Microsoft Windows Server System

Windows PowerShell™ 사용 설명서

Microsoft Corporation

게시일: 2006 년 9 월

요약

Windows PowerShell™은 시스템 관리자를 위해 특별히 설계된 새로운 Windows 명령줄 셸입니다. 이 셸에는 조합하거나 독립적으로 사용할 수 있는 대화형 프롬프트 및 스크립팅 환경이 포함되어 있습니다.

이 설명서에서는 Windows PowerShell 의 기본 개념 및 기능에 대해 설명하고 Windows PowerShell 을 사용하여 시스템을 관리할 수 있는 방법을 제안합니다.

Microsoft

목차

Windows PowerShell 사용 설명서 저작권

Windows PowerShell 소개

Windows PowerShell 은 명령줄 사용자와 스크립트 작성자가 .NET Framework 의 기능을 사용할 수 있도록 해주는 명령줄 셸 및 스크립팅 환경입니다. 여기에는 Windows 명령 프롬프트 및 Windows 스크립트 호스트 환경을 통해 얻은 정보와 작성한 스크립트를 확장할 수 있는 많은 강력한 개념이 새로 도입되었습니다.

대상

Windows PowerShell 사용 설명서는 Windows PowerShell 에 대한 사전 배경 지식이 없는 IT 전문가, 프로그래머 및 고급 사용자를 위한 것입니다. 스크립팅과 WMI 에 대한 배경 지식이 있으면 좋겠지만 이 설명서의 내용을 이해하는 데 꼭 필요하지는 않습니다.

Windows PowerShell 정보

Windows PowerShell 은 오랜 문제를 해결하고 새로운 기능을 추가하여 명령줄 및 스크립팅 환경을 향상시키기 위해 설계되었습니다.

검색 기능

Windows Powershell 에서는 기능을 쉽게 찾을 수 있습니다. 예를 들어 Windows 서비스를 보고 변경하는 cmdlet 목록을 찾으려면 다음과 같이 입력하십시오.

```
get-command *-service
```

원하는 작업을 수행하는 cmdlet 을 찾은 후에는 Get-Help cmdlet 을 사용하여 해당 cmdlet 에 대한 자세한 내용을 볼 수 있습니다. 예를 들어 Get-Service cmdlet 에 대한 도움말을 보려면 다음과 같이 입력하십시오.

```
get-help get-service
```

이 cmdlet 이 출력하는 내용을 모두 보려면 해당 출력을 Get-Member cmdlet 에 파이프하십시오. 예를 들어 다음 명령은 Get-Service cmdlet 이 출력하는 개체의 멤버에 대한 정보를 표시합니다.

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

일관성

시스템 관리에는 많은 노력이 필요할 뿐 아니라 특성 상 복잡한 작업을 관리하는 데 도움이 되는 일관된 인터페이스를 갖춘 도구가 필요합니다. 그러나 명령줄 도구와 스크립트 가능한 COM 개체에는 이러한 일관된 인터페이스가 갖춰져 있지 않습니다.

Windows PowerShell 의 일관성은 Windows PowerShell 이 갖고 있는 장점 중 하나입니다. 예를 들어 Sort-Object cmdlet 을 사용하는 방법을 배운 경우 이 방법을 사용하여 아무 cmdlet 의 출력이나 정렬할 수 있기 때문에 cmdlet 에 따라 각기 다른 정렬 루틴을 배우지 않아도 됩니다.

또한 cmdlet 개발자는 개발한 cmdlet 을 정렬하는 기능을 설계하지 않아도 됩니다. Windows PowerShell 은 기본 기능이 갖춰져 있고 인터페이스의 다양한 측면에 있어서 일관성을 유지할 수 있도록 해주는 프레임워크를 제공합니다. 이 프레임워크는 일반적으로 개발자에게 주어지는 일부 선택권을 배제하지만 그 대신 강력하고 사용하기 쉬운 cmdlet 을 훨씬 더 쉽게 개발할 수 있도록 해줍니다.

대화형 스크립팅 환경

Windows PowerShell 은 명령줄 도구와 COM 개체에 액세스할 수 있을 뿐 아니라 .NET FCL(Framework Class Library)의 기능을 사용할 수 있는 대화형 스크립팅 환경으로,

여러 명령줄 도구와의 대화형 환경을 제공하는 Windows 명령 프롬프트와 여러 명령줄 도구와 COM 자동화 개체를 사용할 수 있지만 대화형 환경을 제공하지 않는 WSH(Windows 스크립트 호스트) 스크립트를 향상시킨 것입니다.

Windows PowerShell 은 이러한 모든 기능에 대한 액세스를 결합하여 대화형 사용자와 스크립트 작성자가 사용할 수 있는 기능을 확장하고 시스템을 보다 쉽게 관리할 수 있도록 합니다.

개체 지향

사용자는 명령을 텍스트로 입력하는 방식으로 Windows PowerShell 과 상호 작용하지만 Windows PowerShell 은 텍스트가 아니라 개체를 기반으로 하기 때문에 명령을 실행하면 개체가 출력되고 이러한 출력 개체는 다른 명령에 입력으로 보낼 수 있습니다. 따라서 Windows PowerShell 은 새롭고 강력한 명령줄 패러다임을 도입하는 동시에 다른 셸을 사용해 본 경험이 있는 사용자에게 친숙한 인터페이스를 제공합니다. 또한 Windows PowerShell 은 텍스트 대신 개체를 보낼 수 있도록 함으로써 명령 간에 데이터를 보내는 개념을 확장합니다.

스크립팅 환경으로의 간편한 전환

Windows PowerShell 에서는 대화형으로 명령을 입력하는 작업을 수행하다가 스크립트를 만들고 실행하는 작업으로 쉽게 전환할 수 있습니다. 이렇게 하려면 Windows PowerShell 명령 프롬프트에서 명령을 입력하여 원하는 작업을 수행하는 명령을 찾은 다음 이러한 명령을 스크립트로 사용하기 위해 파일로 복사하기 전에 스크립트나 기록에 저장하면 됩니다.

Windows PowerShell 설치 및 실행

설치 요구 사항

Windows PowerShell 을 설치하려면 먼저 Windows PowerShell 에 필요한 다음과 같은 소프트웨어 프로그램이 시스템에 설치되어 있어야 합니다.

- Windows XP 서비스 팩 2, Windows 2003 서비스 팩 1 이상
- Microsoft .NET Framework 2.0

컴퓨터에 Windows PowerShell 이 이미 설치되어 있으면 새 버전을 설치하기 전에 제어판의 **프로그램 추가/제거**를 사용하여 기존 버전을 제거하십시오.

Windows PowerShell 설치

Windows PowerShell 을 설치하려면

1. Windows PowerShell 설치 파일을 다운로드합니다. 이 설치 파일의 이름은 플랫폼, 운영 체제 및 언어 팩에 따라 다릅니다.
2. 열기를 클릭하여 설치를 시작합니다.
3. 설치 마법사 페이지에 나오는 지시에 따릅니다.

또한 Windows PowerShell 파일을 네트워크 공유 위치에 저장하여 여러 컴퓨터에 설치할 수도 있습니다.

자동 설치를 수행하려면 다음과 같이 입력하십시오.

```
<PowerShell-exe-file-name> /quiet
```

예를 들면 다음과 같습니다.

```
PowerShellSetup_x86_fre.exe /quiet
```

기본적으로 32 비트 버전의 Windows 에서는 Windows PowerShell 이 %SystemRoot%\System32\WindowsPowerShell\v1.0 디렉터리에 설치되고, 64 비트 버전의 Windows 에서는 32 비트 버전의 Windows PowerShell 은 %SystemRoot%\SysWow64\WindowsPowerShell\v1.0 디렉터리에, 64 비트 버전의 Windows PowerShell 은 %SystemRoot%\System32\WindowsPowerShell\v1.0 디렉터리에 설치됩니다.

Windows PowerShell 실행

시작 메뉴에서 Windows PowerShell 을 시작하려면 **시작**, **모든 프로그램**, **Windows PowerShell 1.0** 을 차례로 클릭하고 **Windows PowerShell** 아이콘을 클릭하십시오.

실행 상자에서 Windows PowerShell 을 시작하려면 **시작**을 클릭하고 **실행**을 클릭한 다음 **powershell** 을 입력하고 **확인**을 클릭하십시오.

명령 프롬프트(cmd.exe) 창에서 Windows PowerShell 을 시작하려면 명령 프롬프트에서 **powershell** 을 입력하십시오. Windows PowerShell 이 콘솔 세션에서 실행되므로 이와 동일한 방법으로 원격 텔넷이나 SSH 세션에서 Windows PowerShell 을 실행할 수도 있습니다. 명령 프롬프트 세션으로 돌아가려면 **exit** 를 입력하십시오.

Windows PowerShell 기본 사항

그래픽 인터페이스는 대부분의 컴퓨터 사용자에게 잘 알려진 몇 가지 기본 개념을 사용합니다. 사용자는 이러한 친숙한 인터페이스를 사용하여 손쉽게 작업을 수행할 수 있습니다. 운영 체제는 대개 특정 기능에 액세스하기 위한 드롭다운 메뉴와 특정 상황에 맞는 기능에 액세스하기 위한 상황에 맞는 메뉴를 사용하여 검색 가능한 항목을 그래픽 방식으로 표시합니다.

그러나 Windows PowerShell 과 같은 CLI(명령줄 인터페이스)는 메뉴나 그래픽 시스템을 사용하지 않으므로 이와 다른 방식으로 정보를 표시해야 합니다. 명령을 사용하려면 먼저 명령 이름을 알아야 합니다. GUI 환경의 기능에 해당하는 복잡한 명령을 입력할 수 있더라도 일반적으로 사용되는 명령과 명령 매개 변수에 대해 잘 알고 있어야 합니다.

대부분의 CLI 에는 인터페이스를 쉽게 익힐 수 있는 일정한 패턴이 없습니다. CLI 는 최초 운영 체제 셸이기 때문에 많은 명령 이름과 매개 변수 이름이 임의로 선택되었습니다. 대개 이러한 이름은 정확성보다는 간결성을 고려하여 만들어졌습니다. 도움말 시스템 및 명령 설계 표준은 대부분의 CLI 에 통합되어 있지만 대개 최초 명령과의 호환성을 고려해 설계되었으므로 명령 집합의 모양이 계속 이전에 결정된 모양을 따릅니다.

Windows PowerShell 은 사용자가 CLI 에 대해 오랫동안 쌓은 지식을 활용하도록 설계되었습니다. 이 장에서는 Windows PowerShell 에 대해 빠르게 배울 수 있는 다음과 같은 몇 가지 기본 도구와 개념에 대해 설명합니다.

- Get-Command 사용
- Cmd.exe 및 UNIX 명령 사용

- 외부 명령 사용
- Tab 자동 채우기 사용
- Get-Help 사용

중요한 **Windows PowerShell** 개념에 대한 설명

Windows PowerShell 설계는 다양한 환경의 개념을 통합한 것으로, 이러한 개념 중 일부는 특정 셸 또는 프로그래밍 환경을 사용해 본 경험이 있는 사용자에게 친숙하겠지만 이러한 개념을 모두 알고 있는 사용자는 거의 없습니다. 이러한 개념 중 몇 가지를 살펴보면 셸을 이해하는 데 도움이 됩니다.

텍스트를 기반으로 하지 않는 명령

이전의 명령줄 인터페이스 명령과 달리 Windows PowerShell cmdlet 은 단순히 화면에 표시되는 문자열이 아니라 개체 즉, 구조화된 정보를 처리하도록 설계되었습니다. 명령 출력에는 필요할 때 사용할 수 있는 추가 정보가 항상 포함됩니다. 이 문서에서는 이 항목에 대해 자세히 설명합니다.

Windows PowerShell 에서는 텍스트 처리 도구가 이전의 명령줄 인터페이스에서와 다른 방식으로 명령줄 데이터를 처리하기 때문에 대부분의 경우 텍스트 처리 도구를 사용하여 특정 정보를 추출할 필요가 없습니다. 표준 Windows PowerShell 개체 조작 명령을 사용하면 데이터의 일부에 직접 액세스할 수 있습니다.

확장 가능한 명령 계열

Cmd.exe 와 같은 인터페이스에서는 기본 제공 명령 집합을 직접 확장할 수 없습니다. Cmd.exe 에서 실행되는 외부 명령줄 도구를 만들 수는 있지만 이러한 외부 도구에는 도움말 통합 같은 서비스가 없기 때문에 Cmd.exe 는 이러한 외부 도구가 유효한 명령인지 자동으로 인식하지 못합니다.

cmdlet(command-let 으로 발음)이라고 하는 Windows PowerShell 의 기본 이진 명령은 cmdlet 을 만든 다음 스냅인을 사용하여 cmdlet 을 Windows PowerShell 에 추가함으로써 확장할 수 있습니다. Windows PowerShell 스냅인은 다른 모든 인터페이스의 이진 도구처럼 컴파일됩니다. Windows PowerShell 스냅인을 사용하면 셸에 새 cmdlet 뿐 아니라 Windows PowerShell 공급자도 추가할 수 있습니다.

이 설명서에서는 Windows PowerShell 내부 명령도 특성상 cmdlet 이라고 합니다.

참고:

Windows PowerShell 에서는 cmdlet 이외의 명령을 실행할 수 있습니다. 이 **Windows PowerShell 사용 설명서**에서 자세히 설명하지 않지만 이러한 명령은 명령 유형을 범주로 구분하는 데 유용합니다. Windows PowerShell 은 UNIX 셸 스크립트 및 Cmd.exe 배치 파일과 유사하지만 파일 이름 확장명이 .ps1 인 스크립트를 지원합니다. 또한 Windows PowerShell 에서는 인터페이스나 스크립트에서 직접 사용할 수 있는 내부 함수를 만들 수도 있습니다.

콘솔 입력 및 표시 처리

사용자가 명령을 입력하면 Windows PowerShell 은 항상 이 명령줄 입력을 직접 처리합니다. Windows PowerShell 은 또한 화면에 표시되는 출력의 형식을 지정합니다. 이 기능은 각 cmdlet 에 대해 수행해야 하는 작업을 줄여주고 사용 중인 cmdlet 에 관계없이 항상 동일한 방식으로 작업을 수행할 수 있도록 해 주기 때문에 중요합니다. 이 기능을 통해 도구 개발자와 도구 사용자의 작업을 단순화하는 방법의 예로 명령줄 도움말을 들 수 있습니다.

이전의 명령줄 도구는 도움말을 요청하고 표시하는 자체 구성을 갖습니다. 일부 명령줄 도구는 **/?**를 사용하여 도움말 표시를 트리거하고 일부 명령줄 도구는 **-?** 또는 **/H** 뿐 아니라 심지어 **//**를 사용하여 도움말 표시를 트리거합니다. 또 일부 명령줄 도구는 콘솔 화면 대신 GUI 창에 도움말을 표시합니다. 응용 프로그램 업데이트 프로그램과 같이 일부 복잡한 도구는 도움말을 표시하기 전에 내부 파일의 압축을 풉니다. 사용자가 잘못된 매개 변수를 사용할 경우 이러한 도구는 사용자의 입력 내용을 무시하고 자동으로 작업을 시작할 수 있습니다.

Windows PowerShell 에서 명령을 입력하면 Windows PowerShell 은 자동으로 이러한 명령을 구문 분석하고 전처리합니다. 예를 들어 Windows PowerShell cmdlet 과 함께 **-?** 매개 변수를 사용할 경우, 이는 항상 "해당 명령에 대한 도움말을 표시" 하라는 의미입니다. Cmdlet 개발자는 명령을 구문 분석할 필요 없이 도움말 텍스트만 제공하면 됩니다.

Windows PowerShell 에서 이전의 명령줄 도구를 사용하는 경우에도 Windows PowerShell 의 도움말 기능은 사용할 수 있습니다. Windows PowerShell 은 매개 변수를 처리하고 그 결과를 외부 도구에 전달합니다.

참고:

Windows PowerShell 에서 그래픽 응용 프로그램을 실행하면 해당 응용 프로그램의 창이 열립니다. Windows PowerShell 은 사용자가 제공하는 명령줄 입력이나 콘솔 창에 반환되는 응용 프로그램 출력을 처리할 때만 개입하고 응용 프로그램이 자체적으로 작업을 수행하는 방식에는 영향을 주지 않습니다.

일부 C# 구문 사용

Windows PowerShell 은 .NET Framework 를 기반으로 하기 때문에 Windows PowerShell 에는 C# 프로그래밍 언어에서 사용하는 것과 매우 유사한 구문 기능과 키워드가 포함되어는 있습니다. C#에 관심이 있을 경우 Windows PowerShell 에 대해 알면 C#을 익히기가 훨씬 더 쉬워집니다.

C# 프로그래머가 아닐 경우 이러한 유사함은 별로 도움이 되지 않지만 C#에 대해 잘 알고 있을 경우 이러한 유사함을 통해 Windows PowerShell 을 훨씬 더 쉽게 익힐 수 있습니다.

Windows PowerShell 이름 익히기

대부분의 명령줄 인터페이스의 경우 명령 이름과 명령 매개 변수 이름을 익히는 데 많은 시간이 걸립니다. 문제는 일정한 패턴이 없기 때문에 정기적으로 사용해야 하는 각 명령과 매개 변수를 익히려면 기억에만 의존해야 한다는 것입니다.

새 명령이나 매개 변수를 사용하여 작업할 경우 대개 기존 명령이나 매개 변수를 사용할 수 없기 때문에 새 이름을 찾아서 익혀야 합니다. 기능을 점차 추가하는 방식으로 인터페이스를 작은 도구 집합에서 점차 확장한다는 사실에 주목하면 인터페이스 구조가 표준 구조가 아닌 이유를 쉽게 이해할 수 있습니다. 특히 명령 이름과 관련해서 각 명령은 별도의 도구이기 때문에 이것이 논리적으로 들릴 수 있지만 명령 이름을 처리할 수 있는 더 좋은 방법이 있습니다.

대부분의 명령은 서비스나 프로세스와 같은 운영 체제 요소나 응용 프로그램을 관리하도록 만들어졌습니다. 명령에는 계열과 맞거나 맞지 않을 수 있는 다양한 이름이 있습니다. 예를 들어 Windows 시스템에서는 **net start** 명령과 **net stop** 명령을 사용하여 서비스를 시작하고 중지할 수도 있지만 **net** 서비스 명령의 이름 패턴과 맞지 않는 완전히 다른 **sc** 라는 이름을 가진 보다 일반화된 서비스 제어 도구를 사용하여 서비스를 시작하고 중지할 수도 있습니다. Windows 프로세스 관리 명령으로는 프로세스를 표시하는 **tasklist** 명령과 프로세스를 종료하는 **taskkill** 명령이 있습니다.

명령에 사용되는 매개 변수는 불규칙적으로 적용됩니다. **net start** 명령으로는 원격 컴퓨터의 서비스를 시작할 수 없지만 **sc** 명령으로는 원격 컴퓨터의 서비스를 시작할 수 있습니다. 그러나 이 경우 원격 컴퓨터를 지정하려면 원격 컴퓨터 이름 앞에 두 개의 백슬래시를 입력해야 합니다. 예를 들어 DC01 이라는 원격 컴퓨터에서 스풀러 서비스를 시작하려면 **sc \\DC01 start spooler** 를 입력하고, DC01 에서 실행 중인 작업을 표시하려면 **tasklist /S DC01** 과 같이 **/S**("시스템"을 나타냄) 매개 변수를 사용하고 DC01 이름을 백슬래시 없이 제공해야 합니다.

서비스와 프로세스 사이에는 중요한 기술적인 차이가 있지만 둘 다 수명 주기가 잘 정의된 컴퓨터에서 관리가 가능한 요소입니다. 서비스나 프로세스를 중지 또는 시작하거나 현재 실행 중인 모든 서비스나 프로세스의 목록을 볼 수 있습니다. 즉, 서비스와 프로세스는 서로 다른 요소이지만 일반적으로 서비스나 프로세스에 대해 수행하는 작업은 개념상 동일합니다. 또한 매개 변수를 지정하여 이러한 작업을 사용자 지정하기 위해 선택할 수 있는 옵션도 개념상 유사합니다.

이러한 유사함을 이용하면 Windows PowerShell 에서 cmdlet 을 익히고 사용하기 위해 알아야 하는 개별 이름의 수를 줄일 수 있습니다.

쉽게 기억할 수 있는 **Cmdlet** 의 동사-명사 이름 사용

Windows PowerShell 은 "동사-명사" 이름 체계를 사용하며, 각 cmdlet 이름은 표준 동사에 특정 명사를 하이픈으로 연결하여 구성됩니다. Windows PowerShell 동사는 반드시 영어가 아니어도 되지만 Windows PowerShell 의 특정 동작을 나타내야 합니다. 명사는 모든 언어의 명사와 매우 유사하며 시스템 관리에 중요한 특정 유형의 개체를 나타냅니다. 동사와 명사로 구성된 이름을 몇 개만 살펴보면 두 부분으로 구성된 이러한 이름이 얼마나 기억하기 쉬운지 쉽게 알 수 있습니다.

명사는 덜 제한적이지만 항상 명령의 실행 대상을 나타내야 합니다. Windows PowerShell 에는 **Get-Process**, **Stop-Process**, **Get-Service** 및 **Stop-Service** 와 같은 명령이 있습니다.

두 개의 명사와 두 개의 동사로 구성된 이름의 경우 일관성이 있어도 기억하기가 쉽지 않습니다. 그러나 10 개의 동사와 10 개의 명사로 구성된 표준 집합이 있다고 가정할 경우 20 개의 단어만 기억하면 되지만 이러한 단어를 사용하여 100 개의 고유한 명령 이름을 만들 수 있습니다.

종종 이름만으로 명령이 수행하는 작업을 알 수 있으며 대개 새 명령에 사용해야 하는 이름도 쉽게 확인할 수 있습니다. 예를 들어 컴퓨터 종료 명령은 **Stop-Computer** 이고, 네트워크에 있는 모든 컴퓨터를 보여 주는 명령은 **Get-Computer** 이며, 시스템 날짜를 표시하는 명령은 **Get-Date** 일 수 있습니다.

Get-Command(다음 절에 자세히 설명되어 있음)와 함께 **-Verb** 매개 변수를 사용하면 특정 동사가 포함된 모든 명령을 표시할 수 있습니다. 예를 들어 동사 **Get** 을 사용하는 모든 cmdlet 을 보려면 다음과 같이 입력하십시오.

```
PS> Get-Command -Verb Get
CommandType     Name                               Definition
-----------     ----                               ----------
Cmdlet          Get-Acl                            Get-Acl [[-Path] <String[]>]...
Cmdlet          Get-Alias                          Get-Alias [[-Name] <String[]...
```

```
Cmdlet          Get-AuthenticodeSignature       Get-AuthenticodeSignature [-...
Cmdlet          Get-ChildItem                   Get-ChildItem [[-Path] <Stri...
...
```

-Noun 매개 변수는 동일한 유형의 개체에 영향을 주는 명령 계열을 볼 수 있게 해 주므로 훨씬 더 유용합니다. 예를 들어 서비스를 관리하는 데 사용할 수 있는 명령을 보려면 다음 명령을 입력하십시오.

```
PS> Get-Command -Noun Service
CommandType     Name                            Definition
-----------     ----                            ----------
Cmdlet          Get-Service                     Get-Service [[-Name] <String...
Cmdlet          New-Service                     New-Service [-Name] <String>...
Cmdlet          Restart-Service                 Restart-Service [-Name] <Str...
Cmdlet          Resume-Service                  Resume-Service [-Name] <Stri...
Cmdlet          Set-Service                     Set-Service [-Name] <String>...
Cmdlet          Start-Service                   Start-Service [-Name] <Strin...
Cmdlet          Stop-Service                    Stop-Service [-Name] <String...
Cmdlet          Suspend-Service                 Suspend-Service [-Name] <Str...
...
```

단지 동사-명사 이름 체계를 사용한다고 해서 명령을 cmdlet 이라고 할 수는 없습니다. cmdlet 이 아니지만 동사-명사 이름을 사용하는 기본 Windows PowerShell 명령의 한 예로 콘솔 창을 지우는 명령인 Clear-Host 를 들 수 있습니다. 다음과 같이 Get-Command 를 실행하면 알 수 있듯이 Clear-Host 명령은 실제로 내부 함수입니다.

```
PS> Get-Command -Name Clear-Host

CommandType     Name                            Definition
-----------     ----                            ----------
Function        Clear-Host                      $spaceType = [System.Managem...
```

표준 매개 변수 사용

앞에서 설명한 대로 이전의 명령줄 인터페이스에 사용되는 명령은 대개 일관된 매개 변수 이름을 사용하지 않으며, 어떤 경우는 매개 변수에 아예 이름이 없습니다. 이름이 있어도 단일 문자나 약어로 되어 있어 신속하게 입력할 수는 있지만 새로운 사용자가 쉽게 이해하기 힘듭니다.

대부분의 다른 이전 명령줄 인터페이스와 달리 Windows PowerShell 에서는 매개 변수를 직접 처리하며, 매개 변수에 대한 이러한 직접 액세스와 함께 개발자 지침을 사용하여 매개 변수 이름을 표준화합니다. 이렇게 해도 모든 cmdlet 이 항상 표준 이름을 사용하는 것은 아니지만 표준화를 촉진합니다.

참고:

매개 변수 이름 앞에는 Windows PowerShell 이 매개 변수로 명확히 식별할 수 있도록 '-'이 옵니다. 예를 들어 **Get-Command -Name Clear-Host** 에서 매개 변수 이름이 실제로는 **Name** 이지만 **–Name** 으로 입력되어 있습니다.

다음은 표준 매개 변수의 이름과 사용 방법에 대한 몇 가지 일반적인 특징입니다.

도움말 매개 변수(?)

cmdlet 에 **-?** 매개 변수를 지정하면 cmdlet 이 실행되는 대신 cmdlet 에 대한 도움말이 표시됩니다.

일반 매개 변수

Windows PowerShell 에는 일반 매개 변수라고 하는 여러 개의 매개 변수가 있습니다. 이러한 매개 변수는 Windows PowerShell 엔진에 의해 제어되므로 cmdlet 에 의해 구현될 때마다 항상 동일한 방식으로 동작합니다. 일반 매개 변수로는 **WhatIf**, **Confirm**, **Verbose**, **Debug**, **Warn**, **ErrorAction**, **ErrorVariable**, **OutVariable** 및 **OutBuffer** 가 있습니다.

권장 매개 변수

Windows PowerShell 핵심 cmdlet 은 유사한 매개 변수에 표준 이름을 사용합니다. 매개 변수 이름을 반드시 사용해야 하는 것은 아니지만 Windows PowerShell 에는 표준 이름 사용을 권장하는 명시적 지침이 있습니다.

예를 들어 이 지침에서는 Server, Host, System, Node 또는 기타 일반적인 대체 단어가 아니라 **ComputerName** 과 같이 컴퓨터를 이름으로 참조하는 매개 변수 이름을 권장합니다. 이러한 권장 매개 변수 이름으로는 **Force**, **Exclude**, **Include**, **PassThru**, **Path** 및 **CaseSensitive** 가 있습니다.

