Microsoft Windows Server System

Windows PowerShell™ 入门指南

Microsoft Corporation

发布日期：2006 年 9 月

摘要

Windows PowerShell™ 是专为系统管理员设计的新 Windows 命令行外壳程序。该外壳程序包括交互式提示和脚本环境，两者既可以独立使用也可以组合使用。

本文档旨在为新用户介绍 Windows PowerShell，以及使他们了解其基本功能。有关更详细的信息，请参阅“Windows PowerShell 入门”。

Microsoft

目录

Windows PowerShell 入门指南版权声明

Windows PowerShell 设计目标

Windows PowerShell 是专为系统管理员设计的新 Windows 命令行外壳程序。该外壳程序包括交互式提示和脚本环境，两者既可以独立使用也可以组合使用。

与接受和返回文本的大多数外壳程序不同，Windows PowerShell 是在 .NET 公共语言运行时 (CLR) 和 .NET Framework 的基础上构建的，它接受和返回 .NET 对象。环境中的这一根本更改带来了管理和配置 Windows 的全新工具和方法。

Windows PowerShell 引入了 cmdlet（读作“command-let”）的概念，这是内置到外壳程序中的一个简单的单一功能命令行工具。可以分别使用每个 cmdlet，但是组合使用这些简单的工具执行复杂任务时才发挥其作用。Windows PowerShell 包括一百多个基本的核心 cmdlet，您可以编写自己的 cmdlet 并与其他用户共享它们。

与许多外壳程序一样，Windows PowerShell 为您提供了对计算机上文件系统的访问。此外，使用 Windows PowerShell 提供程序，还可以访问其他数据存储区，如注册表和数字

签名证书存储区，与访问文件系统一样容易。

本入门指南对 Windows PowerShell 进行了介绍：语言、cmdlet、提供程序和对象的使用。

主旨

本文档的主旨是帮助 Windows PowerShell 用户了解 Windows PowerShell 的入门知识。本文档介绍开始使用外壳程序时所需的外壳程序功能。有关外壳程序的详细分析、其功能以及如何使用外壳程序的示例，请参阅“Windows PowerShell 入门”。

Windows PowerShell 简介

大多数外壳程序（包括 Cmd.exe 以及 SH、KSH、CSH 和 BASH Unix 外壳程序）的运行方式是在新进程中执行命令或实用工具，然后将结果以文本形式显示给用户。经过数年的发展，许多文本处理实用工具（如 sed、AWK 和 PERL）现已发展为支持此交互。

这些外壳程序还具有内置到外壳程序中并在外壳程序进程中运行的命令，如 KSH 中的 **typeset** 命令和 Cmd.exe 中的 dir 命令。在大多数的外壳程序中，由于内置命令很少，因此创建了许多实用工具。

Windows PowerShell 有很大不同。

Windows PowerShell 不处理文本。相反，它基于 .NET 平台处理对象。

Windows PowerShell 附带了具有一致界面的大量内置命令。

所有的外壳程序命令都使用同一命令分析程序，而不是每个工具使用不同的分析程序。这样便可更轻松地学习如何使用每个命令。

其最好的优点在于，您不必忍痛舍弃自己惯用的工具。仍可以在 Windows PowerShell 中使用传统的 Windows 工具，如 Net、SC 和 Reg.exe。

Windows PowerShell Cmdlet

cmdlet（读作“command-let”）是 Windows PowerShell 中用于操作对象的单功能命令。可以通过其名称格式识别 cmdlet -- 由短划线 (-) 分隔的动词和名词，如 Get-Help、Get-Process 和 Start-Service。

在传统的外壳程序中，命令是从非常简单（如 attrib.exe）到非常复杂（如 netsh.exe）的可执行程序。

在 Windows PowerShell 中，大多数 cmdlet 都非常简单，它们设计为与其他 cmdlet 组

合使用。例如，“get”cmdlet 仅检索数据，“set”cmdlet 仅建立或更改数据，“format”cmdlet 仅设置数据格式，“out”cmdlet 仅将输出定向到指定的目标。

每个 cmdlet 都具有一个帮助文件，可以通过键入以下内容访问它：

```
get-help <cmdlet 名称> -detailed
```

cmdlet 帮助文件的详细视图包括 cmdlet 说明、命令语法、参数说明和说明 cmdlet 用法的示例。

新脚本语言

由于以下原因，Windows PowerShell 使用它自己的语言，而不是重用现有的语言：

Windows PowerShell 需要用于管理 .NET 对象的语言。

该语言需要为使用 cmdlet 提供一致的环境。

该语言需要支持复杂的任务，而不会使简单的任务变得更复杂。

该语言需要与在 .NET 编程中使用的高级语言（如 C#）一致。

Windows 命令和实用工具

可以在 Windows PowerShell 中运行 Windows 命令行程序，并可以在外壳程序中启动具有图形用户界面的 Windows 程序（如记事本和计算器）。还可以捕获程序生成的文本，并在外壳程序中使用该文本（与在 Cmd.exe 中很类似）。

处理对象

虽然您最初可能没有意识到，但是在 Windows PowerShell 中工作时，所使用的是 .NET 对象。随着您经验的增加，对象处理能力变得更明显，而且您发现自己使用的是对象，甚至用对象进行思考。

从技术上讲，.NET 对象是 .NET 类的实例，包含数据以及与该数据关联的操作。但是，可以将对象视为具有属性（与特性类似）和方法（可以对对象执行的操作）的数据实体。

例如，在 Windows PowerShell 中获取服务时，实际上是获取表示该服务的对象。查看有关服务的信息时，所查看的是其服务对象的属性。此外，启动服务时（即，在将服务的 Status 属性更改为“started”时），所使用的是服务对象的方法。

类型相同的所有对象都具有相同的属性和方法，但是对象的每个实例可能具有不同的属性值。例如，每个服务对象都具有 Name 和 Status 属性。但是，每个服务都可以具有不同的名称和不同的状态。

准备就绪后，了解对象是很容易的。若要查明 cmdlet 正获取对象的类型，请使用管道运算符 (|) 将 “get”命令的结果发送到 Get-Member 命令。例如，以下命令将 Get-Service 命令检索的对象发送到 Get-Member。

```
get-service | get-member
```

