Get-Command Out*”,或者指定“-verb”参数:“Get-Command-verb Out”。

“Out-Printer”可能是现有“Out-Cmdlets”中最有用的命令了。虽然“Out-GridView”也有类似功能，但是需要安装.NET Framework v3.5 和 Windows PowerShell ISE 之后才能使用，而这些配置在服务器操作系统中是非自带的。如果你安装了这些，可以尝试运行“Get-Service | Out-GridView”看看结果。“Out-Null”和“Out-String”也非常有用，但是暂时我们不深入探讨。如果你愿意，可以先看看它们的帮助文档。

6.4 转换成 HTML

用 PowerShell 生成 HTML 报告可行吗？可行。只需要通过管道将结果传递给“ConvertTo-HTML”命令即可。这个命令可以生成结构良好的、通用的 HTML 数据，并可以在任何 Web 浏览器中打开。但是这只是原始数据，如果需要美观，需要引用 CSS (Cascading Style Sheet) 定制样式。注意，这个命令不需要文件名：

```
Get-Service | ConvertTo-HTML
```

动手实验：确保在阅读本书的时候自己亲手运行命令，我们希望你理解它们之前先知道它们的功能。

在 PowerShell 世界里面，动词“Export”意味着你把数据提取，然后转换成其他格式，最后把转换后的格式存到某些存储介质中，如文件。而动词“ConvertTo”仅仅是处理过程的一部分，它仅转换不保存。当你执行前面的命令时，可以看到全屏的 HTML 数据，明显不是你想要的。那么请思考一下：你应该怎么把 HTML 存入磁盘的文本文件上？

动手实验：如果你想到其他方式，尽管尝试。下面的命令就是其中一种：

```
Get-Service | ConvertTo-HTML | Out-File services.html
```

你现在是否看到越来越多强大的命令了？每个命令单独执行一个处理操作，而整个命令行可以被视为一个整体完成一个任务。

PowerShell 附带其他“ConvertTo-”Cmdlets，包括“ConvertTo-CSV”和“ConvertTo-XML”等。正如“ConvertTo-HTML”一样，这些命令都不在磁盘上创建文件，只是把命令的输出分别转换成 CSV 或 XML。你需要用管道把它们和“Out-File”连接起来以便存储到磁盘上，但是它们比使用“Export-CSV”或“Export-CliXML”更简短。另外，它们能既转换又存储。

补充说明

现在闲聊一些背景知识。在本例中，经常有学生问：为什么微软提供了“Export-CSV”和“ConvertTo-CSV”这两个对于 XML 数据来说看上去几乎一样的功能？

在某些高级场景中，你可能不想把结果存到磁盘文件上。比如你想把数据转换成 XML 然后传输到 Web 服务，或者其他地方。通过使用不需要存储文件的“ConvertTo-”Cmdlets，你可以灵活地实现你的需求。

6.5 使用 Cmdlets 修改系统：终止进程和停止服务

导出和转换不是你希望连接两个命令的唯一目的。比如下面的例子，记住不要运行：

```
Get-Process | Stop-Process
```

你能想象一下使用这个命令会怎样吗？会宕机！它会检索每一个进程，然后尝试逐个终止。这是一个很危险的进程，类似本地安全权限（Local Security Authority），你的电脑很可能进入蓝屏死机状态。如果你在虚拟机中运行 PowerShell 倒是可以尝试一下。

这个例子想说明的是带有相同名词（本例中的进程）的 Cmdlets 可以在彼此之间互传信息。通常情况下，你最好带上特定进程名而不是终止全部：

```
Get-Process -name Notepad | Stop-Process
```

服务也是类似的，“Get-Service”命令的输出结果能和其他 Cmdlets（如 Stop-Service、Start-Service、Set-Service 等）一起被管道传输。

你可能想象得到，命令之间能互相连接是需要符合某些特定规则的。比如，当你看到这样：Get-ADUser | New-SQLDatabase 的指令序列，你会知道它不会实现什么有意义的功能（虽然它的确做了一些无用功）。在第 7 章中，我们会深入解释这些管理命令间互相连接的规则。

下面我们希望你对类似“Stop-Service”和“Stop-Process”这些 Cmdlets 有更深入的了解。这些 Cmdlets 以某些方式修改系统，并且有一个内部定义的影响级别（impact level）。Cmdlet 的创建者已经设定了这些影响级别，并且不允许修改。而 Shell 有一个相应的“\$ConfirmPreference”设置，默认为“High”。可以通过下面的命令查看你的 Shell 的设置：

```
PS C:\> $ConfirmPreference
High
```

工作原理：当 Cmdlet 的内部影响级别大于等于 Shell 的“\$ConfirmPreference”设置时，不管 Cmdlet 正准备做什么，Shell 都会自动询问“你确定要这样做吗？（Are you sure？）”。实际上，如果你使用虚拟机尝试前面提到的那个“宕机”命令，你会发现对于每个进程，都会问一次“Are you sure？”。当 Cmdlet 的内部影响级别小于 Shell 的“\$ConfirmPreference”设置时，不会自动弹出这个提示。

但是如果你“喜欢”它总是弹出，可以使用下面的命令：

```
Get-Service | Stop-Service -confirm
```

我们在这里加了“-confirm”参数，对于某些被支持的用于修改系统的 Cmdlet，会弹出提示，并对这些被支持的 Cmdlet 显示对应的帮助文档。

另外一个类似的参数是“-whatif”，可用于支持“-confirm”的 Cmdlet。但是它并不默认触发，可以在你想用的时候使用：

```
PS C:\> get-process | stop-process -whatif
What if: Performing operation "Stop-Process" on Target "conhost (1920)".
What if: Performing operation "Stop-Process" on Target "conhost (1960)".
What if: Performing operation "Stop-Process" on Target "conhost (2460)".
What if: Performing operation "Stop-Process" on Target "csrss (316)".
```