요약 명령 정보 보기

Windows PowerShell **Get-Command** cmdlet 은 사용 가능한 모든 명령의 이름을 검색합니다. Windows PowerShell 프롬프트에서 **Get-Command** 를 입력하면 다음과 같은 내용이 출력됩니다**.**

```
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...
...
```

이 출력은 내부 명령을 표 형식으로 요약하여 보여 주는 Cmd.exe 의 Help 출력과 매우 유사합니다. 위에 표시된 **Get-Command** 명령의 출력에서는 모든 명령의 CommandType 이 Cmdlet 입니다. Cmdlet 은 Cmd.exe 의 **dir** 및 **cd** 명령과 BASH 같은 UNIX 셸의 기본 제공 명령에 해당하는 Windows PowerShell 의 내장 명령 유형입니다.

Get-Command 명령의 출력에서 모든 정의가 줄임표(...)로 끝나는데, 이는 PowerShell 이 빈 공간에 내용을 모두 표시할 수 없음을 나타냅니다. Windows PowerShell 은 출력을 표시할 때 출력 형식을 텍스트로 지정한 다음 데이터가 콘솔 창 안에 올바로 표시되도록 정렬합니다. 자세한 내용은 이 절의 뒷부분에 나오는 포맷터에 대한 설명을 참조하십시오.

Get-Command cmdlet 에는 각 cmdlet 의 구문을 검색할 수 있는 **Syntax** 매개 변수가 있습니다. 다음과 같이 모든 구문을 출력하려면 **Get-Command -Syntax** 명령을 입력하십시오.

```
PS> Get-Command -Syntax
Add-Content [-Path] <String[]> [-Value] <Object[]> [-PassThru] [-Filter <String>]
[-Include <String[]>] [-Exclude <String[]>] [-Force] [Credential <PSCredential>]
[-Verbose] [-Debug] [-ErrorAction <ActionPreference>] [-ErrorVariable <String>] [-
```

```
OutVariable <String>] [-OutBuffer <Int32>] [-WhatIf] [-Confirm] [-Encoding
<FileSystemCmdletProviderEncoding>]

Add-History [[-InputObject] <PSObject[]>] [-Passthru] [-Verbose] [-Debug] [-
ErrorAction <ActionPreference>] [-ErrorVariable <String>] [-OutVariable
<String>][-OutBuffer <Int32>]...
```

사용 가능한 명령 유형 표시

Get-Command 명령은 Windows PowerShell 에서 사용할 수 있는 명령을 모두 표시하는 대신 Windows PowerShell 세션에 있는 cmdlet 만 표시합니다. 실제로 Windows PowerShell 은 여러 가지 유형의 명령을 지원합니다. 이 Windows PowerShell 사용 설명서에서 자세히 설명하지는 않지만 별칭, 함수 및 스크립트도 Windows PowerShell 명령입니다. 또한 실행 파일이거나 파일 유형 처리기가 등록되어 있는 외부 파일도 명령으로 간주됩니다.

다음 명령을 입력하면 호출할 수 있는 모든 항목의 목록을 반환할 수 있습니다.

```
PS> Get-Command *
```

이 목록에는 검색 경로에 있는 외부 파일도 포함되므로 수천 개의 항목이 포함될 수 있습니다. 따라서 검색되는 명령의 수를 줄이는 것이 좋습니다. 다른 유형의 기본 명령을 찾으려면 **Get-Command** cmdlet 의 **CommandType** 매개 변수를 사용하면 됩니다. 다른 명령 유형에 대해 자세히 설명하지는 않았지만 명령 클래스에 대한 CommandType 의 이름을 알고 있으면 이러한 명령도 표시할 수 있습니다.

참고:

이 설명서에서 자세히 설명하지 않지만 Windows PowerShell 명령 인수의 와일드카드 일치에 별표(*)가 사용됩니다. 별표(*)는 "하나 이상의 문자와 일치한다"는 것을 의미합니다. **Get-Command a***를 입력하여 "a" 문자로 시작하는 모든 명령을 찾을 수 있습니다. Cmd.exe 의 와일드카드 일치와 달리 Windows PowerShell 의 와일드카드 일치에는 마침표도 포함됩니다.

특정 명령 범주 별칭(표준 명령 이름의 대체 이름으로 사용되는 애칭)을 표시하려면 다음 명령을 입력하십시오.

```
PS> Get-Command -CommandType Alias
```

모든 Windows PowerShell 함수를 표시하려면 다음 명령을 입력하십시오.

```
PS> Get-Command -CommandType Function
```

Windows PowerShell 의 검색 경로에 있는 외부 스크립트를 표시하려면 다음 명령을 입력하십시오.

```
PS> Get-Command -CommandType ExternalScript
```

자세한 도움말 정보 보기

Windows PowerShell 에는 모든 cmdlet 에 대한 자세한 도움말 설명서가 있습니다. 도움말 항목을 표시하려면 Get-Help cmdlet 을 사용하십시오. 예를 들어 Get-Childitem cmdlet 에 대한 도움말을 보려면 다음과 같이 입력하십시오.

```
get-help get-childitem
```

또는

```
get-childitem -?
```

또한 **man** 및 **help** 함수를 사용하면 각 도움말 항목을 한 번에 한 페이지씩 표시할 수 있습니다. 이러한 함수를 사용하려면 cmdlet 이름 앞에 **man** 또는 **help** 를 입력하십시오. 예를 들어 Get-Childitem cmdlet 에 대한 도움말을 표시하려면 다음과 같이 입력하십시오.

```
man get-childitem
```

또는

```
help get-childitem
```

Get-Help cmdlet 은 Windows PowerShell 의 개념 항목에 대한 정보도 표시합니다. 개념 도움말 항목은 about_line_editing 과 같이 "about_" 접두사로 시작합니다. 개념 항목 목록을 표시하려면 다음과 같이 입력하십시오.

```
get-help about_*
```

특정 도움말 항목을 표시하려면 다음과 같이 항목 이름을 입력하십시오.

```
get-help about_line_editing
```

친숙한 명령 이름 사용

Windows PowerShell 에서는 별칭이라고 하는 메커니즘을 사용하여 대체 이름으로 명령을 나타낼 수 있습니다. 별칭은 다른 셸을 사용해 본 경험이 있는 사용자가 이미 알고 있는 일반적인 명령 이름을 다시 사용하여 Windows PowerShell 에서 유사한 작업을 수행할 수 있도록 해 줍니다. 따라서 이 설명서에서 Windows PowerShell 별칭을 자세히 설명하지 않지만 Windows PowerShell 시작부터 이러한 별칭을 사용할 수 있습니다.

별칭은 사용자가 입력하는 명령 이름을 다른 명령과 연결합니다. 예를 들어 Windows PowerShell 에는 출력 창을 지우는 **Clear-Host** 라는 내부 함수가 있는데, 명령 프롬프트에서 **cls** 또는 **clear** 명령을 입력하면 Windows PowerShell 은 이러한 명령을 **Clear-Host** 함수의 별칭으로 해석하고 **Clear-Host** 함수를 실행합니다.

이 기능은 사용자가 Windows PowerShell 을 익히는 데 도움이 됩니다. 첫째, 대부분의 Cmd.exe 및 UNIX 사용자는 많은 명령의 이름을 이미 알고 있기 때문에, Windows PowerShell 에서 이러한 명령이 동일한 결과를 나타내지는 않더라도 그 형태가 유사하여 따로 이름을 외우지 않아도 Windows PowerShell 명령을 사용하여 작업을 수행할 수 있습니다. 둘째,

다른 셸에 이미 익숙한 사용자가 새 셸을 익히는 데 있어서 가장 어려운 점은 손에 익은 습관 때문에 발생하는 실수입니다. 예를 들어 Cmd.exe 를 몇 년 동안 사용한 사용자의 경우 화면에 표시된 출력 내용을 모두 지우고자 할 때 반사적으로 **cls** 명령을 입력한 다음 Enter 키를 누를 수 있습니다. Windows PowerShell 에 **Clear-Host** 함수에 대한 별칭이 없다면 **"'cls'는 cmdlet, 함수, 실행할 수 있는 프로그램 또는 스크립트 파일로 인식되지 않습니다."**라는 오류 메시지가 표시되고 출력 내용을 지울 수 있는 어떠한 정보도 표시되지 않습니다.

다음은 Windows PowerShell 에서 사용할 수 있는 일반적인 Cmd.exe 및 UNIX 명령에 대한 간단한 목록입니다.

cat	dir	mount	rm
cd	echo	move	rmdir
chdir	erase	popd	sleep
clear	h	ps	sort
cls	history	pushd	tee
copy	kill	pwd	type
del	lp	r	write
diff	ls	ren	

사용자 자신이 이러한 명령 중 하나를 반사적으로 사용하고 있음을 알아채고 기본 Windows PowerShell 명령의 실제 이름을 확인하려면 다음과 같이 **Get-Alias** 명령을 사용하면 됩니다.

```
PS> Get-Alias cls

CommandType     Name                               Definition
-----------     ----                               ----------
Alias           cls                                Clear-Host
```

일반적으로 이 Windows PowerShell 사용 설명서에서는 읽기 쉽게 하기 위해 예제에 별칭을 사용하지 않습니다. 그러나 별칭을 많이 알고 있으면 다른 곳에서 가져온 임의의 Windows PowerShell 코드 조각을 사용하여 작업하는 경우나 사용자 고유의 별칭을 정의하려는 경우 유용할 수 있습니다. 이 절의 나머지 부분에서는 표준 별칭과 사용자 고유의 별칭을 정의하는 방법을 설명합니다.

표준 별칭 해석

위에서 설명한 별칭은 다른 인터페이스와의 이름 호환성을 위해 개발된 반면 Windows PowerShell 에 기본 제공된 별칭은 주로 명령 이름을 간략하게 표현하기 위해 개발되었습니다. 이러한 약식 이름은 빨리 입력할 수는 있지만 참조 대상을 모르면 해석할 수 없습니다.

Windows PowerShell 에서는 일반적인 동사와 명사에 대한 약식 이름을 기반으로 하는 일련의 표준 별칭을 제공하여 명확하면서도 간결한 별칭을 만들기 때문에 약식 이름만 알면 해석할 수 있는 일반적인 cmdlet 에 대한 핵심 별칭 집합을

사용할 수 있습니다. 예를 들어 표준 별칭에서 동사 **Get** 은 **g** 로, 동사 **Set** 은 **s** 로, 명사 **Item** 은 **i** 로, 명사 **Location** 은 **l** 로, 명사 Command 는 cm 으로 축약됩니다.

다음은 이러한 방식으로 별칭을 만드는 방법을 보여 주는 간단한 예입니다. Get-Item 의 표준 별칭은 Get 을 나타내는 g 와 Item 을 나타내는 i 를 결합한 gi 이고, Set-Item 의 표준 별칭은 Set 을 나타내는 s 와 Item 을 나타내는 i 를 결합한 si 이고, Get-Location 의 표준 별칭은 Get 을 나타내는 g 와 Location 을 나타내는 l 을 결합한 gl 이고, Set-Location 의 표준 별칭은 Set 을 나타내는 s 와 Location 을 나타내는 l 을 결합한 sl 이며, Get-Command 의 표준 별칭은 Get 을 나타내는 g 와 Command 를 나타내는 cm 을 결합한 gcm 입니다. Set-Command cmdlet 은 실제로 없지만 있다고 가정할 경우 표준 별칭은 Set 을 나타내는 s 와 Command 를 나타내는 cm 을 결합한 scm 일 수 있습니다. 또한 Windows PowerShell 별칭에 익숙한 사용자는 scm 을 보면 이 별칭이 Set-Command 를 나타낸다는 것을 짐작할 수 있습니다.

새 별칭 만들기

Set-Alias cmdlet 을 사용하여 사용자 고유의 별칭을 만들 수 있습니다. 예를 들어 다음 문은 **표준 별칭 해석**에서 설명한 표준 cmdlet 별칭을 만듭니다.

```
Set-Alias -Name gi -Value Get-Item
Set-Alias -Name si -Value Set-Item
Set-Alias -Name gl -Value Get-Location
Set-Alias -Name sl -Value Set-Location
Set-Alias -Name gcm -Value Get-Command
```

내부적으로 Windows PowerShell 은 시작할 때 이와 같은 명령을 사용하지만 이러한 별칭은 변경할 수 없습니다. 실제로 이러한 명령 중 하나를 실행하려고 하면 다음과 같이 별칭을 수정할 수 없다는 오류 메시지가 나타납니다.

```
PS> Set-Alias -Name gi -Value Get-Item
Set-Alias : 별칭은 읽기 전용이거나 상수이며 쓸 수 없으므로 별칭에 쓸 수 없습니다.
줄:1 문자:10
+ Set-Alias  <<<< -Name gi -Value Get-Item
```

탭 확장을 사용하여 자동으로 이름 완성

명령줄 셸은 종종 긴 파일 또는 명령의 이름을 자동으로 완성하는 기능을 제공하여 명령 입력 시간을 단축시키고 힌트를 제공합니다. Windows PowerShell 에서는 **Tab** 키를 눌러 파일 이름과 cmdlet 이름을 채울 수 있습니다.

참고:

탭 확장은 내부 함수인 TabExpansion 에 의해 제어됩니다. 이 함수는 수정 또는 재정의가 가능하므로 이 설명서에서는 기본 Windows PowerShell 구성의 동작에 대해 설명합니다.

사용 가능한 선택 항목으로 파일 이름이나 경로를 채우려면 이름의 일부를 입력한 다음 **Tab** 키를 누르십시오. 그러면 이름이 자동으로 일치하는 첫 번째 항목으로 확장됩니다. **Tab** 키를 반복해서 누르면 사용 가능한 모든 선택 항목이 번갈아 표시됩니다.

cmdlet 이름의 탭 확장은 약간 다릅니다. cmdlet 이름에 대해 탭 확장을 사용하려면 이름의 첫 번째 부분(즉, 동사) 전체와 하이픈을 차례로 입력하십시오. 그러면 부분 일치를 위해 더 많은 이름을 채울 수 있습니다. 예를 들어 **get-co** 를 입력한 다음 **Tab** 키를 누르면 자동으로 **Get-Command** cmdlet 으로 확장되고(이때 문자의 대/소문자도 표준 형식으로 변경됨), **Tab** 키를 다시 누르면 일치하는 유일한 다른 cmdlet 이름인 **Get-Content** 로 이 cmdlet 이 대체됩니다.

탭 확장은 동일한 줄에서 반복해서 사용할 수 있습니다. 예를 들어 다음을 입력하여 **Get-Content** cmdlet 의 이름에 대해 탭 확장을 사용할 수 있습니다.

```
PS> Get-Con<Tab>
```

Tab 키를 누르면 이 명령이 다음과 같이 확장됩니다.

```
PS> Get-Content
```

그러면 다음과 같이 Active Setup 로그 파일의 경로를 일부만 지정하고 다시 탭 확장을 사용할 수 있습니다.

```
PS> Get-Content c:\windows\acts<Tab>
```

Tab 키를 누르면 이 명령이 다음과 같이 확장됩니다.

```
PS> Get-Content C:\windows\actsetup.log
```

참고:

탭 확장 프로세스의 유일한 제한 사항은 탭이 항상 단어를 완성하려는 시도로 해석된다는 것입니다. 따라서 명령 예제를 복사하여 Windows PowerShell 콘솔에 붙여 넣을 경우 명령 예제에 탭이 포함되지 않도록 해야 합니다. 명령 예제에 탭이 포함되어 있으면 예측할 수 없는 결과가 발생하고 대부분 의도한 대로 되지 않습니다.

개체 파이프라인

파이프라인은 일련의 파이프 세그먼트가 연결된 것처럼 동작합니다. 파이프라인을 따라 이동하는 항목은 각 세그먼트를 통과합니다. Windows PowerShell 에서 여러 명령을 파이프 연산자 "|"로 연결하여 파이프라인을 만들면 각 명령의 출력이 그 다음 명령의 입력으로 사용됩니다.

파이프라인은 명령줄 인터페이스에 사용되는 가장 중요한 개념이라고 할 수 있습니다. 파이프라인을 올바로 사용하면 복잡한 명령을 보다 쉽게 입력할 수 있을 뿐 아니라 명령의 작업 흐름도 보다 쉽게 확인할 수 있습니다. 또한 파이프라인은 각 항목에 대해 개별적으로 작동하기 때문에 파이프라인에 있는 항목의 수에 따라 파이프라인을 수정하지 않아도 된다는 이점이 있습니다. 게다가 파이프라인에 있는 각 명령(파이프라인 요소)은 해당 출력을 파이프라인에 있는 다음 명령에 항목 단위로 전달하므로 복잡한 명령의 리소스 요구를 줄여 주고 출력을 즉시 볼 수 있게 해 줍니다.

이 장에서는 Windows PowerShell 파이프라인과 가장 보편적으로 사용되는 셸의 파이프라인의 차이점에 대해 설명한 다음 파이프라인 출력을 제어하고 파이프라인의 작동 방법을 확인하는 데 사용할 수 있는 몇 가지 기본 도구를 보여 줍니다.

Windows PowerShell 파이프라인에 대한 설명

파이프는 사실상 Windows PowerShell 의 모든 영역에 사용됩니다. 화면에 텍스트가 표시되어 있는 경우에도 Windows PowerShell 은 명령 사이에 있는 텍스트를 파이프하는 대신 개체를 파이프합니다.

파이프라인에 사용되는 표기법과 다른 셸에 사용되는 표기법이 유사하기 때문에 처음에는 Windows PowerShell 에 새로 도입된 기능을 쉽게 확인하지 못할 수도 있습니다. 예를 들어 **Out-Host** cmdlet 을 사용하여 다른 명령의 출력을 페이지 단위로 표시하면 다음과 같이 일반 텍스트가 페이지 단위로 나뉘어져 화면에 표시된 것처럼 보입니다.

```
PS> Get-ChildItem -Path C:\WINDOWS\System32 | Out-Host -Paging
    Directory: Microsoft.Windows PowerShell.Core\FileSystem::C:\WINDOWS\system32

Mode                 LastWriteTime     Length Name
----                 -------------     ------ ----
-a---         2005-10-22  11:04 PM        315 $winnt$.inf
-a---         2004-08-04   8:00 AM      68608 access.cpl
-a---         2004-08-04   8:00 AM      64512 acctres.dll
-a---         2004-08-04   8:00 AM     183808 accwiz.exe
-a---         2004-08-04   8:00 AM      61952 acelpdec.ax
-a---         2004-08-04   8:00 AM     129536 acledit.dll
-a---         2004-08-04   8:00 AM     114688 aclui.dll
-a---         2004-08-04   8:00 AM     194048 activeds.dll
-a---         2004-08-04   8:00 AM     111104 activeds.tlb
-a---         2004-08-04   8:00 AM       4096 actmovie.exe
-a---         2004-08-04   8:00 AM     101888 actxprxy.dll
-a---         2003-02-21   6:50 PM     143150 admgmt.msc
-a---         2006-01-25   3:35 PM      53760 admparse.dll
<SPACE> next page; <CR> next line; Q quit
...
```

Out-Host -Paging 명령은 긴 출력을 천천히 표시하려는 경우에 유용한 파이프라인 요소로, 특히 CPU 를 많이 사용하는 작업에 유용합니다. Out-Host cmdlet 이 전체 페이지를 표시할 준비가 되면 프로세스가 이 cmdlet 에 전달되므로 파이프라인에서 이 cmdlet 뒤에 있는 cmdlet 은 다음 페이지를 출력할 수 있을 때까지 작업을 중지합니다. Windows 작업 관리자에서 Windows PowerShell 이 사용하는 CPU 와 메모리를 모니터링하면 이를 확인할 수 있습니다.

Get-ChildItem C:\Windows -Recurse 명령을 실행한 다음 CPU 및 메모리 사용량을 **Get-ChildItem C:\Windows -Recurse | Out-Host –Paging** 명령과 비교하십시오. 화면에 텍스트가 표시되지만 이것은 개체를 콘솔 창에 텍스트로 표시해야 하기 때문이며 Windows PowerShell 에서 실제로 진행되고 있는 작업을 보여 주기 위한 것일 뿐입니다. 예를 들어 현재 위치가 C 드라이브의 루트일 때 **Get-Location** 을 입력하면 다음과 같은 내용이 출력됩니다.

```
PS> Get-Location

Path
----
C:\
```

Windows PowerShell 에서 파이프라인으로 결합한 **Get-Location | Out-Host** 와 같은 명령을 실행하면 일련의 문자가 화면에 표시되는 순서대로 **Get-Location** 에서 **Out-Host** 로 전달됩니다. 즉, 머리글 정보를 무시할 경우 **Out-Host** 에 **'C'**

문자, ':' 문자 및 '₩' 문자가 차례로 전달됩니다. **Out-Host** cmdlet 으로는 **Get-Location** cmdlet 이 출력하는 문자에 연결된 의미를 알 수 없습니다.

Windows PowerShell 은 텍스트 대신 개체를 사용하여 파이프라인에 있는 명령과 통신합니다. 사용자의 관점에서 볼 때 개체는 관련 정보를 하나의 단위로 쉽게 조작할 수 있는 형식으로 패키징하고 필요한 특정 항목을 추출합니다.

Get-Location 명령은 현재 경로가 포함된 텍스트를 반환하지 않고 현재 경로와 함께 몇 가지 다른 정보가 포함된 **PathInfo** 라는 정보 패키지를 반환합니다. 그러면 **Out-Host** cmdlet 이 이 **PathInfo** 개체를 화면에 보내고 Windows PowerShell 은 형식 지정 규칙에 따라 표시할 정보와 이러한 정보의 표시 방법을 결정합니다.

사실 **Get-Location** cmdlet 이 출력하는 머리글 정보는 화면에 표시할 데이터의 형식을 지정하는 프로세스의 일부로 해당 프로세스의 끝에만 추가됩니다. 화면에는 출력 개체에 대한 전체 정보 대신 요약 정보가 표시됩니다.

Windows PowerShell 명령이 출력하는 정보가 콘솔 창에 표시되는 정보보다 많을 경우 콘솔 창에 표시되지 않는 요소를 검색하거나 추가 데이터를 표시할 수 있으며 특정 Windows PowerShell 이 일반적으로 사용하는 것과 다른 형식으로 데이터를 표시할 수도 있습니다.

이 장의 나머지 부분에서는 특정 Windows PowerShell 개체의 구조를 검색하는 방법과 이러한 정보를 파일이나 프린터와 같은 다른 출력 위치로 보내는 방법을 설명합니다.

개체 구조 보기(Get-Member)

Windows PowerShell 에서 개체는 중요한 역할을 하므로 임의의 개체 유형에 사용할 여러 개의 기본 명령이 설계되어 있습니다. 이러한 명령 중 가장 중요한 명령은 **Get-Member** 입니다.

명령이 반환하는 개체를 분석하는 가장 간단한 방법은 명령의 출력을 **Get-Member** cmdlet 에 파이프하는 것입니다. **Get-Member** cmdlet 은 개체 유형의 정식 이름과 해당 멤버의 전체 목록을 보여 줍니다. 이때 매우 많은 요소가 반환될 수도 있습니다. 예를 들어 프로세스 개체에는 100 개 이상의 멤버가 있습니다.

프로세스 개체의 멤버를 모두 표시하고 출력을 페이징하여 해당 내용을 모두 표시하려면 다음과 같이 입력하십시오.

```
PS> Get-Process | Get-Member | Out-Host -Paging
```

이 명령을 실행하면 다음과 같은 내용이 출력됩니다.

```
   TypeName: System.Diagnostics.Process

Name                        MemberType     Definition
----                        ----------     ----------
Handles                     AliasProperty  Handles = Handlecount
Name                        AliasProperty  Name = ProcessName
NPM                         AliasProperty  NPM = NonpagedSystemMemorySize
PM                          AliasProperty  PM = PagedMemorySize
VM                          AliasProperty  VM = VirtualMemorySize
WS                          AliasProperty  WS = WorkingSet
add_Disposed                Method         System.Void add_Disposed(Event...
...
```

이 긴 정보 목록은 원하는 요소만 표시되도록 필터링하여 보다 간단하게 만들 수 있습니다. **Get-Member** 명령을 사용하면 속성 멤버만 표시할 수 있습니다. 속성에는 여러 가지 유형이 있는데, **Get-MemberMemberType** 매개 변수를 값 **Properties** 로 설정하면 이 cmdlet 이 모든 유형의 속성을 표시합니다. 결과 목록은 다음과 같이 아직도 매우 길지만 읽기가 조금 더 쉬워집니다.

```
PS> Get-Process | Get-Member -MemberType Properties


   TypeName: System.Diagnostics.Process

Name                       MemberType     Definition
----                       ----------     ----------
Handles                    AliasProperty  Handles = Handlecount
Name                       AliasProperty  Name = ProcessName
...
ExitCode                   Property       System.Int32 ExitCode {get;}
...
Handle                     Property       System.IntPtr Handle {get;}
...
CPU                        ScriptProperty System.Object CPU {get=$this.Total...
...
Path                       ScriptProperty System.Object Path {get=$this.Main...
...
```

참고:

MemberType 에 사용할 수 있는 값으로는 AliasProperty, CodeProperty, Property, NoteProperty, ScriptProperty, Properties, PropertySet, Method, CodeMethod, ScriptMethod, Methods, ParameterizedProperty, MemberSet 및 All 이 있습니다.

프로세스에는 60 개 이상의 속성이 있습니다. Windows PowerShell 이 종종 잘 알려진 개체의 속성을 일부만 표시하는 이유는 속성을 모두 표시하면 관리할 수 없는 정보의 양이 많아지기 때문입니다.

참고:

Windows PowerShell 은 이름이 .format.ps1xml 로 끝나는 XML 파일에 저장된 정보를 사용하여 개체의 표시 방법을 결정합니다. 예를 들어 .NET System.Diagnostics.Process 개체인 프로세스 개체의 서식 데이터는 PowerShellCore.format.ps1xml 에 저장됩니다.

Windows PowerShell 이 기본적으로 표시하는 속성이 아닌 다른 속성을 보려면 출력 데이터의 형식을 사용자가 직접 지정해야 합니다. 이렇게 하려면 format cmdlet 을 사용하면 됩니다.

형식 명령을 사용하여 출력 보기 변경

Windows PowerShell 에는 특정 개체에 대해 표시할 속성을 제어할 수 있는 일련의 cmdlet 이 있습니다. 이러한 cmdlet 의 이름은 모두 동사 **Format** 으로 시작되며, 표시할 속성을 하나 이상 선택하는 데 사용할 수 있습니다.

Format cmdlet 으로는 **Format-Wide**, **Format-List**, **Format-Table** 및 **Format-Custom** 이 있습니다. 이 사용 설명서에서는 **Format-Wide**, **Format-List** 및 **Format-Table** cmdlet 에 대해서만 설명합니다.

각 형식 cmdlet 에는 표시할 특정 속성을 지정하지 않을 경우 사용될 기본 속성이 있습니다. 각 cmdlet 은 또한 동일한 매개 변수 이름인 **Property** 를 사용하여 표시할 속성을 지정합니다. **Format-Wide** 는 하나의 속성만 표시하기 때문에 해당 **Property** 속성에는 하나의 값만 있지만 **Format-List** 및 **Format-Table** 의 속성 매개 변수는 여러 개의 속성 이름을 받아들입니다.

