第 4 章　运行命令

当开始在互联网上查看 PowerShell 示例时，很容易觉得 PowerShell 是某种基于.NET Framework 的脚本或编程语言。我们的伙伴微软最有价值专家（MVP）奖项获得者，以及大量其他 PowerShell 用户都是一本正经的极客（Geek）。我们乐于深入挖掘 Shell 的潜力并发挥它的最大价值。但几乎我们所有人都是以本章标题那样开始：运行命令。这也是本章我们将要做的：没有脚本、没有编程语言，仅仅是运行命令和命令行工具。

4.1　无需脚本，仅仅是运行命令

PowerShell，如其名称所示，是一个 Shell。它和你之前可能使用过的 Cmd.exe 命令行 Shell 类似，甚至更像是与 20 世纪 80 年代第一台 PC 机一起发布的 MS-DOS。它与 Unix 的 Shell 也十分类似，比如说 20 世纪 80 年代后期的 Bash，甚至是 20 世纪 70 年代面世最原始的 Unix Bourne Shell。虽然 PowerShell 更加现代，但最终，PowerShell 并不是一个类似 VBScript 或 KiXtart 的脚本语言。

这些语言和大多数编程语言一样，你在文本编辑器（即使是 Windows 记事本）中键入大量关键字形成脚本。当脚本完成保存为文件后，可能还需要双击该文件进行测试。PowerShell 能够以这种方式工作，但这并不是 PowerShell 的主要工作模式，尤其是当你开始学习 PowerShell 时。使用 PowerShell，你输入一个命令，然后通过添加一些参数来定制化命令行为，点击返回，立刻就能看到结果。

最终，你会厌倦一遍遍输入同样的命令（和参数），然后你会将其复制粘贴到一个文本文件中，并将文件的扩展名更名为.PS1，然后你瞬间就拥有了一个“PowerShell 脚本”。现在，你不再需要一遍遍输入命令，而是直接执行该文件中的脚本。这也和你在 Cmd.exe Shell 中使用的批处理文件是同一种模式，但相较于脚本或编程而言却要简单许多。

别理解错我们的意思：你可以将 PowerShell 用得极其复杂。实际上，PowerShell 支持与 VBScript 和其他脚本或编程语言同一种使用模式。PowerShell 拥有能够访问整个.Net Framework 底层的能力，我们也看到 PowerShell“脚本”实际上与通过 Visual Studio 编写的 C#语言使用模式也十分类似。PowerShell 支持这两种不同的使用模式，是因为其设计目标是为了拥有更广阔的使用场景。关键是，不能仅仅是 PowerShell 可以实现得非常复杂，就意味着你也必须将 PowerShell 使用到这种程度，也并不意味着你不能以更简单的方式实现非常高效的结果。

来看这样一个类比：你或许有一辆车，如果你和我们一样，或许换机油是你对车做过最复杂的机械性工作。我们并不是汽车专家，也不能重建一个引擎。我们也不能完成如你在电影中所见非常酷的漂移。你从未见过我们在汽车广告中的封闭赛道开车，虽然 Jeff 的梦想就是这么做（他看了太多的 Top Gear（译者注：一档由英国 BBC 电台出品的汽车节目））。虽然我们并不是专业的赛车手，但是并不会阻挡我们在日常中以更低的复杂度高效驾驶。如果某天我们决定来一场特技驾驶（我们的保险公司会被吓死的），这时或许多学一点汽车工作的原理并掌握一些新的技巧才更有帮助。留给我们进阶的选择一直就在那儿。但目前为止，我们对完成普通驾驶就非常满意。

目前为止，我们依然是一位普通的“PowerShell 驾驶员”，以比较简单的方式操纵 Shell。无论你是否相信，在该阶段的用户才是 PowerShell 的主要目标用户。你会发现，在这个阶段，你就能够完成很多难以置信的工作。你仅需要掌握如何在 Shell 中运行命令即可。

4.2 剖析一个命令

图 4.1 展示了对复杂 PowerShell 命令的一个基本剖析。我们称之为一个命令的完整语法形式。我们尝试使用一个有点复杂的命令，这样你就能看到可能出现的所有部分。

