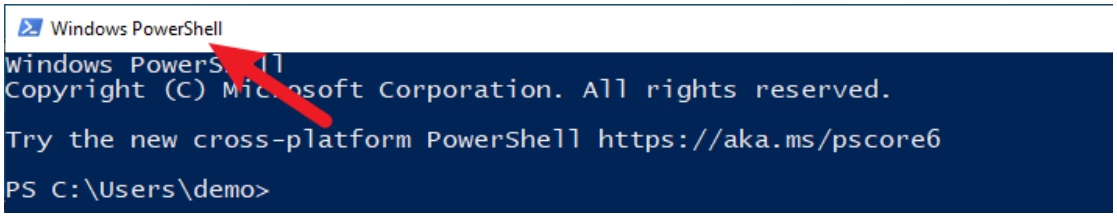
七、 开始使用 PowerShell

(一) 启动 PowerShell

通过单击“Windows PowerShell”快捷方式启动了 PowerShell 控制台，如所示。



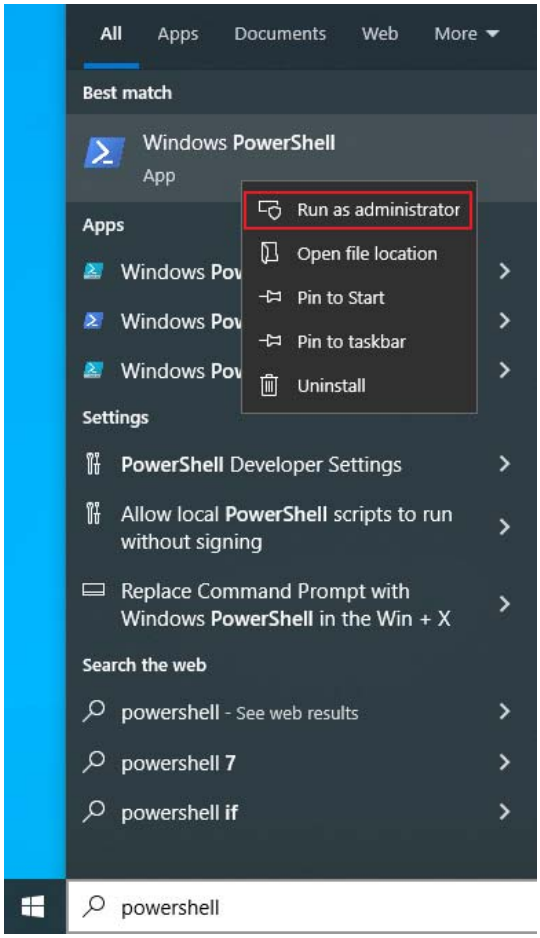
请注意，PowerShell 控制台的标题栏显示为“Windows PowerShell”，如图 1-4 所示。 一些命令运行正常，但 PowerShell 无法参与用户访问控制(UAC)。 这意味着它无法提示用户对需要管理员批准的任务进行提升。 生成以下错误消息：

```
Get-Service -Name W32Time | Stop-Service
# 输出
Stop-Service : Service 'Windows Time (W32Time)' cannot be stopped due to the following error: Cannot open W32Time
service on computer '.'.
At line:1 char:29
+ Get-Service -Name W32Time | Stop-Service
+                             ~~~~~~~~~~~~~
    + CategoryInfo          : CloseError: (System.ServiceProcess.ServiceController:ServiceController) [Stop-Service],
   ServiceCommandException
    + FullyQualifiedErrorId : CouldNotStopService,Microsoft.PowerShell.Commands.StopServiceCommand
```

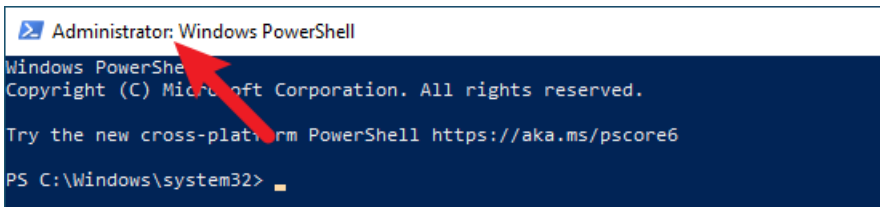
解决此问题的方法是以作为域用户的本地管理员身份运行 PowerShell。

关闭 PowerShell。 重启 PowerShell 控制台，但这次需要右键单击

Windows PowerShell 快捷方式，然后选择“以管理员身份运行”，如所示 。



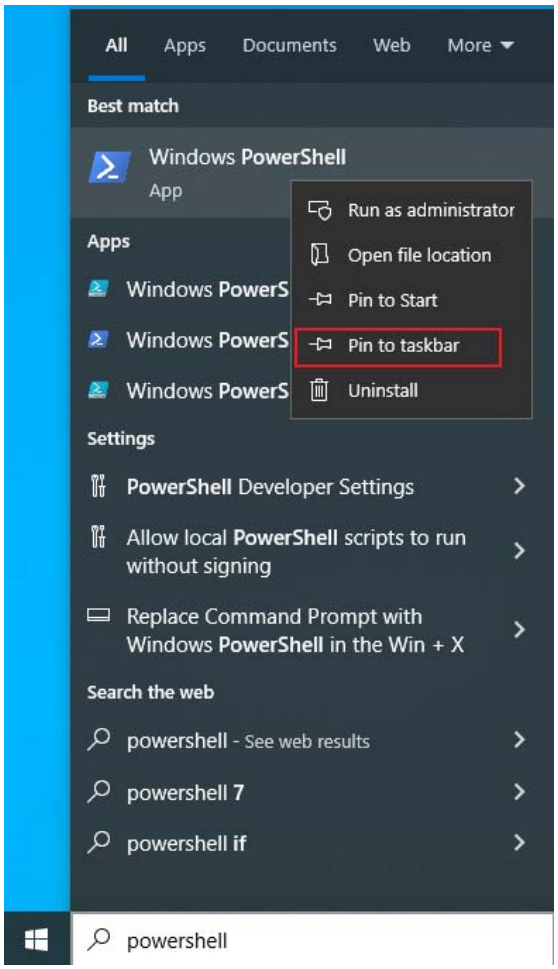
以管理员身份重启 PowerShell 后，标题栏应显示“Administrator: Windows PowerShell”，如所示。



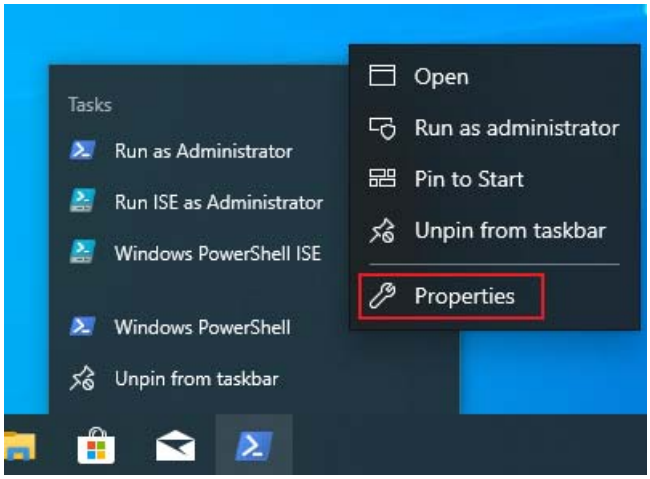
现在，由于以本地管理员身份运行了提升的 PowerShell，因此，当在本地计算机上运行某个命令且需要给予提升提示时，不会再产生 UAC 问题。 请记住，通过这个经过提升的 PowerShell 控制台实例运行的所有命令在运行时也会得到提升。

为了简化查找 PowerShell 并以管理员身份启动它的过程，建议将其固定在任务栏上，并将其设置为在每次运行时自动以管理员身份启动。

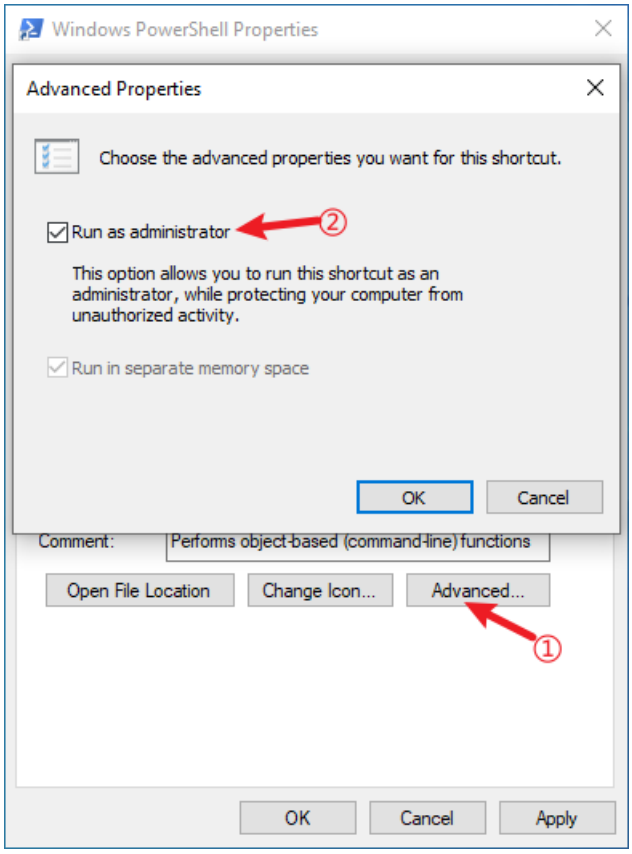
再次搜索 PowerShell，但这次要右键单击它，然后选择“固定到任务栏”，如图所示。



右键单击当前固定在任务栏上的 PowerShell 快捷方式，然后选择属性，如所示。



单击中的 ① 所表示的“高级”，然后选中图 1-10 中的 ② 所表示的“以管理员身份运行”复选框，然后双击“确定”，以接受更改并退出这两个对话框。



无需再次查找 PowerShell 或再次担心它是否以管理员身份运行。

为防止 UAC 出现问题而以管理员身份运行提升的 PowerShell 仅影响针对本地计算机运行的命令。 它不影响以远程计算机为目标的命令。

(二) 查看当前 PowerShell 的版本

PowerShell 中有许多用于存储状态信息的自动变量。 其中某个变量是 $PSVersionTable，它包含可用于显示相关 PowerShell 版本信息的哈希表：