它会告诉你哪些 Cmdlet 会被执行，但是并不真正运行。这个功能为那些可能有潜在风险的 Cmdlet 的预览提供了很好的帮助，并且可以检查是否是你想要的结果。

6.6 常见误区

在 PowerShell 中，其中一个常见的困惑是“Export-CSV”和“Export-CliXML”的异同。这两个命令从技术上都是用于创建文本文件。也就是说，两者的输出结果都能在记事本中查看，如图 6.2 所示。但是你必须承认两个结果有明显的差异——一个是逗号分隔值，而另外一个则是 XML。

这个问题主要关心的是用户如何把文件重复读入 Shell 中。为此你是否使用“Get-Content”（或者它的别名，Type/Cat）？举个例子，假设你这样使用：

```
PS C:\> get-eventlog -LogName security -newest 5 | export-csv events.csv
```

现在你需要使用“Get-Content”命令从 Shell 中读出来：

```
PS C:\> Get-Content .\events.csv
#TYPE System.Diagnostics.EventLogEntry#security/Microsoft-Windows-Security-Auditing/4797
"EventID", "MachineName", "Data", "Index", "Category", "CategoryNumber", "EntryType", "Message", "Source", "ReplacementStrings", "InstanceId", "TimeGenerated", "TimeWritten", "UserName", "Site", "Container"
"4797", "DONJONES1D96", "System.Byte[]", "263", "(13824)", "13824", "SuccessAudit", "An attempt was made to query the existence of a blank password for an account.
```

Subject:

Security ID: S-1-5-21-87969579-3210054174-450162487-100

Account Name: donjones

Account Domain: DONJONES1D96


```

Logon ID:                                0x10526

Additional Information:
    Caller Workstation:      DONJONES1D96
    Target Account Name:     Guest
    Target Account Domain:   DONJONES1D96", "Microsoft-Windows-Security-
auditing
", "System.String[]", "4797", "3/29/2012 9:43:36 AM", "3/29/2012 9:43:36 AM", ,
,
"4616", "DONJONES1D96", "System.Byte[]", "262", "(12288)", "12288", "SuccessAudit",
"The system time was changed.

```

我们截断了前面的输出，但是还是可以看到有很多相同的部分。回顾原始的 CSV 数据，你是否觉得有很多垃圾信息？该命令没有尝试解析、编译这些数据。现在对比一下 “Import-CSV” 的结果：

```
PS C:\> import-csv .\events.csv
```

```

EventID           : 4797
MachineName       : DONJONES1D96
Data              : System.Byte[]
Index             : 263
Category          : (13824)
CategoryNumber    : 13824
EntryType         : SuccessAudit
Message           : An attempt was made to query the existence of a
                   blank password for an account.

                   Subject:
                       Security ID:
                           S-1-5-21-87969579-3210054174-450162487-1001
                           Account Name:      donjones
                           Account Domain:    DONJONES1D96
                           Logon ID:         0x10526
                   Additional Information:
                           Caller Workstation:  DONJONES1D96
                           Target Account Name:  Guest
                           Target Account Domain:  DONJONES1D96
Source            : Microsoft-Windows-Security-Auditing
ReplacementStrings : System.String[]
InstanceId        : 4797
TimeGenerated     : 3/29/2012 9:43:36 AM
TimeWritten       : 3/29/2012 9:43:36 AM
Username          :

```


是不是好很多？“Import-Cmdlets”会关注文件中的内容，尝试解析它们，然后创建一个比原始命令（本例中的“Get-EventLog”）看上去更加顺眼的输出结果。如果你使用“Export-CSV”创建文件，可以使用“Import-CSV”命令来读取它们。如果使用“Export-CliXML”命令创建文件，通常建议使用“Import-CliXML”命令读取。使用这些配套命令可以得到更好的结果。仅在从一个文本文件中读取内容并且不需要PowerShell解析数据时，才使用“Get-Content”命令，也就是你仅需要原始内容。

6.7 动手实验

注意：本实验需要 PowerShell v3 或以上版本。

由于前面演示的例子稍微花时间，所以我们尽可能保证本章文字的简洁，因为我们希望你能把更多精力花在下面的动手实验中。如果你还没完成本章中所有“动手实验”的任务，我们强烈建议你先去完成它们，然后进行下面的任务：

1. 在控制台运行“Get-Service | Export-CSV services.csv | Out-File”时会发生什么情况？为什么会这样？
2. 除了获取一个或多个服务及以管道方式传输到“Stop-Service”之外，“Stop-Service”服务还提供了其他什么方式让你指定服务或停止服务？有什么方式可以在不使用“Get-Service”的前提下停止一个服务？
3. 如何创建一个竖线分隔符文件替代一个逗号分隔符（CSV）文件？你可以依旧使用“Export-CSV”命令，但是应该使用什么参数？
4. 可以在已导出的 CSV 文件头部忽略#命令行吗？这一行通常包含了类型信息，但是如果你想从一个特定文件中获取并忽略时要怎么做？
5. “Export-CliXML”和“Export-CSV”都可以通过创建并覆盖文件来修改系统，你可以用什么参数来阻止它们覆盖现有文件？还有什么参数可以在你输出文件前提醒并请求确认？
6. Windows 维护少数局部配置，包括一个默认分隔符列表。在美国系统中，分隔符是逗号。你如何让“Export-CSV”使用当前系统默认的分隔符而不是逗号？

动手实验：完成上面实验之后，尝试完成本书附录中的实验回顾 1。