실행 중인 두 개의 Windows PowerShell 인스턴스에 **Get-Process -Name powershell** 명령을 사용하면 다음과 같은 내용이 출력됩니다.

```
Handles  NPM(K)    PM(K)      WS(K) VM(M)   CPU(s)     Id ProcessName
-------  ------    -----      ----- -----   ------     -- -----------
    995       9    30308      27996   152     2.73   2760 powershell
    331       9    23284      29084   143     1.06   3448 powershell
```

이 절의 나머지 부분에서는 **Format** cmdlet 을 사용하여 이 명령 출력의 표시 방법을 변경하는 방법을 설명합니다.

Format-Wide 를 사용하여 단일 항목 출력

기본적으로 **Format-Wide** cmdlet 은 개체의 기본 속성만 표시합니다. 다음과 같이 각 개체와 연결된 정보는 하나의 열에 표시됩니다.

```
PS> Get-Process -Name powershell | Format-Wide

powershell                                powershell
```

다음과 같이 기본 속성이 아닌 속성을 지정할 수도 있습니다.

```
PS> Get-Process -Name powershell | Format-Wide -Property Id

2760                                      3448
```

열이 포함된 Format-Wide 표시 제어

Format-Wide cmdlet 을 사용하면 한 번에 하나의 속성만 표시할 수 있습니다. 이 cmdlet 은 한 줄에 하나의 요소만 표시되는 간단한 목록을 표시하는 데 유용합니다. 간단한 목록을 표시하려면 다음과 같이 입력하여 **Column** 매개 변수의 값을 1 로 설정하십시오.

```
Get-Command Format-Wide -Property Name -Column 1
```

Format-List 를 사용하여 목록 보기

Format-List cmdlet 은 다음과 같이 각 속성에 레이블이 지정되어 있고 이러한 각 속성이 별도의 줄에 표시되는 목록 형식으로 개체를 표시합니다.

```
PS> Get-Process -Name powershell | Format-List
```

```
Id      : 2760
Handles : 1242
CPU     : 3.03125
Name    : powershell

Id      : 3448
Handles : 328
CPU     : 1.0625
Name    : powershell
```

다음과 같이 속성을 원하는 대로 지정할 수 있습니다.

```
PS> Get-Process -Name powershell | Format-List -Property ProcessName,FileVersion
,StartTime,Id


ProcessName : powershell
FileVersion : 1.0.9567.1
StartTime   : 2006-05-24 13:42:00
Id          : 2760

ProcessName : powershell
FileVersion : 1.0.9567.1
StartTime   : 2006-05-24 13:54:28
Id          : 3448
```

와일드카드와 함께 Format-List 를 사용하여 자세한 정보 보기

Format-List cmdlet 을 사용하면 와일드카드를 해당 **Property** 매개 변수의 값으로 사용할 수 있습니다. 이 경우 자세한 정보를 표시할 수 있습니다. 필요한 것보다 많은 정보가 개체에 포함되는 경우가 종종 있는데, 이것은 Windows PowerShell 이 기본적으로 모든 속성 값을 표시하지 않기 때문입니다. 개체의 속성을 모두 표시하려면 **Format-List -Property *** 명령을 사용하십시오. 다음 명령은 단일 프로세스의 출력을 위해 60 개 이상의 줄을 생성합니다.

```
Get-Process -Name powershell | Format-List -Property *
```

Format-List 명령은 자세한 정보를 표시하는 데 유용하지만 많은 항목의 개요를 출력하려는 경우에는 대개 간단한 표 형식의 보기가 더 유용합니다.

Format-Table 을 사용하여 표 형식으로 출력

속성 이름이 지정되지 않은 **Format-Table** cmdlet 을 사용하여 **Get-Process** 명령의 출력 형식을 지정하면 형식을 지정하지 않고 이와 같은 작업을 수행할 때와 똑같은 내용이 출력됩니다. 그 이유는 대부분의 Windows PowerShell 개체와 마찬가지로 프로세스도 대개 표 형식으로 표시되기 때문입니다.

```
PS> Get-Process -Name powershell | Format-Table

Handles  NPM(K)    PM(K)      WS(K) VM(M)   CPU(s)     Id ProcessName
```

```
-------  ------    -----      -----  -----   ------     -- -----------
   1488       9    31568      29460    152     3.53   2760 powershell
    332       9    23140        632    141     1.06   3448 powershell
```

Format-Table 출력 향상(AutoSize)

표 형식의 보기는 비교 가능한 많은 정보를 표시하는 데 유용하지만 열이 너무 좁아 데이터를 모두 표시할 수 없는 경우에는 정보를 해석하기 어려울 수 있습니다. 예를 들어 프로세스의 Path, ID, Name 및 Company 를 표시하려고 하면 다음과 같이 프로세스의 Path 열과 Company 열의 출력이 잘립니다.

```
PS> Get-Process -Name powershell | Format-Table -Property Path,Name,Id,Company

Path                Name                                Id Company
----                ----                                -- -------
C:\Program Files... powershell                        2836 Microsoft Corpor...
```

Format-Table 명령을 실행할 때 **AutoSize** 매개 변수를 지정하면 Windows PowerShell 은 표시할 실제 데이터를 기반으로 열 너비를 계산합니다. 이렇게 하면 다음과 같이 **Path** 열은 읽을 수 있지만 Company 열은 여전히 잘린 채로 있습니다.

```
PS> Get-Process -Name powershell | Format-Table -Property Path,Name,Id,Company -
AutoSize

Path                                                 Name         Id Company
----                                                 ----         -- -------
C:\Program Files\Windows PowerShell\v1.0\powershell.exe powershell 2836 Micr...
```

Format-Table cmdlet 을 실행하면 여전히 데이터가 잘릴 수 있지만 화면 끝에 있는 데이터만 잘립니다. 마지막으로 표시되는 속성을 제외하고 속성에는 가장 긴 데이터 요소를 올바로 표시하는 데 필요한 충분한 크기가 지정됩니다. **Property** 값 목록에서 **Path** 와 **Company** 의 위치를 바꾸면 회사 이름은 보이지만 경로는 잘립니다.

```
PS> Get-Process -Name powershell | Format-Table -Property Company,Name,Id,Path -
AutoSize

Company               Name         Id Path
-------               ----         -- ----
Microsoft Corporation powershell 2836 C:\Program Files\Windows PowerShell\v1...
```

Format-Table 명령은 속성 목록의 시작 부분에 더 가까이 있을수록 더 중요한 속성으로 간주하고 시작 부분에 가장 가까이 있는 속성을 완전히 표시하려고 합니다. **Format-Table** 명령은 속성을 모두 표시할 수 없으면 일부 열을 표시하지 않는 대신 경고를 표시합니다. 다음과 같이 **Name** 을 목록의 마지막 속성으로 지정하면 이 동작을 재현할 수 있습니다.

```
PS> Get-Process -Name powershell | Format-Table -Property Company,Path,Id,Name -
AutoSize

WARNING: column "Name" does not fit into the display and was removed.

Company               Path                                                  I
                                                                            d
-------               ----                                                  -
```

```
Microsoft Corporation C:\Program Files\Windows PowerShell\v1.0\powershell.exe 6
```

위의 출력에서는 목록에 맞게 조정하기 위해 ID 열이 잘리고 해당 열 머리글이 두 줄로 표시됩니다. 열 크기를 자동으로 조정해도 항상 원하는 대로 표시되지는 않습니다.

열에 Format-Table 출력 래핑(Wrap)

Wrap 매개 변수를 사용하여 긴 **Format-Table** 데이터를 해당 열 안에서 래핑할 수 있습니다. 다음과 같이 **Wrap** 매개 변수를 단독으로 사용하면 **AutoSize** 를 지정하지 않은 경우에도 예상되는 작업을 수행하지 않아도 됩니다.

```
PS> Get-Process -Name powershell | Format-Table -Wrap -Property Name,Id,Company,
Path

Name                                Id Company             Path
----                                -- -------             ----
powershell                        2836 Microsoft Corporati C:\Program Files\Wi
                                       on                  ndows PowerShell\v1
                                                           .0\powershell.exe
```

Wrap 매개 변수를 단독으로 사용하면 프로세스 속도가 크게 느려지지 않는다는 이점이 있습니다. 큰 디렉터리 시스템의 파일을 반복해서 표시할 때 **AutoSize** 를 사용하면 첫 번째 출력 항목을 표시할 때까지 매우 많은 시간이 걸리고 많은 메모리가 사용될 수 있습니다.

시스템 로드에 신경 쓰지 않아도 되는 경우에는 **AutoSize** 가 **Wrap** 매개 변수와 함께 잘 작동합니다. **Wrap** 매개 변수 없이 **AutoSize** 를 지정한 경우와 같이 초기 열에는 항상 하나의 줄에 항목을 표시하는 데 필요한 충분한 너비가 할당됩니다. 유일한 차이점은 다음과 같이 마지막 열이 필요에 따라 래핑된다는 것입니다.

```
PS> Get-Process -Name powershell | Format-Table -Wrap -AutoSize -Property Name,I
d,Company,Path

Name          Id Company               Path
----          -- -------               ----
powershell 2836 Microsoft Corporation C:\Program Files\Windows PowerShell\v1.0\
                                      powershell.exe
```

가장 긴 열을 가장 먼저 지정하면 일부 열이 표시되지 않을 수 있으므로 가장 작은 데이터 요소를 가장 먼저 지정하는 것이 안전합니다. 다음 예제에서는 매우 긴 경로 요소를 가장 먼저 지정했기 때문에 래핑을 사용했지만 마지막 **Name** 요소가 여전히 표시되지 않습니다.

```
 PS> Get-Process -Name powershell | Format-Table -Wrap -AutoSize -Property Path,I
d,Company,Name

WARNING: column "Name" does not fit into the display and was removed.

Path                                                        Id Company
----                                                        -- -------
C:\Program Files\Windows PowerShell\v1.0\powershell.exe 2836 Microsoft Corporat
                                                             ion
```

표 형식의 출력 구성(-GroupBy)

표 형식의 출력을 제어하는 데 사용할 수 있는 또 다른 유용한 매개 변수는 **GroupBy** 입니다. 특히 긴 표 형식의 목록은 비교하기 어려울 수 있지만 **GroupBy** 매개 변수를 사용하면 속성 값을 기반으로 출력을 그룹화할 수 있습니다. 예를 들어 다음과 같이 프로세스를 더 쉽게 검사하기 위해 속성 목록에서 회사 값을 제거하여 해당 프로세스를 회사별로 그룹화할 수 있습니다.

```
PS> Get-Process -Name powershell | Format-Table -Wrap -AutoSize -Property Name,I
d,Path -GroupBy Company


   Company: Microsoft Corporation

Name         Id Path
----         -- ----
powershell 1956 C:\Program Files\Windows PowerShell\v1.0\powershell.exe
powershell 2656 C:\Program Files\Windows PowerShell\v1.0\powershell.exe
```

Out-* Cmdlet 을 사용하여 데이터 리디렉션

Windows PowerShell 에는 데이터 출력을 직접 제어할 수 있는 여러 가지 cmdlet 이 있습니다. 이러한 cmdlet 은 두 가지 중요한 특성을 공유합니다.

첫째, 일반적으로 이러한 cmdlet 은 데이터를 특정 텍스트 형식으로 변환합니다. 이것은 텍스트 입력을 필요로 하는 시스템 구성 요소에 데이터를 출력하기 위한 것입니다. 즉, 개체를 텍스트로 표시해야 합니다. 따라서 Windows PowerShell 콘솔 창에 표시되는 대로 텍스트의 형식이 지정됩니다.

둘째, 이러한 cmdlet 은 Windows PowerShell 에서 다른 위치로 정보를 내보내기 때문에 Windows PowerShell 의 동사 **Out** 을 사용합니다. **Out-Host** cmdlet 은 항상 호스트 창을 Windows PowerShell 의 외부에 표시합니다. 이것은 Windows PowerShell 에서 데이터를 내보내면 실제로 데이터가 제거되기 때문에 중요합니다. 다음과 같이 데이터를 호스트 창으로 페이징하는 파이프라인을 만든 다음 목록 형식으로 지정해 보면 이 동작을 재현할 수 있습니다.

```
PS> Get-Process | Out-Host -Paging | Format-List
```

이 명령을 실행하면 프로세스 정보 페이지가 목록 형식으로 표시되어야 하지만 다음과 같이 기본 표 형식 목록으로 표시됩니다.

```
Handles  NPM(K)    PM(K)      WS(K) VM(M)   CPU(s)     Id ProcessName
-------  ------    -----      ----- -----   ------     -- -----------
    101       5     1076       3316    32     0.05   2888 alg
...
    618      18    39348      51108   143   211.20    740 explorer
    257       8     9752      16828    79     3.02   2560 explorer
...
<SPACE> next page; <CR> next line; Q quit
...
```

Out-Host cmdlet 은 데이터를 콘솔에 직접 보내기 때문에 **Format-List** 명령에는 서식을 지정할 데이터가 전달되지 않습니다.

이 명령을 구성하는 올바른 방법은 다음과 같이 **Out-Host** cmdlet 을 파이프라인 끝에 배치하는 것입니다. 이렇게 하면 프로세스 데이터가 페이징되어 표시되기 전에 목록으로 형식이 지정됩니다.

```
PS> Get-Process | Format-List | Out-Host -Paging


Id      : 2888
Handles : 101
CPU     : 0.046875
Name    : alg
...

Id      : 740
Handles : 612
CPU     : 211.703125
Name    : explorer

Id      : 2560
Handles : 257
CPU     : 3.015625
Name    : explorer
...
<SPACE> next page; <CR> next line; Q quit
...
```

이러한 내용은 모든 **Out** cmdlet 에 적용되므로 **Out** cmdlet 을 항상 파이프라인 끝에 배치해야 합니다.

참고:

모든 **Out** cmdlet 은 줄 길이 제한과 같은 콘솔 창에 적용되는 형식을 사용하여 출력을 텍스트로 렌더링합니다.

콘솔 출력 페이징(Out-Host)

기본적으로 Windows PowerShell 은 데이터를 호스트 창으로 보내는데, 이것은 Out-Host cmdlet 이 수행하는 작업과 똑같습니다. Out-Host cmdlet 의 주된 용도는 앞에서 설명한 대로 데이터를 페이징하는 것입니다. 예를 들어 다음 명령은 Out-Host 를 사용하여 Get-Command cmdlet 의 출력을 페이징합니다.

```
PS> Get-Command | Out-Host -Paging
```

또한 **more** 함수를 사용하여 데이터를 페이징할 수도 있습니다. Windows PowerShell 에서 **more** 는 **Out-Host –Paging** 을 호출하는 함수입니다. 다음 명령은 **more** 함수를 사용하여 Get-Command 의 출력을 페이징하는 방법을 보여 줍니다.

```
PS> Get-Command | more
```

하나 이상의 파일 이름을 more 함수에 대한 인수로 포함하면 more 함수는 이러한 파일을 읽고 해당 내용을 호스트로 페이징합니다.

```
PS> more c:\boot.ini
```

```
[boot loader]
timeout=5
default=multi(0)disk(0)rdisk(0)partition(1)\WINDOWS
[operating systems]
...
```

출력 삭제(Out-Null)

Out-Null cmdlet 은 수신한 입력을 즉시 삭제하도록 설계되었습니다. 이 cmdlet 은 명령 실행의 부작용으로 수신되는 불필요한 데이터를 삭제하는 데 유용합니다. 다음 명령을 입력하면 명령에서 아무 것도 반환되지 않습니다.

```
PS> Get-Command | Out-Null
```

Out-Null cmdlet 은 오류 출력을 삭제하지 않습니다. 예를 들어 다음 명령을 입력하면 Windows PowerShell 이 'Is-NotACommand'를 인식할 수 없다는 메시지가 나타납니다.

```
PS> Get-Command Is-NotACommand | Out-Null
Get-Command : 'Is-NotACommand'는 cmdlet, 함수, 실행할 수 있는 프로그램 또는 스크립트 파일로
인식되지 않습니다.
줄:1 문자:12
+ Get-Command  <<<< Is-NotACommand | Out-Null
```

데이터 인쇄(Out-Printer)

Out-Printer cmdlet 을 사용하여 데이터를 인쇄할 수 있습니다. 프린터 이름을 제공하지 않으면 **Out-Printer** cmdlet 은 기본 프린터를 사용합니다. 표시 이름만 지정하면 아무 Windows 기반 컴퓨터나 사용할 수 있습니다. 프린터 포트 매핑이나 심지어 실제 프린터도 필요하지 않습니다. 예를 들어 Microsoft Office Document Imaging 도구가 설치되어 있으면 다음과 같이 입력하여 데이터를 이미지 파일로 보낼 수 있습니다.

```
PS> Get-Command Get-Command | Out-Printer -Name "Microsoft Office Document Image
Writer"
```

데이터 저장(Out-File)

Out-File cmdlet 을 사용하여 출력을 콘솔 창이 아니라 파일로 보낼 수 있습니다. 다음 명령줄은 프로세스 목록을 **C:\temp\processlist.txt** 로 보냅니다.

```
PS> Get-Process | Out-File -FilePath C:\temp\processlist.txt
```

이전의 출력 리디렉션에 익숙한 경우 **Out-File** cmdlet 을 사용하면 알고 있는 것과 다른 결과가 나타날 수 있습니다. 이 동작을 이해하려면 **Out-File** cmdlet 이 작동하는 컨텍스트를 알고 있어야 합니다.

기본적으로 **Out-File** cmdlet 은 유니코드 파일을 만듭니다. 이 형식은 최선의 기본 출력 형식이지만 이 기본 출력 형식을 사용할 경우 ASCII 파일을 출력하는 도구가 올바로 작동하지 않습니다. 다음과 같이 **Encoding** 매개 변수를 사용하면 기본 출력 형식을 ASCII 로 변경할 수 있습니다.

```
PS> Get-Process | Out-File -FilePath C:\temp\processlist.txt -Encoding ASCII
```

Out-file 은 콘솔에 표시되는 것처럼 파일 내용의 형식을 지정하기 때문에 대부분의 경우 콘솔 창에서처럼 출력이 잘립니다. 예를 들어 다음 명령을 실행할 수 있습니다.

```
PS> Get-Command | Out-File -FilePath c:\temp\output.txt
```

그러면 다음과 같은 내용이 출력됩니다.

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

출력할 때 화면 너비에 맞추기 위해 강제로 줄을 바꾸지 않으려면 **Width** 매개 변수를 사용하여 줄 너비를 지정하면 됩니다. 32 비트 정수 매개 변수이기 때문에 **Width** 에 지정할 수 있는 최대값은 2147483647 입니다. 줄 길이를 최대값으로 설정하려면 다음과 같이 입력하십시오.

```
Get-Command | Out-File -FilePath c:\temp\output.txt -Width 2147483647
```

Out-File cmdlet 은 콘솔에 표시된 상태로 출력을 저장하려는 경우에 가장 유용합니다. 출력 형식을 보다 자세히 제어하려면 고급 도구가 필요합니다. 이러한 도구에 대해서는 다음 장에서 개체 조작에 대해 설명할 때 함께 다룹니다.

Windows PowerShell 탐색

폴더는 Windows 탐색기 인터페이스, Cmd.exe 및 BASH 같은 UNIX 도구에 사용되는 친숙한 관리 기능입니다. 폴더나 디렉터리는 많이 알려져 있는 것처럼 파일과 기타 디렉터리를 관리하는 데 유용한 개념입니다. UNIX 계열 운영 체제에서는 이 개념을 확장하여 가능한 모든 것들을 파일로 취급합니다. 예를 들어 특정 하드웨어와 네트워크 연결을 특정 폴더 안에 파일로 표시합니다. 이 방법을 사용하면 일부 응용 프로그램에서 파일 또는 폴더의 내용을 읽거나 사용하지 못할 수 있지만 원하는 항목을 손쉽게 찾을 수 있습니다. 파일과 폴더를 열거하거나 검색하는 도구는 이러한 장치에서도 작동합니다. 또한 특정 항목을 나타내는 파일의 경로를 사용하여 해당 항목을 지정할 수도 있습니다.

마찬가지로 Windows PowerShell 인프라는 표준 Microsoft Windows 디스크 드라이브나 UNIX 파일 시스템과 같이 탐색 가능한 모든 것들을 가상 Windows PowerShell 드라이브로 표시하는 것을 지원합니다. Windows PowerShell 드라이브는 로컬 드라이브나 네트워크 드라이브와 같은 실제 드라이브가 아닌 드라이브를 나타낼 수 있습니다. 이 장에서는 주로 파일 시스템의 탐색에 대해 설명하지만 그 내용은 파일 시스템과 연결되어 있지 않은 Windows PowerShell 드라이브에 적용됩니다.

Windows Powershell 에서 현재 위치 관리

Windows 탐색기에서 폴더 시스템을 탐색할 때는 일반적으로 현재 열려 있는 폴더라는 특정 작업 위치를 사용합니다. 현재 폴더에 있는 항목은 마우스를 사용하여 손쉽게 조작할 수 있습니다. Cmd.exe 와 같은 명령줄 인터페이스의 경우, 특정

폴더와 같은 폴더에 있을 때는 전체 파일 경로를 지정하는 대신 약식 이름을 지정하여 해당 파일에 액세스할 수 있습니다. 현재 디렉터리는 작업 디렉터리라고 합니다.

Windows PowerShell 은 명사 **Location** 을 사용하여 작업 디렉터리를 참조하고 cmdlet 계열을 구현하여 현재 위치를 검사하고 조작합니다.

현재 위치 보기(Get-Location)

현재 디렉터리 위치의 경로를 확인하려면 다음과 같이 **Get-Location** 명령을 입력하십시오.

```
PS> Get-Location
Path
----
C:\Documents and Settings\PowerUser
```

참고:

Get-Location cmdlet 은 BASH 셸의 pwd 명령과 유사하고 Set-Location cmdlet 은 Cmd.exe 의 cd 명령과 유사합니다.

현재 위치 설정(Set-Location)

Get-Location 명령은 **Set-Location** 명령과 함께 사용됩니다. **Set-Location** 명령을 사용하면 현재 디렉터리 위치를 지정할 수 있습니다.

```
PS> Set-Location -Path C:\Windows
```

이 명령을 입력해도 아무 내용도 출력되지 않는데, 작업을 수행하는 대부분의 Windows PowerShell 명령은 경우에 따라 유용하지 않을 수 있으므로 거의 아무 내용도 출력하지 않습니다. **Set-Location** 명령을 입력한 후 디렉터리가 성공적으로 변경되었는지 확인하려면 다음과 같이 **Set-Location** 명령을 입력할 때 **-PassThru** 매개 변수를 포함하십시오.

```
PS> Set-Location -Path C:\Windows -PassThru
Path
----
C:\WINDOWS
```

-PassThru 매개 변수는 Windows PowerShell 에서 명령을 실행할 때 기본적으로 출력되는 내용이 없을 경우 명령의 실행 결과에 대한 정보를 반환하기 위해 다양한 Set 명령과 함께 사용될 수 있습니다.

대부분의 UNIX 및 Windows 명령 셸에서와 동일한 방식으로 현재 위치에 상대적인 경로를 지정할 수 있습니다. 상대 경로에 대한 표준 표기법에서 마침표(**.**)는 현재 폴더를, 이중 마침표(**..**)는 현재 위치의 부모 디렉터리를 나타냅니다.

예를 들어 **C:\Windows** 폴더에 있을 경우 마침표(**.**)는 **C:\Windows** 를, 이중 마침표는(**..**) **C:**를 나타냅니다. 다음과 같이 입력하면 현재 위치를 C: 드라이브의 루트로 변경할 수 있습니다.

```
PS> Set-Location -Path .. -PassThru

Path

----
```

```
C:\
```

위와 같은 방법을 **HKLM:**과 같이 파일 시스템 드라이브가 아닌 Windows PowerShell 드라이브에서도 사용할 수 있습니다. 다음과 같이 입력하면 현재 위치를 레지스트리의 HKLM\Software 키로 설정할 수 있습니다.

```
 PS> Set-Location -Path HKLM:\SOFTWARE -PassThru

Path
----
HKLM:\SOFTWARE
```

그런 다음 아래와 같이 상대 경로를 사용하여 디렉터리 위치를 Windows PowerShell HKLM: 드라이브의 루트인 부모 디렉터리로 변경할 수 있습니다.

```
PS> Set-Location -Path .. -PassThru

Path
----
HKLM:\
```

다음과 같이 Set-Location 을 사용하거나 Set-Location 에 대한 Windows PowerShell 의 기본 제공 별칭(cd, chdir, sl)을 사용할 수 있습니다.

```
cd -Path C:\Windows
```

```
chdir -Path .. -PassThru
```

```
sl -Path HKLM:\SOFTWARE -PassThru
```

최근 위치 저장 및 다시 불러오기(Push-Location 및 Pop-Location)

지나온 경로를 추적하고 이전 위치로 돌아갈 수 있으면 위치를 변경할 때 도움이 됩니다. Windows PowerShell 의 **Push-Location** cmdlet 이 지나온 디렉터리 경로의 순서가 지정된 기록("스택")을 만들면 **Push-Location** cmdlet 과 보완 관계에 있는 **Pop-Location** cmdlet 을 사용하여 디렉터리 경로의 기록을 통해 이전 위치로 돌아갈 수 있습니다.

예를 들어 Windows PowerShell 세션은 대개 사용자의 홈 디렉터리에서 시작됩니다.

```
 PS> Get-Location

Path
----
C:\Documents and Settings\PowerUser
```

참고:

스택이라는 단어는 .NET 을 포함하여 다양한 프로그래밍 설정에서 특별한 의미를 지닙니다. 실제로 물건을 쌓아놓은 것과 마찬가지로 스택에 마지막으로 올려놓은 항목은 스택에서 내려놓을 수 있는 첫 번째 항목이 됩니다. 일반적으로 스택에 항목을 추가하는 것을 스택에 항목을 "올려놓는다"라고 하고 스택에서 항목을 제거하는 것을 스택에서 항목을 "내려놓는다"라고 합니다.

스택에 현재 위치를 올려놓은 다음 Local Settings 폴더로 이동하려면 다음과 같이 입력하십시오.

```
PS> Push-Location -Path "Local Settings"
```

그런 다음 스택에 Local Settings 위치를 올려놓은 후 다음과 같이 입력하여 Temp 폴더로 이동할 수 있습니다.

```
PS> Push-Location -Path Temp
```

다음과 같이 **Get-Location** 명령을 입력하여 디렉터리가 변경되었는지 확인할 수 있습니다.

```
PS> Get-Location

Path
----
C:\Documents and Settings\PowerUser\Local Settings\Temp
```

그런 다음 **Pop-Location** 명령을 입력하여 가장 최근에 방문한 디렉터리로 돌아간 후 **Get-Location** 명령을 입력하여 디렉터리가 변경되었는지 확인할 수 있습니다.

```
PS> Pop-Location
PS> Get-Location

Path
----
C:\Documents and Settings\me\Local Settings
```

Set-Location cmdlet 을 사용할 때와 마찬가지로 **Pop-Location** cmdlet 을 입력할 때 **-PassThru** 매개 변수를 포함하여 현재 위치한 디렉터리를 표시할 수 있습니다.

```
PS> Pop-Location -PassThru

Path
----
C:\Documents and Settings\PowerUser
```

또한 Location cmdlet 을 네트워크 경로와 함께 사용할 수도 있습니다. 예를 들어 Public 이라는 공유가 있고 이름이 FS01 인 서버가 현재 위치일 경우 다음과 같이 입력하여 이 위치를 변경할 수 있습니다.

```
Set-Location \\FS01\Public
```

또는

```
Push-Location \\FS01\Public
```

Push-Location 및 **Set-Location** 명령을 사용하여 현재 위치를 사용 가능한 아무 드라이브로나 변경할 수 있습니다. 예를 들어 드라이브 문자가 D 인 로컬 CD-ROM 에 데이터 CD 가 들어 있는 경우 **Set-Location D:** 명령을 입력하여 현재 위치를 CD 드라이브로 변경할 수 있습니다.