```
$PSVersionTable


# 输出
Name                           Value
----                           -----
PSVersion                      5.1.19041.1
PSEdition                      Desktop
PSCompatibleVersions           {1.0, 2.0, 3.0, 4.0...}
```

```
BuildVersion                10.0.19041.1
CLRVersion                  4.0.30319.42000
WSManStackVersion           3.0
PSRemotingProtocolVersion   2.3
SerializationVersion        1.1.0.1
```

（三） PowerShell 执行策略

与通常的看法相反，PowerShell 中的执行策略不是安全边界。 它的作用是防止用户无意间运行脚本。 已确定的用户可以轻松绕过 PowerShell 中的执行策略。 表 1-2 显示当前 Windows 操作系统的默认执行策略。

Windows 操作系统版本	默认执行策略
Server 2019	RemoteSigned
Server 2016	RemoteSigned
Windows 10	Restricted

无论采用怎样的执行策略设置，任何 PowerShell 命令都可以通过交互方式运行。 执行策略仅影响脚本中运行的命令。 Get-ExecutionPolicy cmdlet 用于确定当前的执行策略设置，而 Set-ExecutionPolicy cmdlet 用于更改执行策略。 建议使用 RemoteSigned 策略，该策略要求下载的脚本必须由受信任的发布者签名才能运行。

检查当前的执行策略：

```
Get-ExecutionPolicy


# 输出
Restricted
```

当执行策略设置为“Restricted”时，PowerShell 脚本根本无法运行。 这是

所有 Windows 客户端操作系统上的默认设置。 为了演示该问题，将以下代码另存为名为 Stop-TimeService.ps1 的 .ps1 文件。

```
Get-Service -Name W32Time | Stop-Service -PassThru
```

只要以管理员身份运行提升的 PowerShell，该命令就可通过交互方式运行而不会出错。 不过，一旦将其保存为脚本文件并尝试执行该脚本，就会生成错误：

```
.\Stop-TimeService.ps1


# 输出
.\Stop-TimeService.ps1 : File C:\demo\Stop-TimeService.ps1 cannot be
loaded because
running scripts is disabled on this system. For more information, see
about_Execution_Policies at
http://go.microsoft.com/fwlink/?LinkID=135170.
At line:1 char:1
+ .\Stop-TimeService.ps1
+
    + CategoryInfo          : SecurityError: (:) [], PSSecurityException
    + FullyQualifiedErrorId : UnauthorizedAccess
```

请注意，上一组结果中显示的错误明确指示了发生了什么问题（在此系统上禁用了运行脚本）。 在 PowerShell 中运行命令后如果生成错误消息，请确保阅读该错误消息，而不是只重新运行该命令并希望它成功运行。

将 PowerShell 执行策略更改为远程签名。

```
Set-ExecutionPolicy -ExecutionPolicy RemoteSigned


# 输出
Execution Policy Change
The execution policy helps protect you from scripts that you do not trust.
```

```
Changing the execution
    policy might expose you to the security risks described in the
about_Execution_Policies help topic
    at http://go.microsoft.com/fwlink/?LinkID=135170. Do you want to change
the execution policy?
    [Y] Yes [A] Yes to All [N] No [L] No to All [S] Suspend [?] Help (default is
"N"):y
```

由于已将执行策略设置为 RemoteSigned，Stop-TimeService.ps1 脚本将正常运行。

```
.\Stop-TimeService.ps1


# 输出
Status     Name                    DisplayName
------     ----                    -----------
Stopped    W32Time                 Windows Time
```

在继续之前，请务必启动 Windows 时间服务，否则可能会遇到无法预料的问题。

```
Start-Service -Name w32time
```

八、 帮助系统

(一) 可发现性

PowerShell 中的编译命令称为 cmdlet。 Cmdlet 的发音为“command-let”（而不是 CMD-let）。 Cmdlet 名称采用单数形式的“动词-名词”命令形式，这样更易于被发现。例如，用于确定正在运行哪些进程的 cmdlet 是 Get-Process，而用于检索服务及其状态的列表的 cmdlet 是 Get-Service。 PowerShell 中还有

其他类型的命令（例如别名和函数），本书后面部分将对其进行介绍。 术语 PowerShell 命令是一个通用术语，通常用于指代 PowerShell 中任何类型的命令，不管是 cmdlet、函数还是别名。

（二） PowerShell 中的三个核心 Cmdlet

- Get-Command
- Get-Help
- Get-Member（在任务三中介绍此内容）

> NOTE
> 如何确定 PowerShell 中的命令是什么？
> 答：Get-Command 和 Get-Help 均可用于确定命令。

（三） Get-Help

Get-Help 是多用途命令。 Get-Help 帮助你了解找到命令后如何使用它们。Get-Help 也可用于帮助查找命令，但与 Get-Command 相比，它采用不同且较为间接的方式。

使用 Get-Help 查找命令时，它首先根据提供的输入来搜索命令名称的通配符匹配项。 如果找不到匹配项，它将搜索帮助主题本身；如果还找不到匹配项，则返回错误。 与通常的看法相反，Get-Help 可用于查找没有帮助主题的命令。

关于 PowerShell 中的帮助系统，首先需要了解如何使用 Get-Help cmdlet。以下命令用于显示 Get-Help 的帮助主题。

```
Get-Help -Name Get-Help


# 输出
Do you want to run Update-Help?
The Update-Help cmdlet downloads the most current Help files for
```

```
Windows PowerShell
    modules, and installs them on your computer. For more information about
the Update-Help
    cmdlet, see http://go.microsoft.com/fwlink/?LinkId=210614.
    [Y] Yes  [N] No  [S] Suspend  [?] Help (default is "Y"):
```

从 PowerShell 版本 3 开始，操作系统不随附 PowerShell 帮助功能。 第一次为命令运行 Get-Help 时，系统将显示上一条消息。 如果使用的是 help 函数或 man 别名（而不是 Get-Help cmdlet），则不会收到此提示。

若要回答“是”，请按 Y 以运行 Update-Help cmdlet，默认情况下该 cmdlet 需要 Internet 访问权限。 可以大写或小写形式指定 Y。

下载帮助并完成更新后，将为指定的命令返回帮助主题：