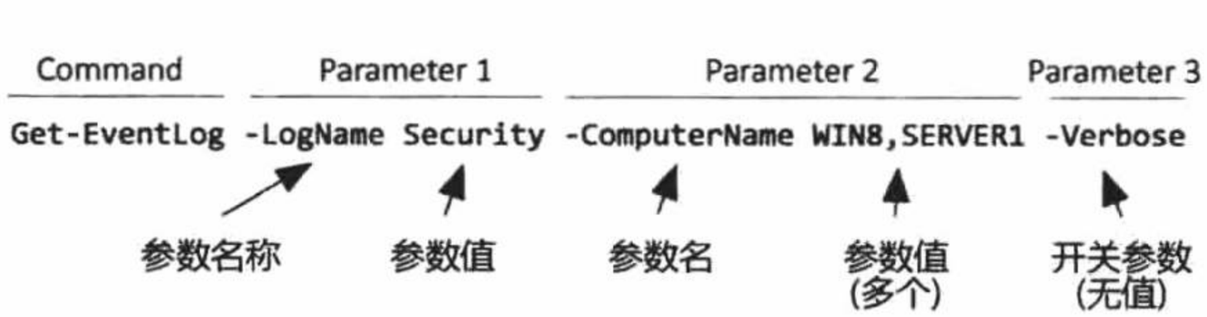

图 4.1 剖析一个 PowerShell 命令

为了确保你能够完全熟悉 PowerShell 的规则，下面更详细地阐述上一张图中每一部分。

- 名称为 Get-EventLog 的 Cmdlet。PowerShell Cmdlet 总是以这种动词-名词形式命名。我们将在下一章关于 Cmdlet 的章节进一步解释。
- 第一个参数名称为-LogName，并赋值为 Security。由于参数值中并不包含任何空格或标点符号，因此并不需要用引号括起来。

- 第二个参数名称为-ComputerName，以逗号分隔列表的形式赋了两个值：Win8和 Server1。由于这两个参数中都不包含空格或标点符号，因此这两个参数都不需要用引号括起来。
- 最后一个参数是-Verbose，是一个开关参数。这意味着该参数无须赋值，仅仅指定参数即可。
- 注意：在命令名称和第一个参数之间必须有空格。
- 参数名总是以英文短横线（-）开头。
- 参数名之间必须有空格，多个参数值之间也必须有空格。
- 无论参数名之前的破折号，还是参数值本身包含的破折号，都不需要加空格。
- PowerShell 不区分大小写。

请逐渐习惯这些规则，并开始对这种精确优雅的输入方式敏感。多注意空格、破折号和其他部分可以最大程度减少 PowerShell 报低级错误的机会。

4.3 Cmdlet 命名惯例

首先，让我们以讨论一些术语开始。据目前我们所知，下面的术语仅仅用于 PowerShell。但为了确保对属于理解的统一性，我们还需要详细解释一下：

- Cmdlet 是一个原生的 PowerShell 命令行工具。该术语仅仅存在于 PowerShell 和类似 C#的.Net Framework 语言中。Cmdlet 仅仅出现在 PowerShell 中，所以当你在 Google 或 Bing 搜索该关键字时，返回结果主要是关于 PowerShell 的。该术语读音为“command-let”。
- 函数和 Cmdlet 类似，但不是以.Net 语言编写，而是以 PowerShell 自己的脚本语言编写。
- 工作流是嵌入 PowerShell 的工作流执行系统的一类特殊函数。
- 应用程序是任意类型的外部可执行程序，包括类似 PING、Ipconfig 等命令行工具。
- 命令是一个通用的术语，用于代表任何或所有上面提到的术语。

微软已经为 Cmdlet 建了一个命名惯例。因此同样的命名管理也应该被用于函数和工作流。虽然微软并没有强制要求，但开发人员应该遵循该惯例。

规则应该以标准的动词开始，比如 Get、Set、New 或 Pause。你可以运行 Get-Verb 查看允许使用的动词列表（虽然只有少部分是常用的，但你大概会看到 100 个左右）。在动词之后紧接着一个破折号，然后是一个单数形式的名词，比如说 Service 或 Process 或 EventLog。由于 PowerShell 允许开发人员自己命名名词，因此并没有一个“Get-Noun”的 Cmdlet 来显示所有名词。

这个规则的妙处在哪里？假设我们告诉你如下几个 Cmdlet 名称：New-Service、Get-Service、Get-Process、Set-Service 等。你能够猜出哪一个命令可以创建一个新的

Exchange 邮箱？你是否能够猜出哪一个命令可以修改活动目录用户？如果你猜是“Get-Mailbox”，那么第一个就猜对了。如果你猜“Set-User”，那么第二个就非常接近了。实际上是 Set-ADUser，你可以在活动目录模块的域控制器中找到该用户。重点是，通过一致的命名规则以及有限的动词集合，猜测命令名称变为可能，在此之后才是使用帮助或“Get-Command”加通配符验证猜想。估计你所需要的命令名称会变得更加简单，而无须每次都去搜索 Google 或 Bing。