드라이브가 비어 있으면 다음과 같은 오류 메시지가 나타납니다.

```
PS> Set-Location D:
Set-Location : 'D:\' 경로는 존재하지 않으므로 찾을 수 없습니다.
```

명령줄 인터페이스를 사용하는 경우 Windows 탐색기를 사용하여 사용 가능한 실제 드라이브를 찾기가 쉽지 않습니다. 또한 Windows 탐색기에 일부 Windows PowerShell 드라이브가 표시되지 않을 수도 있습니다. Windows PowerShell 에는 Windows PowerShell 드라이브를 조작할 수 있는 일련의 명령이 포함되어 있습니다. 이러한 명령에 대해서는 다음 장에서 자세히 설명합니다.

Windows PowerShell 드라이브 관리

Windows PowerShell 드라이브는 Windows Powershell 에서 파일 시스템 드라이브처럼 액세스할 수 있는 데이터 저장소 위치입니다. Windows PowerShell 드라이브에는 파일 시스템 드라이브(C: 및 D:), 레지스트리 드라이브(HKCU: 및 HKLM:) 및 인증서 드라이브(Cert:)와 같이 Windows PowerShell 공급자가 자동으로 만드는 드라이브와 사용자가 직접 만드는 드라이브가 있습니다. 이러한 드라이브는 매우 유용하지만 Windows PowerShell 내에서만 사용할 수 있고 Windows 탐색기나 Cmd.exe 와 같은 다른 Windows 도구를 사용하여 액세스할 수 없습니다.

Windows PowerShell 에서는 Windows PowerShell 드라이브에 사용되는 명령에 명사 **PSDrive** 를 사용합니다. 현재 세션에 있는 Windows PowerShell 드라이브의 목록을 보려면 **Get-PSDrive** cmdlet 을 사용하십시오.

```
PS> Get-PSDrive

Name     __ Provider      Root                                 CurrentLocation
----       --------      ----                                 ---------------
A           FileSystem    A:\
Alias       Alias
C           FileSystem    C:\                               ...And Settings\me
cert        Certificate   \
D           FileSystem    D:\
Env         Environment
Function    Function
HKCU        Registry      HKEY_CURRENT_USER
HKLM        Registry      HKEY_LOCAL_MACHINE
Variable    Variable
```

위에 나온 드라이브는 사용자 시스템의 드라이브와 다르지만 **Get-PSDrive** 명령이 출력하는 내용은 위와 유사합니다.

파일 시스템 드라이브는 Windows PowerShell 드라이브의 하위 집합입니다. 파일 시스템 드라이브는 Provider 열의 FileSystem 항목으로 식별할 수 있습니다. Windows PowerShell 의 파일 시스템 드라이브는 PowerShell 파일 시스템 공급자에 의해 지원됩니다.

Get-PSDrive cmdlet 의 구문을 보려면 다음과 같이 **Syntax** 매개 변수와 함께 **Get-Command** 명령을 입력하십시오.

```
PS> Get-Command -Name Get-PSDrive -Syntax
Get-PSDrive [[-Name] <String[]>] [-Scope <String>] [-PSProvider <String[]>] [-V
erbose] [-Debug] [-ErrorAction <ActionPreference>] [-ErrorVariable <String>] [-
OutVariable <String>] [-OutBuffer <Int32>]
```

PSProvider 매개 변수를 사용하면 특정 공급자가 지원하는 Windows PowerShell 드라이브만 표시할 수 있습니다. 예를 들어 Windows PowerShell 파일 시스템 공급자가 지원하는 Windows PowerShell 드라이브만 표시하려면 다음과 같이 **PSProvider** 매개 변수 및 **FileSystem** 값과 함께 **Get-PSDrive** 명령을 입력하십시오.

```
PS> Get-PSDrive -PSProvider FileSystem

Name       Provider      Root                                    CurrentLocation
----       --------      ----                                    ---------------
A          FileSystem    A:\
C          FileSystem    C:\                           ...nd Settings\PowerUser
D          FileSystem    D:\
```

예를 들어 Windows PowerShell 파일 시스템 공급자가 지원하는 Windows PowerShell 드라이브만 표시하려면 다음과 같이 **PSProvider** 매개 변수 및 **FileSystem** 값과 함께 **Get-PSDrive** 명령을 입력하십시오.

```
PS> Get-PSDrive -PSProvider Registry


Name       Provider      Root                                    CurrentLocation

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

HKCU       Registry      HKEY_CURRENT_USER

HKLM       Registry      HKEY_LOCAL_MACHINE
```

또한 다음과 같이 Windows PowerShell 드라이브에 표준 Location cmdlet 을 사용할 수도 있습니다.

```
PS> Set-Location HKLM:\SOFTWARE

PS> Push-Location .\Microsoft

PS> Get-Location


Path

----

HKLM:\SOFTWARE\Microsoft
```

새 Windows PowerShell 드라이브 추가(New-PSDrive)

New-PSDrive 명령을 사용하여 사용자 고유의 Windows PowerShell 드라이브를 추가할 수 있습니다. **New-PSDrive** 명령의 구문을 보려면 **Syntax** 매개 변수와 함께 **Get-Command** 명령을 입력하십시오.

```
PS> Get-Command -Name New-PSDrive -Syntax
New-PSDrive [-Name] <String> [-PSProvider] <String> [-Root] <String> [-Description <String>] [-Scope <String>] [-Credential <PSCredential>] [-Verbose] [-Debug] [-ErrorAction <ActionPreference>] [-ErrorVariable <String>] [-OutVariable <String>] [-OutBuffer <Int32>] [-WhatIf] [-Confirm]
```

새 Windows PowerShell 드라이브를 만들려면 다음과 같은 세 개의 매개 변수를 제공해야 합니다.

- 드라이브 이름(임의의 유효한 Windows PowerShell 이름을 사용할 수 있음)
- PSProvider(파일 시스템 위치의 경우 "FileSystem"을 사용하고 레지스트리 위치의 경우 "Registry"를 사용함)
- 루트, 즉 새 드라이브의 루트 경로

예를 들어 사용자 시스템에서 **C:\Program Files\Microsoft Office\OFFICE11** 과 같이 Microsoft Office 응용 프로그램이 들어 있는 폴더에 매핑되는 "Office"라는 드라이브를 만들 수 있습니다. 이 드라이브를 만들려면 다음 명령을 입력하십시오.

```
PS> New-PSDrive -Name Office -PSProvider FileSystem -Root "C:\Program Files\Microsoft Office\OFFICE11"

Name       Provider      Root                                    CurrentLocation
----       --------      ----                                    ---------------
Office     FileSystem    C:\Program Files\Microsoft Offic...
```

참고:

일반적으로 경로는 대/소문자를 구분하지 않습니다.

모든 Windows PowerShell 드라이브와 마찬가지로 새 Windows PowerShell 드라이브는 뒤에 콜론(:)이 오는 이름으로 참조됩니다.

Windows PowerShell 드라이브에서는 다양한 작업을 훨씬 더 쉽게 수행할 수 있습니다. 예를 들어 Windows 레지스트리에 있는 가장 중요한 키 중 일부는 경로가 매우 길어 액세스하거나 기억하기 어려울 수 있습니다. 즉, **HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion** 아래에는 중요한 구성 정보가 있습니다. CurrentVersion 레지스트리 키에 있는 항목을 보고 변경하려면 다음과 같이 입력하여 이 키를 루트로 하는 Windows PowerShell 드라이브를 만들면 됩니다.

```
PS> New-PSDrive -Name cvkey -PSProvider Registry -Root HKLM\Software\Microsoft\Windows\CurrentVersion

Name       Provider      Root                                    CurrentLocation
----       --------      ----                                    ---------------
cvkey      Registry      HKLM\Software\Microsoft\Windows\...
```

그런 다음 아래와 같이 입력하여 이 위치를 다른 드라이브와 마찬가지로 **cvkey:** 드라이브로 변경합니다.

```
PS> cd cvkey:
```

또는

```
PS> Set-Location cvkey: -PassThru


Path

----

cvkey:\
```

New-PsDrive cmdlet 은 현재 콘솔 세션에만 새 드라이브를 추가합니다. 콘솔을 종료하거나 Windows PowerShell 창을 닫으면 새 드라이브는 손실됩니다. Windows PowerShell 드라이브를 저장하려면 Export-Console cmdlet 을 사용하여 현재 콘솔을 내보낸 다음 PowerShell.exe 의 PSConsoleFile 매개 변수를 사용하여 새 세션으로 가져오십시오. 또는 Windows PowerShell 프로필에 새 드라이브를 추가하십시오.

Windows PowerShell 드라이브 삭제(Remove-PSDrive)

Remove-PSDrive cmdlet 을 사용하여 Windows PowerShell 에서 드라이브를 삭제할 수 있습니다. **Remove-PSDrive** cmdlet 은 사용이 간편하기 때문에 특정 Windows PowerShell 드라이브를 삭제하려면 Windows PowerShell 드라이브 이름만 제공하면 됩니다.

예를 들어 **New-PSDrive** 항목에서 설명한 대로 Office: Windows PowerShell 드라이브를 추가한 경우 다음과 같이 입력하여 이 드라이브를 삭제할 수 있습니다.

```
PS> Remove-PSDrive -Name Office
```

cvkey: Windows PowerShell 드라이브를 삭제하려면 마찬가지로 **New-PSDrive** 항목에서 설명한 대로 다음 명령을 사용하십시오.

```
PS> Remove-PSDrive -Name cvkey
```

Windows PowerShell 드라이브는 쉽게 삭제할 수 있지만 해당 드라이브가 현재 위치인 경우에는 삭제할 수 없습니다. 예를 들면 다음과 같습니다.

```
PS> cd office:
PS Office:\> remove-psdrive -name office
Remove-PSDrive : 'Office' 드라이브는 사용 중이므로 제거할 수 없습니다.
줄:1 문자:15
+ remove-psdrive  <<<< -name office
```

Windows PowerShell 외부의 드라이브 추가 또는 제거

Windows PowerShell 은 매핑된 네트워크 드라이브, 연결된 USB 드라이브 및 **net use** 명령 또는 WSH(Windows 스크립트 호스트) 스크립트의 **WScript.NetworkMapNetworkDrive** 및 **RemoveNetworkDrive** 메서드를 사용하여 삭제한 드라이브를 포함하여 Windows 에서 추가하거나 제거한 파일 시스템 드라이브를 검색합니다.

파일, 폴더 및 레지스트리 키 작업 수행

Windows PowerShell 은 명사 **Item** 을 사용하여 Windows PowerShell 드라이브에서 찾은 항목을 참조합니다. Windows PowerShell 파일 시스템 공급자에서 사용할 경우 **Item** 은 파일, 폴더 또는 Windows PowerShell 드라이브일 수 있습니다. 이러한 항목을 표시하고 사용하는 작업은 대부분의 관리 설정에서 중요한 기본 작업이므로 이 설명서에서 자세히 설명합니다.

파일, 폴더 및 레지스트리 키 열거(Get-ChildItem)

특정 위치에서 항목 컬렉션을 가져오는 작업이 일반 작업이 된 후 **Get-ChildItem** cmdlet 은 폴더와 같은 컨테이너 내에서 찾은 모든 항목을 반환하도록 특별히 설계되었습니다.

C:\Windows 폴더 바로 아래에 들어 있는 모든 파일과 폴더를 반환하려면 다음과 같이 입력하십시오.

```
PS> Get-ChildItem -Path C:\Windows
    Directory: Microsoft.Windows PowerShell.Core\FileSystem::C:\Windows
Mode                LastWriteTime     Length Name
----                -------------     ------ ----
-a---         2006-05-16   8:10 AM          0 0.log
-a---         2005-11-29   3:16 PM         97 acc1.txt
-a---         2005-10-23  11:21 PM       3848 actsetup.log
...
```

이 목록은 **Cmd.exe** 에서 **dir** 명령을 입력하거나 UNIX 명령 셸에서 **ls** 명령을 입력한 경우에 표시되는 내용과 유사합니다.

Get-ChildItem cmdlet 의 매개 변수를 사용하면 매우 복잡한 목록 작업을 수행할 수 있습니다. 이 설명서의 뒷부분에는 이러한 복잡한 목록 작업에 대한 몇 가지 시나리오가 나와 있습니다. **Get-ChildItem** cmdlet 의 구문을 보려면 다음과 같이 입력하십시오.

```
PS> Get-Command -Name Get-ChildItem -Syntax
```

고도로 사용자 지정된 출력을 얻기 위해 이러한 매개 변수들은 혼합 및 일치시킬 수 있습니다.

포함된 모든 항목 표시(-Recurse)

Windows 폴더에 들어 있는 항목과 그 하위 폴더에 들어 있는 항목을 모두 보려면 **Get-ChildItem** 의 **Recurse** 매개 변수를 사용하십시오. 그러면 다음과 같이 Windows 폴더에 들어 있는 항목과 그 하위 폴더에 들어 있는 항목이 모두 표시됩니다.

```
PS> Get-ChildItem -Path C:\WINDOWS -Recurse


    Directory: Microsoft.Windows PowerShell.Core\FileSystem::C:\WINDOWS
    Directory: Microsoft.Windows PowerShell.Core\FileSystem::C:\WINDOWS\AppPatch
Mode                LastWriteTime     Length Name
----                -------------     ------ ----
-a---         2004-08-04   8:00 AM    1852416 AcGenral.dll
...
```

이름을 기준으로 항목 필터링(-Name)

항목의 이름만 표시하려면 다음과 같이 **Get-Childditem** 의 **Name** 매개 변수를 사용하십시오.

```
PS> Get-ChildItem -Path C:\WINDOWS -Name
addins
AppPatch
assembly
...
```

숨겨진 항목 강제 표시(-Force)

일반적으로 Windows 탐색기나 Cmd.exe 에 표시되지 않는 항목은 **Get-ChildItem** 명령의 출력에도 표시되지 않습니다. 숨겨진 항목을 표시하려면 다음과 같이 **Get-ChildItem** 의 **Force** 매개 변수를 사용하십시오.

```
Get-ChildItem -Path C:\Windows -Force
```

이 매개 변수의 이름은 **Get-ChildItem** 명령의 정상적인 동작을 강제로 무시할 수 있으므로 Force 입니다. Force 매개 변수는 cmdlet 이 일반적으로 수행하지 않는 작업을 강제로 수행하기 위해 널리 사용되는 매개 변수입니다.

와일드카드로 항목 이름 일치

Get-ChildItem 명령은 표시할 항목의 경로에 와일드카드를 허용합니다.

와일드카드 일치가 Windows PowerShell 엔진에 의해 처리되므로 와일드카드를 허용하는 모든 cmdlet 은 동일한 표기법과 일치 동작을 사용합니다. Windows PowerShell 와일드카드 표기법에는 다음과 같은 문자가 사용됩니다.

- 별표(*) – 0 개 이상의 문자와 일치시킵니다.
- 물음표(?) – 정확히 하나의 문자와 일치시킵니다.
- 왼쪽 대괄호([) 문자와 오른쪽 대괄호(]) 문자 – 일치시킬 일련의 문자를 묶습니다.

다음은 와일드카드를 지정하는 방법을 보여 주는 몇 가지 예제입니다.

Windows 디렉터리에서 파일 확장명이 **.log** 이고 기본 이름이 정확히 5 자인 파일을 모두 찾으려면 다음 명령을 입력하십시오.

```
PS> Get-ChildItem -Path C:\Windows\?????.log
    Directory: Microsoft.Windows PowerShell.Core\FileSystem::C:\Windows
Mode                LastWriteTime     Length Name
----                -------------     ------ ----
...
-a---         2006-05-11   6:31 PM     204276 ocgen.log
-a---         2006-05-11   6:31 PM      22365 ocmsn.log
...
-a---         2005-11-11   4:55 AM         64 setup.log
-a---         2005-12-15   2:24 PM      17719 VxSDM.log
...
```

Windows 디렉터리에서 **x** 문자로 시작하는 파일을 모두 찾으려면 다음과 같이 입력하십시오.

```
 Get-ChildItem -Path C:\Windows\x*
```

이름이 **x** 또는 **z** 로 시작하는 파일을 모두 찾으려면 다음과 같이 입력하십시오.

```
Get-ChildItem -Path C:\Windows\[xz]*
```

항목 제외(-Exclude)

Get-ChildItem 의 Exclude 매개 변수를 사용하여 특정 항목을 제외시킬 수 있습니다. 이 기능을 사용하면 단일 문으로 복잡한 필터링을 수행할 수 있습니다.

예를 들어 System32 폴더에서 Windows 시간 서비스 DLL 을 찾으려고 하는데, DLL 이름이 "W"로 시작하고 이름에 "32"가 포함되어 있다는 것만 생각날 경우 필터링을 수행할 수 있습니다.

w*32*.dll 과 같은 식은 조건을 만족하는 모든 DLL 을 찾지만 이름에 "95" 또는 "16"이 포함되어 있는 Windows 95 및 16 비트 Windows 호환 DLL 도 반환할 수 있습니다. 다음과 같이 Exclude 매개 변수를 *[9516]* 패턴과 함께 사용하여 이러한 이름을 가진 파일을 제외시킬 수 있습니다.

```
 PS> Get-ChildItem -Path C:\WINDOWS\System32\w*32*.dll -Exclude *[9516]*


    Directory: Microsoft.PowerShell.Core\FileSystem::C:\WINDOWS\System32


Mode                 LastWriteTime     Length Name
----                 -------------     ------ ----
-a---         2004-08-04   8:00 AM     174592 w32time.dll
-a---         2004-08-04   8:00 AM      22016 w32topl.dll
-a---         2004-08-04   8:00 AM     101888 win32spl.dll
-a---         2004-08-04   8:00 AM     172032 wldap32.dll
-a---         2004-08-04   8:00 AM     264192 wow32.dll
-a---         2004-08-04   8:00 AM      82944 ws2_32.dll
-a---         2004-08-04   8:00 AM      42496 wsnmp32.dll
-a---         2004-08-04   8:00 AM      22528 wsock32.dll
-a---         2004-08-04   8:00 AM      18432 wtsapi32.dll
```

Get-ChildItem 매개 변수 혼합

동일한 명령에서 **Get-ChildItem** cmdlet 의 매개 변수를 여러 개 사용할 수 있습니다. 매개 변수를 혼합하려면 먼저 와일드카드 일치에 대해 알고 있어야 합니다. 예를 들어 다음 명령은 결과를 반환하지 않습니다.

```
PS> Get-ChildItem -Path C:\Windows\*.dll -Recurse -Exclude [a-y]*.dll
```

이 경우 Windows 폴더에 "z" 문자로 시작하는 DLL 두 개가 들어 있어도 아무 결과도 반환되지 않습니다.

아무 결과도 반환되지 않는 이유는 와일드카드를 경로의 일부로 지정했기 때문입니다. 위의 명령을 반복해서 실행해도 **Get-ChildItem** cmdlet 은 Windows 폴더에서 이름이 ".dll"로 끝나는 항목만 반환합니다.

이름이 특수 패턴과 일치하는 파일을 반복해서 검색하려면 다음과 같이 **-Include** 매개 변수를 사용하십시오.

```
PS> Get-ChildItem -Path C:\Windows -Include *.dll -Recurse -Exclude [a-y]*.dll

    Directory: Microsoft.Windows
PowerShell.Core\FileSystem::C:\Windows\System32\Setup


Mode                 LastWriteTime     Length Name
----                 -------------     ------ ----
-a---         2004-08-04   8:00 AM       8261 zoneoc.dll


    Directory: Microsoft.Windows PowerShell.Core\FileSystem::C:\Windows\System32


Mode                 LastWriteTime     Length Name
----                 -------------     ------ ----
-a---         2004-08-04   8:00 AM     337920 zipfldr.dll
```

항목 직접 조작

Windows PowerShell 에서는 파일 시스템 드라이브의 파일 및 폴더와 Windows PowerShell 레지스트리 드라이브의 레지스트리와 같이 Windows PowerShell 드라이브에 표시되는 요소를 항목이라고 합니다. 이러한 항목에 사용되는 cmdlet 의 이름에는 명사 **Item** 이 포함됩니다.

Get-Command -Noun Item 명령을 실행하면 다음과 같이 아홉 개의 Windows PowerShell 항목 cmdlet 에 대한 내용이 출력됩니다.

```
PS> Get-Command -Noun Item

CommandType     Name                              Definition
-----------     ----                              ----------
Cmdlet          Clear-Item                        Clear-Item [-Path] <String[]...
Cmdlet          Copy-Item                         Copy-Item [-Path] <String[]>...
Cmdlet          Get-Item                          Get-Item [-Path] <String[]> ...
Cmdlet          Invoke-Item                       Invoke-Item [-Path] <String[...
Cmdlet          Move-Item                         Move-Item [-Path] <String[]>...
Cmdlet          New-Item                          New-Item [-Path] <String[]> ...
Cmdlet          Remove-Item                       Remove-Item [-Path] <String[...
Cmdlet          Rename-Item                       Rename-Item [-Path] <String>...
```

```
Cmdlet          Set-Item                          Set-Item [-Path] <String[]> ...
```

새 항목 만들기(New-Item)

파일 시스템에서 새 항목을 만들려면 **New-Item** cmdlet 을 사용해야 합니다. 이때 Path 매개 변수 다음에 항목의 경로를 입력하고 ItemType 매개 변수 다음에 값으로 "file" 또는 "directory"를 입력해야 합니다.

예를 들어 C:\Temp 디렉터리에 "New.Directory"라는 새 디렉터리를 만들려면 다음과 같이 입력하십시오.

```
PS> New-Item -Path c:\temp\New.Directory -ItemType Directory

    Directory: Microsoft.Windows PowerShell.Core\FileSystem::C:\temp

Mode                LastWriteTime     Length Name
----                -------------     ------ ----
d----         2006-05-18  11:29 AM            New.Directory
```

파일을 만들려면 ItemType 매개 변수의 값을 "file"로 변경하십시오. 예를 들어 New.Directory 디렉터리에 "file1.txt"라는 파일을 만들려면 다음과 같이 입력하십시오.

```
PS> New-Item -Path C:\temp\New.Directory\file1.txt -ItemType file

    Directory: Microsoft.Windows PowerShell.Core\FileSystem::C:\temp\New.Directory

Mode                LastWriteTime     Length Name
----                -------------     ------ ----
-a---         2006-05-18  11:44 AM          0 file1
```

위와 동일한 방법을 사용하여 새 레지스트리 키를 만들 수 있습니다. 사실 Windows 레지스트리에는 키 항목 유형만 있기 때문에 레지스트리 키는 더 쉽게 만들 수 있습니다(레지스트리 입력 항목은 항목 속성임). 예를 들어 CurrentVersion 하위 키에 "_Test"라는 키를 만들려면 다음과 같이 입력하십시오.

```
PS> New-Item -Path HKLM:\SOFTWARE\Microsoft\Windows\CurrentVersion\_Test

   Hive: Microsoft.PowerShell.Core\Registry::HKEY_LOCAL_MACHINE\SOFTWARE\Micros
oft\Windows\CurrentVersion

SKC  VC Name                           Property
---  -- ----                           --------
  0   0 _Test                         {}
```

레지스트리 경로를 입력할 때는 Windows PowerShell 드라이브 이름인 HKLM: 및 HKCU:에 콜론을 포함해야 합니다. 드라이브 이름에 콜론이 없으면 Windows PowerShell 은 경로에 있는 드라이브 이름을 인식하지 못합니다.

레지스트리 값이 항목이 아닌 이유

Get-ChildItem cmdlet 을 사용하여 레지스트리 값에서 항목을 찾으면 실제 레지스트리 입력 항목이나 해당 값이 표시되지 않습니다.

예를 들어 **HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Run** 레지스트리 키에는 대개 시스템이 시작될 때 실행되는 응용 프로그램을 나타내는 레지스트리 입력 항목이 포함되어 있습니다.

그러나 다음과 같이 **Get-ChildItem** 을 사용하여 이 키에서 자식 항목을 찾으면 이 키의 **OptionalComponents** 하위 키만 표시됩니다.

```
PS> Get-ChildItem HKLM:\Software\Microsoft\Windows\CurrentVersion\Run
   Hive: Microsoft.PowerShell.Core\Registry::HKEY_LOCAL_MACHINE\Software\Micros
oft\Windows\CurrentVersion\Run
SKC  VC Name                           Property
---  -- ----                           --------
  3   0 OptionalComponents             {}
```

레지스트리 입력 항목을 항목으로 취급하면 편리할 수는 있지만 레지스트리 입력 항목에 고유한 경로를 지정할 수 없습니다. 경로 표기법은 **Run** 이라는 레지스트리 하위 키와 이 **Run** 하위 키에 있는 **(Default)** 레지스트리 입력 항목을 구별하지 않습니다. 또한 레지스트리 입력 항목 이름에는 백슬래시 문자가 포함되어 있기 때문에 경로 표기법을 사용하여 **Windows\CurrentVersion\Run** 이라는 레지스트리 입력 항목과 이 경로에 있는 하위 키를 구별할 수도 없습니다.

기존 항목 이름 바꾸기(Rename-Item)

파일이나 폴더의 이름을 바꾸려면 **Rename-Item** cmdlet 을 사용하십시오. 다음 명령은 file1.txt 파일의 이름을 fileOne.txt 로 바꿉니다.

```
PS> Rename-Item -Path C:\temp\New.Directory\file1.txt fileOne.txt
```

Rename-Item cmdlet 은 파일이나 폴더의 이름을 바꿀 수는 있지만 항목을 이동할 수는 없습니다. 다음 명령은 파일을 New.Directory 디렉터리에서 Temp 디렉터리로 이동하려고 하기 때문에 실패합니다.

```
PS> Rename-Item -Path C:\temp\New.Directory\fileOne.txt c:\temp\fileOne.txt
Rename-Item : 지정한 대상이 경로가 아니므로 이름을 바꿀 수 없습니다.
줄:1 문자:12
+ Rename-Item  <<<< -Path C:\temp\New.Directory\fileOne c:\temp\fileOne.txt
```

항목 이동(Move-Item)

파일이나 폴더를 이동하려면 **Move-Item** cmdlet 을 사용하십시오.