Get-Member 显示有关服务对象的信息，其中包括对象的类型名称及其属性和方法的列表。

```
TypeName:System.ServiceProcess.ServiceController

Name           MemberType    Definition
----           ----------    ----------
Name           AliasProperty Name = ServiceName
add_Disposed   Method        System.Void add_Disposed(EventHandler
value)
Close          Method        System.Void Close()
Continue       Method        System.Void Continue()
...
```

若要获取有关对象类的信息，请在 MSDN 中复制并粘贴类型名称，如 System.ServiceProcess.ServiceController。找到类后，可以阅读 MSDN 副主题，以了解基于该类的对象（如 Windows PowerShell 中的对象）的属性和方法。

若要查找特定对象的所有属性的值，请使用管道运算符 (|) 将 “get”命令的结果发送到 Format-List 或 Format-Table 命令。将 format cmdlet 的 Property 参数与表示所有的值 (*) 一起使用。例如，若要查找系统上 Schedule 服务的所有属性，请键入：

```
get-service schedule | format-list -property *
```

以下显示一个结果示例。

```
Name                  :Schedule
CanPauseAndContinue   :True
CanShutdown           :True
CanStop               :True
DisplayName           :Task Scheduler
DependentServices     :
MachineName           : .
ServiceName           :Schedule
ServicesDependedOn    :{RpcSs}
ServiceHandle         :SafeServiceHandle
Status                :Running
ServiceType           :Win32ShareProcess
Site                  :
Container             :
```

最初学习 Windows PowerShell 时，无需了解有关对象的任何信息，但是要意识到该概念。您将很快能够充分利用对象。

对象管道

使用对象的一个主要优点是，它使得用管道传输命令（即，将一个命令的输出作为输入传递到另一命令）容易得多。通信通常需要字符串操作，以便将输出从一种格式转换为另一种格式，并删除标题和列标题。

Windows PowerShell 提供了一个基于对象而不是基于文本的新交互模型。接收对象的 cmdlet 可以直接作用于其属性和方法，而无需进行转换或操作。用户可以通过名称引用对象的属性和方法，而不是计算数据在输出中的位置。

在以下示例中，将 IpConfig 命令的结果传递到 Findstr 命令。管道运算符 (|) 将其左侧命令的结果发送到其右侧的命令。在 Microsoft® Windows® PowerShell 中，无需操作字符串或计算数据偏移量。

```
PS> ipconfig | findstr "Address"
IP Address. . . . . . . . . . . . : 172.28.21.5
IP Address. . . . . . . . . . . . : 172.30.160.225
```

交互和脚本

交互式环境

与其他外壳程序一样，Windows PowerShell 支持完全交互式环境。在提示符下键入命令后，将处理该命令并在外壳程序窗口中显示输出。可以将命令输出发送到文件或打印机，也可以使用管道运算符 (|) 将输出发送到其他命令。

对脚本的支持

如果重复运行特定的命令或命令序列，或者如果开发一系列命令来执行复杂的任务，则会希望在文件中保存命令并执行命令文件，而不是在提示符下键入命令。保存有命令的文件称为脚本。

Windows PowerShell 除了提供交互式界面外，还完全支持脚本。在 Windows PowerShell 中，脚本文件的文件扩展名为 **.ps1**。若要运行脚本，请在命令提示符下键入该脚本的名称。文件扩展名是可选的。

例如：

```
c:\test\testscript.ps1
```

或

```
c:\test\testscript
```

即使脚本在当前目录中，也必须指定脚本文件的完全限定路径。若要指示当前目录，请键入目录名称或使用点 (.) 表示当前目录。例如：

```
.\testscript.ps1
```

虽然脚本在一些企业中非常有用 -- 甚至是必需的，但是它们可以用于传播恶意代码。因此，Windows PowerShell 中的安全策略（称为执行策略）允许您确定脚本是否可以运行，以及它们是否必须包括数字签名。为了消除明显的风险，Windows PowerShell 中的执行策略都不允许通过双击脚本的图标来运行它。有关详细信息，请键入：

```
get-help about_signing
```

Windows PowerShell 还包括一种非常丰富的脚本语言，使用该语言可以创建从最简单到非常复杂的脚本。它支持用于循环、条件、流控制和变量赋值的语言结构。

启动 Windows PowerShell

若要从“开始”菜单启动 Windows PowerShell，请依次单击“开始”、“所有程序”、**Windows PowerShell 1.0** 和 **Windows PowerShell**。

若要从“运行”框启动 Windows PowerShell，请单击“开始”，再单击“运行”，然后键入：

```
powershell
```

若要从命令提示符 (cmd.exe) 窗口启动 Windows PowerShell，请在命令提示符下键入：

```
powershell
```

若要查看用于启动 Windows PowerShell 的选项，请在命令提示符窗口中键入：

```
powershell -?
```

在 Windows PowerShell 打开时，可以使用 Get-Help cmdlet 查找帮助。在 Windows PowerShell 命令提示符下，键入：

```
get-help
```

使用 Windows PowerShell

本节介绍使用 Windows PowerShell 的基本知识。首先介绍 Get-Help cmdlet，该 cmdlet 显示有关 Windows PowerShell 中 cmdlet 和概念性主题的信息。然后，介绍几个基本的 cmdlet，说明如何使用 cmdlet 参数，然后说明如何设置 cmdlet 输出的格式以获取有用显示中所需的数据。最后的主题说明如何使用别名以便更容易地使用 Windows PowerShell，如何在 Windows PowerShell 中运行传统的 Windows 程序，以及如何管理错误。

Get-Help：获取帮助

Get-Help cmdlet 是用于了解 Windows PowerShell 的有用工具。通过阅读 cmdlet 的说明、了解有关概念并浏览语言主题，您可以开始了解如何使用 Windows PowerShell 了。

感兴趣的第一个主题可能是帮助系统。若要显示有关 Windows PowerShell 中帮助系统的信息，请键入：

```
get-help
```

然后，您可能会对了解几个基本的 cmdlet（如 Get-Help、Get-Command、Get-Process、Get-Service 和 Get-Eventlog）感兴趣。

若要显示 cmdlet 的帮助的最简单视图，请键入“get-help”，后跟该 cmdlet 的名称。例如，若要获取 Get-Command cmdlet 的帮助，请键入：

```
get-help get-command
```