```
Get-Help -Name Get-Help
```

花几分钟在计算机上运行该示例，查看输出，并注意信息的分组方式：

- 名称
- 摘要
- SYNTAX
- DESCRIPTION
- 相关链接
- REMARKS

指定 Get-Help 的 Full 参数后，可返回整个帮助主题。

```
Get-Help -Name Get-Help -Full
```

花几分钟在计算机上运行该示例，查看输出，并注意信息的分组方式：

- 名称
- 摘要
- SYNTAX
- DESCRIPTION
- PARAMETERS
- 输入

- 输出
- 注释
- 示例
- 相关链接

> NOTE
>
> 使用 Full 参数返回了几个额外部分，其中一个部分是 PARAMETERS 部分，该部分提供的信息比晦涩的 SYNTAX 部分还要多。
>
> Full 参数是一个开关参数。 不需要值的参数称为开关参数。 如果指定了某个开关参数，其值为 true；否则为 false。

要使用 Get-Help 查找命令，请对 Name 参数使用星号 (*) 通配符。 指定搜索命令时使用的字词，作为 Name 参数的值。

示例：

```
get-help *hotfix*

# 输出
NAME
    Get-HotFix

SYNOPSIS
    Gets the hotfixes that have been applied to the local and remote
computers.



SYNTAX
    Get-HotFix [-ComputerName <String[]>] [-Credential <PSCredential>]
[-Description
    <String[]>] [<CommonParameters>]
```

Get-HotFix [[-Id] <String[]>] [-ComputerName <String[]>] [-Credential
<PSCredential>] [<CommonParameters>]

DESCRIPTION

The Get-Hotfix cmdlet gets hotfixes (also called updates) that have been installed
on either the local computer (or on specified remote computers) by Windows Update,
Microsoft Update, or Windows Server Update Services; the cmdlet also gets hotfixes
or updates that have been installed manually by users.

RELATED LINKS

Online Version: http://go.microsoft.com/fwlink/?LinkId=821586

Win32_QuickFixEngineering
http://go.microsoft.com/fwlink/?LinkID=145071

Get-ComputerRestorePoint

Add-Content

REMARKS

To see the examples, type: "get-help Get-HotFix -examples".

For more information, type: "get-help Get-HotFix -detailed".

For technical information, type: "get-help Get-HotFix -full".

For online help, type: "get-help Get-HotFix -online"

（四） Get-Command

Get-Command 的作用是帮助查找命令。 运行不带任何参数的 Get-Command 会返回系统上所有命令的列表。 以下示例演示使用 Get-Command cmdlet 确定存在的用于处理进程的命令：

```
Get-Command -Noun Process


# 输出
CommandType     Name                Version    Source
-----------     ----                -------    ------
Cmdlet          Debug-Process       3.1.0.0    Microsof...
Cmdlet          Get-Process         3.1.0.0    Microsof...
Cmdlet          Start-Process       3.1.0.0    Microsof...
Cmdlet          Stop-Process        3.1.0.0    Microsof...
Cmdlet          Wait-Process        3.1.0.0    Microsof...
```

请注意，在前面运行了 Get-Command 的示例中，使用了 Noun 参数，并且将 Process 指定为 Noun 参数的值 。 如果不知道如何使用 Get-Command cmdlet，该怎么办？ 可以使用 Get-Help 显示 Get-Command 的帮助主题。

Name、Noun 和 Verb 参数均接受通配符 。 下面的示例演示与 Name 参数一起使用的通配符：

```
Get-Command -Name *service*


# 输出
CommandType     Name                                Version    Source
-----------     ----                                -------    ---------
Function        Get-NetFirewallServiceFilter        2.0.0.0    NetSecurity
Function        Set-NetFirewallServiceFilter        2.0.0.0    NetSecurity
Cmdlet          Get-Service                         3.1.0.0    Microsof...
```

Cmdlet	New-Service	3.1.0.0	Microsof...
Cmdlet	New-WebServiceProxy	3.1.0.0	Microsof...
Cmdlet	Restart-Service	3.1.0.0	Microsof...
Cmdlet	Resume-Service	3.1.0.0	Microsof...
Cmdlet	Set-Service	3.1.0.0	Microsof...
Cmdlet	Start-Service	3.1.0.0	Microsof...
Cmdlet	Stop-Service	3.1.0.0	Microsof...
Cmdlet	Suspend-Service	3.1.0.0	Microsof...
Application	AgentService.exe	10.0.14...	C:\Windo...
Application	SensorDataService.exe	10.0.14...	C:\Windo...
Application	services.exe	10.0.14...	C:\Windo...
Application	services.msc	0.0.0.0	C:\Windo...
Application	TieringEngineService.exe	10.0.14...	C:\Windo...

我不喜欢将通配符与 Get-Command 的 Name 参数一起使用，因为它还返回不是本机 PowerShell 命令的可执行文件。

如果要在 Name 参数中使用通配符，建议使用 CommandType 参数来限定结果 。

```
Get-Command -Name *service* -CommandType Cmdlet, Function, Alias
```

更好的选择是使用 Verb 或/和 Noun 参数，因为只有 PowerShell 命令同时具有谓词和名词 。

九、 发现对象、属性和方法

（一） Get-Member

Get-Member 可帮助发现可用于命令的对象、属性和方法。 任何生成基于对象的输出的命令都可以通过管道传递到 Get-Member。 属性是有关某个项的特征。 驾驶证上有一个属性名为“眼睛颜色”，而该属性最常见的值是棕色。 方法是可以对某个项执行的操作。 以驾驶证为例，方法之一是“吊销”，因为机动车

管理部门可以吊销驾驶证。

(二) 属性

在下面的示例中，我将检索计算机上运行的 Windows 时间服务的相关信息。

```
Get-Service -Name w32time

# 输出
Status     Name               DisplayName
------     ----               -----------
Running    w32time            Windows Time
```

Status、Name 和 DisplayName 是上一组结果中显示的属性示例 。 Status 属性的值为 Running，Name 属性的值为 w32time，而 DisplayName 的值为 Windows Time 。

现在，我通过管道将同一命令传递给 Get-Member：

```
Get-Service -Name w32time | Get-Member

# 输出
TypeName: System.ServiceProcess.ServiceController

Name                      MemberType     Definition
----                      ----------     ----------
Name                      AliasProperty  Name = ServiceName
RequiredServices          AliasProperty  RequiredServices
=ServicesDependedOn
Disposed                  Event          System.EventHandler
Disposed(System.Object, Sy...
Close                     Method         void Close()
Continue                  Method         void Continue()
```

CreateObjRef Method System.Runtime.Remoting.ObjRef CreateObjRef(ty...

Dispose Method void Dispose(), void IDisposable.Dispose()

Equals Method bool Equals(System.Object obj)

ExecuteCommand Method void ExecuteCommand(int command)

GetHashCode Method int GetHashCode()

GetLifetimeService Method System.Object GetLifetimeService()

GetType Method type GetType()

InitializeLifetimeService Method System.Object InitializeLifetimeService()

Pause Method void Pause()

Refresh Method void Refresh()

Start Method void Start(), void Start(string[] args)

Stop Method void Stop()

WaitForStatus Method void WaitForStatus(System.ServiceProcess.Servi...

CanPauseAndContinue Property bool CanPauseAndContinue {get;}

CanShutdown Property bool CanShutdown {get;}

CanStop Property bool CanStop {get;}

Container Property System.ComponentModel.IContainer Container {get;}

DependentServices Property System.ServiceProcess.ServiceController[] Depe...

DisplayName Property string DisplayName {get;set;}

MachineName Property string MachineName {get;set;}

ServiceHandle Property System.Runtime.InteropServices.SafeHandle Serv...