예를 들어 다음 명령은 New.Directory 디렉터리를 C:\temp 디렉터리에서 C: 드라이브의 루트로 이동합니다. 항목이 이동되었는지 확인하려면 **Move-Item** cmdlet 의 **PassThru** 매개 변수를 포함하십시오. Passthru 가 없으면 **Move-Item** cmdlet 은 결과를 표시하지 않습니다.

```
PS> Move-Item -Path C:\temp\New.Directory -Destination C:\ -PassThru

    Directory: Microsoft.Windows PowerShell.Core\FileSystem::C:\

Mode                LastWriteTime     Length Name
----                -------------     ------ ----
d----         2006-05-18  12:14 PM            New.Directory
```

항목 복사(Copy-Item)

다른 셸의 복사 작업에 익숙한 사용자에게는 Windows PowerShell 의 **Copy-Item** cmdlet 동작이 생소하게 느껴질 수 있습니다. 한 위치에서 다른 위치로 항목을 복사할 때 Copy-Item 은 기본적으로 항목의 내용을 복사하지 않습니다.

예를 들어 C: 드라이브에서 C:\temp 디렉터리로 **New.Directory** 디렉터리를 복사하면 명령은 성공하지만 New.Directory 디렉터리에 들어 있는 파일은 복사되지 않습니다.

```
PS> Copy-Item -Path C:\New.Directory -Destination C:\temp
```

다음과 같이 **C:\temp\New.Directory** 의 내용을 표시하면 아무 파일도 들어 있지 않습니다.

```
PS> Get-ChildItem -Path C:\temp\New.Directory
PS>
```

Copy-Item cmdlet 이 새 위치로 내용을 복사하지 않는 이유는 다음과 같습니다.

Copy-Item cmdlet 이 단지 파일과 폴더를 복사하기 위한 것이 아니라 다양한 용도로 사용하기 위해 설계되었고 사용자도 파일과 폴더를 복사할 때 컨테이너만 복사하고 그 안에 들어 있는 항목은 복사하지 않으려는 경우가 있기 때문입니다.

폴더의 내용을 모두 복사하려면 명령에 **Copy-Item** cmdlet 의 **Recurse** 매개 변수를 포함하십시오. 이미 내용을 포함하지 않은 채 디렉터리를 복사한 경우 다음과 같이 빈 폴더를 덮어쓸 수 있는 **Force** 매개 변수를 포함하십시오.

```
PS> Copy-Item -Path C:\New.Directory -Destination C:\temp -Recurse -Force -
Passthru
    Directory: Microsoft.Windows PowerShell.Core\FileSystem::C:\temp

Mode                LastWriteTime     Length Name
----                -------------     ------ ----
d----        2006-05-18   1:53 PM            New.Directory

    Directory: Microsoft.Windows PowerShell.Core\FileSystem::C:\temp\New.Directory

Mode                LastWriteTime     Length Name
----                -------------     ------ ----
-a---        2006-05-18  11:44 AM          0 file1
```

항목 삭제(Remove-Item)

파일과 폴더를 삭제하려면 **Remove-Item** cmdlet 을 사용하십시오. 중요하면서도 되돌릴 수 없는 변경 작업을 수행할 수 있는 **Remove-Item** 과 같은 Windows PowerShell cmdlet 은 종종 해당 명령을 입력하면 확인 메시지를 표시합니다. 예를 들어 **New.Directory** 폴더를 제거하려고 하면 이 폴더 안에 파일이 들어 있으므로 다음과 같은 명령 확인 메시지가 나타납니다.

```
PS> Remove-Item C:\New.Directory

Confirm
C:\temp\New.Directory 의 항목에는 하위 항목이 있으며 -recurse 매개 변수를 지정하지 않았습니다.
```

```
계속하면 해당 항목과 모든 하위 항목이 제거됩니다. 계속하시겠습니까?
[Y] 예 [A] 모두 예 [N] 아니요 [L] 모두 아니요 [S] 일시 중단 [?] 도움말
(기본값은 "Y"임):
```

기본 응답이 **예**이므로 폴더와 해당 파일을 삭제하려면 **Enter** 키를 누르십시오. 폴더를 제거할 때 확인 메시지가 나타나지 않도록 하려면 **-Recurse** 매개 변수를 사용하십시오.

```
PS> Remove-Item C:\temp\New.Directory -Recurse
```

항목 실행(Invoke-Item)

Windows PowerShell 은 **Invoke-Item** cmdlet 을 사용하여 파일 또는 폴더에 대한 기본 작업을 수행합니다. 이러한 기본 작업은 레지스트리에 등록된 기본 응용 프로그램 처리기에 의해 결정되며, Windows 탐색기에서 항목을 두 번 클릭한 것과 같습니다.

예를 들어 다음 명령을 실행할 수 있습니다.

```
PS> Invoke-Item C:\WINDOWS
```

그러면 C:\Windows 폴더를 두 번 클릭한 것처럼 C:\Windows 가 표시된 탐색기 창이 열립니다.

예를 들어 Windows Vista 이전 시스템에서 다음과 같이 **Boot.ini** 파일을 호출할 수 있습니다.

```
PS> Invoke-Item C:\boot.ini
```

.ini 파일 유형이 메모장과 연결되어 있으면 boot.ini 파일이 메모장에서 열립니다.

개체 작업 수행

이 설명서에서는 Windows PowerShell 에서 개체를 사용하여 cmdlet 간에 데이터를 전송하는 방법을 설명하고 개체의 특정 속성을 보기 위해 Get-Member 및 Format cmdlet 을 사용하여 개체에 대한 자세한 정보를 표시하는 몇 가지 방법을 살펴보았습니다.

개체 사용의 이점은 많은 복잡한 데이터에 액세스할 수 있을 뿐 아니라 이미 상관 관계가 있다는 것입니다. 간단한 조작만으로 훨씬 더 많은 작업을 수행할 수 있습니다. 이 장에서는 몇 가지 특정 유형의 개체와 이러한 개체를 조작하는 방법을 설명합니다.

WMI 개체 보기(Get-WmiObject)

WMI 개체 보기(Get-WmiObject)

WMI(Windows Management Instrumentation)는 다양한 정보를 일관되게 표시하므로 Windows 시스템 관리를 위한 핵심 기술입니다. WMI 가 표시하는 정보의 양이 제한되어 있기 때문에 WMI 개체에 액세스하기 위한 Windows PowerShell cmdlet 인 **Get-WmiObject** 는 실제 작업을 수행하는 데 있어서 가장 유용한 도구 중 하나입니다. 이 장에서는 Get-WmiObject 를 사용하여 WMI 개체에 액세스하는 방법과 WMI 개체를 사용하여 특정 작업을 수행하는 방법을 차례로 설명합니다.

WMI 클래스 표시

대부분의 WMI 사용자가 겪는 첫 번째 문제는 WMI 를 사용하여 수행할 수 있는 작업을 검색하려고 할 때 발생합니다. WMI 클래스는 관리 가능한 리소스를 설명하는데, 이러한 WMI 클래스는 수백 개가 있고 일부 클래스에는 수십 개의 속성이 포함되어 있습니다.

Get-WmiObject 는 이 문제를 해결하기 위해 WMI 를 검색 가능하게 합니다. 다음과 같이 입력하면 로컬 컴퓨터에서 사용할 수 있는 WMI 클래스의 목록을 볼 수 있습니다.

```
PS> Get-WmiObject -List


__SecurityRelatedClass                  __NTLMUser9X
__PARAMETERS                            __SystemSecurity
__NotifyStatus                          __ExtendedStatus
Win32_PrivilegesStatus                  Win32_TSNetworkAdapterSettingError
Win32_TSRemoteControlSettingError       Win32_TSEnvironmentSettingError
...
```

다음과 같이 ComputerName 매개 변수를 사용하여 컴퓨터 이름과 IP 주소를 지정하면 원격 컴퓨터에서도 이 정보를 검색할 수 있습니다.

```
PS> Get-WmiObject -List -ComputerName 192.168.1.29


__SystemClass                           __NAMESPACE
__Provider                              __Win32Provider
__ProviderRegistration                  __ObjectProviderRegistration
...
```

원격 컴퓨터에서 반환되는 클래스 목록은 컴퓨터가 실행 중인 운영 체제와 설치된 응용 프로그램에서 추가한 WMI 확장에 따라 다를 수 있습니다.

참고:

Get-WmiObject 를 사용하여 원격 컴퓨터에 연결할 때는 원격 컴퓨터에서 WMI 를 실행 중이어야 하고, 기본 구성에서는 사용 중인 계정이 원격 컴퓨터의 로컬 관리자 그룹에 있어야 합니다. 원격 시스템에는 Windows PowerShell 을 설치하지 않아도 됩니다. 이 경우 관리자는 Windows PowerShell 을 실행 중이지 않지만 WMI 를 사용할 수 있는 운영 체제를 관리할 수 있습니다.

로컬 시스템에 연결할 때 ComputerName 을 포함할 수도 있습니다. 로컬 컴퓨터의 이름, IP 주소(또는 루프백 주소 127.0.0.1) 또는 WMI 스타일 '.'를 컴퓨터 이름으로 사용할 수 있습니다. IP 주소가 192.168.1.90 이고 이름이 Admin01 인 컴퓨터에서 Windows PowerShell 을 실행 중인 경우 다음 명령을 실행하면 이 컴퓨터에서 사용할 수 있는 모든 WMI 클래스 목록이 반환됩니다.

```
Get-WmiObject -List
Get-WmiObject -List -ComputerName .
Get-WmiObject -List -ComputerName Admin01
Get-WmiObject -List -ComputerName 192.168.1.90
Get-WmiObject -List -ComputerName 127.0.0.1
Get-WmiObject -List -ComputerName localhost
```

Get-WmiObject 는 기본적으로 root/cimv2 네임스페이스를 사용합니다. 다른 WMI 네임스페이스를 지정하려면 다음과 같이 **Namespace** 매개 변수를 사용하고 해당 네임스페이스 경로를 지정하십시오.

```
PS> Get-WmiObject -List -ComputerName 192.168.1.29 -Namespace root


__SystemClass                           __NAMESPACE
__Provider                              __Win32Provider
...
```

WMI Class 세부 정보 표시

WMI 클래스의 이름을 알고 있으면 이 이름을 사용하여 정보를 즉시 볼 수 있습니다. 예를 들어 컴퓨터에 대한 정보를 검색하는 데 일반적으로 사용되는 WMI 클래스 중 하나는 **Win32_OperatingSystem** 입니다.

```
PS> Get-WmiObject -Class Win32_OperatingSystem -Namespace root/cimv2 -
ComputerName .


SystemDirectory : C:\WINDOWS\system32
Organization    : Global Network Solutions
BuildNumber     : 2600
RegisteredUser  : Oliver W. Jones
SerialNumber    : 12345-678-9012345-67890
Version         : 5.1.2600
```

이 명령에는 매개 변수가 모두 나와 있지만 불필요한 매개 변수를 표시하지 않을 수 있습니다. 로컬 시스템에 연결할 때는 **ComputerName** 매개 변수가 필요하지 않습니다. 여기에 이 매개 변수를 표시한 것은 가장 일반적인 경우를 보여 주고 이 매개 변수가 있다는 것을 알려 주기 위한 것입니다. **Namespace** 는 기본적으로 root/cimv2 로 설정되며 마찬가지로 생략될

수 있습니다. 마지막으로 대부분의 cmdlet 에서는 일반적인 매개 변수의 이름을 생략할 수 있습니다. Get-WmiObject 를 사용할 때 첫 번째 매개 변수의 이름을 지정하지 않으면 Windows PowerShell 은 이 매개 변수를 **Class** 매개 변수로 취급합니다. 즉, 다음과 같이 입력하여 위의 명령을 실행할 수도 있습니다.

```
Get-WmiObject Win32_OperatingSystem
```

Win32_OperatingSystem 클래스에는 여기에 표시된 것보다 훨씬 더 많은 속성이 있습니다. Get-Member 를 사용하면 이러한 속성을 모두 볼 수 있습니다. 다음과 같이 WMI 클래스의 속성도 다른 개체 속성처럼 자동으로 사용할 수 있습니다.

```
PS> Get-WmiObject -Class Win32_OperatingSystem -Namespace root/cimv2 -
ComputerName . | Get-Member -MemberType Property


   TypeName: System.Management.ManagementObject#root\cimv2\Win32_OperatingSyste
m

Name                                       MemberType Definition
----                                       ---------- ----------
__CLASS                                    Property   System.String __CLASS {...
...
BootDevice                                 Property   System.String BootDevic...
BuildNumber                                Property   System.String BuildNumb...
...
```

Format Cmdlet 을 사용하여 기본 속성이 아닌 속성 표시

기본적으로 표시되지 않는 **Win32_OperatingSystem** 클래스에 포함된 정보를 보려면 **Format** cmdlet 을 사용하여 표시할 수 있습니다. 예를 들어 사용 가능한 메모리 데이터를 보려면 다음과 같이 입력하십시오.

```
PS> Get-WmiObject -Class Win32_OperatingSystem -Namespace root/cimv2 -
ComputerName . | Format-Table -Property
TotalVirtualMemorySize,TotalVisibleMemorySize,FreePhysicalMemory,FreeVirtualMemory
,FreeSpaceInPagingFiles

TotalVirtualMemorySize TotalVisibleMem FreePhysicalMem FreeVirtualMemo
FreeSpaceInPagi
                            ory              ry         ngFiles
---------------- --------------- --------------- --------------- ---------------
         2097024          785904          305808         2056724         1558232
```

참고:

Format-Table 의 속성 이름에 와일드카드를 사용할 수 있으므로 마지막 파이프라인 요소를 **Format-Table -Property TotalV*,Free*** 로 줄일 수 있습니다.

다음과 같이 입력하여 메모리 데이터를 목록으로 표시하면 더 쉽게 읽을 수 있습니다.

```
PS> Get-WmiObject -Class Win32_OperatingSystem -Namespace root/cimv2 -
ComputerName . | Format-List
```

```
TotalVirtualMemorySize,TotalVisibleMemorySize,FreePhysicalMemory,FreeVirtualMemory
,FreeSpaceInPagingFiles


TotalVirtualMemorySize : 2097024
TotalVisibleMemorySize : 785904
FreePhysicalMemory     : 301876
FreeVirtualMemory      : 2056724
FreeSpaceInPagingFiles : 1556644
```

.NET 및 COM 개체 만들기(New-Object)

.NET Framework 및 COM 인터페이스에는 다양한 시스템 관리 작업을 수행할 수 있는 소프트웨어 구성 요소가 포함되어 있습니다. Windows PowerShell 에서는 이러한 구성 요소를 사용할 수 있기 때문에 cmdlet 을 사용하지 않고도 작업을 수행할 수 있습니다. Windows PowerShell 의 초기 릴리스에 포함된 대부분의 cmdlet 은 원격 컴퓨터에 사용할 수 없습니다. 이 설명서에서는 Windows PowerShell 에서 .NET **System.Diagnostics.EventLog** 클래스를 직접 사용하여 이벤트 로그를 관리할 때 이러한 제한을 해결하는 방법을 보여 줍니다.

New-Object 를 사용하여 이벤트 로그에 액세스

.NET Framework 클래스 라이브러리에는 이벤트 로그를 관리하는 데 사용할 수 있는 **System.Diagnostics.EventLog** 라는 클래스가 포함되어 있습니다. **TypeName** 매개 변수와 함께 **New-Object** cmdlet 을 사용하면 .NET 클래스의 새 인스턴스를 만들 수 있습니다. 예를 들어 다음 명령은 이벤트 로그 참조를 만듭니다.

```
PS> New-Object -TypeName System.Diagnostics.EventLog

  Max(K) Retain OverflowAction        Entries Name
  ------ ------ --------------        ------- ----
```

이 명령이 EventLog 클래스의 인스턴스를 만들었지만 이 인스턴스에는 아무 데이터도 포함되어 있습니다. 이것은 특정 이벤트 로그를 지정하지 않았기 때문입니다. 실제 이벤트 로그를 보는 방법은 다음 절에서 설명합니다.

New-Object 와 함께 생성자 사용

특정 이벤트 로그를 참조하려면 이벤트 로그의 이름을 지정해야 합니다. **New-Object** 에는 **ArgumentList** 라는 매개 변수가 있습니다. 이 매개 변수에 값으로 전달되는 인수는 개체의 특수 시작 메서드에서 사용합니다. 이러한 메서드는 개체를 생성하는 데 사용되므로 생성자라고 합니다. 예를 들어 응용 프로그램 로그에 대한 참조를 보려면 다음과 같이 'Application'이라는 문자열을 인수로 지정하면 됩니다.

```
PS> New-Object -TypeName System.Diagnostics.EventLog -ArgumentList Application

Max(K) Retain OverflowAction        Entries Name
------ ------ --------------        ------- ----
16,384      7 OverwriteOlder          2,160 Application
```

참고:

대부분의 .NET 핵심 클래스가 System 네임스페이스에 들어 있으므로 Windows PowerShell 은 사용자가 지정한 클래스를 찾을 수 없으면 자동으로 System 네임스페이스에서 이 클래스를 찾으려고 시도합니다. 따라서 System.Diagnostics.EventLog 대신 Diagnostics.EventLog 를 지정할 수 있습니다.

변수에 개체 저장

Windows PowerShell 세션 동안 사용할 수 있도록 개체에 대한 참조를 저장할 수 있습니다. Windows PowerShell 에서는 파이프라인을 통해 많은 작업을 수행할 수 있지만 변수에 대한 필요성을 줄이고 경우에 따라 개체에 대한 참조를 변수에 저장하면 이러한 개체를 훨씬 더 쉽게 조작할 수 있습니다.

Windows PowerShell 에서는 이름이 기본적으로 지정되는 개체인 변수를 만들 수 있습니다. 올바른 Windows PowerShell 명령의 출력은 변수에 저장될 수 있습니다. 변수 이름은 항상 $로 시작됩니다. $AppLog 라는 변수에 응용 프로그램 로그를 저장하려면 다음과 같이 변수 이름과 등호를 차례로 입력한 다음 응용 프로그램 로그 개체를 만드는 데 사용되는 명령을 입력합니다.

```
PS> $AppLog = New-Object -TypeName System.Diagnostics.EventLog -ArgumentList
Application
```

그런 다음 아래와 같이 $AppLog 를 입력하면 $AppLog 변수에 응용 프로그램 로그가 포함되어 있는 것을 확인할 수 있습니다.

```
PS> $AppLog

  Max(K) Retain OverflowAction        Entries Name
  ------ ------ --------------        ------- ----
  16,384      7 OverwriteOlder          2,160 Application
```

New-Object 를 사용하여 원격 이벤트 로그에 액세스

앞 절에서 설명한 명령은 로컬 컴퓨터를 대상으로 하는 반면 **Get-EventLog** cmdlet 은 원격 컴퓨터를 대상으로 합니다. 원격 컴퓨터에 있는 응용 프로그램 로그에 액세스하려면 로그 이름과 컴퓨터 이름(또는 IP 주소)을 인수로 제공해야 합니다.

```
PS> $RemoteAppLog = New-Object -TypeName System.Diagnostics.EventLog
Application,192.168.1.81
PS> $RemoteAppLog

  Max(K) Retain OverflowAction        Entries Name
  ------ ------ --------------        ------- ----
     512      7 OverwriteOlder            262 Application
```

이제 $RemoteAppLog 변수에 이벤트 로그에 대한 참조가 저장되었으므로 다음 절에서는 이러한 이벤트 로그와 관련된 작업에 대해 설명합니다.

개체 메서드를 사용하여 이벤트 로그 지우기

일반적으로 개체에는 작업을 수행하기 위해 호출할 수 있는 메서드가 있습니다. **Get-Member** 를 사용하면 개체와 연결된 메서드를 볼 수 있습니다. 다음 명령과 일부 출력 내용은 EventLog 클래스의 일부 메서드를 보여 줍니다.

```
PS> $RemoteAppLog | Get-Member -MemberType Method


   TypeName: System.Diagnostics.EventLog

Name                       MemberType Definition
----                       ---------- ----------
...
Clear                      Method     System.Void Clear()
Close                      Method     System.Void Close()
...
GetType                    Method     System.Type GetType()
...
ModifyOverflowPolicy       Method     System.Void ModifyOverflowPolicy(Overfl...
RegisterDisplayName        Method     System.Void RegisterDisplayName(String ...
...
ToString                   Method     System.String ToString()
WriteEntry                 Method     System.Void WriteEntry(String message),...
WriteEvent                 Method     System.Void WriteEvent(EventInstance in...
```

Clear() 메서드는 이벤트 로그를 지우는 데 사용할 수 있습니다. 메서드를 호출할 때는 인수가 필요 없어도 항상 메서드 이름 뒤에 괄호를 입력해야 합니다. 이렇게 하면 Windows PowerShell 이 메서드와 동일한 이름을 가진 잠재적 속성과 메서드를 구별할 수 있게 됩니다. 다음 명령을 입력하면 **Clear** 메서드를 호출할 수 있습니다.

```
PS> $RemoteAppLog.Clear()
```

다음 명령을 입력하면 로그를 표시할 수 있습니다. 그러면 다음과 같이 이벤트 로그가 지워지고 항목 수로 262 대신 **0** 이 표시됩니다.

```
PS> $RemoteAppLog

  Max(K) Retain OverflowAction        Entries Name
  ------ ------ --------------        ------- ----
     512      7 OverwriteOlder              0 Application
```

New-Object 를 사용하여 COM 개체 만들기

New-Object 를 사용하여 COM(구성 요소 개체 모델) 구성 요소와 관련된 작업을 수행할 수 있습니다. 구성 요소는 WSH(Windows 스크립트 호스트)에 포함된 다양한 라이브러리부터 대부분의 시스템에 설치되어 있는 Internet Explorer 와 같은 ActiveX 응용 프로그램까지 다양합니다.

New-Object 는 .NET Framework 런타임 호출 가능 래퍼를 사용하여 COM 개체를 만들기 때문에 COM 개체를 호출할 때 .NET 에 적용되는 것과 동일한 제한을 받습니다. To create a COM 개체를 만들려면 사용할 COM 클래스의 ProgId(프로그래밍 ID)를 사용하여 **ComObject** 매개 변수를 지정해야 합니다. 이 설명서에서 COM 사용과 관련된 제한

사항과 시스템에서 사용할 수 있는 ProgId 를 확인하는 방법에 대해 자세히 설명하지는 않지만 WSH 와 같은 환경에서 가져온 잘 알려진 개체는 Windows PowerShell 에서 사용할 수 있습니다.

WScript.Shell, **WScript.Network**, **Scripting.Dictionary** 및 **Scripting.FileSystemObject** 와 같은 Progid 를 지정하여 WSH 개체를 만들 수 있습니다. 다음 명령은 이러한 개체를 만듭니다.

```
New-Object -ComObject WScript.Shell
New-Object -ComObject WScript.Network
New-Object -ComObject Scripting.Dictionary
New-Object -ComObject Scripting.FileSystemObject
```

Windows PowerShell 에서 이러한 클래스의 기능 대부분을 다양하게 사용할 수 있지만 바로 가기 만들기와 같은 일부 작업은 WSH 클래스를 사용하여 더 쉽게 수행할 수 있습니다.

WScript.Shell 을 사용하여 데스크톱 바로 가기 만들기

COM 개체를 사용하여 신속하게 수행할 수 있는 작업 중 하나는 바로 가기 만들기입니다. 예를 들어 데스크톱에 Windows PowerShell 의 홈 폴더와 연결된 바로 가기를 만들려면 먼저 **WScript.Shell** 에 대한 참조를 만든 후 다음과 같이 **$WshShell** 이라는 변수에 저장해야 합니다.

```
$WshShell = New-Object -ComObject WScript.Shell
```

Get-Member 를 COM 개체에 사용할 수 있으므로 다음과 같이 입력하여 개체의 멤버를 검색할 수 있습니다.

```
PS> $WshShell | Get-Member


   TypeName: System.__ComObject#{41904400-be18-11d3-a28b-00104bd35090}

Name                      MemberType            Definition
----                      ----------            ----------
AppActivate               Method                bool AppActivate (Variant, Va...
CreateShortcut            Method                IDispatch CreateShortcut (str...
...
```

참고:

Get-Member 에는 파이프 대신 사용하여 **Get-Member** 에 개체를 입력할 수 있는 선택적 **InputObject** 매개 변수가 있습니다. 따라서 **Get-Member -InputObject $WshShell** 명령을 대신 사용해도 위와 동일한 내용이 출력됩니다. **InputObject** 를 사용할 경우 **Get-Member** 는 해당 인수를 단일 항목으로 취급합니다. 즉, 변수에 여러 개의 개체가 있을 경우 **Get-Member** 는 이러한 개체를 하나의 개체 배열로 취급합니다**.** 예를 들면 다음과 같습니다.

```
PS> $a = 1,2,"three"

PS> Get-Member -InputObject $a


   TypeName: System.Object[]
```

```
Name                MemberType     Definition
----                ----------     ----------
Count               AliasProperty Count = Length


...
```

WScript.Shell CreateShortcut 메서드는 만들려는 바로 가기 파일의 경로로 단일 인수를 받아들입니다. 데스크톱에 대한 전체 경로를 입력할 수도 있지만 더 쉬운 방법이 있습니다. 일반적으로 데스크톱은 현재 사용자의 홈 폴더에 들어 있는 Desktop 이라는 폴더로 표시됩니다. Windows PowerShell 에는 이 폴더의 경로가 들어 있는 **$Home** 이라는 변수가 있습니다. 다음과 같이 이 변수를 사용하여 홈 폴더의 경로를 지정한 다음 Desktop 폴더의 이름과 만들려는 바로 가기의 이름을 추가할 수 있습니다.

```
$lnk = $WshShell.CreateShortcut("$Home\Desktop\PSHome.lnk")
```

참고:

변수 이름과 유사한 이름을 큰따옴표로 묶으면 Windows PowerShell 은 이 이름과 일치하는 변수의 값을 대체하려고 시도합니다. 작은따옴표를 사용하면 Windows PowerShell 은 변수 값을 대체하려고 시도하지 않습니다. 예를 들어 다음 명령을 입력해 보십시오.

```
PS> "$Home\Desktop\PSHome.lnk"
C:\Documents and Settings\aka\Desktop\PSHome.lnk
PS> '$Home\Desktop\PSHome.lnk'
$Home\Desktop\PSHome.lnk
```

그러면 새 바로 가기 참조가 들어 있는 **$lnk** 라는 변수가 만들어집니다. 해당 멤버를 보려면 이 변수를 **Get-Member** 에 파이프하면 됩니다. 다음 출력은 바로 가기 만들기를 완료하기 위해 사용해야 하는 멤버를 보여 줍니다.

```
PS> $lnk | Get-Member


   TypeName: System.__ComObject#{f935dc23-1cf0-11d0-adb9-00c04fd58a0b}


Name              MemberType   Definition
----              ----------   ----------
...
Save              Method       void Save ()
...
TargetPath        Property     string TargetPath () {get} {set}
...
```

Windows PowerShell 의 응용 프로그램 폴더인 **TargetPath** 를 지정한 다음 **Save** 메서드를 호출하여 **$lnk** 바로 가기를 저장해야 합니다. Windows PowerShell 응용 프로그램 폴더 경로가 **$PSHome** 변수에 저장되므로 이렇게 하려면 다음과 같이 입력해야 합니다.

```
$lnk.TargetPath = $PSHome
```

```
$lnk.Save()
```

Windows PowerShell 에서 Internet Explorer 사용

COM 을 사용하면 Microsoft Office 제품군 응용 프로그램이나 Internet Explorer 와 같은 다양한 응용 프로그램을 자동화할 수 있습니다. Internet Explorer 는 COM 기반 응용 프로그램 작업과 관련된 몇 가지 일반적인 기술과 문제를 보여 줍니다.