如果 cmdlet 帮助的格式设置不正确（即，如果它以 XMLNS 标记开头），则可能是系统上的 Windows PowerShell 执行策略阻止系统加载用于设置 cmdlet 帮助格式的配置文件。有关执行策略的信息，请键入：

```
get-help about_signing
```

若要显示 cmdlet 的详细帮助，包括参数说明和示例，请使用 Get-Help 的 Detailed 参数。例如，若要获取 Get-Command cmdlet 的详细帮助，请键入：

```
get-help get-command -detailed
```

若要显示 cmdlet 的所有可用帮助，包括有关 cmdlet 及其参数的技术信息，请使用 Full 参数。例如，若要获取 Get-Command cmdlet 的完整帮助，请键入：

```
get-help get-command -full
```

也可以显示帮助文件的所选部分。若要仅查看示例，请使用 Examples 参数。例如，若要显示 Get-Command cmdlet 的示例，请键入：

```
get-help get-command -examples
```

若要仅查看详细的参数说明，请使用 Get-Help 的 Parameter 参数。可以指定参数的名称，或者使用通配符 (*) 指定所有参数。例如，若要查看 Get-Command 的 TotalCount 参数说明，请键入：

```
get-help get-command -parameter totalcount
```

若要查看 Get-Command cmdlet 的所有参数，请键入：

```
get-help get-command -parameter *
```

也可以使用调用 Get-Help 的 Windows PowerShell 函数之一。Help 函数一次显示一整屏帮助内容。Man 函数显示与 Unix 中的手册页类似的帮助。若要使用 Help 和 Man 函数显示 Get-Command cmdlet 的帮助，请键入：

```
man get-command
```

或

```
help get-command
```

请求特定的帮助主题时，Get-Help 将显示该主题的内容。但是使用通配符请求多个主题时，Get-Help 将显示一个主题列表。例如，若要查看“Get”cmdlet 的帮助主题列表，请键入：

get-help get-*

有关 Windows PowerShell 中概念的帮助以“about_”开头。若要显示有关 Windows PowerShell 概念的帮助，请键入“get-help”，后跟概念名称。例如，若要获取有关通配符的帮助，请键入：

```
get-help about_wildcard
```

若要显示 Windows PowerShell 中所有概念性帮助主题的列表，请键入：

```
get-help about_*
```

通过阅读帮助主题并尝试示例，您将了解 Windows PowerShell 的工作原理以及在您的工作中如何使用它。

使用 Cmdlet

cmdlet（读作“command-let”）是一个内置到外壳程序中的简单的单一功能命令行工具。可以就像使用传统的命令和实用工具那样使用 cmdlet。首先在 Windows PowerShell 命

令提示符下键入 cmdlet 的名称。Windows PowerShell 命令不区分大小写，因此可以用任意大小写键入。

例如，可以尝试 Get-Date cmdlet：

```
C:\PS> get-date
2005 年 11 月 10 日，星期四，下午 4:43:50
```

若要在会话中列出 cmdlet，请使用 Get-Command cmdlet，且不带任何命令参数。

```
PS> get-command

CommandType     Name                              Definition
-----------     ----                              ----------
Cmdlet          Add-Content                       Add-Content [-Path] <String[...
Cmdlet          Add-History                       Add-History [[-InputObject] ...
Cmdlet          Add-Member                        Add-Member [-MemberType] <PS...
...
…
```

默认的 Get-Command 显示有以下三列：CommandType、Name 和 Definition。列出 cmdlet 时，Definition 列显示 cmdlet 的语法。语法中的省略号 (…) 指示数据被截断。

Get-Command cmdlet 还获取除 cmdlet 之外的命令和命令元素，其中包括在 Windows PowerShell 中可用的别名（命令昵称）、函数和可执行文件。

通过使用 Get-Command 的 Name 参数，以下命令列出了在 Windows PowerShell 中可用的可执行文件。

```
PS> get-command *.exe

CommandType Name                    Definition
----------- ----                    ----------
Application 000StTHK.exe            C:\WINDOWS\system32\000StTHK.exe
Application 00THotkey.exe           C:\WINDOWS\system32\00THotkey.exe
Application accwiz.exe              C:\WINDOWS\system32\accwiz.exe
...
```

列出可执行文件时，Defintion 列包含可执行文件的完整路径。

然后，尝试一些其他 cmdlet，如 Get-Process、Get-Service、Get-EventLog 和 Get-Alias。

如果对简单的“Get-”cmdlet 已非常熟悉，请尝试更有趣的 cmdlet，如 Get-WmiObject。此 cmdlet 非常有用，因为使用它可以查看和更改远程计算机的组件。例如，以下命令获取有关 Server01 远程计算机上 BIOS 的信息：

```
get-wmiobject win32_bios -computername server01
```