```
ServiceName             Property        string ServiceName {get;set;}
ServicesDependedOn      Property
System.ServiceProcess.ServiceController[] Serv...
ServiceType             Property        System.ServiceProcess.ServiceType
ServiceType ...
Site                    Property        System.ComponentModel.ISite Site
{get;set;}
StartType               Property    System.ServiceProcess.ServiceStartMode
StartTy...
Status          Property    System.ServiceProcess.ServiceControllerStatus ...
ToString        ScriptMethod    System.Object ToString();
```

上一个示例中的结果的第一行包含一条非常重要的信息。 TypeName 指示返回的对象类型。 在本示例中，返回了 System.ServiceProcess.ServiceController 对象。 该对象通常缩写为 TypeName 最后一个句点之后的部分；在本示例中，即为 ServiceController 。

在了解了命令生成的对象类型之后，就可以使用此信息查找接受该类型的对象作为输入的命令。

```
Get-Command -ParameterType ServiceController


# 输出
CommandType     Name            Version     Source
-----------     ----            -------     ------
Cmdlet          Get-Service     3.1.0.0     Microsof...
Cmdlet          Restart-Service 3.1.0.0     Microsof...
Cmdlet          Resume-Service  3.1.0.0     Microsof...
Cmdlet          Set-Service     3.1.0.0     Microsof...
Cmdlet          Start-Service   3.1.0.0     Microsof...
Cmdlet          Stop-Service    3.1.0.0     Microsof...
```

```
Cmdlet          Suspend-Service    3.1.0.0       Microsof...
```

所有这些命令都有一个参数，该参数通过管道或/和参数输入接受 ServiceController 对象类型。

请注意，实际的属性数量比默认显示的多。 尽管默认情况下未显示全部属性，但可以通过将命令通过管道传递到 Select-Object cmdlet 并使用 Property 参数，从管道中选择未显示的属性。 以下示例通过将 Get-Service 的结果通过管道传递到 Select-Object 并将 * 通配符指定为 Property 参数的值来选择所有属性。

```
Get-Service -Name w32time | Select-Object -Property *


# 输出
Name                  : w32time
RequiredServices      : {}
CanPauseAndContinue : False
CanShutdown           : True
CanStop               : True
DisplayName           : Windows Time
DependentServices     : {}
MachineName           : .
ServiceName           : w32time
ServicesDependedOn    : {}
ServiceHandle         : SafeServiceHandle
Status                : Running
ServiceType           : Win32ShareProcess
StartType             : Manual
Site                  :
Container             :
```

也可以使用以逗号分隔的列表，选择特定的属性，作为 Property 参数的值。

```
Get-Service -Name w32time | Select-Object -Property Status, Name,
```

```
DisplayName, ServiceType


    # 输出
    Status       Name      DisplayName            ServiceType
     ------      ----       -----------            -----------
    Running    w32time    Windows Time      Win32ShareProcess
```

默认情况下，表中会返回四个属性，而列表中会返回五个及以上的属性。一些命令使用自定义格式来覆盖表中默认显示的属性数量。 有几个 Format-* cmdlet 可用于手动覆盖这些默认值。 下一章将介绍最常见的 Format-Table 和 Format-List 这两种命令。

使用 Select-Object 指定属性名称时可以使用通配符。

```
    Get-Service -Name w32time | Select-Object -Property Status, DisplayName,
Can*


    # 输出
    Status                  : Running
    DisplayName             : Windows Time
    CanPauseAndContinue : False
    CanShutdown             : True
    CanStop                 : True
```

在上一个示例中，Can* 被用作 Property 参数的某个值，用于返回所有以 Can 开头的属性。其中包括 CanPauseAndContinue、CanShutdown 和 CanStop 。

(三) 方法

方法是可执行的操作。 使用 MemberType 参数来缩小 Get-Member 的结果范围，使其仅显示 Get-Service 的方法。

```
    Get-Service -Name w32time | Get-Member -MemberType Method


```

```
# 输出
TypeName: System.ServiceProcess.ServiceController


Name                 MemberType          Definition
----                 ----------          ----------
Close                Method        void Close()
Continue             Method        void Continue()
CreateObjRef         Method        System.Runtime.Remoting.ObjRef
CreateObjRef(type ...
Dispose              Method        void Dispose(), void
IDisposable.Dispose()
Equals               Method        bool Equals(System.Object obj)
ExecuteCommand       Method        void ExecuteCommand(int command)
GetHashCode          Method        int GetHashCode()
GetLifetimeService   Method        System.Object GetLifetimeService()
GetType              Method        type GetType()
InitializeLifetimeService  Method  System.Object
InitializeLifetimeService()
Pause                Method        void Pause()
Refresh              Method        void Refresh()
Start                Method        void Start(), void Start(string[] args)
Stop                 Method        void Stop()
WaitForStatus        Method        void
WaitForStatus(System.ServiceProcess.ServiceC...
```

正如所见，存在多种方法。 Stop 方法可用于停止 Windows 服务。

```
(Get-Service -Name w32time).Stop()
```

现在可验证 Windows 时间服务确实已停止。

```
Get-Service -Name w32time
```

```
# 输出
Status     Name               DisplayName
------         ----               -----------
Stopped    w32time            Windows Time
```

需要了解方法的另一个原因是，许多初学者认为无法使用 Get-* 命令进行颠覆性更改。 但是，如果方法使用不当，也会带来严重后果。

更好的选择是使用 cmdlet 执行该操作（如果存在）。 继续操作，并启动 Windows 时间服务，但这次使用 cmdlet 启动服务。

```
Get-Service -Name w32time | Start-Service -PassThru

# 输出
Status     Name               DisplayName
------         ----               -----------
Running    w32time            Windows Time
```

默认情况下，Start-Service 不会像 Get-Service 的启动方法那样返回任何结果。但使用 cmdlet 的好处之一是，很多时候 cmdlet 提供了方法无法提供的其他功能。 上一个示例中使用了 PassThru 参数。 这会导致 cmdlet 生成输出，而它通常不生成输出。

如果命令没有生成输出，则无法通过管道将其传递到 Get-Member。 由于 Start-Service 默认不生成任何输出，在尝试通过管道将其传递到 Get-Member 时会生成错误。

```
Start-Service -Name w32time | Get-Member

# 输出
Get-Member : You must specify an object for the Get-Member cmdlet.
At line:1 char:31
+ Start-Service -Name w32time | Get-Member
```

```
    +
        + CategoryInfo          : CloseError: (:) [Get-Member],
InvalidOperationException
        + FullyQualifiedErrorId :
NoObjectInGetMember,Microsoft.PowerShell.Commands.GetMembe
    rCommand
```

可以使用 Start-Service cmdlet 指定 PassThru 参数，以使其生成输出，然后通过管道将输出传递到 Get-Member，这样不会出错。

```
    Start-Service -Name w32time -PassThru | Get-Member

    # 输出
    TypeName: System.ServiceProcess.ServiceController

    Name                MemberType      Definition
    ----                ----------      ----------
    Name                AliasProperty Name = ServiceName
    RequiredServices    AliasProperty RequiredServices =
ServicesDependedOn
    Disposed            Event           System.EventHandler
Disposed(System.Object, Sy...
    Close               Method          void Close()
    Continue            Method          void Continue()
    CreateObjRef        Method          System.Runtime.Remoting.ObjRef
CreateObjRef(ty...
    Dispose             Method          void Dispose(), void
IDisposable.Dispose()
    Equals              Method          bool Equals(System.Object obj)
    ExecuteCommand      Method          void ExecuteCommand(int
command)
```

GetHashCode Method int GetHashCode()

GetLifetimeService Method System.Object GetLifetimeService()

GetType Method type GetType()

InitializeLifetimeService Method System.Object InitializeLifetimeService()

Pause Method void Pause()

Refresh Method void Refresh()

Start Method void Start(), void Start(string[] args)

Stop Method void Stop()

WaitForStatus Method void WaitForStatus(System.ServiceProcess.Servi...

CanPauseAndContinue Property bool CanPauseAndContinue {get;}

CanShutdown Property bool CanShutdown {get;}

CanStop Property bool CanStop {get;}

Container Property System.ComponentModel.IContainer Container {get;}

DependentServices Property System.ServiceProcess.ServiceController[] Depe...

DisplayName Property string DisplayName {get;set;}

MachineName Property string MachineName {get;set;}

ServiceHandle Property System.Runtime.InteropServices.SafeHandle Serv...

ServiceName Property string ServiceName {get;set;}

ServicesDependedOn Property System.ServiceProcess.ServiceController[] Serv...

ServiceType Property System.ServiceProcess.ServiceType ServiceType ...