注意：并不是所有所谓的动词都是动词。虽然微软官方使用术语“动词-名词命名规范”，你仍然能看到类似 New、Where 等“动词”，请逐渐习惯吧。

4.4 别名：命令的昵称

虽然 PowerShell 命令名称足够好，并具有良好的一致性，但仍然可能很长。类似 Set-WinDefaultInputMethodOverride 的命令名称，即使有 Tab 键补全，对于输入来说也是太长。虽然命令名称非常清晰——看到名称就能大概猜到其功能，但对于输入来说还是太长。

这也是为什么需要 PowerShell 别名。别名仅仅是命令的昵称。厌倦了输入 Get-Service？尝试下面的代码：

```
PS C:\> get-alias -Definition "Get-Service"
Capability      Name
----------      ----
Cmdlet          gsv -> Get-Service
```

现在你知道 Gsv 是 Get-Service 的别名了。

无论是否使用别名，命令的工作方式不会变。参数还是原来的参数，其他部分也不会有任何改变——仅仅是命令名称变得更短。

如果你看到一个别名（网上的一些家伙倾向于使用别名，就好像我们都能够记住所有 150 个内置别名）而不知道其含义，请查阅帮助：

```
PS C:\> help gsv
NAME
    Get-Service
SYNOPSIS
    Gets the services on a local or remote computer.

SYNTAX
    Get-Service [[-Name] <String[]>] [-ComputerName <String[]>]
    [-DependentServices [<SwitchParameter>]] [-Exclude <String[]>]
    [-Include <String[]>] [-RequiredServices [<SwitchParameter>]]
    [<CommonParameters>]
```

```
Get-Service [-ComputerName <String[]>] [-DependentServices
[<SwitchParameter>]] [-Exclude <String[]>] [-Include <String[]>]
[-RequiredServices [<SwitchParameter>]] -DisplayName <String[]>
[<CommonParameters>]

Get-Service [-ComputerName <String[]>] [-DependentServices
[<SwitchParameter>]] [-Exclude <String[]>] [-Include <String[]>]
[-InputObject <ServiceController[]>] [-RequiredServices
[<SwitchParameter>]] [<CommonParameters>]
```

在根据别名查阅帮助时，帮助系统将会显示完整命令的帮助，其中也会包含命令的完整名称。

补充说明

你可以使用 New-Alias 创建自定义别名，使用 Export-Alias 导出别名列表。当创建一个别名时，其生命周期只能持续到当前的 Shell 会话结束。一旦关闭窗口，别名就会不复存在。这也是你需要导出别名的原因，以便后续重新导入。

我们通常会避免创建和使用自定义别名，因为这些别名除我们之外的别人无法使用。如果某个用户无法查到 xtd 的含义，这会导致混淆。

xtd 仅仅是我们编造的一个假的别名，不会做任何工作。

4.5 使用快捷方式

这也是 PowerShell 奇妙的地方。之前提到过 PowerShell 唯一的方式就是我们之前给你展示的那样，但实际上我们撒谎了。如果你希望在网络上偷取（或者再利用）其他人的示例代码，那首先需要懂得如何看懂它。

除了作为快捷方式的命令的别名之外，参数也同样可以使用别名。总共有三种方式可以实现这一点，每一种都可能造成混淆。

4.5.1 简化参数名称

PowerShell 并不强制要求输入完整的参数名称。例如，你可以通过输入-comp 代替-ComputerName，简化的规则是必须输入足够的字母让 PowerShell 可以识别不同参数。如果既存在-composite 参数，也存在-computerName 以及-common 参数，你至少要输入-compu、-commo 和-compo。这是由于上述值是唯一识别参数所需要输入的最少部分。

如果你很希望使用简便方式，那上面就是一个不错的选择。如果你在输入最少部分的参数之后记得按 Tab 键，PowerShell 会帮你自动完成余下的输入。

4.5.2 参数名称别名

尽管参数的别名不在帮助文件或任何方便查阅的地方而难以识别，但参数也拥有别名。比如说，Get-EventLog 命令有-ComputerName 参数。可以运行下述命令，查阅该参数别名。

```
PS C:\> (get-command get-eventlog | select -ExpandProperty parameters). Comp
utername.aliases
```

上述命令已经用粗体标出命令和参数名称。你可以用任意你希望了解的命令和参数名称进行替换。在本例中，数据结果展示了-Cn 是-computerName 的别名，所以你可以运行下述命令：

```
PS C:\> Get-EventLog -LogName Security -Cn SERVER2 -Newest 10
```