如果需要任何 cmdlet 的帮助，请键入：

get-help <cmdlet 名称> -detailed

例如：

get-help get-alias -detailed。

了解对象：Get-Member

Get-Member 是最有用的 cmdlet 之一，它显示有关命令返回的 .NET 对象的信息。该信息包括对象的类型、属性和方法。

若要使用 Get-Member，请使用管道运算符 (|) 将命令结果发送到 Get-Member。例如：

get-service | get-member

此命令显示 Get-Service 实际上返回了一组 System.ServiceProcess.ServiceController 对象 -- 计算机上的每个服务都有一个对象。

```
TypeName:System.ServiceProcess.ServiceController

Name                      MemberType    Definition
----                      ----------    ----------
Name                      AliasProperty Name = ServiceName
add_Disposed              Method        System.Void add_Disposed(EventHandler
value)
Close                     Method        System.Void Close()
Continue                  Method        System.Void Continue()
CreateObjRef              Method        System.Runtime.Remoting.ObjRef
CreateObjRef(Type requestedType)
Dispose                   Method        System.Void Dispose()
Equals                    Method        System.Boolean Equals(Object obj)
ExecuteCommand            Method        System.Void ExecuteCommand(Int32 command)
get_CanPauseAndContinue   Method        System.Boolean get_CanPauseAndContinue()
get_CanShutdown           Method        System.Boolean get_CanShutdown()
get_CanStop               Method        System.Boolean get_CanStop()
get_Container             Method        System.ComponentModel.IContainer
get_Container()
get_DependentServices     Method        System.ServiceProcess.ServiceController[]
get_DependentServices()
get_DisplayName           Method        System.String get_DisplayName()
get_MachineName           Method        System.String get_MachineName()
get_ServiceHandle         Method        System.Runtime.InteropServices.SafeHandle
get_ServiceHandle()
get_ServiceName           Method        System.String get_ServiceName()
get_ServicesDependedOn    Method        System.ServiceProcess.ServiceController[]
get_ServicesDependedOn()
get_ServiceType           Method        System.ServiceProcess.ServiceType
get_ServiceType()
get_Site                  Method        System.ComponentModel.ISite get_Site()
get_Status                Method
System.ServiceProcess.ServiceControllerStatus get_Status()
```

```
GetHashCode                Method        System.Int32 GetHashCode()
GetLifetimeService         Method        System.Object GetLifetimeService()
GetType                    Method        System.Type GetType()
InitializeLifetimeService Method        System.Object InitializeLifetimeService()
Pause                      Method        System.Void Pause()
Refresh                    Method        System.Void Refresh()
remove_Disposed            Method        System.Void remove_Disposed(EventHandler
value)
set_DisplayName            Method        System.Void set_DisplayName(String value)
set_MachineName            Method        System.Void set_MachineName(String value)
set_ServiceName            Method        System.Void set_ServiceName(String value)
set_Site                   Method        System.Void set_Site(ISite value)
Start                      Method        System.Void Start(), System.Void
Start(String[] args)
Stop                       Method        System.Void Stop()
ToString                   Method        System.String ToString()
WaitForStatus              Method        System.Void
WaitForStatus(ServiceControllerStatus desiredStatus), System.Voi...
CanPauseAndContinue        Property      System.Boolean CanPauseAndContinue {get;}
CanShutdown                Property      System.Boolean CanShutdown {get;}
CanStop                    Property      System.Boolean CanStop {get;}
Container                  Property      System.ComponentModel.IContainer Container
{get;}
DependentServices          Property      System.ServiceProcess.ServiceController[]
DependentServices {get;}
DisplayName                Property      System.String DisplayName {get;set;}
MachineName                Property      System.String MachineName {get;set;}
ServiceHandle              Property      System.Runtime.InteropServices.SafeHandle
ServiceHandle {get;}
ServiceName                Property      System.String ServiceName {get;set;}
ServicesDependedOn         Property      System.ServiceProcess.ServiceController[]
ServicesDependedOn {get;}
ServiceType                Property      System.ServiceProcess.ServiceType
ServiceType {get;}
Site                       Property      System.ComponentModel.ISite Site
{get;set;}
Status                     Property
System.ServiceProcess.ServiceControllerStatus Status {get;}
```

此信息看起来技术性很强，但是它实际上非常实用。

通过类型名称（如“System.ServiceProcess.ServiceController”）可以知道 cmdlet 返回什么类型的 .NET 对象。若要获取有关此 .NET 类中对象的信息，请将类型名称粘贴在 MSDN 上的“**Search**”（搜索）文本框中。关联的 MSDN 主题包括有关此类中对象的属性和方法的信息，其中包括 Get-Service 返回的对象。

Property 类型表示对象的属性。每个属性的值是有关服务对象的信息。例如，Service Controller 对象具有 **CanPauseAndContinue** 属性。该属性的 MSDN 说明解释，该属性指示是否可以暂停和恢复服务。

若要列出**特定服务的属性**值，请键入：

```
(get-service <服务名称>).<属性名称>
```

例如：

```
(get-service alerter).canpauseandcontinue
```

若要显示 Alerter 服务的 **CanPauseAndContinue** 属性的名称和值列表，请键入：

```
get-service alerter | format-list -property name, CanPauseAndContinue
```

若要显示 Alerter 服务的所有属性值的列表，请键入：

```
get-service alerter | format-list -property *
```

若要显示所有服务的 **CanPauseAndContinue** 属性的名称和值表，请键入：

```
get-service | format-table -property name, CanPauseAndContinue
```

Method 类型表示对象的方法，即可以对对象执行的操作。例如，ServiceController 对象具有 Stop 方法，使用该方法可以停止服务。

若要调用服务对象的方法，请使用以下格式。（务必包括圆括号）。

```
(get-service <服务名称>).<方法名称>()
例如，
(get-service schedule).stop()
```

有关 Get-Member 命令的信息，请键入：

get-help get-member -detailed。

使用 Cmdlet 参数

Cmdlet 参数的标识方法是在参数名称前放一个连字符 (-)。（在 Windows PowerShell 中斜杠（/ 和 \）不用于参数。）

键入参数名称时，可以键入整个名称，但是只需键入必要的字符数，能够将该参数名称与 cmdlet 的其他参数名称区分开即可。

例如，Get-Help cmdlet 具有一个名为“Detailed”的参数，但是可以键入“-det”，这就足以将它与 Get-Help 的 Debug 参数区分开。

一些参数名称是可选的。可以通过键入参数值而不是键入参数名称来使用参数。但是，如果省略参数名称，则参数值出现在命令中的位置必须与它出现在语法图表中的位置相同。

例如，Get-Help cmdlet 具有一个 Name 参数，该参数指定 cmdlet 或概念的名称。可以

键入 Name 参数的名称，也可以省略它。若要获取 Get-Alias 命令的帮助，可以键入：

```
get-help -name get-alias
```

或

```
get-help get-alias
```

若要查找可选参数名称，请参阅帮助文件中的语法块。可选参数名称出现在方括号中，例如：

```
Get-Help [[-Name] <字符串>]...
```

通用参数

所有 cmdlet 都支持一组称为通用参数的参数。此功能提供了通向 Windows PowerShell 的统一接口。

如果 cmdlet 支持通用参数，则使用该参数将不会出错。但是，该参数在某些 cmdlet 中可能没有任何作用。有关通用参数的说明，请键入：

get-help about_commonparameters

设置命令输出的格式

在传统的外壳程序中，每个工具或命令确定其输出的格式。一些工具允许自定义输出，因此它们包括用于控制输出格式的特殊参数。

在 Windows PowerShell 中，只有下列 Format cmdlet 才设置输出格式：

Format-List

Format-Custom

Format-Table

Format-Wide

其他 cmdlet 都不设置输出格式。因此，无需了解多个工具的格式设置例程和参数。只需了解 Format cmdlet 及其参数即可。

运行命令时，Windows PowerShell 调用默认的格式化程序，该程序由所显示的数据类型确定。格式化程序确定显示输出的哪些属性以及用列表还是表显示它们。

例如，使用 Get-Service cmdlet 时，默认显示是一个包含三列的表，如下所示：

```
C:\PS> get-service
Status   Name               DisplayName
```

```
------   ----                 -----------
Running  AdtAgent             Event Forwarder
Stopped  Alerter              Alerter
Running  ALG                  Application Layer Gateway Service
```

...