```
    Site                    Property        System.ComponentModel.ISite
Site {get;set;}
    StartType               Property
System.ServiceProcess.ServiceStartMode StartTy...
    Status                  Property
System.ServiceProcess.ServiceControllerStatus ...
    ToString                ScriptMethod    System.Object ToString();
```

若要通过管道将命令传递到 Get-Member，命令必须生成基于对象的输出。

```
    Get-Service -Name w32time | Out-Host | Get-Member


    # 输出
    Status     Name                DisplayName
    ------     ----                -----------
    Running    w32time             Windows Time


    Get-Member : You must specify an object for the Get-Member cmdlet.
    At line:1 char:40
    + Get-Service -Name w32time | Out-Host | Get-Member
    +
        + CategoryInfo          : CloseError: (:) [Get-Member],
InvalidOperationException
        + FullyQualifiedErrorId :
NoObjectInGetMember,Microsoft.PowerShell.Commands.GetMemberCommand
```

Out-Host 直接写入 PowerShell 主机，但它不会为管道生成基于对象的输出。因此无法通过管道将该命令传输到 Get-Member。

十、 单行命令和管道

(一) 单行命令

PowerShell 单行是一个连续管道，不一定是位于一个物理行上的命令。 并非一个物理行上的所有命令都是单行命令。

即使以下命令位于多个物理行上，它也是 PowerShell 单行命令，因为它是一个连续管道。 它可以写在一个物理行上，但我已选择在管道符号处换行。 管道符号是 PowerShell 中允许自然换行处的某个字符。

```
Get-Service |
   Where-Object CanPauseAndContinue -eq $true |
      Select-Object -Property *


# 输出
Name                  : LanmanWorkstation
RequiredServices      : {NSI, MRxSmb20, Bowser}
CanPauseAndContinue : True
CanShutdown           : False
CanStop               : True
DisplayName           : Workstation
DependentServices     : {SessionEnv, Netlogon, Browser}
MachineName           : .
ServiceName           : LanmanWorkstation
ServicesDependedOn    : {NSI, MRxSmb20, Bowser}
ServiceHandle         : SafeServiceHandle
Status                : Running
ServiceType           : Win32ShareProcess
StartType             : Automatic
Site                  :
Container             :
```

```
Name                           : Netlogon
RequiredServices       : {LanmanWorkstation}
CanPauseAndContinue : True
CanShutdown              : False
CanStop                     : True
DisplayName              : Netlogon
DependentServices     : {}
MachineName              : .
ServiceName              : Netlogon
ServicesDependedOn    : {LanmanWorkstation}
ServiceHandle           : SafeServiceHandle
Status                       : Running
ServiceType              : Win32ShareProcess
StartType                  : Automatic
Site                           :
Container                   :


Name                           : vmicheartbeat
RequiredServices       : {}
CanPauseAndContinue : True
CanShutdown              : False
CanStop                     : True
DisplayName              : Hyper-V Heartbeat Service
DependentServices     : {}
MachineName              : .
ServiceName              : vmicheartbeat
ServicesDependedOn    : {}
ServiceHandle           : SafeServiceHandle
Status                       : Running
```

```
ServiceType              : Win32ShareProcess
StartType                : Manual
Site                     :
Container                :

Name                     : vmickvpexchange
RequiredServices         : {}
CanPauseAndContinue      : True
CanShutdown              : False
CanStop                  : True
DisplayName              : Hyper-V Data Exchange Service
DependentServices        : {}
MachineName              : .
ServiceName              : vmickvpexchange
ServicesDependedOn       : {}
ServiceHandle            : SafeServiceHandle
Status                   : Running
ServiceType              : Win32ShareProcess
StartType                : Manual
Site                     :
Container                :

Name                     : vmicrdv
RequiredServices         : {}
CanPauseAndContinue      : True
CanShutdown              : False
CanStop                  : True
DisplayName              : Hyper-V Remote Desktop Virtualization Service
DependentServices        : {}
MachineName              : .
```

```
ServiceName               : vmicrdv
ServicesDependedOn    : {}
ServiceHandle             : SafeServiceHandle
Status                    : Running
ServiceType               : Win32ShareProcess
StartType                 : Manual
Site                      :
Container                 :

Name                      : vmicshutdown
RequiredServices      : {}
CanPauseAndContinue : True
CanShutdown               : False
CanStop                   : True
DisplayName               : Hyper-V Guest Shutdown Service
DependentServices     : {}
MachineName               : .
ServiceName               : vmicshutdown
ServicesDependedOn    : {}
ServiceHandle             : SafeServiceHandle
Status                    : Running
ServiceType               : Win32ShareProcess
StartType                 : Manual
Site                      :
Container                 :

Name                      : vmictimesync
RequiredServices      : {VmGid}
CanPauseAndContinue : True
CanShutdown               : False
```

```
CanStop                 : True
DisplayName             : Hyper-V Time Synchronization Service
DependentServices       : {}
MachineName             : .
ServiceName             : vmictimesync
ServicesDependedOn      : {VmGid}
ServiceHandle           : SafeServiceHandle
Status                  : Running
ServiceType             : Win32ShareProcess
StartType               : Manual
Site                    :
Container               :

Name                    : vmicvss
RequiredServices        : {}
CanPauseAndContinue     : True
CanShutdown             : False
CanStop                 : True
DisplayName             : Hyper-V Volume Shadow Copy Requestor
DependentServices       : {}
MachineName             : .
ServiceName             : vmicvss
ServicesDependedOn      : {}
ServiceHandle           : SafeServiceHandle
Status                  : Running
ServiceType             : Win32ShareProcess
StartType               : Manual
Site                    :
Container               :
```

```
Name                          : Winmgmt
RequiredServices       : {RPCSS}
CanPauseAndContinue : True
CanShutdown               : True
CanStop                      : True
DisplayName               : Windows Management Instrumentation
DependentServices     : {wscsvc, NcaSvc, iphlpsvc}
MachineName               : .
ServiceName               : Winmgmt
ServicesDependedOn    : {RPCSS}
ServiceHandle             : SafeServiceHandle
Status                        : Running
ServiceType               : Win32ShareProcess
StartType                   : Automatic
Site                           :
Container                   :
```

自然换行可以出现在常用字符处，包括逗号 (,) 和方左括号 ([)、大括号 ({) 和圆括号 (()。 其他不太常见的字符包括分号 (;)、等于号 (=) 以及左单引号和双引号（'、"）。

将反引号 (`) 或重音符用作续行符是一个有争议的话题。 建议尽量避免这样做。 我经常会看到使用反引号编写的 PowerShell 命令紧跟在自然换行符之后。 没有理由把它放在那里。

```
Get-Service -Name w32time |
>> Select-Object -Property *


# 输出
Name                          : w32time
RequiredServices       : {}
CanPauseAndContinue : False
```

```
CanShutdown              : True
CanStop                  : True
DisplayName              : Windows Time
DependentServices        : {}
MachineName              : .
ServiceName              : w32time
ServicesDependedOn       : {}
ServiceHandle            : SafeServiceHandle
Status                   : Running
ServiceType              : Win32ShareProcess
StartType                : Manual
Site                     :
Container                :
```

上面两个示例中所示的命令在 PowerShell 控制台中正常工作。 但如果尝试在 PowerShell ISE 的控制台窗格中运行它们，则会出现错误。 PowerShell ISE 的控制台窗格不会等待命令的其余部分在下一行（如 PowerShell 控制台）中输入。 要避免 PowerShell ISE 的控制台窗格中出现此问题，请使用 Shift+Enter，而不是只是在继续执行另一行上的命令时按 Enter。

下一个示例不是 PowerShell 单行命令，因为它不是一个连续管道。 它是一行上的两个单独命令，用分号分隔。