Tab 键补全将会展示出-Cn 这个别名。如果你输入 Get-EventLog –C 并开始按 Tab 键，该别名将会出现。但是命令的帮助并不会显示关于-Cn 的任何信息，且 Tab 键补全并不会显示-Cn 和-ComputerName 实际上 hi 同一个命令。

4.5.3 定位参数

当你在帮助文件中查看命令语法时，你可以很容易认出定位参数：

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

在上述语法中，-Path 和-Filter 参数是定位参数，这是由于参数名称被中括号给括起来。在完整的帮助文档（本例是 help Get-ChildItem-Full）中会有更清晰的解释，如下：

```
-Path <String[]>
   Specifies a path to one or more locations. Wildcards are
   permitted. The default location is the current directory (.).

   Required?                    false
   Position?                    1
   Default value                Current directory
   Accept pipeline input?       true (ByValue, ByPropertyName)
   Accept wildcard characters?  True
```

上述帮助明显解释了-Path 参数在位置 1。对于定位参数来说，你无须输入参数名称——仅需要在正确的位置提供参数值，例如：

```
PS C:\> Get-ChildItem c:\users
    Directory: C:\users
Mode                 LastWriteTime     Length   Name
------               -------------     -------  -------
d----          3/27/2012  11:20 AM              donjones
d-r--          2/18/2012   2:06 AM              Public
```

和下述命令完全相同：

```
PS C:\> Get-ChildItem -path c:\users
    Directory: C:\users
Mode                 LastWriteTime        Length   Name
------               -------------        -------  -------
d----          3/27/2012   11:20 AM                donjones
d-r--          2/18/2012    2:06 AM                Public
```

定位参数的一个弊端是你必须记住每一个位置所代表的参数。你还必须首先按照正确的顺序输入定位参数，然后才能输入命名（非定位）参数。如果你将定位参数的顺序搞混，命令则会失败。对于你可能已经使用多年的简单 DIR 命令来说，如果提供-Path 参数将会变得很怪异，没有人会这么做。但对于更复杂的命令来说，比如一行包含 3 至 4 个定位参数的命令，将难以记住每一个位置所代表的参数。

比如说，下面命令将会难以阅读和理解：

```
PS C:\> move file.txt users\donjones\
```

下面的版本显式指定了参数名称，将会更容易理解：

```
PS C:\> move -Path c:\file.txt -Destination \users\donjones\
```

下面的版本将参数调换顺序，只有在指定参数名称时才允许这么做：

```
PS C:\> move -Destination \users\donjones\ -Path c:\file.txt
```

我们倾向于不推荐使用定位（也就是不指定参数名）参数，除非你仅仅是即时输入一个命令并不会带来任何后续影响。任何将命令长期保存的方式，包括在批处理文件中或是写入博客中，都要把所有的参数名称带上。我们在本书中尽量不使用不指定参数名称的方式，只有在少数示例中由于命令过长影响到排版时，我们才会使用。

4.6 小小作弊一下：Show-Command

尽管我们拥有多年使用 PowerShell 的经验，但命令语法的复杂度有时依然会让我们抓狂。PowerShell v3 提供的一个非常棒的特性是 Show-Command commlet。如果你在命令语法方面遇到困难，包括空格、破折号、逗号、引号或是其他方面，Show-Command 将成为你的助手。该命令允许你指定你无法用对的命令名称，并以图形化的方式将命令