다음과 같이 Internet Explorer ProgId 인 **InternetExplorer.Application** 을 지정하여 Internet Explorer 인스턴스를 만들 수 있습니다.

```
$ie = New-Object -ComObject InternetExplorer.Application
```

이 명령은 Internet Explorer 를 시작하기만 하고 보여 주지는 않습니다. Get-Process 를 입력하면 iexplore 라는 프로세스가 실행 중인 것을 확인할 수 있습니다. 실제로 iexplore 프로세스는 Windows PowerShell 을 종료해도 계속 실행되므로 이 프로세스를 종료하려면 컴퓨터를 다시 시작하거나 작업 관리자와 같은 도구를 사용해야 합니다.

참고:

개별 프로세스로 시작되는 COM 개체(보통 ActiveX 실행 파일이라고 함)는 시작 시 사용자 인터페이스 창을 표시하거나 표시하지 않을 수 있습니다. Internet Explorer 와 마찬가지로 창을 만들기만 하고 표시하지 않으면 일반적으로 포커스가 Windows 바탕 화면으로 이동하므로 이 창을 사용하기 위해서는 먼저 이 창이 보이게 해야 합니다.

$ie | Get-Member 를 입력하면 Internet Explorer 의 속성과 메서드를 볼 수 있습니다. Internet Explorer 창을 표시하려면 다음과 같이 입력하여 변수 속성을 $true 로 설정하십시오.

```
$ie.Visible = $true
```

그러면 다음과 같이 Navigate 메서드를 사용하여 특정 웹 주소로 이동할 수 있습니다.

```
$ie.Navigate("http://www.microsoft.com/technet/scriptcenter/default.mspx")
```

Internet Explorer 개체 모델의 다른 멤버를 사용하면 웹 페이지에서 텍스트 내용을 검색할 수 있습니다. 다음 명령은 현재 웹 페이지의 본문에 HTML 텍스트를 표시합니다.

```
$ie.Document.Body.InnerText
```

PowerShell 에서 Internet Explore 를 종료하려면 다음과 같이 해당 Quit() 메서드를 호출하십시오.

```
$ie.Quit()
```

그러면 Internet Explore 가 강제로 종료됩니다. COM 개체에 $ie 변수가 아직 표시되어 있어도 $ie 변수에는 더 이상 유효한 참조가 없습니다. 이 변수를 사용하려고 하면 다음과 같은 자동화 오류 메시지가 나타납니다.

```
PS> $ie | Get-Member
Get-Member : "Application" 속성에 대한 문자열 표현을 검색하는 동안 예외가 발생했습니다:
"호출된 개체와 해당 클라이언트의 연결이 끊어졌습니다. (예외가
발생한 HRESULT: 0x80010108 (RPC_E_DISCONNECTED))"
줄:1 문자:16
+ $ie | Get-Member <<<<
```

$ie = $null 과 같은 명령을 사용하여 나머지 참조를 제거하거나 다음 명령을 사용하여 변수를 완전히 제거할 수 있습니다.

```
Remove-Variable ie
```

참고:

해당 참조를 제거할 때 ActiveX 실행 파일을 종료할지 아니면 계속 실행할지 여부를 결정하는 일반적인 규칙은 없습니다. 응용 프로그램이 보이는지 여부, 응용 프로그램에 편집된 문서가 열려 있는지 여부 및 Windows PowerShell 이 계속 실행 중인지 여부와 같이 상황에 따라 응용 프로그램을 종료하거나 종료하지 않을 수 있습니다. 따라서 Windows PowerShell 에서 사용할 각 ActiveX 실행 파일의 종료 동작을 테스트해야 합니다.

.NET-Wrapped COM 개체에 대한 경고 보기

경우에 따라 COM 개체에는 **New-Object** 가 사용하는 .NET RCW(런타임 호출 가능 래퍼)가 포함되어 있을 수 있습니다. RCW 의 동작이 일반적인 COM 개체와 다를 수 있기 때문에 **New-Object** 에는 RCW 액세스에 대해 경고하는 **Strict** 매개 변수가 포함되어 있습니다. **Strict** 매개 변수를 지정한 다음 RCW 를 사용하는 COM 개체를 만들면 다음과 같은 경고 메시지가 나타납니다.

```
PS> $xl = New-Object -ComObject Excel.Application -Strict
New-Object : 파이프라인에 기록된 개체는 구성 요소의 기본 interop 어셈블리에서 가져온
"Microsoft.Office.Interop.Excel.ApplicationClass" 유형의 인스턴스입니다. 이 유형이
IDispatch 멤버와 다른 멤버를 노출하면 기본 interop 어셈블리가 설치되지 않은 경우에 이 개체를
사용하도록 기록된 스크립트가 작동하지 않을 수 있습니다.
줄:1 문자:17
+ $xl = New-Object  <<<< -ComObject Excel.Application -Strict
```

개체가 만들어진 후에도 표준 COM 개체가 아니라는 경고 메시지는 계속 나타납니다.

정적 클래스 및 메서드 사용

일부 .NET Framework 클래스는 **New-Object** 를 사용하여 만들 수 없습니다. 예를 들어 **New-Object** 를 사용하여 **System.Environment** 또는 **System.Math** 개체를 만들려고 하면 다음과 같은 오류 메시지가 나타납니다.

```
PS> New-Object System.Environment
New-Object : 생성자가 없습니다. System.Environment 유형에 적절한 생성자를 찾을 수 없습니다.
줄:1 문자:11
+ New-Object  <<<< System.Environment
PS> New-Object System.Math
New-Object : 생성자가 없습니다. System.Math 유형에 적절한 생성자를 찾을 수 없습니다.
```

```
줄:1 문자:11
+ New-Object  <<<< System.Math
```

이러한 오류는 .NET Framework 클래스에서 새 개체를 만들 수 없기 때문에 발생합니다. .NET Framework 클래스는 상태가 바뀌지 않는 메서드와 속성의 참조 라이브러리로, 직접 만들지 않아도 사용할 수 있습니다. 만들거나, 제거하거나, 변경할 수 없으므로 이러한 클래스와 메서드는 정적 클래스라고 합니다. 이해를 돕기 위해 이 설명서에서는 정적 클래스를 사용하는 예제를 제공합니다.

System.Environment 를 사용하여 환경 데이터 보기

일반적으로 Windows PowerShell 에서 개체를 사용할 때 수행하는 첫 단계는 Get-Member 를 사용하여 해당 개체 안에 들어 있는 멤버를 확인하는 것입니다. 정적 클래스를 사용할 경우 실제 클래스가 개체가 아니므로 약간 다른 방식으로 이 작업을 수행합니다.

정적 System.Environment 클래스 참조

클래스 이름을 대괄호로 묶으면 정적 클래스를 참조할 수 있습니다. 예를 들어 대괄호 안에 이름을 입력하여 **System.Environment** 를 참조할 수 있습니다. 이렇게 하면 다음과 같은 일반적인 유형 정보가 표시됩니다.

```
PS> [System.Environment]

IsPublic IsSerial Name                                     BaseType
-------- -------- ----                                     --------
True     False    Environment                              System.Object
```

참고:

앞에서 언급한 것처럼 Windows PowerShell 에서 **New-Object** 를 사용하면 자동으로 유형 이름 앞에 '**System.**'이 추가됩니다. 대괄호로 묶은 유형 이름을 사용하는 경우에도 마찬가지이므로 **[System.Environment]**를 **[Environment]**로 지정할 수 있습니다.

System.Environment 클래스에는 현재 프로세스(Windows PowerShell 에서 작업하는 경우 powershell.exe 임)의 작업 환경에 대한 일반적인 정보가 포함되어 있습니다.

[System.Environment] | Get-Member 를 입력하여 이 클래스에 대한 세부 정보를 표시하면 다음과 같이 개체 유형이 **System.Environment** 가 아니라 **System.RuntimeType** 으로 표시됩니다.

```
PS> [System.Environment] | Get-Member

   TypeName: System.RuntimeType
```

Get-Member 를 사용하여 정적 멤버를 보려면 다음과 같이 **Static** 매개 변수를 지정하십시오.

```
PS> [System.Environment] | Get-Member -Static


   TypeName: System.Environment
```

```
Name                        MemberType Definition
----                        ---------- ----------
Equals                      Method     static System.Boolean Equals(Object ob...
Exit                        Method     static System.Void Exit(Int32 exitCode)
...
CommandLine                 Property   static System.String CommandLine {get;}
CurrentDirectory            Property   static System.String CurrentDirectory ...
ExitCode                    Property   static System.Int32 ExitCode {get;set;}
HasShutdownStarted          Property   static System.Boolean HasShutdownStart...
MachineName                 Property   static System.String MachineName {get;}
NewLine                     Property   static System.String NewLine {get;}
OSVersion                   Property   static System.OperatingSystem OSVersio...
ProcessorCount              Property   static System.Int32 ProcessorCount {get;}
StackTrace                  Property   static System.String StackTrace {get;}
SystemDirectory             Property   static System.String SystemDirectory {...
TickCount                   Property   static System.Int32 TickCount {get;}
UserDomainName              Property   static System.String UserDomainName {g...
UserInteractive             Property   static System.Boolean UserInteractive ...
UserName                    Property   static System.String UserName {get;}
Version                     Property   static System.Version Version {get;}
WorkingSet                  Property   static System.Int64 WorkingSet {get;}
TickCount                                ExitCode
```

이제 System.Environment 를 사용하여 표시할 속성을 선택할 수 있습니다.

System.Environment 의 정적 속성 표시

System.Environment 의 속성도 정적이므로 일반적인 속성과 다른 방식으로 지정해야 합니다. Windows PowerShell 에서는 **::**을 사용하여 작업할 정적 메서드나 속성을 나타냅니다. Windows PowerShell 을 시작하는 데 사용된 명령을 보려면 다음과 같이 입력하여 **CommandLine** 속성을 표시하십시오.

```
PS> [System.Environment]::Commandline
"C:\Program Files\Windows PowerShell\v1.0\powershell.exe"
```

운영 체제 버전을 확인하려면 다음과 같이 입력하여 OSVersion 속성을 표시하십시오.

```
PS> [System.Environment]::OSVersion

          Platform ServicePack         Version             VersionString
          -------- -----------         -------             -------------
           Win32NT Service Pack 2      5.1.2600.131072     Microsoft Window...
```

다음과 같이 **HasShutdownStarted** 속성을 표시하면 컴퓨터가 종료 중인지 여부를 확인할 수 있습니다.

```
PS> [System.Environment]::HasShutdownStarted
False
```

System.Math 를 사용하여 산술 연산 수행

System.Math 정적 클래스는 일부 산술 연산을 수행하는 데 유용합니다. **System.Math** 의 중요한 멤버는 대부분 **Get-Member** 를 사용하여 표시할 수 있는 메서드입니다.

참고:

System.Math 에는 동일한 이름을 가진 메서드가 여러 개 있지만 유형이 서로 다릅니다.

다음 명령을 입력하면 **System.Math** 클래스의 메서드를 표시할 수 있습니다.

```
PS> [System.Math] | Get-Member -Static -MemberType Methods


   TypeName: System.Math

Name              MemberType Definition
----              ---------- ----------
Abs               Method     static System.Single Abs(Single value), static Sy...
Acos              Method     static System.Double Acos(Double d)
Asin              Method     static System.Double Asin(Double d)
Atan              Method     static System.Double Atan(Double d)
Atan2             Method     static System.Double Atan2(Double y, Double x)
BigMul            Method     static System.Int64 BigMul(Int32 a, Int32 b)
Ceiling           Method     static System.Double Ceiling(Double a), static Sy...
Cos               Method     static System.Double Cos(Double d)
Cosh              Method     static System.Double Cosh(Double value)
DivRem            Method     static System.Int32 DivRem(Int32 a, Int32 b, Int3...
Equals            Method     static System.Boolean Equals(Object objA, Object ...
Exp               Method     static System.Double Exp(Double d)
Floor             Method     static System.Double Floor(Double d), static Syst...
IEEERemainder     Method     static System.Double IEEERemainder(Double x, Doub...
Log               Method     static System.Double Log(Double d), static System...
Log10             Method     static System.Double Log10(Double d)
Max               Method     static System.SByte Max(SByte val1, SByte val2), ...
Min               Method     static System.SByte Min(SByte val1, SByte val2), ...
Pow               Method     static System.Double Pow(Double x, Double y)
ReferenceEquals   Method     static System.Boolean ReferenceEquals(Object objA...
Round             Method     static System.Double Round(Double a), static Syst...
Sign              Method     static System.Int32 Sign(SByte value), static Sys...
Sin               Method     static System.Double Sin(Double a)
Sinh              Method     static System.Double Sinh(Double value)
Sqrt              Method     static System.Double Sqrt(Double d)
Tan               Method     static System.Double Tan(Double a)
Tanh              Method     static System.Double Tanh(Double value)
Truncate          Method     static System.Decimal Truncate(Decimal d), static...
```

여기에는 여러 가지 산술 메서드가 포함되어 있습니다. 다음은 일반적인 메서드의 작동 방법을 보여 주는 명령 목록입니다.

```
PS> [System.Math]::Sqrt(9)
3
PS> [System.Math]::Pow(2,3)
8
```

```
PS> [System.Math]::Floor(3.3)
3
PS> [System.Math]::Floor(-3.3)
-4
PS> [System.Math]::Ceiling(3.3)
4
PS> [System.Math]::Ceiling(-3.3)
-3
PS> [System.Math]::Max(2,7)
7
PS> [System.Math]::Min(2,7)
2
PS> [System.Math]::Truncate(9.3)
9
PS> [System.Math]::Truncate(-9.3)
-9
```

파이프라인에서 개체 제거(Where-Object)

일반적으로 Windows PowerShell 은 필요한 것보다 많은 개체를 만들어 파이프라인에 전달합니다. **Format** cmdlet 을 사용하면 표시할 특정 개체의 속성을 지정할 수 있지만 전체 개체를 표시하지 않는 문제에는 도움이 되지 않습니다. 파이프라인 끝에 도달하기 전에 개체를 필터링하여 처음 만들어진 개체의 하위 집합에 대해서만 특정 작업을 수행할 수 있습니다.

Windows PowerShell 에는 파이프라인에 있는 각 개체를 테스트하고 특정 테스트 조건을 통과하는 개체만 파이프라인을 통해 전달할 수 있는 **Where-Object** cmdlet 이 포함되어 있습니다. 테스트를 통과하지 못한 개체는 파이프라인에서 제거됩니다. 테스트 조건은 **Where-ObjectFilterScript** 매개 변수의 값으로 지정됩니다.

Where-Object 를 사용하여 간단한 테스트 수행

FilterScript 의 값은 하나의 스크립트 블록으로, true 또는 false 로 평가되는 하나 이상의 Windows PowerShell 명령이 중괄호({})로 묶여 있습니다. 이러한 스크립트 블록은 매우 간단할 수 있지만 비교 연산자와 같은 다른 Windows PowerShell 개념을 알고 있어야 만들 수 있습니다. 비교 연산자는 자신을 중심으로 양쪽에 표시되는 두 항목을 비교합니다. 비교 연산자의 이름은 '-' 문자로 시작됩니다. 기본 비교 연산자는 거의 모든 유형의 개체에서 작동하지만 고급 비교 연산자는 텍스트나 배열에서만 작동할 수 있습니다.

참고:

기본적으로 Windows PowerShell 비교 연산자는 텍스트를 비교할 때 대/소문자를 구분합니다.

구문 오류가 발생할 수 있으므로 **<**, **>**, **=** 등의 기호는 비교 연산자로 사용할 수 없고 문자만 비교 연산자로 사용할 수 있습니다. 다음 표에는 기본 비교 연산자가 나와 있습니다.

비교 연산자	의미	예제(true 반환)
-eq	같음	1 -eq 1
-ne	같지 않음	1 -ne 2
-lt	보다 작음	1 -lt 2
-le	작거나 같음	1 -le 2
-gt	보다 큼	2 -gt 1
-ge	크거나 같음	2 -ge 1
-like	비슷함(텍스트의 와일드카드 비교)	"file.doc" -like "f*.do?"
-notlike	비슷하지 않음(텍스트의 와일드카드 비교)	"file.doc" -notlike "p*.doc"
-contains	포함	1,2,3 -contains 1
-notcontains	포함 안 함	1,2,3 -notcontains 4

Where-Object 스크립트 블록은 특수 변수 '$_'를 사용하여 파이프라인에 있는 현재 개체를 참조합니다. 다음은 이러한 동작을 보여 주는 예제입니다. 일련의 숫자 중에서 3 보다 작은 숫자만 반환하려면 다음과 같이 Where-Object 를 사용하여 숫자를 필터링하면 됩니다.

```
PS> 1,2,3,4 | Where-Object -FilterScript {$_ -lt 3}
1
2
```

개체 속성을 기반으로 필터링

$_가 현재 파이프라인 개체를 참조하므로 테스트를 위해 이 개체의 속성에 액세스할 수 있습니다.

예를 들어 WMI 에서 Win32_SystemDriver 클래스를 살펴보면 특정 시스템에 수백 개의 시스템 드라이버가 있을 수 있지만 현재 실행 중인 드라이버와 같은 특정 시스템 드라이버 집합만 사용할 수 있습니다. Get-Member 를 사용하여 Win32_SystemDriver 멤버(**Get-WmiObject -Class Win32_SystemDriver | Get-Member -MemberType Property**)를 보면 관련 속성이 State 이고 드라이버가 실행 중일 때 해당 값이 "Running"인 것을 확인할 수 있습니다. 다음과 같이 입력하면 시스템 드라이버를 필터링하여 실행 중인 드라이버만 선택할 수 있습니다.

```
Get-WmiObject -Class Win32_SystemDriver | Where-Object -FilterScript {$_.State -eq
"Running"}
```

그러나 이 목록에는 아직도 필요한 것보다 많은 항목이 포함되어 있으므로 다음과 같이 StartMode 값도 테스트하여 자동으로 시작되도록 설정된 드라이버만 선택하도록 필터링할 수 있습니다.

```
PS> Get-WmiObject -Class Win32_SystemDriver | Where-Object -FilterScript {$_.State
```

```
-eq "Running"} | Where-Object -FilterScript {$_.StartMode -eq "Auto"}

DisplayName : RAS Asynchronous Media Driver
Name        : AsyncMac
State       : Running
Status      : OK
Started     : True

DisplayName : Audio Stub Driver
Name        : audstub
State       : Running
Status      : OK
Started     : True
```

드라이버가 실행 중이라는 것을 앞에서 확인했기 때문에 이 목록에는 아직도 불필요한 정보가 포함되어 있습니다. 사실 이 시점에서 필요한 정보는 이름과 표시 이름뿐입니다. 다음 명령은 목록에 이러한 두 속성만 포함하여 출력을 훨씬 더 간단하게 만듭니다.

```
PS> Get-WmiObject -Class Win32_SystemDriver | Where-Object -FilterScript {$_.State
-eq "Running"} | Where-Object -FilterScript {$_.StartMode -eq "Manual"} | Format-
Table -Property Name,DisplayName

Name                                DisplayName
----                                -----------
AsyncMac                            RAS Asynchronous Media Driver
Fdc                                 Floppy Disk Controller Driver
Flpydisk                            Floppy Disk Driver
Gpc                                 Generic Packet Classifier
IpNat                               IP Network Address Translator
mouhid                              Mouse HID Driver
MRxDAV                              WebDav Client Redirector
mssmbios                            Microsoft System Management BIOS Driver
```

위의 명령에는 두 개의 Where-Object 요소가 있지만 다음과 같이 -and 논리 연산자를 사용하면 단일 Where-Object 요소로 표시할 수 있습니다.

```
Get-WmiObject -Class Win32_SystemDriver | Where-Object -FilterScript { ($_.State -
eq "Running") -and ($_.StartMode -eq "Manual") } | Format-Table -Property
Name,DisplayName
```

다음 표에는 기본 논리 연산자가 나와 있습니다.

논리 연산자	의미	예제(true 반환)
-and	논리곱(둘 다 true 일 경우 true 임)	(1 -eq 1) -and (2 -eq 2)
-or	논리합(둘 중 하나만 true 여도 true 임)	(1 -eq 1) -or (1 -eq 2)

논리 연산자	의미	예제(true 반환)
-not	논리 부정(true 와 false 를 반대로 바꿈)	-not (1 -eq 2)
!	논리 부정(true 와 false 를 반대로 바꿈)	!(1 -eq 2)

여러 개체에 대해 작업 반복(ForEach-Object)

ForEach-Object cmdlet 을 사용하면 현재 파이프라인 개체의 스크립트 블록과 $_ 설명자를 통해 파이프라인에 있는 각 개체에 대해 명령을 실행할 수 있습니다. 이 cmdlet 은 복잡한 작업을 수행하는 데 사용할 수 있습니다.

이 cmdlet 은 데이터를 조작하여 보다 사용하기 쉽게 만드는 데 유용할 수 있습니다. 예를 들어 WMI 의 Win32_LogicalDisk 클래스를 사용하여 각 로컬 디스크의 사용 가능한 공간 정보를 반환할 수 있습니다. 그러나 이 정보는 다음과 같이 쉽게 읽을 수 없는 바이트 단위로 반환됩니다.

```
PS> Get-WmiObject -Class Win32_LogicalDisk


DeviceID     : C:
DriveType    : 3
ProviderName :
FreeSpace    : 50665070592
Size         : 203912880128
VolumeName   : Local Disk
```

이러한 값을 1024 로 차례로 두 번 나누면 MB 단위로 변환할 수 있습니다. 즉, 첫 번째 나누기에서 KB 단위로 변환되고 두 번째 나누기에서 MB 단위로 변환됩니다. 다음과 같이 입력하면 ForEach-Object 스크립트 블록에서 이렇게 할 수 있습니다.

```
Get-WmiObject -Class Win32_LogicalDisk | ForEach-Object –Process
{($_.FreeSpace)/1024.0/1024.0}
48318.01171875
```

그러나 이 출력에는 데이터를 나타내는 레이블이 포함되지 않습니다. 이러한 WMI 속성은 읽기 전용이므로 직접 변환할 수 없습니다. 예를 들어 다음과 같이 입력할 수 있습니다.

```
Get-WmiObject -Class Win32_LogicalDisk | ForEach-Object -Process {$_.FreeSpace =
($_.FreeSpace)/1024.0/1024.0}
```

그러면 다음과 같은 오류메시지가 나타납니다.

```
"FreeSpace"은(는) ReadOnly 속성입니다.
줄:1 문자:70
+ Get-WmiObject -Class Win32_LogicalDisk | ForEach-Object -Process {$_.F <<<< r
eeSpace = ($_.FreeSpace)/1024.0/1024.0}
```

고급 기술을 사용하여 데이터를 다시 구성할 수도 있지만 **Select-Object** 를 사용하여 새 개체를 만드는 것이 훨씬 더 간단합니다.

개체의 일부 선택(Select-Object)

Select-Object cmdlet 을 사용하면 새로운 사용자 지정 Windows PowerShell 개체를 만들 수 있습니다. 이러한 개체에는 해당 개체를 만드는 데 사용된 개체에서 선택한 속성이 포함되어 있습니다. 다음 명령을 입력하면 Win32_LogicalDisk WMI 클래스의 Name 및 FreeSpace 속성만 포함된 새 개체를 만들 수 있습니다.

```
PS> Get-WmiObject -Class Win32_LogicalDisk | Select-Object -Property
Name,FreeSpace

Name                                    FreeSpace
----                                    ---------
C:                                      50664845312
```

이 명령을 실행해도 데이터 유형이 표시되지 않지만 다음과 같이 Select-Object 뒤에서 결과를 Get-Member 에 파이프하면 새 유형의 개체인 PSCustomObject 가 만들어지는 것을 확인할 수 있습니다.

```
PS> Get-WmiObject -Class Win32_LogicalDisk | Select-Object -Property
Name,FreeSpace| Get-Member



   TypeName: System.Management.Automation.PSCustomObject

Name        MemberType   Definition
----        ----------   ----------
Equals      Method       System.Boolean Equals(Object obj)
GetHashCode Method       System.Int32 GetHashCode()
GetType     Method       System.Type GetType()
ToString    Method       System.String ToString()
FreeSpace   NoteProperty  FreeSpace=...
Name        NoteProperty System.String Name=C:
```

Select-Object 는 다양한 용도로 사용할 수 있는데, 그 중 하나는 데이터를 복제하는 것입니다(이러한 데이터는 나중에 수정할 수 있음). 이렇게 하면 앞 절에서 설명한 문제를 해결할 수 있습니다. 새로 만든 개체에서 FreeSpace 값을 업데이트하면 다음과 같이 설명이 포함된 레이블이 출력됩니다.

```
Get-WmiObject -Class Win32_LogicalDisk | Select-Object -Property Name,FreeSpace |
ForEach-Object -Process {$_.FreeSpace = ($_.FreeSpace)/1024.0/1024.0; $_}
Name                                                                  FreeSpace
----                                                                  ---------
C:                                                                  48317.7265625
```

개체 정렬

Sort-Object cmdlet 을 사용하여 표시된 데이터를 검색하기 쉽게 구성할 수 있습니다. **Sort-Object** 는 정렬 기준으로 사용할 하나 이상의 속성 이름을 선택하고 이러한 속성의 값을 기준으로 정렬된 데이터를 반환합니다.

예를 들어 Win32_SystemDriver 인스턴스를 표시할 때 정렬 기준으로 **State** 와 **Name** 을 차례로 사용하여 정렬하려면 다음과 같이 입력하면 됩니다.

```
Get-WmiObject -Class Win32_SystemDriver | Sort-Object -Property State,Name |
Format-Table -Property Name,State,Started,DisplayName -AutoSize -Wrap
```

이 목록에는 필요한 것보다 많은 항목이 포함되어 있지만 다음과 같이 상태가 동일한 항목이 그룹화되어 표시됩니다.

```
Name            State   Started DisplayName
----            -----   ------- -----------
ACPI            Running    True Microsoft ACPI Driver
AFD             Running    True AFD
AmdK7           Running    True AMD K7 Processor Driver
AsyncMac        Running    True RAS Asynchronous Media Driver
...
Abiosdsk        Stopped   False Abiosdsk
ACPIEC          Stopped   False ACPIEC
aec             Stopped   False Microsoft Kernel Acoustic Echo Canceller
...
```

또한 **Descending** 매개 변수를 지정하여 개체를 반대 순서로 정렬할 수도 있습니다. 이렇게 하면 이름은 사전 반대순으로 정렬되고 번호는 내림차순으로 정렬됩니다.

```
PS> Get-WmiObject -Class Win32_SystemDriver | Sort-Object -Property State,Name -
Descending | Format-Table -Property Name,State,Started,DisplayName -AutoSize -Wrap

Name            State   Started DisplayName
----            -----   ------- -----------
WS2IFSL         Stopped   False Windows Socket 2.0 Non-IFS Service Provider Supp
                                ort Environment
wceusbsh        Stopped   False Windows CE USB Serial Host Driver...
...
wdmaud          Running    True Microsoft WINMM WDM Audio Compatibility Driver
Wanarp          Running    True Remote Access IP ARP Driver
...
```

변수를 사용하여 개체 저장

Windows PowerShell 에서는 개체에 대한 작업을 수행합니다. Windows PowerShell 에서는 이름이 기본적으로 지정되는 개체인 변수를 만들어 출력을 나중에 사용하기 위해 저장할 수 있습니다. 다른 셸에서 변수에 대한 작업을 수행하는 데 익숙한 경우 Windows PowerShell 변수는 텍스트가 아닌 개체라는 사실에 주의해야 합니다.