```
$Service = 'w32time'; Get-Service -Name $Service


# 输出
Status     Name               DisplayName
------     ----               -----------
Running    w32time            Windows Time
```

许多编程和脚本语言要求在每一行的末尾使用分号。 尽管可以在 PowerShell 中以这种方法使用它们，但不建议这样做，因为它们不是必需的。

(二) 筛选左侧

本章中所示的命令结果已筛选为一个子集。 例如，Get-Service 与 Name 参数一起使用，筛选返回的服务列表，以仅显示 Windows 时间服务。

在管道中，你始终希望尽快将结果筛选为你要查找的内容。 可使用第一个命令或最左侧命令的参数来完成此操作。 这有时称为“筛选左侧”。

下面的示例使用 Get-Service 的 Name 参数来立即筛选结果，以仅显示 Windows 时间服务。

```
Get-Service -Name w32time


# 输出
Status     Name               DisplayName
------         ----               -----------
Running    w32time            Windows Time
```

通过管道将命令传递到 Where-Object 来执行筛选的例子很常见。

```
Get-Service | Where-Object Name -eq w32time


# 输出
Status     Name               DisplayName
------         ----               -----------
Running    W32Time             Windows Time
```

第一个示例筛选源，只返回 Windows 时间服务的结果。 第二个示例返回所有服务，然后通过管道将它们传递到另一个命令来执行筛选。 虽然在本例中这似乎不是什么大问题，但是想象一下，如果你正在查询 Active Directory 用户列表。 你真的想要从 Active Directory 返回数千个用户帐户的信息，而只是通过管道将它们传递到另一个命令，从而筛选出一个很小的子集吗？ 我的建议是始终筛选左侧，即使它似乎并不重要。 你会非常习惯它，并在真正需要时自动筛选

左侧。

执行筛选时，指定命令的顺序确实很重要。 例如，假设你使用 Select-Object 仅选择几个属性，并使用 Where-Object 筛选不在选择内容中的属性。 在这种情况下，必须首先进行筛选，否则在尝试执行筛选时，该属性将不存在于管道中。

```
Get-Service |
Select-Object -Property DisplayName, Running, Status |
Where-Object CanPauseAndContinue
```

上一示例中的命令不会返回任何结果，因为当 Select-Object 的结果通过管道传递到 Where-Object 时，CanStopAndContinue 属性不存在。 该特定属性未被“选定”。 从本质上讲，它已被筛除了。反转 Select-Object 和 Where-Object 的顺序将产生所需结果。

```
Get-Service |
Where-Object CanPauseAndContinue |
Select-Object -Property DisplayName, Status


# 输出
DisplayName                                 Status
-----------                                 ------
Workstation                                 Running
Netlogon                                    Running
Hyper-V Heartbeat Service                   Running
Hyper-V Data Exchange Service               Running
Hyper-V Remote Desktop Virtualization Service Running
Hyper-V Guest Shutdown Service              Running
Hyper-V Time Synchronization Service        Running
Hyper-V Volume Shadow Copy Requestor        Running
Windows Management Instrumentation          Running
```

（三） 管道

很多时候，一个命令的输出可以用作另一个命令的输入。 在任务三中，Get-Member 用于确定命令所生成的对象类型。 任务三还介绍了如何使用 Get-Command 的 ParameterType 参数来确定哪些命令接受该类型的输入，尽管不一定是通过管道输入。

根据命令提供帮助的程度，它可能包含 INPUTS 和 OUTPUTS 部分 。

```
help Stop-Service -Full


# 输出
...
INPUTS
    System.ServiceProcess.ServiceController, System.String
        You can pipe a service object or a string that contains the name of
a service
        to this cmdlet.


OUTPUTS
    None, System.ServiceProcess.ServiceController
        This cmdlet generates a System.ServiceProcess.ServiceController
object that
        represents the service, if you use the PassThru parameter.
Otherwise, this
        cmdlet does not generate any output.
...
```

之前的结果中只显示了帮助的相关部分。 正如你所看到的，INPUTS 部分表明，ServiceController 或 String 对象可以通过管道传递到 Stop-Service cmdlet 。它不会告诉你哪些参数接受该类型的输入。 确定该信息的最简单方法之一是查看完整版本的 Stop-Service cmdlet 帮助中的不同参数。

```
help Stop-Service -Full

# 输出
...
-DisplayName <String[]>
    Specifies the display names of the services to stop. Wildcard characters
are
    permitted.

    Required?                    true
    Position?                    named
    Default value                None
    Accept pipeline input?       False
    Accept wildcard characters?  false

-InputObject <ServiceController[]>
    Specifies ServiceController objects that represent the services to stop.
Enter a
    variable that contains the objects, or type a command or expression that
gets the
    objects.

    Required?                    true
    Position?                    0
    Default value                None
    Accept pipeline input?       True (ByValue)
    Accept wildcard characters?  false

-Name <String[]>
```

```
        Specifies the service names of the services to stop. Wildcard characters
are
        permitted.

        Required?                    true
        Position?                    0
        Default value                None
        Accept pipeline input?       True (ByPropertyName, ByValue)
        Accept wildcard characters?  false
    ...
```

同样，我仅在以前的结果集中显示了帮助的相关部分。请注意，DisplayName 参数不接受管道输入，InputObject 参数按 ServiceController 对象的值接受管道输入，而 Name 参数按 String 对象的值接受管道输入 。 它还按属性名称接受管道输入。

当参数按属性名称和按值接受管道输入时，将始终首先尝试按值接受。 如果按值接受管道输入失败，则会尝试按属性名称接受管道输入 。 按值接受管道输入会产生误导。 我更喜欢按类型调用它。 这意味着，如果通过管道将生成 ServiceController 对象类型的命令结果传递到 Stop-Service，则会将该输入绑定到 InputObject 参数 。 但如果通过管道将生成 String 输出的命令结果传递到 Stop-Service，则将其绑定到 Name 参数 。 如果通过管道将不生成 ServiceController 或 String 对象的命令结果传递到 Stop-Service，但该命令确实生成了包含名为 Name 的属性的输出，则它会将输出的 Name 属性绑定到 Stop-Service 的 Name 参数 。

确定 Get-Service 命令生成的输出类型。

```
Get-Service -Name w32time | Get-Member

# 输出
```

```
TypeName: System.ServiceProcess.ServiceController
```

Get-Service 生成 ServiceController 对象类型。

正如你之前在帮助中看到的那样，Stop-Service 的 InputObject 参数通过管道按值（按类型）接受 ServiceController 对象 。 这意味着，当 Get-Service cmdlet 的结果通过管道传递到 Stop-Service 时，这些结果将绑定到 Stop-Service 的 InputObject 参数。

```
Get-Service -Name w32time | Stop-Service
```

现在，尝试字符串输入。 通过管道将 w32time 传递到 Get-Member，以确认它是一个字符串。

```
'w32time' | Get-Member


# 输出
TypeName: System.String
```

如前面的帮助中所示，通过管道将字符串传递到 Stop-Service 会将其按值绑定到 Stop-Service 的 Name 参数 。 通过管道将 w32time 传递到 Stop-Service，以进行测试。

```
'w32time' | Stop-Service
```

请注意，在上面的示例中，我在字符串 w32time 两侧使用了单引号。 在 PowerShell 中，应始终使用单引号而不是双引号，除非带引号的字符串的内容包含需要扩展为其实际值的变量。 通过使用单引号，PowerShell 不必分析引号中包含的内容，因此可稍微加快代码运行速度。

按 Stop-Service 的 Name 参数的属性名称创建自定义对象，以测试管道输入。

```
$CustomObject = [pscustomobject]@{
 Name = 'w32time'
 }
```

CustomObject 变量的内容是 PSCustomObject 对象类型，并且它包含名为

Name 的属性 。