的参数名称展示出来。如图 4.2 所示，Tab 键补全不会跨越参数集（在前一章学到的），因此不同参数集之间的参数不会搞混——使用 Tab 键补全并坚持使用。

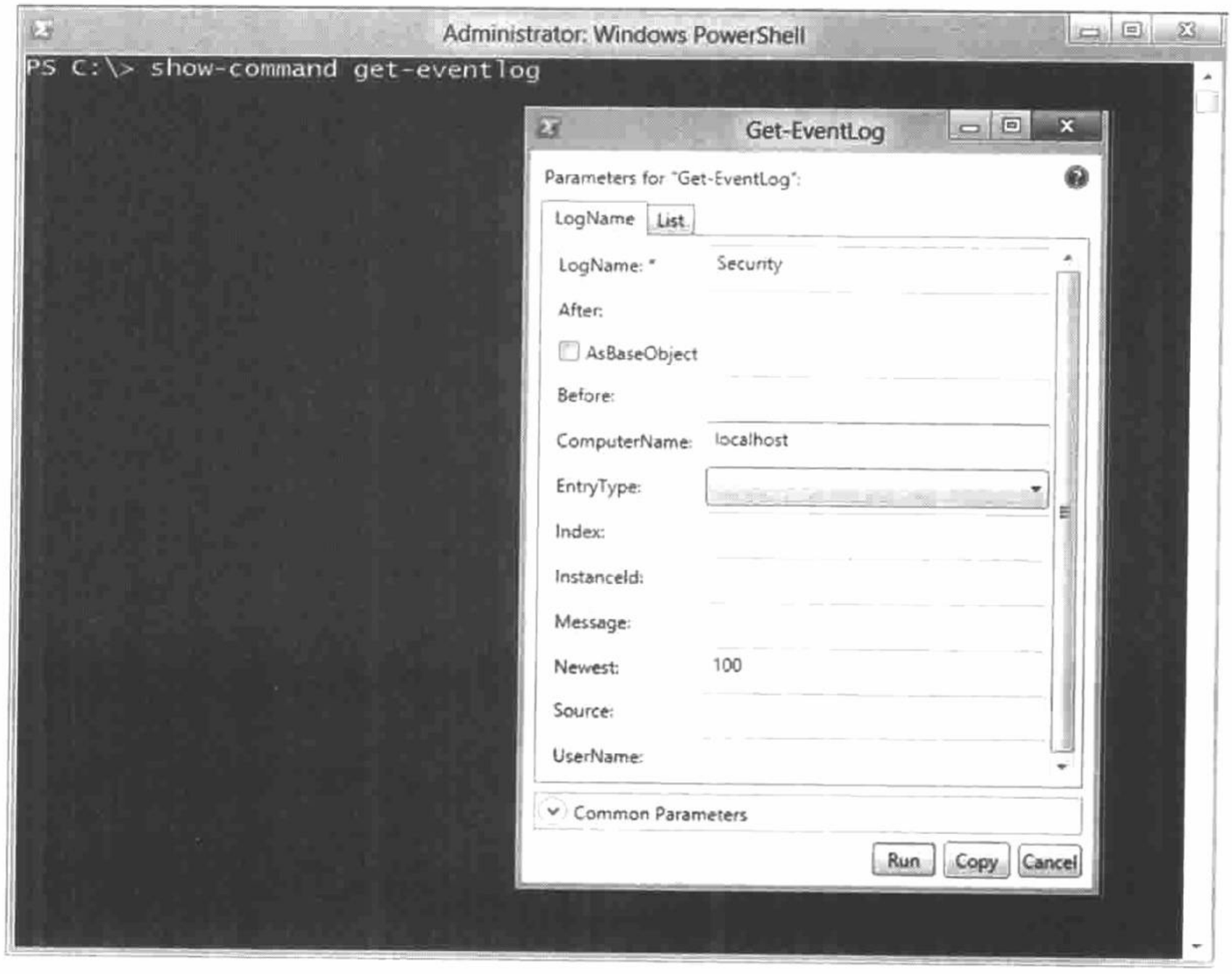

图 4.2 Show-Command 命令使用图形提示帮助填写命令参数

当完成后，你可以单击运行来执行命令，或使用我更喜欢的选项——单击复制将完成后的命令复制到剪贴板，返回 Shell，将命令剪贴（右击控制台，或是在 ISE 中使用 Ctrl+V 组合键）到 Shell 中进行查看。这也是自学 PowerShell 语法最好的方式，如图 4.3 所示。你每次都能获得正确的语法。