변수 이름은 항상 **$** 문자로 시작되며 모든 영숫자 문자와 밑줄을 사용할 수 있습니다.

변수 만들기

변수를 만들려면 다음과 같이 유효한 변수 이름을 입력해야 합니다.

```
PS> $loc
PS>
```

이 경우 **$loc** 에 지정된 값이 없기 때문에 아무 결과도 표시되지 않습니다. 동일한 단계에서 변수를 만드는 작업과 변수에 값을 할당하는 작업을 수행할 수 있습니다. Windows PowerShell 은 변수가 없으면 변수를 만들기만 하고, 변수가 있으면 기존 변수에 지정된 값을 할당합니다. **$loc** 변수에 현재 위치를 저장하려면 다음과 같이 입력하십시오.

```
$loc = Get-Location
```

이 명령을 입력하면 출력이 $loc 에 보내지기 때문에 아무 결과도 표시되지 않습니다. 출력이 표시된다면 이는 달리 리디렉션되지 않은 데이터가 항상 화면에 표시되기 때문에 발생하는 의도하지 않은 결과입니다. 다음과 같이 $loc 를 입력하면 현재 위치가 표시됩니다.

```
PS> $loc

Path
----
C:\temp
```

Get-Member 를 사용하여 변수의 내용에 대한 정보를 표시할 수 있습니다. 다음과 같이 $loc 를 Get-Member 에 파이프하면 Get-Location 을 입력한 것과 마찬가지로 변수에 **PathInfo** 개체가 포함되어 있는 것을 확인할 수 있습니다.

```
PS> $loc | Get-Member -MemberType Property


   TypeName: System.Management.Automation.PathInfo

Name         MemberType Definition
----         ---------- ----------
Drive        Property   System.Management.Automation.PSDriveInfo Drive {get;}
Path         Property   System.String Path {get;}
Provider     Property   System.Management.Automation.ProviderInfo Provider {...
ProviderPath Property   System.String ProviderPath {get;}
```

변수 조작

Windows PowerShell 에는 변수를 조작할 수 있는 여러 명령이 포함되어 있습니다. 다음과 같이 입력하면 이러한 명령의 전체 목록을 쉽게 확인할 수 있습니다.

```
Get-Command -Noun Variable | Format-Table -Property Name,Definition -AutoSize -
Wrap
```

현재 Windows PowerShell 세션에서 사용자가 직접 만든 변수 외에도 여러 시스템 정의 변수가 있습니다. **Remove-Variable** cmdlet 을 사용하여 Windows PowerShell 에서 제어하지 않는 모든 변수를 지울 수 있습니다. 다음 명령을 입력하면 모든 변수를 지울 수 있습니다.

```
Remove-Variable -Name * -Force -ErrorAction SilentlyContinue
```

이 경우 다음과 같은 확인 메시지가 나타납니다.

```
Confirm
이 작업을 수행하시겠습니까?
대상 "Name: Error"에서 "Remove Variable" 작업을 수행합니다.
[Y] 예  [A] 모두 예  [N] 아니요  [L] 모두 아니요  [S] 일시 중단  [?] 도움말
(기본값은 "Y"임):A
```

Get-Variable cmdlet 을 실행하면 나머지 Windows PowerShell 변수를 볼 수 있습니다. 또한 변수 Windows PowerShell 드라이브도 있으므로 다음과 같이 입력하면 모든 Windows PowerShell 변수를 표시할 수도 있습니다.

```
Get-ChildItem variable:
```

Cmd.exe 변수 사용

Windows PowerShell 은 Cmd.exe 가 아니지만 명령 셸 환경에서 실행되며 Windows 환경에서 사용할 수 있는 것과 동일한 변수를 사용할 수 있습니다. 이러한 변수는 **env**:라는 드라이브를 통해 표시됩니다. 다음과 같이 입력하면 이러한 변수를 볼 수 있습니다.

```
Get-ChildItem env:
```

표준 변수 cmdlet 은 원래 설계 의도와 달리 **env:** 접두사를 지정하면 **env:** 변수와 함께 사용할 수 있습니다. 예를 들어 운영 체제 루트 디렉터리를 보려면 다음과 같이 입력하여 Windows PowerShell 에서 명령 셸 **%SystemRoot%** 변수를 사용하면 됩니다.

```
PS> $env:SystemRoot
C:\WINDOWS
```

또한 Windows PowerShell 에서 환경 변수를 만들고 수정할 수도 있습니다. Windows PowerShell 에서 액세스하는 환경 변수는 다른 Windows 환경 변수에 대한 일반적인 규칙을 따릅니다.

Windows PowerShell 을 사용하여 관리 작업 수행

Windows PowerShell 의 기본적인 목표는 대화형 환경이나 스크립트에서 시스템을 보다 쉽고 보다 잘 관리하는 것입니다. 이 장에서는 Windows PowerShell 을 사용하여 Windows 시스템을 관리할 때 발생하는 다양한 문제를 해결하는 방법을 설명합니다. 이 설명서에는 스크립트나 함수에 대한 설명이 나와 있지 않지만 이 장에서 설명하는 문제 해결 방법에는 스크립트나 함수가 사용될 수 있습니다. 여기에는 문제 해결 방법의 일부로 함수를 사용하는 예제도 제공됩니다.

이 장에서 설명하는 문제 해결 방법에는 특정 cmdlet 이나 일반 Get-WmiObject cmdlet 뿐 아니라 Windows 및 .NET 인프라에 포함된 외부 도구를 사용하는 다양한 문제 해결 방법이 포함되어 있습니다. 외부 도구의 사용은 Windows PowerShell 의 장기적인 설계 의도에 포함된 내용입니다. 시스템이 커지기만 해도 사용 가능한 도구 집합으로 필요한 작업을 모두 수행할 수 없는 문제가 지속적으로 발생하게 됩니다. Windows PowerShell 은 cmdlet 으로만 문제를 해결하려고 하는 대신 가능한 모든 방법을 동원하여 문제를 해결하려고 합니다.

로컬 프로세스 관리

Process cmdlet 에는 두 개의 핵심 cmdlet, **Get-Process** 와 **Stop-Process** 가 있습니다. 매개 변수나 Object cmdlet 을 사용하여 프로세스를 검토하거나 필터링할 수 있으므로 이러한 두 cmdlet 만으로도 복잡한 작업을 수행할 수 있습니다.

프로세스 표시(Get-Process)

매개 변수 없이 **Get-Process** 를 실행하여 로컬 시스템에서 실행 중인 모든 프로세스의 목록을 볼 수 있습니다.

또한 Id 매개 변수를 사용하여 ProcessId 를 지정하면 단일 프로세스를 반환할 수도 있습니다. 다음 예제는 유휴 상태의 시스템 프로세스를 반환합니다.

```
PS> Get-Process -Id 0
Handles  NPM(K)    PM(K)      WS(K) VM(M)   CPU(s)     Id ProcessName
-------  ------    -----      ----- -----   ------     -- -----------
      0       0        0         16     0                0 Idle
```

일반적으로 cmdlet 이 아무 데이터도 반환하지 않아도 오류 메시지가 나타나지 않지만 ProcessId 를 사용하여 프로세스를 지정한 경우 **Get-Process** 가 일치하는 항목을 찾지 못하면 오류 메시지가 나타납니다. 이것은 **Get-Process** 의 기본 목적이 실행 중인 프로세스를 검색하는 것이기 때문입니다. 해당 Id 를 가진 프로세스가 없으면 Id 가 잘못되었거나 해당 프로세스가 이미 종료된 것일 수 있습니다.

```
PS> Get-Process -Id 99
Get-Process : ID가 99인 프로세스를 찾을 수 없습니다.
줄:1 문자:12
+ Get-Process  <<<< -Id 99
```

Name 매개 변수를 사용하면 프로세스 이름을 기반으로 프로세스의 하위 집합을 지정할 수 있습니다. 동일한 이름을 가진 프로세스가 여러 개 있을 수 있으므로 둘 이상의 프로세스가 출력될 수도 있습니다. 해당 이름을 가진 프로세스가 없으면 ProcessId 를 지정했을 때와 마찬가지로 **Get-Process** 도 오류를 반환합니다. 예를 들어 프로세스 이름을 **explorer** 대신 **explore** 로 지정하면 다음과 같은 오류 메시지가 나타납니다.

```
PS> Get-Process -Name explore
Get-Process : 이름이 'explore'인 프로세스를 찾을 수 없습니다.
줄:1 문자:12
+ Get-Process  <<<< -Name explore
```

Name 매개 변수는 와일드카드의 사용을 허용하므로 다음과 같이 이름의 첫 몇 자 뒤에 별표를 입력하여 목록을 표시할 수 있습니다.

```
PS> Get-Process -Name ex*
Handles  NPM(K)    PM(K)      WS(K) VM(M)   CPU(s)     Id ProcessName
-------  ------    -----      ----- -----   ------     -- -----------
    234       7     5572      12484   134     2.98   1684 EXCEL
    555      15    34500      12384   134   105.25    728 explorer
```

참고:

.NET System.Diagnostics.Process 클래스는 Windows PowerShell 프로세스의 기초이므로 System.Diagnostics.Process 가 사용하는 일부 규칙을 따릅니다. 이러한 규칙 중 하나는 실행 파일의 프로세스 이름에 ".exe" 확장명을 포함하지 않는 것입니다.

Get-Process 는 또한 Name 매개 변수에 대해 여러 값을 허용합니다. 단일 이름을 지정한 것처럼 일치하는 이름을 찾을 수 없으면 다음과 같이 일반적으로 일치 프로세스의 결과로 출력되는 내용과 함께 오류 메시지가 나타납니다.

```
PS> Get-Process -Name exp*,power*,NotAProcess
Get-Process : 프로세스 이름이 'NotAProcess'인 프로세스를 찾지 못했습니다.
줄:1 문자:12
+ Get-Process  <<<< -Name exp*,power*,svchost,NotAProcess
Handles  NPM(K)    PM(K)      WS(K) VM(M)   CPU(s)     Id ProcessName
-------  ------    -----      ----- -----   ------     -- -----------
    540      15    35172      48148   141    88.44    408 explorer
    605       9    30668      29800   155     7.11   3052 powershell
```

프로세스 중지(Stop-Process)

Windows PowerShell 에서는 다양한 방법으로 프로세스를 표시할 수 있을 뿐 아니라 프로세스를 중지할 수도 있습니다.

Stop-Process cmdlet 은 이름이나 Id 를 사용하여 중지할 프로세스를 지정합니다. 프로세스를 중지할 수 있는지 여부는 사용 권한에 의해 결정됩니다. 일부 프로세스는 중지할 수 없습니다. 예를 들어 유휴 프로세스를 중지하려고 하면 다음과 같은 오류 메시지가 나타납니다.

```
PS> Stop-Process -Name Idle
Stop-Process : 다음 오류로 인해 'Idle (0)' 프로세스를 중지할 수 없습니다:
 액세스가 거부되었습니다.
줄:1 문자:13
+ Stop-Process  <<<< -Name Idle
```

또한 강제로 **Confirm** 매개 변수를 사용하도록 지정할 수도 있습니다. 이 매개 변수는 프로세스 이름을 지정할 때 와일드카드를 사용할 경우 원하지 않는 프로세스가 중지될 수 있으므로 특히 유용합니다.

```
PS> Stop-Process -Name t*,e* -Confirm
Confirm
이 작업을 수행하시겠습니까?
대상 "explorer (408)"에서 "Stop-Process" 작업을 수행합니다.
[Y] 예  [A] 모두 예  [N] 아니요  [L] 모두 아니요  [S] 일시 중단  [?] 도움말
(기본값은 "Y"임):n
Confirm
```

```
이 작업을 수행하시겠습니까?
대상 "taskmgr (4072)"에서 "Stop-Process" 작업을 수행합니다.
[Y] 예  [A] 모두 예  [N] 아니요  [L] 모두 아니요  [S] 일시 중단  [?] 도움말
(기본값은 "Y"임):n
```

일부 개체 필터링 cmdlet 을 사용하면 복잡한 프로세스 조작이 가능합니다. 프로세스 개체에는 더 이상 응답이 없을 때 true 인 응답 속성이 있으므로 다음 명령을 사용하여 응답이 없는 모든 응용 프로그램을 중지할 수 있습니다.

```
Get-Process | Where-Object -FilterScript {$_.Responding -eq $false} | Stop-Process
```

위와 동일한 방법을 다른 경우에 사용할 수도 있습니다. 예를 들어 다른 응용 프로그램을 시작하면 보조 시스템 트레이 응용 프로그램이 자동으로 실행된다고 가정할 경우 이 보조 시스템 트레이 응용 프로그램이 터미널 서비스 세션에서 올바로 작동하지 않는 것을 알았지만 실제 컴퓨터 콘솔에서 실행되는 세션에서 이 응용 프로그램을 계속 실행할 수 있습니다. 실제 컴퓨터의 데스크톱에 연결된 세션에는 항상 세션 ID 로 0 이 지정되므로 다음과 같이 **Where-Object** 와 **SessionId** 프로세스를 사용하여 다른 세션에 있는 프로세스의 인스턴스를 모두 중지할 수 있습니다.

```
Get-Process -Name BadApp | Where-Object -FilterScript {$_.SessionId -neq 0} |
Stop-Process
```

다른 모든 Windows PowerShell 세션 중지

경우에 따라 현재 세션을 제외하고 실행 중인 다른 모든 Windows PowerShell 세션을 중지할 수 있는 것이 유용할 수 있습니다. 세션에서 너무 많은 리소스를 사용하고 있거나 세션에 액세스할 수 없으면(원격에서 실행 중이거나 다른 데스크톱 세션에서 실행 중인 경우) 세션을 직접 중지하지 못할 수 있습니다. 그러나 실행 중인 세션을 중지하려고 하면 대신 현재 세션이 중지될 수 있습니다.

각 Windows PowerShell 세션에는 Windows PowerShell 프로세스의 Id 가 들어 있는 환경 변수 PID 가 있습니다. 각 세션의 Id 에 대해 $PID 를 확인하고 다른 Id 를 가진 Windows PowerShell 세션만 중지할 수 있습니다. 다음 파이프라인 명령은 이 작업을 수행하고 종료된 세션의 목록을 반환합니다. 이 명령이 종료된 세션의 목록을 반환하는 것은 **PassThru** 매개 변수를 사용하기 때문입니다.

```
PS> Get-Process -Name powershell | Where-Object -FilterScript {$_.Id -ne $PID} |
Stop-Process -
PassThru
Handles  NPM(K)    PM(K)      WS(K) VM(M)   CPU(s)     Id ProcessName
-------  ------    -----      ----- -----   ------     -- -----------
    334       9    23348      29136   143     1.03    388 powershell
    304       9    23152      29040   143     1.03    632 powershell
    302       9    20916      26804   143     1.03   1116 powershell
    335       9    25656      31412   143     1.09   3452 powershell
    303       9    23156      29044   143     1.05   3608 powershell
    287       9    21044      26928   143     1.02   3672 powershell
```

로컬 서비스 관리

핵심 Service cmdlet 에는 다양한 서비스 작업을 위해 설계된 여덟 개의 cmdlet 이 있습니다. 이 설명서에서는 실행 중인 서비스의 상태를 표시하고 변경하는 방법에 대해서만 설명하지만 **Get-Help *-Service** 를 사용하여 Service cmdlet 의 목록을 보거나 **Get-Help New-Service** 와 같은 **Get-Help<Cmdlet-Name>**을 사용하여 각 Service cmdlet 에 대한 정보를 찾을 수도 있습니다.

서비스 표시

Get-Service 를 사용하여 컴퓨터에 있는 로컬 서비스를 나열할 수 있습니다. **Get-Process** 와 마찬가지로 매개 변수 없이 **Get-Service** 명령을 사용하면 모든 서비스가 반환됩니다. 이때 다음과 같이 별표를 와일드카드로 사용하여 이름을 기준으로 필터링할 수도 있습니다.

```
PS> Get-Service -Name se*
Status   Name               DisplayName
------   ----               -----------
Running  seclogon           Secondary Logon
Running  SENS               System Event Notification
Stopped  ServiceLayer       ServiceLayer
```

서비스의 실제 이름이 항상 분명한 것은 아니기 때문에 서비스를 실제 이름으로 찾아야 할 수도 있습니다. 이렇게 하려면 다음과 같이 와일드카드를 사용하거나 표시 이름 목록을 사용하여 특정 이름으로 찾으면 됩니다.

```
PS> Get-Service -DisplayName se*
Status   Name               DisplayName
------   ----               -----------
Running  lanmanserver       Server
Running  SamSs              Security Accounts Manager
Running  seclogon           Secondary Logon
Stopped  ServiceLayer       ServiceLayer
Running  wscsvc             Security Center
PS> Get-Service -DisplayName ServiceLayer,Server
Status   Name               DisplayName
------   ----               -----------
Running  lanmanserver       Server
Stopped  ServiceLayer       ServiceLayer
```

서비스 중지, 시작, 일시 중단 및 다시 시작

Service cmdlet 이 모두 동일한 일반 형식으로 되어 있기 때문에 서비스는 일반 이름이나 표시 이름으로 지정될 수 있으며 목록과 와일드카드를 값으로 사용할 수 있습니다. 인쇄 스풀러를 중지하려면 다음 명령을 사용하십시오.

```
Stop-Service -Name spooler
```

인쇄 스풀러를 중지한 후 다시 시작하려면 다음 명령을 사용하십시오.

```
Start-Service -Name spooler
```

인쇄 스풀러를 일시 중단하려면 다음 명령을 사용하십시오.

```
Suspend-Service -Name spooler
```

Restart-Service cmdlet 은 다른 Service cmdlet 과 동일한 방식으로 작동하지만 이 설명서에서는 이 cmdlet 에 대해 약간 더 복잡한 예제를 보여 줍니다. 가장 간단한 사용 시나리오에서는 다음과 같이 서비스 이름을 지정합니다.

```
PS> Restart-Service -Name spooler
경고: 'Print Spooler (Spooler)' 서비스가 시작될 때까지 기다리는 중...
경고: 'Print Spooler (Spooler)' 서비스가 시작될 때까지 기다리는 중...
PS>
```

그러면 인쇄 스풀러 시작에 대한 경고 메시지가 반복해서 나타나는데, 이것은 다소 시간이 걸리는 서비스 작업을 수행하면 Windows PowerShell 이 계속 작업을 시도하고 있다는 내용의 메시지를 표시하기 때문입니다.

여러 서비스를 다시 시작하려면 다음과 같이 서비스 목록을 보고, 필터링한 다음 원하는 서비스를 다시 시작하면 됩니다.

```
PS> Get-Service | Where-Object -FilterScript {$_.CanStop} | Restart-Service
경고: 'Computer Browser (Browser)' 서비스가 중지될 때까지 기다리는 중...
경고: 'Computer Browser (Browser)' 서비스가 중지될 때까지 기다리는 중...
Restart-Service : 'Logical Disk Manager (dmserver)' 서비스는 종속 서비스를 포함하므로
중지할 수 없습니다. Force 플래그가 설정된 경우에만 이 서비스를 중지할 수 있습니다.
줄:1 문자:57
+ Get-Service | Where-Object -FilterScript {$_.CanStop} | Restart-Service <<<<
경고: 'Print Spooler (Spooler)' 서비스가 시작될 때까지 기다리는 중...
경고: 'Print Spooler (Spooler)' 서비스가 시작될 때까지 기다리는 중...
```

컴퓨터에 대한 정보 수집

Get-WmiObject 는 일반적인 시스템 관리 작업에 가장 중요한 cmdlet 입니다. 중요한 하위 시스템 설정은 모두 WMI 를 통해 표시됩니다. 게다가 WMI 는 데이터를 하나 이상의 항목 컬렉션에 있는 데이터로 취급합니다. 또한 Windows PowerShell 에서는 개체에 대한 작업을 수행하고 하나 또는 여러 개의 개체를 동일한 방식으로 취급할 수 있는 파이프라인을 사용하므로 일반적인 WMI 액세스를 통해 거의 자동으로 고급 작업을 수행할 수 있습니다.

다음 예제에서는 임의의 컴퓨터에 대해 **Get-WmiObject** 를 사용하여 특정 정보를 수집하는 방법을 보여 줍니다. 이러한 예제에서는 **ComputerName** 매개 변수가 로컬 컴퓨터를 나타내는 점(**.**)으로 지정되어 있지만 WMI 를 통해 액세스할 수 있는 컴퓨터에 연결된 이름이나 IP 주소를 지정할 수도 있습니다. 로컬 컴퓨터에 대한 정보를 검색하려는 경우에는 **–ComputerName** 을 생략해도 됩니다.

데스크톱 설정 표시

다음 명령은 로컬 컴퓨터에 있는 데스크톱에 대한 정보를 수집합니다.

```
Get-WmiObject -Class Win32_Desktop -ComputerName .
```

이 명령은 사용 여부에 관계없이 모든 데스크톱에 대한 정보를 반환합니다.

참고:

일부 WMI 클래스가 반환하는 정보에는 많은 내용이 포함될 수 있으며 종종 WMI 클래스에 대한 메타데이터가 포함되기도 합니다. 이러한 메타데이터 속성은 대부분 이름이 두 개의 밑줄로 시작되므로 Select-Object 를 사용하여 속성을 필터링할 수 있습니다. 다음과 같이 속성 값으로 **[a-z]*** 를 사용하면 알파벳 문자로 시작하는 속성만 지정하십시오.

```
Get-WmiObject -Class Win32_Desktop -ComputerName . | Select-Object -Property [a-
z]*
```

메타데이터를 필터링하여 제외시키려면 파이프 연산자(|)를 사용하여 Get-WmiObject 명령의 결과를 **Select-Object -Property [a-z]*** 에 보내십시오.

BIOS 정보 표시

다음 WMI Win32_BIOS 클래스는 로컬 컴퓨터에 있는 시스템 BIOS 에 대해 매우 간단하지만 완전한 정보를 반환합니다.

```
Get-WmiObject -Class Win32_BIOS -ComputerName .
```

프로세서 정보 표시

다음 WMI **Win32_Processor** 클래스는 일반적인 프로세서 정보를 반환합니다. 이러한 정보는 나중에 필터링할 수 있습니다.

```
Get-WmiObject -Class Win32_Processor -ComputerName . | Select-Object -Property [a-
z]*
```

프로세서 계열에 대한 일반적인 설명을 보려면 **Win32_ComputerSystemSystemType** 속성만 반환하면 됩니다.

```
PS> Get-WmiObject -Class Win32_ComputerSystem -ComputerName . | Select-Object -
Property SystemType
SystemType
----------
X86-based PC
```

컴퓨터 제조업체 및 모델 표시

Win32_ComputerSystem 을 사용하여 컴퓨터 모델 정보를 볼 수도 있습니다. 다음과 같이 표준 표시 출력 결과는 필터링 없이 OEM 데이터를 제공합니다.

```
PS> Get-WmiObject -Class Win32_ComputerSystem
Domain              : WORKGROUP
Manufacturer        : Compaq Presario 06
Model               : DA243A-ABA 6415cl NA910
Name                : MyPC
```

```
PrimaryOwnerName    : Jane Doe
TotalPhysicalMemory : 804765696
```

일부 하드웨어에서 직접 정보를 반환하는 이와 같은 명령을 실행하면 사용자가 가지고 있는 것과 동일한 데이터가 출력됩니다. 일부 정보는 하드웨어 제조업체에서 제대로 구성하지 않아 사용하지 못할 수 있습니다.