```
$CustomObject | Get-Member


# 输出
TypeName: System.Management.Automation.PSCustomObject


Name              MemberType      Definition
----              ----------      ----------
Equals            Method            bool Equals(System.Object obj)
GetHashCode Method             int GetHashCode()
GetType         Method             type GetType()
ToString        Method            string ToString()
Name              NoteProperty string Name=w32time
```

如果要在 $CustomObject 变量两边使用引号，则需要使用双引号。 否则，如果使用单引号，则会将文本字符串 $CustomObject 通过管道传递到 Get-Member，而不是传递变量包含的值。

尽管将 $CustomObject 的内容通过管道传递到 Stop-Service cmdlet 会将该内容绑定到 Name 参数，但这次它会按属性名称绑定，而不是按值，因为 $CustomObject 的内容是一个具有名为 Name 的属性的对象 。

在此示例中，我使用其他属性名称（如 Service）创建了另一个自定义对象。

```
$CustomObject = [pscustomobject]@{
   Service = 'w32time'
}
```

尝试通过管道将 $CustomObject 传递到 Stop-Service 时，会产生错误，因为它不会生成 ServiceController 或 String 对象，并且没有名为 Name 的属性 。

```
$CustomObject | Stop-Service
```

```
    # 输出
    Stop-Service : Cannot find any service with service name
'@{Service=w32time}'.
    At line:1 char:17
    + $CustomObject | Stop-Service
    +
        + CategoryInfo                  : ObjectNotFound:
(@{Service=w32time}:String) [Stop-Service]
       , ServiceCommandException
        + FullyQualifiedErrorId :
NoServiceFoundForGivenName,Microsoft.PowerShell.Commands.S
       topServiceCommand
```

如果一个命令的输出与另一个命令的管道输入选项不相符，则可以使用 Select-Object 重命名属性，以便正确地配置属性。

```
    $CustomObject |
      Select-Object -Property @{name='Name';expression={$_.Service}} |
        Stop-Service
```

在此示例中，使用 Select-Object 将 Service 属性重命名为名为 Name 的属性 。

有时，你可能需要使用不接受管道输入的参数。 下面的示例演示如何使用一个命令的输出作为另一个命令的输入。 首先将几个 Windows 服务的显示名称保存到一个文本文件中。

```
    'Background Intelligent Transfer Service', 'Windows Time' |
    Out-File -FilePath $env:TEMP\services.txt
```

可以运行命令，在括号中提供所需输出，作为需要输入的命令的参数值。

```
    Stop-Service -DisplayName (Get-Content -Path $env:TEMP\services.txt)
```

十一、 格式设置、别名、提供程序、比较

(一) 右对齐格式

最常见的格式命令是 Format-Table 和 Format-List。 还可以使用 Format-Wide 和 Format-Custom，但不太常见。

如任务三中所述，除非使用自定义格式设置，否则返回超过四个属性的命令默认为列表。

```
Get-Service -Name w32time | Select-Object -Property Status, DisplayName, Can*

# 输出
Status                    : Running
DisplayName               : Windows Time
CanPauseAndContinue : False
CanShutdown               : True
CanStop                   : True
```

使用 Format-Table cmdlet 手动替代格式设置，并在表中而不是在列表中显示输出。

```
Get-Service -Name w32time | Select-Object -Property Status, DisplayName, Can* |
Format-Table

# 输出
Status DisplayName    CanPauseAndContinue CanShutdown CanStop
------ -----------    ------------------- ----------- -------
Running Windows Time                 False        True    True
```

Get-Service 的默认输出是表中的三个属性。

```
Get-Service -Name w32time


# 输出
Status     Name                 DisplayName
------     ----                 -----------
Running    w32time              Windows Time
```

使用 Format-List cmdlet 替代默认格式设置，并在列表中返回结果。

```
Get-Service -Name w32time | Format-List


# 输出
Name                  : w32time
DisplayName           : Windows Time
Status                : Running
DependentServices     : {}
ServicesDependedOn    : {}
CanPauseAndContinue   : False
CanShutdown           : True
CanStop               : True
ServiceType           : Win32ShareProcess
```

请注意，仅通过管道将 Get-Service 传送到 Format-List 会使其返回其他属性。并非每个命令都会出现这种情况，因为特定命令的格式是在后台设置的。

使用 cmdlet 格式时，首先需要注意的是，它们生成的格式对象与 PowerShell 中的普通对象不同。

```
Get-Service -Name w32time | Format-List | Get-Member


# 输出
TypeName:
Microsoft.PowerShell.Commands.Internal.Format.FormatStartData
```

Name MemberType Definition

---- ---------- ----------

Equals Method bool Equals(System.Object obj)

GetHashCode Method int GetHashCode()

GetType Method type GetType()

ToString Method string ToString()

autosizeInfo Property Microsoft.PowerShell.Commands.Inter...

ClassId2e4f51ef21dd47e99d3c952918aff9cd Property string ClassId2e4f51ef21dd47e99d3c9...

groupingEntry Property Microsoft.PowerShell.Commands.Inter...

pageFooterEntry Property Microsoft.PowerShell.Commands.Inter...

pageHeaderEntry Property Microsoft.PowerShell.Commands.Inter...

shapeInfo Property Microsoft.PowerShell.Commands.Inter...

TypeName: Microsoft.PowerShell.Commands.Internal.Format.GroupStartData

Name MemberType Definition

---- ---------- ----------

Equals Method bool Equals(System.Object obj)

GetHashCode Method int GetHashCode()

GetType Method type GetType()

ToString Method string ToString()

ClassId2e4f51ef21dd47e99d3c952918aff9cd Property string ClassId2e4f51ef21dd47e99d3c9...

groupingEntry Property Microsoft.PowerShell.Commands.Inter...

shapeInfo Property Microsoft.PowerShell.Commands.Inter...

TypeName: Microsoft.PowerShell.Commands.Internal.Format.FormatEntryData

```
    Name                MemberType Definition
    ----                ---------- ----------
    Equals              Method     bool Equals(System.Object obj)
    GetHashCode         Method     int GetHashCode()
    GetType             Method     type GetType()
    ToString            Method     string ToString()
    ClassId2e4f51ef21dd47e99d3c952918aff9cd Property   string
ClassId2e4f51ef21dd47e99d3c9...
    formatEntryInfo     Property   Microsoft.PowerShell.Commands.Inter...
    outOfBand           Property   bool outOfBand {get;set;}
    writeStream         Property   Microsoft.PowerShell.Commands.Inter...


        TypeName:
Microsoft.PowerShell.Commands.Internal.Format.GroupEndData

    Name                MemberType Definition
    ----                ---------- ----------
    Equals              Method     bool Equals(System.Object obj)
    GetHashCode         Method     int GetHashCode()
    GetType             Method     type GetType()
    ToString            Method     string ToString()
    ClassId2e4f51ef21dd47e99d3c952918aff9cd Property   string
ClassId2e4f51ef21dd47e99d3c9...
    groupingEntry       Property   Microsoft.PowerShell.Commands.Inter...


        TypeName:
Microsoft.PowerShell.Commands.Internal.Format.FormatEndData

    Name                MemberType Definition
```

```
----                    ---------- ----------
Equals                  Method     bool Equals(System.Object obj)
GetHashCode             Method     int GetHashCode()
GetType                 Method     type GetType()
ToString                Method     string ToString()
ClassId2e4f51ef21dd47e99d3c952918aff9cd Property   string
ClassId2e4f51ef21dd47e99d3c9...
groupingEntry           Property   Microsoft.PowerShell.Commands.Inter...
```

这意味着无法通过管道将格式命令传送给大多数其他命令。 可以将格式命令通过管道传送给一些 Out-* 命令，但仅此而已。 因此，建议在行尾执行任何格式设置（右对齐格式）。

(二) 别名

PowerShell 的别名是命令的较短名称。 PowerShell 包含一组内置别名，你也可以定义自己的别名。

Get-Alias cmdlet 用于查找别名。 如果你已知道命令的别名，则 Name 参数用于确定与该别名关联的命令。

```
Get-Alias -Name gcm

# 输出
CommandType     Name                    Version    Source
-----------     ----                    -------    ------
Alias           gcm -> Get-Command
```

可以为 Name 参数的值指定多个别名。

```
Get-Alias -Name gcm, gm

# 输出
CommandType     Name                    Version    Source
```

```
-----------          ----                    -------     ------
Alias                gcm -> Get-Command
Alias                gm -> Get-Member
```

你会经常看到 Name 参数被忽略，因为它是位置参数。