若要更改任何 cmdlet 的输出格式，请使用管道运算符 (|) 将命令输出发送到 Format cmdlet。

例如，以下命令将 Get-Service 命令的输出发送到 Format-List cmdlet。因此，对于每个服务，服务数据被设置为列表格式。

```
C:\PS> get-service | format-list
Name                  :AdtAgent
DisplayName           :Event Forwarder
Status                :Running
DependentServices     : {}
ServicesDependedOn    :{eventlog, dnscache}
CanPauseAndContinue   :False
CanShutdown           :True
CanStop               :True
ServiceType           :Win32OwnProcess

Name                  :Alerter
DisplayName           :Alerter
Status                :Stopped
DependentServices     : {}
ServicesDependedOn    :{LanmanWorkstation}
CanPauseAndContinue   :False
CanShutdown           :False
CanStop               :False
ServiceType           :Win32ShareProcess

Name               :ALG
DisplayName           :Application Layer Gateway Service
Status             :Running
DependentServices     : {}
```

在此格式中，不仅数据出现在列表（而不是表）中，而且存在有关每个服务的详细信息。每个服务的数据不是三列，而是有九行数据。Format-List 不检索额外的服务信息。该数据一直在 Get-Service 检索的对象中，但是默认的格式化程序 Format-Table 因无法在屏幕上显示三列以上而忽略了它。

除了确定数据出现在列表中还是表中之外，还可以确定显示对象的哪些属性。例如，Get-Service 的默认显示仅显示服务对象的 Status、Name 和 DisplayName 属性。

若要查看对象的所有属性，请使用管道运算符 (|) 将命令输出发送到 Get-Member cmdlet。例如，若要查看服务对象的所有属性，请键入：

```
get-service | get-member -membertype *property

TypeName:System.ServiceProcess.ServiceController

Name                MemberType    Definition
----                ----------    ----------
Name                AliasProperty Name = ServiceName
CanPauseAndContinue Property      System.Boolean CanPauseAndContinue {get;}
CanShutdown         Property      System.Boolean CanShutdown {get;}
CanStop             Property      System.Boolean CanStop {get;}
Container           Property      System.ComponentModel.IContainer Container
{get;}
DependentServices   Property      System.ServiceProcess.ServiceController[]
DependentServices {get;}
DisplayName         Property      System.String DisplayName {get;set;}
MachineName         Property      System.String MachineName {get;set;}
ServiceHandle       Property      System.Runtime.InteropServices.SafeHandle
ServiceHandle {get;}
ServiceName         Property      System.String ServiceName {get;set;}
ServicesDependedOn  Property      System.ServiceProcess.ServiceController[]
ServicesDependedOn {get;}
ServiceType         Property      System.ServiceProcess.ServiceType ServiceType
{get;}
Site                Property      System.ComponentModel.ISite Site {get;set;}
Status              Property      System.ServiceProcess.ServiceControllerStatus
Status {get;}
```

由于所有这些属性都在 Get-Service 为每个服务检索的对象中，因此可以显示其中任一属性或所有属性。使用 Format cmdlet 的 Property 参数可以选择要显示的属性以及在其中显示它们的其他属性。例如，以下命令使用 Format-Table 命令仅显示服务的 Name、ServiceType 和 CanShutDown 属性。

```
get-service | format-table name, Servicetype, Canshutdown
```

这仅仅是可以对 Windows PowerShell 显示所执行操作的开始。有关更多详细信息，请使用以下命令来阅读有关 Format cmdlet 的帮助：

```
get-help format-list
get-help format-table
get-help format-wide
get-help format-custom
```

使用别名

键入 Cmdlet 名称很麻烦。为最大限度地减少键入内容，并使习惯其他外壳程序的用户更容易使用 Windows PowerShell，Windows PowerShell 支持别名（即命令的替代名称）的概念。可以为 cmdlet 名称、函数名称或可执行文件的名称创建别名，然后在任何命令中

键入别名而不是名称。

Windows PowerShell 包括许多内置的别名，而且您可以创建自己的别名。您创建的别名仅在当前会话内有效。若要创建持久性别名，请将别名添加到 Windows PowerShell 配置文件。

若要查找会话中的所有别名，请键入：

```
get-alias
```

若要查找 cmdlet 的别名，请键入：

```
get-alias | where-object {$_.definition -eq "<cmdlet 名称>"}
```

例如：

```
get-alias | where-object {$_.definition -eq "set-location"}
```

Windows PowerShell 中的别名由 Windows PowerShell Alias 提供程序支持，该提供程序是一个 .NET 程序集，通过它可以查看驱动器（与 Windows 中的文件系统驱动器非常类似）中的别名。用于别名的驱动器是 Alias:。

若要转到 Alias 驱动器，请键入：

```
set-location alias:
```

若要查看别名，即 Alias: 驱动器中的子项，请键入

```
get-childitem
```

若要从另一驱动器查看 Alias: 驱动器中的子项，请在命令中包括该驱动器名称。例如：

```
get-childitem alias:
```

创建别名

若要在 Windows PowerShell 中为 cmdlet 和命令创建别名，请使用 **Set-Alias** cmdlet。例如，若要为 Get-Help cmdlet 创建“gh”别名，请键入：

```
set-alias gh get-help
```

也可以为命令（例如，用于启动程序的命令）创建别名。例如，若要为 Notepad 创建别名“np”，请键入：

```
set-alias np c:\windows\notepad.exe
```