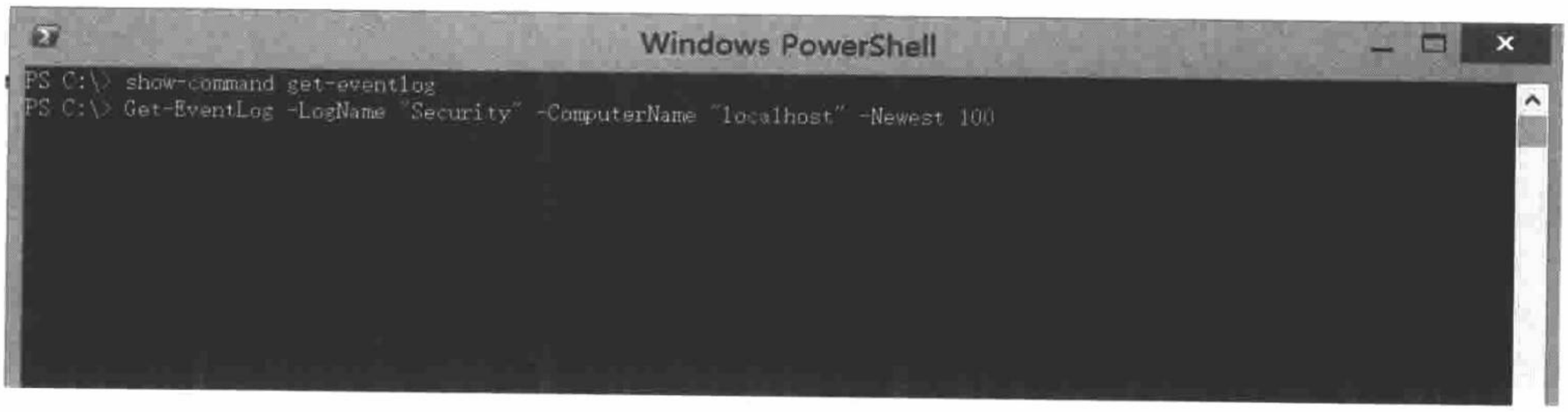

图 4.3 基于初始条目对话框，Show-Command 生成合适的命令行语法

以这种方式产生的命令，总会是命令的完整形式。完整的命令名称，完整的参数名称，所有的参数名称都显式输入（即，不会出现定位参数）。因此，这种方式可以看到使用 PowerShell 最完美、推荐并符合最佳实践的方式。

不幸的是，Show-Command 一次只能展示一个命令。因此，当希望了解多个命令时，只能逐个使用该命令。

4.7 对扩展命令的支持

目前为止，你所有在 Shell 中运行的命令（至少是我们建议你运行的命令）都是内置 Cmdlet。大约 400 个 Cmdlet 都被集成到最新版本的 Windows 客户端操作系统中，上千个被集成到 Windows 服务器版本的操作系统中，并且你还能添加更多——类似 Exchange Server、Sharepoint Server 和 SQL Server 都包含数以百计的额外 Cmdlet。

但是你并不会被局限在仅仅使用随 PowerShell 一同发行的 Cmdlet——你还可以使用一些或许你已经使用多年的外置命令行工具，包括 Ping、Nslookup、Ipconfig、Net 等。由于这些都不是原生 PowerShell Cmdlet，因此你可以按照原来使用这些命令的方法继续使用这些命令。PowerShell 将会在后台启动 Cmd.exe。由于 PowerShell 知道如何运行扩展命令，因此返回的结果都会被显示在 PowerShell 窗口。请尝试运行一些你已经熟悉的 CMD 命令。我们经常会被问到如何使用 PowerShell 关联一个普通的网络驱动器——你可以在对象资源管理器中看到那个。我们经常使用的 Net Use 命令在 PowerShell 中也能正常工作。

动手实验： 在 PowerShell 中运行一些之前你熟知的外部命令行工具。是否能够正常工作？哪些命令执行失败？

Net Use 示例传递出一个重要信息：使用 PowerShell，微软并不是说“你必须重新来过，重新学习所有的一切”，而是说“如果你已经知道了如何完成工作，请继续保持。我们会提供更好、更完整的工具帮助你，但你之前所学依然可用”。PowerShell 中不存在“Map-Drive”命令的一个原因是 Net Use 已经可以很好地完成工作，那为什么不继续使用该命令呢？

注意： 我们已经使用 Net Use 多年，甚至是 PowerShell v1 发行之前，该命令依然是非常好的命令。但 PowerShell v3 证实微软开始寻找合适的时机推出 PowerShell 风格的方式来完成这些传统任务。现在你可以发现 New-PSDrive 命令多了一个-Persisit 参数，该参数决定是否启用文件系统提供程序。这样一来，新的磁盘将会在文件资源管理器中被查看到。

有一些确定的例子说明微软已经为一些已经存在的老的命令提供了一些更好的替代工具。比如说，原生的 Test-Connection Cmdlet 相比之前的提供了更多选项和更灵活的输出方式。还有外部的 Ping 命令，如果你知道如何使用 Ping 命令，它可以解决你的所有需求，请立刻使用它。Ping 命令在 PowerShell 中可以正常工作。

综合上面，我们必须透漏出一个严酷的事实：并不是所有的外部命令都可以流畅地运行在 PowerShell 中，至少如果你不做一些调整是不行的。这是由于 PowerShell 解析器——Shell 的该部分读取你输入的内容并尝试解析出你希望 Shell 执行什么——并不是每次都能猜对。有时你输入一个外部命令，就会导致 PowerShell 产生混乱，输出错误信息，因此命令不会生效。