```
Get-Alias gm

# 输出
CommandType     Name                    Version     Source
-----------     ----                    -------     ------
Alias           gm -> Get-Member
```

如果要查找命令的别名，需要使用 Definition 参数。

```
Get-Alias -Definition Get-Command, Get-Member

# 输出
CommandType     Name                    Version     Source
-----------     ----                    -------     ------
Alias           gcm -> Get-Command
Alias           gm -> Get-Member
```

不能按位置使用 Definition 参数，因此必须指定它。

使用别名可以减少按键次数，并且在将命令输入到控制台时也可行。 它们不应在脚本或任何要保存或与他人共享的代码中使用。 如本书前面所述，使用完整的 cmdlet 和参数名称是自文档化，并且更易于理解。

创建自己的别名时要小心，因为它们将仅存在于计算机上的当前 PowerShell 会话中。

(三) 提供程序

PowerShell 中的提供程序是一种允许文件系统访问数据存储的接口。

PowerShell 中提供了许多内置提供程序。

```
Get-PSProvider


# 输出
Name             Capabilities                       Drives
----             ------------                       ------
Registry         ShouldProcess, Transactions        {HKLM, HKCU}
Alias            ShouldProcess                      {Alias}
Environment      ShouldProcess                      {Env}
FileSystem       Filter, ShouldProcess, Credentials {C, A, D}
Function         ShouldProcess                      {Function}
Variable         ShouldProcess                      {Variable}
Certificate      ShouldProcess                      {Cert}
WSMan            Credentials                        {WSMan}
```

如上面的结果所示，存在用于注册表、别名、环境变量、文件系统、函数、变量、证书和 WSMan 的内置提供程序。

这些提供程序用于公开其数据存储的实际驱动器可使用 Get-PSDrive cmdlet 确定。 Get-PSDrive cmdlet 不仅显示由提供程序公开的驱动器，而且还显示 Windows 逻辑驱动器，其中包括映射到网络共享的驱动器。

```
Get-PSDrive


# 输出
Name       Used (GB)     Free (GB)  Provider       Root
----       ---------     ---------  --------       ----
A                                   FileSystem     A:\
Alias                               Alias
C              14.41         112.10 FileSystem     C:\
Cert                                Certificate    \
D                                   FileSystem     D:\
```

```
Env                                 Environment
Function                            Function
HKCU                                Registry    HKEY_CURRENT_USER
HKLM                                Registry    HKEY_LOCAL_MACHINE
Variable                            Variable
WSMan                               WSMan
```

可以像访问传统文件系统一样访问 PSDrive。

```
Get-ChildItem -Path Cert:\LocalMachine\CA


# 输出
PSParentPath: Microsoft.PowerShell.Security\Certificate::LocalMachine\CA


Thumbprint                                Subject
----------                                -------
FEE449EE0E3965A5246F000E87FDE2A065FD89D4    CN=Root Agency
D559A586669B08F46A30A133F8A9ED3D038E2EA8
OU=www.verisign.com/CPS Incorporated LIABI...
109F1CAED645BB78B3EA2B94C0697C740733031C    CN=Microsoft
Windows Hardware Compatibility,...
```

（四） 比较运算符

PowerShell 包含许多比较运算符，用于比较值或查找与特定模式匹配的值。表 5-1 包含 PowerShell 中的比较运算符的列表。

操作	定义
-eq	等于

-ne	不等于
-gt	大于
-ge	大于或等于
-lt	小于
-le	小于或等于
-Like	使用 * 通配符进行匹配
-NotLike	不适用 * 通配符进行匹配
-Match	匹配指定的正则表达式
-NotMatch	不匹配指定的正则表达式
-Contains	确定集合中是否包含指定的值
-NotContains	确定集合是否不包含特定值
-In	确定指定的值是否在集合中
-NotIn	确定指定的值是否不在集合中
-Replace	替换指定的值

表 5-1 中列出的所有运算符都不区分大小写。 将 c 放置在表 5-1 中列出的运算符之前，使其区分大小写。 例如，-ceq 是区分大小写的 -eq 比较运算符。

首字母大写的“PowerShell”等效于使用等于比较运算符的小写的“powershell”。

```
'PowerShell' -eq 'powershell'


# 输出
True
```

使用区分大小写的等于比较运算符时，不等效。

```
'PowerShell' -ceq 'powershell'
```

```
# 输出
False
```

不等于比较运算符反转条件。

```
'PowerShell' -ne 'powershell'

# 输出
False
```

大于、大于或等于、小于和小于或等于均可用于字符串或数值。

```
5 -gt 5

# 输出
False
```

在上一个示例中，使用大于或等于而不是大于，会返回布尔值 true，因为 5 等于 5。

```
5 -ge 5

# 输出
True
```

根据上述两个示例中的结果，你可能会猜到使用小于和小于或等于的情况。

```
5 -lt 10

# 输出
True
```

即使对于经验丰富的 PowerShell 用户，-Like 和 -Match 运算符也可能会造成混淆。 -Like 与通配符 * 和 ？结合使用，以执行“like”匹配。

```
'PowerShell' -like '*shell'
```

```
# 输出
True
```

-Match 使用正则表达式执行匹配。

```
'PowerShell' -match '^*.shell$'


# 输出
True
```

使用范围运算符将数字 1 到 10 存储在变量中。

```
$Numbers = 1..10
```

确定 $Numbers 变量是否包含 15。

```
$Numbers -contains 15


# 输出
False
```

确定它是否包含数字 10。

```
$Numbers -contains 10


# 输出
True
```

-NotContains 反转逻辑，以查看 $Numbers 变量是否不包含值。

```
$Numbers -notcontains 15


# 输出
True
```

前面的示例返回布尔值 true，因为 $Numbers 变量不包含 15。 但它包含数字 10，因此在进行测试时，结果为 false。

```
$Numbers -notcontains 10
```

```
# 输出
False
```

PowerShell 版本 3.0 首次引入了“in”比较运算符。 它用于确定某个值是否“位于”数组中。 $Numbers 变量是数组，因为它包含多个值。

```
15 -in $Numbers

# 输出
False
```

换言之，-in 执行与 contains 比较运算符相同的测试，不过方向相反。

```
10 -in $Numbers

# 输出
True
```

15 不在 $Numbers 数组中，因此在下面的示例中返回 false。

```
15 -in $Numbers

# 输出
False
```

与 -contains 运算符一样，not 反转 -in 运算符的逻辑。

```
10 -notin $Numbers

# 输出
False
```

上面的示例返回 false，因为 $Numbers 数组包含 10，并且条件进行了测试以确定它是否不包含 10。

15“不位于”$Numbers 数组中，因此它返回布尔值 true。

```
15 -notin $Numbers
```

```
# 输出
True
```

-replace 运算符所执行的操作和你想象的一样。 它用于替换内容。 如果指定一个值，则会将该值替换为空值。 在下面的示例中，我将“Shell”替换为空值。

```
'PowerShell' -replace 'Shell'


# 输出
Power
```

如果要将值替换为其他值，请在要替换的模式之后指定新值。 SQL Saturday in Baton Rouge is an event that I try to speak at every year. 在下面的示例中，我将“Saturday”这个词替换为缩写“Sat”。

```
'SQL Saturday - Baton Rouge' -Replace 'saturday','Sat'


# 输出
SQL Sat - Baton Rouge
```

还有一些可用于替换内容的方法（如 Replace()），其工作原理类似于替换运算符。 但是，默认情况下，-Replace 运算符不区分大小写，而 Replace() 方法区分大小写。

```
'SQL Saturday - Baton Rouge'.Replace('saturday','Sat')


# 输出
SQL Sat - Baton Rouge
```

请注意，上述示例中未替换“Saturday”一词。 这是因为所指定的大小写格式与原始格式不同。当指定的“Saturday”的大小写格式与原始格式相同时，Replace() 方法会按预期替换它。

```
'SQL Saturday - Baton Rouge'.Replace('Saturday','Sat')
```

```
# 输出
SQL Sat - Baton Rouge
```