（在您的系统上，Notepad 的路径可能是不同的。）

删除别名

若要删除别名，请使用 Remove-Item cmdlet 从 Alias: 驱动器中删除别名。例如，若要删除“ls”别名，请键入

```
remove-item alias:ls
```

使用函数创建替代名称

可以为 cmdlet、函数或可执行文件创建别名，但不能为带参数的命令创建别名。但是，可以创建其行为与别名非常类似的函数。

例如，若要在运行 Windows XP 的计算机上使用记事本打开 Boot.ini 文件，请键入：

```
notepad c:\boot.ini
```

不能为“notepad c:\boot.ini”创建别名，但是可以创建函数。以下命令创建 **bootini** 函数。

```
function bootini {notepad c:\boot.ini}
```

此函数的行为与别名类似。如果在 Windows PowerShell 提示符下键入 **bootini**，则 Boot.ini 将在记事本中打开。

使用 Windows 程序

在 Windows PowerShell 中，可以运行 Windows 命令行程序和启动 Windows 图形程序。如果程序生成文本输出，则可以捕获该文本，并在新外壳程序中使用它，就像在任何外壳程序中一样。

若要在 Windows PowerShell 中运行程序（如记事本），该程序的可执行文件必须位于 Path 环境变量所含的目录中，因为 Path 环境变量的值确定 Windows PowerShell 查找应用程序、实用工具和脚本的位置。（Cmdlet 不必位于 Path 目录中。）

若要查看 Path 环境变量中的路径，请键入：

```
PS> $env:path
```

若要将目录添加到 Path 环境变量，请键入：

```
PS> $env:path += ";newdirectory"
```

例如，若要将 WordPad.exe 文件的目录添加到 Path 变量，请键入：

```
PS> $env:path += ";C:\Program Files\Windows NT\Accessories"
```

与 **set** 命令一样，此赋值语句仅更改当前 Windows PowerShell 会话的 Path 值。若要使

该更改成为永久性更改，请将赋值语句添加到 Windows PowerShell 配置文件。有关详细信息，请参阅“Windows PowerShell 配置文件”主题。

管理错误

使用外壳程序时偶尔会出现错误，如尝试将位置设置为不存在的目录或者在没有所需特权的情况下尝试删除文件时。

在 Windows PowerShell 中，有以下两种类型的错误：

终止性错误：终止命令执行的错误。

非终止性错误：不终止命令执行的错误。

例如，如果要删除目录中的所有 .TMP 文件，则当其中一个文件无法删除时，您可能不希望该操作停止。通常，您希望删除可以删除的所有文件，然后再回来处理无法删除的文件。

无法删除文件时出现的错误称为非终止性错误。出现非终止性错误时，Windows PowerShell 会继续运行，而不管该错误，然后显示该错误以及输出。

更严重的错误将停止命令处理。这些错误称为终止性错误。终止性错误会停止命令的处理。例如，如果提交的数据无效或者您没有执行命令所需的权限，则 Windows PowerShell 将生成非终止性错误。

在 Windows PowerShell 中导航

Windows PowerShell 的最强大功能之一是，它允许您使用在文件系统中用来导航的相同熟悉方法在许多不同的数据存储区中导航。

除了熟悉的文件系统驱动器（如 C: 和 D:）之外，Windows PowerShell 还包括表示 HKEY_LOCAL_MACHINE (HKLM:) 和 HKEY_CURRENT_USER (HKCU:) 注册表配置单元、计算机上的数字签名证书存储区 (Cert:) 以及当前会话中函数 (Function:) 等的驱动器。这些驱动器称为 Windows PowerShell 驱动器。

Windows PowerShell 附带有 Windows PowerShell 提供程序支持的数个有用驱动器。若要查看 Windows PowerShell 驱动器的列表，请键入：

```
get-psdrive
```

在文件系统中导航

启动 Windows PowerShell 时，您可能很想键入熟悉的 **cd**、**dir** 或 **ls**。可以这样做！**cd** 是 Set-Location cmdlet（将当前位置更改为指定路径的 cmdlet）的别名。**dir** 和 **ls** 是 Get-Childitem cmdlet（获取某个位置中子项的 cmdlet）的别名。

若要在文件系统驱动器中导航，请使用 Set-Location (cd) 和 Get-Childitem（dir、ls）cmdlet。在 Windows PowerShell 中，驱动器由驱动器名称后跟冒号 (:) 表示，如 **C:**，父项与子项用反斜杠 (\) 或正斜杠 (/) 隔开，如 **C:\Windows\System32**。

有几项功能可以使在 Windows PowerShell 中导航更容易：

有表示当前目录 (.) 和目录内容 (*) 的符号。

有表示主目录的内置变量 **$home** 和表示 Windows PowerShell 安装目录的内置变量 **$pshome**。

与在其他外壳程序中一样，可以更改位置，创建、删除、移动和复制目录及文件，以及更改其属性。甚至可以将 Tab 补齐功能用于路径名。有关详细信息，请参阅 Item cmdlet（Get-Item、Get-Childitem、New-Item、Remove-Item、Set-Item、Move-Item 和 Copy-Item）的帮助。

在注册表中导航

可以使用与在文件系统驱动器中用来导航的相同方法在 Windows 注册表中进行导航。在 Windows PowerShell 中，HKEY_LOCAL_MACHINE 配置单元映射到 Windows PowerShell HKLM: 驱动器，而 HKEY_CURRENT_USER 驱动器映射到 Windows PowerShell HKCU: 驱动器。

例如：

```
PS C:\> cd hklm:
PS HKLM:\> dir
PS HKLM:\> dir
Hive:Microsoft.PowerShell.Core\Registry::HKEY_LOCAL_MACHINE
SKC  VC Name                           Property
---  -- ----                           --------
4    0 HARDWARE                       {}
1    0 SAM                            {}
Get-ChildItem :不允许所请求的注册表访问权。
所在行:1 字符:3
+ dir <<<<
39   2 SOFTWARE                       {flash, (default)}
8    0 SYSTEM                         {}
PS HKLM:\> cd system\currentcontrolset\control
PS HKLM:\system\currentcontrolset\control> dir
```