比如说，当一个外部命令拥有很多参数时，事情就变得很难办。这也是 PowerShell 在大多数场景下无法工作的情形。我们深入其不能正常工作的细节，但可以提供下面的命令，从而确保运行一个命令且其参数可以准确无误：

```
$exe = "C:\Vmware\vcbMounter.exe"
$host = "server"
$user = "joe"
$password = "password"
$machine = "somepc"
$location = "somelocation"
$backupType = "incremental"

& $exe -h $host -u $user -p $password -s "name:$machine" -r $location -t
$backupType
```

假设你有一个名为 vcbMounter.exe 的外部命令（这是一个由某些 VMWare 虚拟化产品所提供的真实命令；如果你从未使用或安装过该命令，没有关系——大多数传统的命令行工具都以同样的方式工作，所以这依然是一个很好的教学案例），该命令接受 6 个参数：

- -h for the host name
- -u for the user name
- -p for the password
- -s for the server name
- -r for a location
- -t for a backup type

我们所做的是将不同的元素——可执行路径和名称，以及所有的参数值放入容器。这部分操作以$开始。这使得 PowerShell 将这些值当作一个单元，而不是尝试对其进行解析来发现是否包含命令或特殊字符等。然后我们使用调用操作符，将该值传递给可执行名称，还有所有的参数值。这种方式可以在 PowerShell 中执行的几乎所有的命令行工具中生效。

听上去很复杂？好吧，我们有一些好消息：在 PowerShell 第三版中，你不必再如此纠结，仅需要在外部命令名称之后加两个破折号。如果你这么做，PowerShell 甚至不会解析该命令，仅仅是将该命令传递到 Cmd.exe 中。这意味着你基本上可以使用 Cmd.exe 的语法在 PowerShell 中运行任何命令，而不用担心该命令是如何被 PowerShell 解析的。

4.8 处理错误

在刚开始使用 PowerShell 时无可避免地会遇见丑陋的红色文本提示，在不同水平阶段依然可以遇到，甚至当你成为专家级的 Shell 用户时也避免不了。我们都能遇到，但不要让红字把你逼疯。

先不管用于警告目的的红字，PowerShell 的错误信息的目的是用于帮助。例如，如图 4.4 所示，红字尝试展示给你 PowerShell 错误的地方。

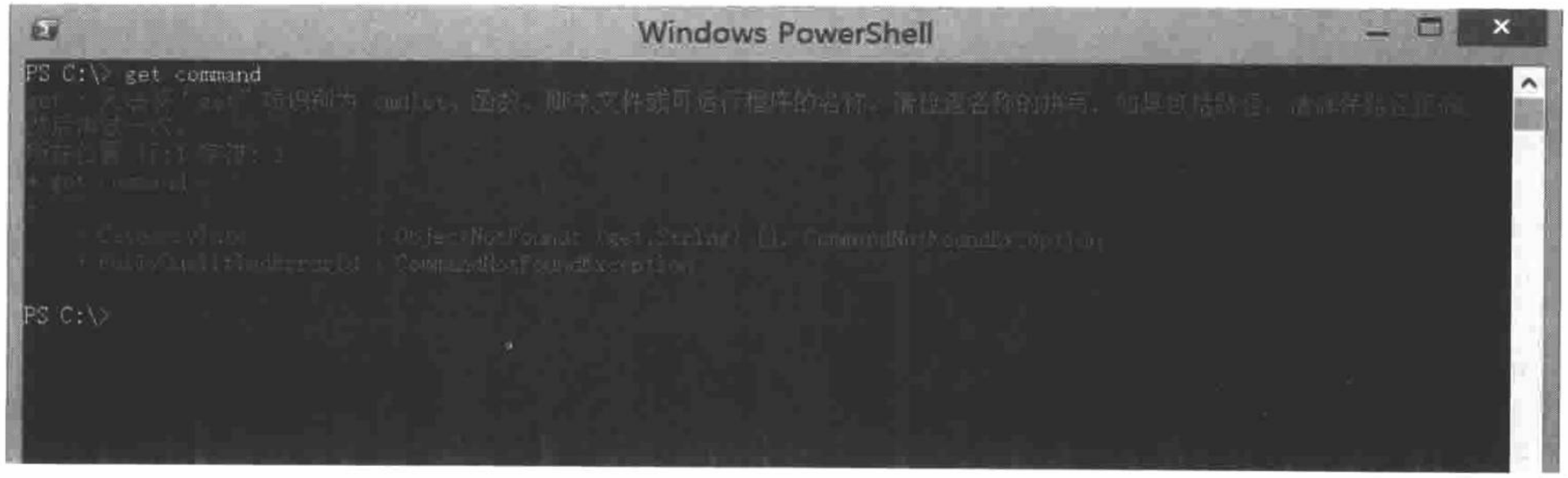

图 4.4 解释 PowerShell 的错误信息

错误信息几乎总是会包括 PowerShell 认为有歧义地方的行数和字符数。在图 4.4 中，是第一行，字符 1——就是命令开始部分。其表达的意思为“你输入了‘get’，我不知道该词的意思”。这是由于我们输错了命令，正确应该是 Get-Command，而不是 Get Command。那么图 4.5 是什么情况?