在导航时，您将注意到，**dir** (Get-Childitem) 的输出在注册表驱动器中与在文件系统中是不同的。由于注册表具有包含不同信息的不同驱动器，因此外壳程序提供了数据的不同视图。在这种情况下，知道存在多少子项和项是很重要的，因此除了子项和项的名称外，输出还包括子项计数 (SKC) 和值项计数 (VC)。

```
PS> cd "CurrentControlSet\Control\Session Manager"
PS> dir
Hive:Registry::HKEY_LOCAL_MACHINE\system\CurrentControlSet\Control\Session
Manager

SKC  VC ChildName                      Property
---  -- ---------                      --------
0    1 AppCompatibility               {AppCompatCache}
15   0 AppPatches                      {}
0    7 DOS Devices                    {AUX, MAILSLOT, NUL, PIPE, PRN, UNC, f...
```

到达注册表项之前，不会在导航中遇到很多的差异。注册表项中的项被认为是它们所在项的属性。因此，使用 Get-ItemProperty cmdlet 可以检索它们。

例如，如果要查看 Windows PowerShell 执行策略的值，可以使用 Get-ExecutionPolicy cmdlet，或导航到在 HKLM:\Software\Microsoft\PowerShell\1\ShellIds\Microsoft.PowerShell 中存储值的 **ExecutionPolicy** 注册表项。

```
PS C:\> cd hklm:
PS HKLM:\> cd software\microsoft\powershell\1\ShellIds\Microsoft.PowerShell
PS HKLM:\software\microsoft\powershell\1\ShellIds\Microsoft.PowerShell> dir
PS HKLM:\software\microsoft\powershell\1\ShellIds\Microsoft.PowerShell> get-
itemproperty -path .-name executionpolicy

PSPath
:Microsoft.PowerShell.Core\Registry::HKEY_LOCAL_MACHINE\software\microsoft\powersh
ell\1\ShellIds\Micro
soft.PowerShell
PSParentPath
:Microsoft.PowerShell.Core\Registry::HKEY_LOCAL_MACHINE\software\microsoft\powersh
ell\1\ShellIds
PSChildName     :Microsoft.PowerShell
PSDrive         :HKLM
PSProvider      :Microsoft.PowerShell.Core\Registry
ExecutionPolicy :RemoteSigned
```

在证书存储区中导航

也可以在计算机上的数字签名证书存储区中导航。证书存储区映射到 theWindows PowerShell Cert: 驱动器。以下示例说明如何使用 Set-Location (cd) 和 Get-Childitem（dir、ls）在 Cert: 驱动器中导航。

```
PS C:\> cd cert:
PS cert:\> dir
Location   :CurrentUser
StoreNames :{TrustedPeople, _NMSTR, Trust, REQUEST...}

Location   :LocalMachine
StoreNames :{_NMSTR, Trust, REQUEST, TrustedPeople...}

PS cert:\> cd currentuser
PS cert:\currentuser> dir

Name :TrustedPeople
Name :_NMSTR
Name :Trust
Name :REQUEST
Name :AuthRoot
Name :ACRS
Name :My
Name :addressbook
Name :Disallowed
Name :CA
Name :UserDS
Name :Root
Name :TrustedPublisher

PS cert:\currentuser> cd authroot
PS cert:\currentuser\authroot> dir
Directory:Microsoft.PowerShell.Security\Certificate::currentuser\authroot
Thumbprint                                Subject
----------                                -------
F88015D3F98479E1DA553D24FD42BA3F43886AEF  O=C&W HKT SecureNet CA SGC Root, C=hk
F44095C238AC73FC4F77BF8F98DF70F8F091BC52  CN=Class 3TS Primary CA, O=Certplus,
C=FR
EF2DACCBEABB682D32CE4ABD6CB90025236C07BC  O="Colegio Nacional de Correduria
Publica Mexicana, A.C.", CN="Autoridad C...
...
PS cert:\currentuser\authroot> get-childitem
F88015D3F98479E1DA553D24FD42BA3F43886AEF
Directory:Microsoft.PowerShell.Security\Certificate::currentuser\authroot
Thumbprint                              Subject
----------                              -------
F88015D3F98479E1DA553D24FD42BA3F43886AEF  O=C&W HKT SecureNet CA SGC Root, C=hk

PS cert:\currentuser\authroot> get-childitem
F88015D3F98479E1DA553D24FD42BA3F43886AEF | format-list -property *


PSPath
:Microsoft.PowerShell.Security\Certificate::currentuser\authroot\F88015D3F98479E1D
A553D24FD42BA3F43
886AEF
PSParentPath
```

```
:Microsoft.PowerShell.Security\Certificate::currentuser\authroot
PSChildName         :F88015D3F98479E1DA553D24FD42BA3F43886AEF
PSDrive             :cert
PSProvider          :Microsoft.PowerShell.Security\Certificate
PSIsContainer       :False
Archived            :False
Extensions          : {}
FriendlyName        :CW HKT SecureNet CA SGC Root
IssuerName
:System.Security.Cryptography.X509Certificates.X500DistinguishedName
NotAfter            :10/16/2009 2:59:00 AM
NotBefore           :6/30/1999 3:00:00 AM
HasPrivateKey       :False
PrivateKey          :
PublicKey           :System.Security.Cryptography.X509Certificates.PublicKey
RawData             : {48, 130, 2, 235...}
SerialNumber        : 00
SubjectName
:System.Security.Cryptography.X509Certificates.X500DistinguishedName
SignatureAlgorithm :System.Security.Cryptography.Oid
Thumbprint          :F88015D3F98479E1DA553D24FD42BA3F43886AEF
Version             : 1
Handle              : 1577256
Issuer              :O=C&W HKT SecureNet CA SGC Root, C=hk
Subject             :O=C&W HKT SecureNet CA SGC Root, C=hk
```

在其他驱动器中导航

除了文件系统、注册表和证书驱动器外，Windows PowerShell 还附带有几个其他有用的驱动器，其中包括别名 (Alias:)、环境提供程序 (Env:)、函数 (Function:) 和变量 (Variable:) 驱动器。使用相同的基本方法可以在这些驱动器中导航。

关于 Windows PowerShell 驱动器

Windows PowerShell 中扩展导航功能背后的概念是 Windows PowerShell 驱动器。

可以在 Windows PowerShell 的任何数据存储区中创建 Windows PowerShell 驱动器，而且它们可以具有任何有效名称，如 C: 或 “My Drive”后跟冒号 (:)。可以使用在文件系统驱动器中所用的相同方法在这些驱动器中导航。但是，Windows PowerShell 驱动器仅在 Windows PowerShell 中是可见的。无法在 Windows 资源管理器或 Cmd.exe 中查看或访问它们。