图 4.5 “第二路径片段”是什么?

图 4.5 中所示的错误信息“第二路径不得为驱动器或 UNC 名称”让人感到困惑，什么第二路径？我们并没有输入第二路径。我们输入了一个路径 c:\windows 和一个命令行参数/s，不是吗？

当然不是。解决该类问题最简便的方式就是阅读帮助，并完整输入命令。如果你输入 Get-ChildItem-path C:\Windows，就会发现/s 并不是正确的语法。我们希望该值对应的参数是-recurse。有时，错误信息并不一定很有帮助，就好像你和 PowerShell 说的不是同一种语言。当然，PowerShell 不可能改变其语言，那么只能是你错了，所以你得去改变。通过咨询帮助并拼写出完整的命令和参数，通常都是解决问题最快的方式。还有不要忘了使用 Show-Command 来找出正确的语法。

4.9 常见误区

当合适时，我们会在一章中安排一个简短的小节，包含我们在教学过程中存在的一些常见误区。这样做的目的是帮助其他像你一样的管理员，从而避免该类问题——或是至少在你开始使用 Shell 时对这些问题找到解决办法。

4.9.1 输入 Cmdlet 名称

首先是输入 Cmdlet 名称。该名称永远是动词-名词形式，比如说 Get-Content。下面是我看到的一些新手尝试输入的命令，但显然难以奏效：

- `Get Content`
- `GetContent`
- `Get=Content`
- `Get_Content`

其中一些问题是由于输入错误（比如说“=”，而不是“-”），还有一些是省略破折号。我们都会将命令读成“Get Content”，省略了破折号。但输入时必须输入破折号。

4.9.2 输入参数

参数同样需要正确书写。参数可以不赋值，比如说-recurse，在参数名称之前加上破折号。但必须在 Cmdlet 名称和参数之间加空格，参数之间也需要空格。下述命令都正确：

- `Dir-rec`（可以使用参数名称的简写）
- `New-PSDrive-name DEMO-psprovider FileSystem-root \\Server\Share`

但下述写法不正确：

- `Dir-rec`（在名称和参数之间没有空格）

- New-PSDrive-nameDEMO（参数和值之间没有空格）
- New-PSDrive-name DEMO-psprovider FileSystem（在第一个参数值和第二个参数名之间没有空格）

PowerShell 并不会挑剔大小写问题，也就是说 dir 和 DIR 并无不同。-RECURSE、-recurse 和-Recurse 也是如此。但 PowerShell 会挑剔空格和破折号的写法。

4.10 动手实验

注意：对于本次实验，你需要 Windows 8（或更新版本）或 Windows 2012（或更新版本）的计算器来运行 PowerShell。

请使用在本章以及之前关于帮助系统章节所学的内容，使用 PowerShell 完成下述任务：

1. 显示正在运行的进程列表。

2. 显示最新的 100 个应用程序日志（请不要使用 Get-WinEvent，我们已经为你展示过完成该任务的另一个命令）。

3. 显示所有类型为“Cmdlet”的命令（我们已经展示了 Get-Command，你还需要阅读帮助文档，从而找出缩小该列表范围，正如本次动手实验所要求的）。

4. 显示所有的别名。

5. 创建一个新的别名。使用该别名，你可以运行“D”获取目录列表。

6. 显示以字母 M 开头的服务名称。同样，你需要阅读帮助文档找出所需的命令。请不要忘了星号（*），这是 PowerShell 中通用的通配符。

7. 显示所有的 Windows 防火墙规则，你需要使用 Help 或 Get-Command 找出所需的 Cmdlet。

8. 显示所有 Windows 防火墙的入站规则。可以使用和之前任务同样的 Cmdlet，但你需要阅读帮助文档找出所需的参数以及可选值。

我们希望上述任务对你来说很直白。如果是这样，那就太好了。你已经利用现有的命令行技巧来使得 PowerShell 帮助你完成实际的工作。如果你是命令行世界的新手，那么上述任务将会是你学习本书其他章节的敲门砖。