Windows PowerShell 附带有 Windows PowerShell 提供程序支持的数个有用驱动器。若要查看 Windows PowerShell 驱动器的列表，请键入：

```
get-psdrive
```

也可以使用 **New-PsDrive** cmdlet 创建自己的 Windows PowerShell 驱动器。例如，若要创建名为“MyDocs:”的新驱动器（位于 **My Documents** 目录中），请键入：

```
new-psdrive -name MyDocs -psprovider FileSystem -root "$home\My Documents"
```

现在，可以像使用任何其他驱动器那样使用 MyDocs: 驱动器。可以将您的位置转到该驱动器，枚举其内容以及更改其属性。

驱动器和提供程序

Windows PowerShell 提供程序使 Windows PowerShell 中的驱动器可用于您的会话，这些提供程序是 .NET 程序集，它们使在专用数据存储区中的数据在 Windows PowerShell 中可用，以便您可以轻松地查看和管理该数据。有关 Windows PowerShell 提供程序的信息，请键入：

```
get-help about_psprovider
```

若要查看 Windows PowerShell 提供程序的列表，请键入：

```
get-psprovider
```

有关提供程序帮助文件的列表，请键入：

```
get-help -category provider
```

有关特定提供程序的信息，请键入：

```
get-help <提供程序名称>
```

例如，

```
get-help registry
```

自定义 Windows PowerShell

本节介绍可以自定义 Windows PowerShell 以最大限度地满足您的需要的几种方法。

检查执行策略

脚本是一种功能非常强大的工具，但是它可能被滥用于恶意目的。为保护用户数据和操作系统的完整性，Windows PowerShell 包括了若干安全功能，其中之一是执行策略。

Windows PowerShell 执行策略确定是否允许脚本运行，如果它们可以运行，则确定它们是否必须经过数字签名。它还确定是否可以加载配置文件。

默认的执行策略 Restricted 是最安全的执行策略。它不允许任何脚本运行，而且不允许加载任何配置文件，其中包括 Windows PowerShell 配置文件。您仍然能够以交互方式使用 Windows PowerShell。

但是，如果要运行脚本或加载配置文件，则可以更改系统上的执行策略。有关信息和说明，请键入：

```
get-help about_signing
```

若要查找系统上的执行策略，请键入：

```
get-executionpolicy
```

若要更改系统上的执行策略，请使用 Set-ExecutionPolicy cmdlet。例如，若要将执行策略更改为 RemoteSigned，请键入：

```
set-executionpolicy remotesigned
```

Windows PowerShell 执行策略保存在 Windows 注册表中，即使您卸载并重新安装 Windows PowerShell 也保留执行策略。

Windows Powershell 配置文件

将别名、函数和变量添加到 Windows PowerShell 时，实际上仅将它们添加到当前的 Windows PowerShell 会话。如果退出会话或者关闭 Windows PowerShell，则更改将丢失。

若要保留这些更改，可以创建 Windows PowerShell 配置文件，然后将别名、函数和变量添加到配置文件。每次启动 Windows PowerShell 时，都会加载该配置文件。

若要加载配置文件，Windows PowerShell 执行策略必须允许您加载配置文件。如果它不允许，则加载配置文件的尝试将失败，而且 Windows PowerShell 显示一条错误消息。

了解配置文件

在 Windows PowerShell 中可以有四个不同的配置文件。配置文件按加载顺序列出。较特定的配置文件优先于较不特定的配置文件（如果它们适用）。

%windir%\system32\WindowsPowerShell\v1.0\profile.ps1

此配置文件适用于所有用户和所有外壳程序。

%windir%\system32\WindowsPowerShell\v1.0\ Microsoft.PowerShell_profile.ps1

此配置文件适用于所有用户，但仅适用于 Microsoft.PowerShell 外壳程序。

%UserProfile%\My Documents\WindowsPowerShell\profile.ps1

此配置文件仅适用于当前用户，但影响所有外壳程序。

%UserProfile%\\My Documents\WindowsPowerShell\Microsoft.PowerShell_profile.ps1

此配置文件仅适用于当前用户和 Microsoft.PowerShell 外壳程序。

创建配置文件

可以创建、共享和分发配置文件，以便在较大的企业中强制实施 Windows PowerShell 的一致视图。

配置文件不是自动创建的。若要创建配置文件，请在指定位置中创建具有指定名称的文本文件。

通常，将使用特定于用户、特定于外壳程序的配置文件（称为用户配置文件）。此配置文件的位置存储在 $profile 变量中。

若要确定是否已创建用户配置文件，请键入：

```
test-path $profile
```

如果存在该配置文件，则响应为 **True**；否则响应为 **False**。

若要创建用户配置文件，请键入：

```
new-item -path $profile -itemtype file -force
```

若要在记事本中打开配置文件，请键入：

```
notepad $profile
```

若要创建其他配置文件之一，如适用于所有用户和所有外壳程序的配置文件，请键入：

```
new-item -path C:\Windows\System32\WindowsPowerShell\v1.0\profile.ps1 -itemtype
file -force
```

对于 Windows PowerShell 中的环境变量，不能使用“%”表示法。若要标识 Windows 环境变量，请使用以下格式：$env:<变量>，如 $env:windir：

```
new-item -path C:\Windows\System32\WindowsPowerShell\v1.0\profile.ps1 -itemtype
file -force
```

如果在记事本中创建配置文件，然后保存它，请务必将文件名括在引号中。例如：

```
"profile.ps1"
```

如果没有引号，则记事本会将 .txt 文件扩展名追加到文件，而 Windows PowerShell 将无法识别它。

使用配置文件存储日常使用的别名、函数和变量。一个非常有用的函数是用于在喜爱的文本编辑器中打开配置文件的函数。例如，以下命令创建一个名为 pro 的函数，该函数用于在记事本中打开用户配置文件。

```
function pro { notepad $profile }
```

有了设计良好的配置文件，就可以更轻松地使用 Windows PowerShell 和管理系统。