설치된 핫픽스 표시

다음과 같이 **Win32_QuickFixEngineering** 을 사용하여 설치된 핫픽스를 모두 표시할 수 있습니다.

```
Get-WmiObject -Class Win32_QuickFixEngineering -ComputerName .
```

이 클래스는 다음과 같은 핫픽스 목록을 반환합니다.

```
Description          : Update for Windows XP (KB910437)
FixComments          : Update
HotFixID             : KB910437
Install Date         :
InstalledBy          : Administrator
InstalledOn          : 12/16/2005
Name                 :
ServicePackInEffect  : SP3
Status               :
```

일부 속성을 제외하여 출력을 더 간단하게 만들 수 있습니다. 다음과 같이 **Get-WmiObject Property** 매개 변수를 사용하여 **HotFixID** 만 선택할 수 있지만, 모든 메타데이터가 기본적으로 표시되므로 실제로는 더 많은 정보가 반환됩니다.

```
PS> Get-WmiObject -Class Win32_QuickFixEngineering -ComputerName . -Property
HotFixId
HotFixID          : KB910437
__GENUS           : 2
__CLASS           : Win32_QuickFixEngineering
__SUPERCLASS      :
__DYNASTY         :
__RELPATH         :
__PROPERTY_COUNT  : 1
__DERIVATION      : {}
__SERVER          :
__NAMESPACE       :
__PATH            :
```

데이터가 더 반환되는 것은 **Get-WmiObject** 의 Property 매개 변수가 Windows PowerShell 에 반환된 개체 대신 WMI 클래스 인스턴스에서 반환된 속성을 제한하기 때문입니다. 출력을 더 간단하게 만들려면 다음과 같이 **Select-Object** 를 사용하십시오.

```
PS> Get-WmiObject -Class Win32_QuickFixEngineering -ComputerName . -Property Hot
FixId | Select-Object -Property HotFixId
HotFixId
--------
```

```
KB910437
```

운영 체제의 버전 정보 표시

Win32_OperatingSystem 클래스 속성에는 버전 및 서비스 팩 정보가 포함되어 있습니다. 다음과 같이 이러한 속성만 명시적으로 선택하여 **Win32_OperatingSystem** 의 버전에 대한 요약 정보를 볼 수 있습니다.

```
Get-WmiObject -Class Win32_OperatingSystem -ComputerName . | Select-Object -
Property
BuildNumber,BuildType,OSType,ServicePackMajorVersion,ServicePackMinorVersion
```

또한 **Select-Object Property** 매개 변수와 함께 와일드카드를 사용할 수 있습니다. **Build** 나 **ServicePack** 으로 시작하는 모든 속성을 사용해야 하므로 이 속성을 다음과 같이 간단하게 만들 수 있습니다.

```
PS> Get-WmiObject -Class Win32_OperatingSystem -ComputerName . | Select-Object -
Property Build*,OSType,ServicePack*

BuildNumber             : 2600
BuildType               : Uniprocessor Free
OSType                  : 18
ServicePackMajorVersion : 2
ServicePackMinorVersion : 0
```

로컬 사용자 및 소유자 표시

Win32_OperatingSystem 속성을 선택하면 정식 사용자 수, 현재 사용자 수 및 소유자 이름 등의 일반적인 로컬 사용자 정보를 찾을 수 있습니다. 속성을 명시적으로 선택하여 다음과 같이 표시할 수 있습니다.

```
Get-WmiObject -Class Win32_OperatingSystem -ComputerName . | Select-Object -
Property NumberOfLicensedUsers,NumberOfUsers,RegisteredUser
```

와일드카드를 사용한 더 간단한 버전은 다음과 같습니다.

```
Get-WmiObject -Class Win32_OperatingSystem -ComputerName . | Select-Object -
Property *user*
```

사용 가능한 디스크 공간 보기

WMI Win32_LogicalDisk 클래스를 사용하여 로컬 드라이브의 디스크 공간과 사용 가능한 공간을 확인할 수 있습니다. WMI 가 고정 하드 디스크에 대해 사용하는 값인 DriveType 3 을 갖는 인스턴스만 확인하면 됩니다.

```
Get-WmiObject -Class Win32_LogicalDisk -Filter "DriveType=3" -ComputerName .

DeviceID     : C:
DriveType    : 3
ProviderName :
```

```
FreeSpace    : 65541357568
Size         : 203912880128
VolumeName   : Local Disk

DeviceID     : Q:
DriveType    : 3
ProviderName :
FreeSpace    : 44298250240
Size         : 122934034432
VolumeName   : New Volume

PS> Get-WmiObject -Class Win32_LogicalDisk -Filter "DriveType=3" -ComputerName . |
Measure-Object -Property FreeSpace,Size -Sum

Get-WmiObject -Class Win32_LogicalDisk -Filter "DriveType=3" -ComputerName . |
Measure-Object -Property FreeSpace,Size -Sum | Select-Object -Property
Property,Sum
```

로그온 세션 정보 보기

다음과 같이 WMI Win32_LogonSession 클래스를 통해 사용자와 관련된 로그온 세션의 일반 정보를 볼 수 있습니다.

```
Get-WmiObject -Class Win32_LogonSession -ComputerName .
```

컴퓨터에 로그온한 사용자 보기

Win32_ComputerSystem 을 사용하여 특정 컴퓨터 시스템에 로그온한 사용자를 표시할 수 있습니다. 이 명령은 다음과 같이 시스템 데스크톱에 로그온한 사용자만 반환합니다.

```
Get-WmiObject -Class Win32_ComputerSystem -Property UserName -ComputerName .
```

컴퓨터에서 현지 시간 보기

WMI Win32_LocalTime 클래스를 사용하여 특정 컴퓨터에서 현재 현지 시간을 검색할 수 있습니다. 이 클래스는 기본적으로 모든 메타데이터를 표시하므로 다음과 같이 **Select-Object** 를 사용하여 이러한 메타데이터를 필터링할 수 있습니다.

```
PS> Get-WmiObject -Class Win32_LocalTime -ComputerName . | Select-Object -Property
[a-z]*


Day          : 15
DayOfWeek    : 4
Hour         : 12
Milliseconds :
Minute       : 11
Month        : 6
```

```
Quarter      : 2
Second       : 52
WeekInMonth  : 3
Year         : 2006
```

서비스 상태 표시

앞에서 설명한 대로 **Get-Service** cmdlet 을 로컬로 사용하여 특정 컴퓨터의 모든 서비스 상태를 볼 수 있습니다. 원격 시스템의 경우 WMI Win32_Service 클래스를 사용할 수 있습니다. 또한 **Select-Object** 를 사용하여 **Status**, **Name** 및 **DisplayName** 에 대한 결과를 필터링하는 경우 출력 형식은 다음과 같이 **Get-Service** 와 거의 동일합니다.

```
Get-WmiObject -Class Win32_Service -ComputerName . | Select-Object -Property
Status,Name,DisplayName
```

매우 긴 이름으로 일반 서비스의 전체 이름을 표시하는 경우 다음과 같이 **AutoSize** 및 **Wrap** 매개 변수와 함께 **Format-Table** 을 사용하여 열 너비를 최적화하고 긴 이름을 자르는 대신 래핑할 수 있습니다.

```
Get-WmiObject -Class Win32_Service -ComputerName . | Format-Table -Property
Status,Name,DisplayName -AutoSize -Wrap
```

소프트웨어 설치 작업 수행

Windows Installer 를 사용할 수 있도록 올바로 설계된 응용 프로그램은 WMI **Win32_Product** 클래스를 통해 액세스할 수 있지만 현재 사용 중인 일부 응용 프로그램에서는 Windows Installer 를 사용하지 않습니다. Windows Installer 가 설치 가능한 응용 프로그램과 관련된 작업에 대해 광범위한 표준 방법을 제공하므로 이 설명서에서는 이러한 응용 프로그램을 중점적으로 설명합니다. 일반적으로 대체 설치 루틴을 사용하는 응용 프로그램은 Windows Installer 가 관리하지 않습니다. 응용 프로그램의 작업 수행에 필요한 특정 방법은 설치 소프트웨어 및 응용 프로그램 개발자의 결정에 따라 달라집니다.

참고:

일반적으로 응용 프로그램 파일을 컴퓨터에 복사하여 설치한 응용 프로그램은 여기서 설명하는 방법으로 관리할 수 없습니다. 이러한 응용 프로그램은 "파일 및 폴더 작업 수행" 절에서 설명하는 방법을 사용하여 파일과 폴더로 관리할 수 있습니다.

Windows Installer 응용 프로그램 표시

Windows installer 를 사용하여 로컬 또는 원격 시스템에 설치한 응용 프로그램은 다음과 같이 간단한 WMI 쿼리로 손쉽게 열거할 수 있습니다.

```
PS> Get-WmiObject -Class Win32_Product -ComputerName .
IdentifyingNumber : {7131646D-CD3C-40F4-97B9-CD9E4E6262EF}
Name              : Microsoft .NET Framework 2.0
Vendor            : Microsoft Corporation
Version           : 2.0.50727
```

```
Caption              : Microsoft .NET Framework 2.0
```

기본적으로 표시되지 않는 다른 정보를 보려는 경우 **Select-Object** 를 계속 사용할 수도 있습니다. Microsoft .NET Framework 2.0 의 패키지 캐시 위치를 찾으려면 다음 명령을 사용하면 됩니다.

```
PS> Get-WmiObject -Class Win32_Product -ComputerName . | Where-Object -
FilterScript {$_.Name -eq "Microsoft .NET Framework 2.0"} | Select-Object -
Property [a-z]*
Name               : Microsoft .NET Framework 2.0
Version            : 2.0.50727
InstallState       : 5
Caption            : Microsoft .NET Framework 2.0
Description        : Microsoft .NET Framework 2.0
IdentifyingNumber  : {7131646D-CD3C-40F4-97B9-CD9E4E6262EF}
InstallDate        : 20060506
InstallDate2       : 20060506000000.000000-000
InstallLocation    :
PackageCache       : C:\WINDOWS\Installer\619ab2.msi
SKUNumber          :
Vendor             : Microsoft Corporation
```

또는 **Get-WmiObject Filter** 매개 변수를 사용하여 Microsoft .NET Framework 2.0 만 선택할 수 있습니다. 이 명령에 사용된 필터는 WMI 필터이므로 Windows PowerShell 필터링 구문을 사용할 수 없습니다. 이 필터는 다음과 같은 WQL(WMI 쿼리 언어)을 사용합니다.

```
Get-WmiObject -Class Win32_Product -ComputerName . -Filter "Name='Microsoft .NET
Framework 2.0'"| Select-Object -Property [a-z]*
```

참고:

WQL 쿼리는 Windows PowerShell 에서 특별한 의미를 갖는 공백 또는 등호와 같은 문자를 자주 사용하므로 Filter 매개 변수 값을 항상 따옴표로 묶어야 합니다. 또한 Windows PowerShell 이스케이프 문자, 즉 역음(`)을 사용할 수도 있습니다. 역음을 사용한다고 해서 쿼리가 항상 읽기 쉽게 되는 것은 아닙니다. 다음 명령은 이전 명령과 동일한 결과를 반환하지만 전체 필터 문자열을 따옴표로 묶는 대신 역음을 사용하여 특수 문자를 이스케이프 처리합니다.

```
Get-WmiObject -Class Win32_Product -ComputerName . -Filter Name`=`'Microsoft` .NET` Framework` 2.0`' |
Select-Object -Property [a-z]*
```

목록을 간단하게 만드는 또 다른 방법은 표시 형식을 명시적으로 선택하는 것입니다. 다음 출력 결과는 설치된 특정 패키지를 식별할 때 가장 유용한 속성의 일부를 보여 줍니다.

```
Get-WmiObject -Class Win32_Product -ComputerName . | Format-List
Name,InstallDate,InstallLocation,PackageCache,Vendor,Version,IdentifyingNumber
...
Name               : HighMAT Extension to Microsoft Windows XP CD Writing Wizard
InstallDate        : 20051022
InstallLocation    : C:\Program Files\HighMAT CD Writing Wizard\
```

```
PackageCache      : C:\WINDOWS\Installer\113b54.msi
Vendor            : Microsoft Corporation
Version           : 1.1.1905.1
IdentifyingNumber : {FCE65C4E-B0E8-4FBD-AD16-EDCBE6CD591F}
...
```

마지막으로, 설치된 응용 프로그램의 이름만 찾으려면 다음과 같이 간단한 **Format-Wide** 문으로 출력되는 내용을 줄이면 됩니다.

```
Get-WmiObject -Class Win32_Product -ComputerName .  | Format-Wide -Column 1
```

여기서는 Windows Installer 를 사용하여 설치된 응용 프로그램만 다루고 다른 응용 프로그램은 다루지 않습니다. 이것은 대부분의 표준 응용 프로그램이 Windows 에 해당 설치 제거 관리자를 등록하므로 Windows 레지스트리에서 이러한 설치 제거 관리자를 찾아 로컬로 작업할 수 있기 때문입니다.

제거할 수 있는 모든 응용 프로그램 표시

시스템에 설치된 모든 응용 프로그램을 찾을 수 있는 방법은 없지만 프로그램 추가/제거 대화 상자를 사용하면 시스템에 설치된 응용 프로그램을 최대한 많이 찾을 수 있습니다. 시스템에 설치된 응용 프로그램은 프로그램 추가/제거를 사용하여 **HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Uninstall** 레지스트리 키에서 찾거나 사용자가 직접 이 키를 검사하여 찾을 수 있습니다. 다음과 같이 Windows PowerShell 드라이브를 이 레지스트리 위치에 매핑하면 Uninstall 키를 쉽게 볼 수 있습니다.

```
PS> New-PSDrive -Name Uninstall -PSProvider Registry -Root
HKLM:\SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall

Name       Provider      Root                                    CurrentLocation
----       --------      ----                                    ---------------
Uninstall  Registry      HKEY_LOCAL_MACHINE\SOFTWARE\Micr...
```

참고:

HKLM: 드라이브가 **HKEY_LOCAL_MACHINE** 의 루트에 매핑되므로 Uninstall 키의 경로에 이 드라이브가 사용되었습니다. **HKLM:** 대신 **HKLM** 이나 **HKEY_LOCAL_MACHINE** 을 사용하여 레지스트리 경로를 지정할 수 있습니다. 기존 레지스트리 드라이브를 사용하면 탭 완성 기능을 사용하여 키 이름을 채울 수 있으므로 키 이름을 직접 입력할 필요가 없습니다.

이제 응용 프로그램 설치를 쉽고 빠르게 찾을 수 있는 "Uninstall" 드라이브가 있으므로 다음과 같이 Uninstall: Windows PowerShell 드라이브에서 레지스트리 키의 수를 세면 시스템에 설치된 응용 프로그램의 수를 확인할 수 있습니다.

```
PS> (Get-ChildItem -Path Uninstall:).Length
459
```

Get-ChildItem 을 비롯한 다양한 방법을 사용하여 이 응용 프로그램 목록을 자세히 검색할 수 있습니다. **$UninstallableApplications** 변수로 응용 프로그램 목록을 가져오려면 다음을 입력하십시오.

```
$UninstallableApplications = Get-ChildItem -Path Uninstall:
```

참고:

여기서는 이해를 돕기 위해 긴 변수 이름을 사용하지만 실제로는 긴 이름을 사용할 필요가 없습니다. 변수 이름에 탭 완성 기능을 사용할 수도 있지만 빠른 입력을 위해 1–2 자로 된 이름을 사용할 수도 있습니다. 길고 설명이 포함된 이름은 다시 사용할 코드를 개발하는 경우에 유용합니다.

다음 명령을 사용하면 Uninstall 키에서 응용 프로그램의 표시 이름을 찾을 수 있습니다.

```
PS> Get-ChildItem -Path Uninstall: | ForEach-Object -Process
{ $_.GetValue("DisplayName") }
```

이러한 값은 고유한 값이 아닐 수도 있습니다. 다음 예제에서는 설치된 두 항목이 "Windows Media Encoder 9 Series"로 나타납니다.

```
PS> Get-ChildItem -Path Uninstall: | Where-Object -FilterScript
{ $_.GetValue("DisplayName") -eq "Windows Media Encoder 9 Series"}


   Hive: Microsoft.PowerShell.Core\Registry::HKEY_LOCAL_MACHINE\SOFTWARE\Micros
oft\Windows\CurrentVersion\Uninstall

SKC  VC Name                           Property
---  -- ----                           --------
  0   3 Windows Media Encoder 9        {DisplayName, DisplayIcon, UninstallS...
  0  24 {E38C00D0-A68B-4318-A8A6-F7... {AuthorizedCDFPrefix, Comments, Conta...
```

응용 프로그램 설치

Win32_Product 를 사용하여 Windows Installer 패키지를 원격 또는 로컬로 설치할 수 있습니다. WMI 하위 시스템은 Windows PowerShell 경로를 인식하지 못하므로 원격으로 설치하는 경우 .msi 패키지의 경로를 지정하여 일반 UNC(Universal Naming Convention) 네트워크 경로로 설치해야 합니다. 예를 들어 네트워크 공유 \\AppServ\dsp 에 있는 MSI 패키지 NewPackage.msi 를 PC01 원격 컴퓨터에 설치하려면 Windows PowerShell 프롬프트에서 다음 명령을 입력하십시오.

```
(Get-WMIObject -ComputerName PC01 -List | Where-Object -FilterScript {$_.Name -eq
"Win32_Product"}).InvokeMethod("Install","\\AppSrv\dsp\NewPackage.msi")
```

Windows Installer 기술을 사용하지 않는 응용 프로그램에는 자동화된 배포에 사용할 수 있는 응용 프로그램 관련 메서드가 있습니다. 배치 자동화에 관한 메서드가 있는지 확인하려면 응용 프로그램 설명서를 확인하거나 응용 프로그램 공급업체에 지원 시스템이 있는지 문의해야 합니다. 응용 프로그램 공급업체가 설치 자동화를 위해 응용 프로그램을 특별히 설계하지 않았지만 설치 프로그램 소프트웨어 제조업체에서 몇 가지 자동화 방법을 제공할 수도 있습니다.

응용 프로그램 제거

Windows PowerShell 을 사용하여 Windows Installer 패키지를 제거하는 작업은 **InvokeMethod** 를 사용하여 패키지를 설치하는 작업과 거의 동일합니다. 다음 예제에서는 이름 속성을 기반으로 제거할 패키지를 선택하는 방법을 보여 줍니다. 그러나 경우에 따라 **IdentifyingNumber** 로 필터링하는 것이 더 쉬울 수 있습니다.

```
(Get-WmiObject -Class Win32_Product -Filter "Name='ILMerge'" -
ComputerName . ).InvokeMethod("Uninstall",$null)
```

다른 응용 프로그램의 제거는 로컬로 제거하는 경우에도 매우 복잡합니다. **UninstallString** 속성을 추출하면 이러한 응용 프로그램의 명령줄 제거 문자열을 찾을 수 있습니다. 이 방법은 다음과 같이 Windows Installer 응용 프로그램과 Uninstall 키 아래에 나타나는 기존 프로그램에 사용할 수 있습니다.

```
Get-ChildItem -Path Uninstall: | ForEach-Object -Process
{ $_.GetValue("UninstallString") }
```

필요에 따라 다음과 같이 표시 이름으로 출력을 필터링할 수 있습니다.

```
Get-ChildItem -Path Uninstall: | Where-Object -FilterScript
{ $_.GetValue("DisplayName") -like "Win*"} | ForEach-Object -Process
{ $_.GetValue("UninstallString") }
```

그러나 이러한 문자열은 Windows PowerShell 프롬프트에서 직접 사용하려면 약간의 수정 작업을 거쳐야 합니다.

Windows Installer 응용 프로그램 업그레이드

응용 프로그램을 업그레이드하려면 업그레이드할 설치 응용 프로그램의 이름과 응용 프로그램 업그레이드 패키지의 경로를 알아야 합니다. 그러면 Windows PowerShell 에서 하나의 명령줄만으로 업그레이드를 수행할 수 있습니다.

```
(Get-WmiObject -Class Win32_Product -ComputerName . -Filter
"Name='OldAppName'").InvokeMethod("Upgrade","\\AppSrv\dsp\OldAppUpgrade.msi")
```

컴퓨터 상태 변경: 잠금, 로그오프, 종료 및 다시 시작

Windows PowerShell 에서는 여러 가지 방법으로 컴퓨터를 다시 설정할 수 있지만 초기 릴리스에서는 표준 명령줄 도구나 WMI 를 사용해야 합니다. Windows PowerShell 만 사용하여 특정 도구를 호출하는 경우에도 컴퓨터 전원 상태를 단계별로 변경해 보면 외부 도구 작업과 관련된 중요한 정보를 확인할 수 있습니다.

컴퓨터 잠금

사용 가능한 표준 도구로 컴퓨터를 직접 잠그는 유일한 방법은 다음과 같이 **user32.dll** 에서 **LockWorkstation()** 함수를 직접 호출하는 것입니다.

```
rundll32.exe user32.dll,LockWorkStation
```

이 명령은 워크스테이션을 즉시 잠급니다. Windows XP 와 같은 운영 체제에서 빠른 사용자 전환이 활성화되어 있으면 컴퓨터가 현재 사용자의 화면 보호기를 시작하는 대신 사용자 로그온 화면으로 돌아갑니다. 또한 특정 세션을 연결 해제하려는 터미널 서버에서 **tsshutdn.exe** 명령줄 도구를 사용할 수도 있습니다.

현재 세션 로그오프

여러 가지 방법으로 로컬 시스템의 세션에서 로그오프할 수 있습니다. 가장 간단한 방법은 원격 데스크톱/터미널 서비스 명령줄 도구 logoff.exe 를 사용하는 것입니다. 사용 정보를 보려면 Windows PowerShell 또는 명령 셸 프롬프트에서 logoff /?를 입력하십시오. 현재 활성 세션에서 로그오프하려면 인수 없이 **logoff** 를 입력하십시오.

또는 다음과 같이 해당 로그오프 옵션과 함께 shutdown.exe 도구를 사용할 수도 있습니다.

```
shutdown.exe -l
```

세 번째 옵션은 WMI 를 사용하는 것입니다. Win32_OperatingSystem 클래스에는 Win32Shutdown 메서드가 있는데, 다음과 같이 이 메서드를 인수 0 과 함께 호출하면 로그오프가 시작됩니다.

```
(Get-WmiObject -Class Win32_OperatingSystem -
ComputerName .).InvokeMethod("Win32Shutdown",0)
```

컴퓨터 종료 또는 다시 시작

일반적으로 컴퓨터 종료하는 작업과 컴퓨터를 다시 시작하는 작업은 동일한 유형의 작업입니다. 즉, 컴퓨터를 종료하는 도구로 컴퓨터를 다시 시작할 수도 있고 컴퓨터를 다시 시작하는 도구로 컴퓨터를 종료할 수도 있습니다. Windows PowerShell 에서 컴퓨터를 다시 시작하려면 해당 인수와 함께 tsshutdn.exe 또는 shutdown.exe 를 사용하면 됩니다. **tsshutdn.exe /?** 또는 **shutdown.exe /?**를 사용하면 자세한 사용 정보를 볼 수 있습니다.

또한 Windows PowerShell 에서 **Win32_OperatingSystem** 을 직접 사용하여 컴퓨터를 종료하거나 다시 시작할 수도 있습니다. 그러나 이 설명서에서는 이러한 방법에 대한 내용을 다루지 않습니다.

프린터 작업 수행

Windows PowerShell 에서는 WMI 및 WSH 의 **WScript.Network** COM 개체를 사용하여 프린터 관리 작업을 수행할 수 있습니다. 이 설명서에서는 두 도구를 모두 사용하여 프린터를 관리하는 방법을 보여 줍니다.

프린터 연결 표시

컴퓨터에 설치된 프린터를 표시하는 가장 간단한 방법은 다음과 같이 WMI **Win32_Printer** 클래스를 사용하는 것입니다.

```
Get-WmiObject -Class Win32_Printer -ComputerName .
```

WSH 스크립트에 사용된 **WScript.Network** COM 개체를 사용하여 프린터를 표시할 수 있습니다.

```
(New-Object -ComObject WScript.Network).EnumPrinterConnections()
```

이 명령을 사용하면 고유 레이블 없이 포트 이름 및 프린터 장치 이름의 간단한 문자열 컬렉션만 반환되기 때문에 프린터를 쉽게 찾을 수 없습니다.

네트워크 프린터 추가

다음과 같이 **WScript.Network** 로 새 네트워크 프린터를 쉽게 추가할 수 있습니다.

```
(New-Object -ComObject
WScript.Network).AddWindowsPrinterConnection("\\Printserver01\Xerox5")
```

기본 프린터 설정

WMI 로 기본 프린터를 설정하려면 다음과 같이 원하는 프린터에 **Win32_Printer** 컬렉션을 필터링한 다음 **SetDefaultPrinter** 메서드를 호출해야 합니다.

```
(Get-WmiObject -ComputerName . -Class Win32_Printer -Filter "Name='HP LaserJet
5Si'").InvokeMethod("SetDefaultPrinter",$null)
```

WScript.Network 는 사용하기 간단하고 **SetDefaultPrinter** 메서드가 있어 다음과 같이 인수로 프린터 이름만 지정하면 됩니다.

```
(New-Object -ComObject WScript.Network).SetDefaultPrinter('HP LaserJet 5Si')
```

프린터 연결 제거

다음과 같이 **WScript.Network RemovePrinterConnection** 메서드로 프린터 연결을 제거할 수 있습니다.

```
(New-Object -ComObject
WScript.Network).RemovePrinterConnection("\\Printserver01\Xerox5")
```

네트워크 작업 수행

TCP/IP 는 가장 일반적으로 사용되는 프로토콜이므로 대부분의 간단한 네트워크 프로토콜 관리 작업은 TCP/IP 와 관련이 있습니다. 이 설명서에서는 Windows PowerShell 에서 WMI 를 사용하여 이러한 작업을 수행하는 방법을 설명합니다.

컴퓨터의 IP 주소 표시

다음 명령으로 컴퓨터에서 사용 중인 모든 IP 주소를 반환할 수 있습니다.

```
Get-WmiObject -Class Win32_NetworkAdapterConfiguration -Filter IPEnabled=TRUE -
ComputerName . | Select-Object -Property IPAddress
```

다음과 같이 이 명령의 출력 결과는 값이 대괄호로 묶여 있기 때문에 일반적인 속성 목록과 다릅니다.

```
IPAddress
---------
{192.168.1.80}
{192.168.148.1}
{192.168.171.1}
{0.0.0.0}
```

다음과 같이 **Get-Member** 를 사용하여 **IPAddress** 속성을 자세히 살펴보면 대괄호로 묶은 이유를 알 수 있습니다.

```
PS> Get-WmiObject -Class Win32_NetworkAdapterConfiguration -Filter IPEnabled=TRUE -ComputerName . | Get-Member -
Name IPAddress



   TypeName: System.Management.ManagementObject#root\cimv2\Win32_NetworkAdapter
Configuration


Name      MemberType Definition
----      ---------- ----------
IPAddress Property   System.String[] IPAddress {get;}
```

각 네트워크 어댑터의 IP 주소 속성은 실제 배열입니다. 정의에 있는 괄호는 **IPAddress** 가 **System.String** 값이 아니라 **System.String** 값의 배열임을 나타냅니다.

다음과 같이 **Select-Object ExpandProperty** 매개 변수를 사용하여 이 값을 확장할 수 있습니다.

```
PS> Get-WmiObject -Class Win32_NetworkAdapterConfiguration -Filter IPEnabled=TRUE
-ComputerName . | Select-Object -ExpandProperty IPAddress
192.168.1.80
192.168.148.1
192.168.171.1
0.0.0.0
```

IP 구성 데이터 표시

다음 명령을 사용하여 각 네트워크 어댑터의 자세한 IP 구성 데이터를 표시할 수 있습니다.

```
Get-WmiObject -Class Win32_NetworkAdapterConfiguration -Filter IPEnabled=TRUE -
ComputerName .
```

참고:

네트워크 어댑터 구성에는 매우 간단한 정보만 표시됩니다. 자세한 검사 및 문제 해결 정보를 보려면 **Select-Object** 를 사용하여 더 많은 속성을 표시하십시오. 또한 최신 TCP/IP 네트워크를 사용하는 경우와 같이 IPX 또는 WINS 속성이 필요 없는 경우 다음과 같이 WINS 또는 IPX 로 시작하는 모든 속성을 제외할 수 있습니다.

```
Get-WmiObject -Class Win32_NetworkAdapterConfiguration -Filter IPEnabled=TRUE -ComputerName . | Select-Object -Property [a-z]* -ExcludeProperty IPX*,WINS*
```

이 명령은 DHCP, DNS, 라우팅 및 기타 보조 IP 구성 속성에 대한 세부 정보를 반환합니다.

컴퓨터에 대해 ping 수행

Win32_PingStatus 를 사용하여 컴퓨터에 대해 간단한 ping 을 수행할 수 있습니다. 다음 명령은 ping 을 수행하지만 긴 출력 결과를 반환합니다.

```
Get-WmiObject -Class Win32_PingStatus -Filter "Address='127.0.0.1'" -
ComputerName .
```

이 출력을 보기 쉽게 요약하려면 다음과 같이 Address, ResponseTime 및 StatusCode 속성만 표시하며 됩니다.

```
PS> Get-WmiObject -Class Win32_PingStatus -Filter "Address='127.0.0.1'" -
ComputerName . | Select-Object -Property Address,ResponseTime,StatusCode

Address                                  ResponseTime               StatusCode
-------                                  ------------               ----------
127.0.0.1                                           0                          0
```

상태 코드 0 은 ping 을 성공적으로 수행했음을 나타냅니다.

배열을 사용하여 전체 계열의 컴퓨터에 대해 ping 을 수행할 수 있습니다. 여러 주소를 사용하므로 다음과 같이 **ForEach-Object** 를 사용하여 각 주소에 대해 개별적으로 ping 을 수행해야 합니다.

```
"127.0.0.1","localhost","research.microsoft.com" | ForEach-Object -Process {Get-
WmiObject -Class Win32_PingStatus -Filter ("Address='" + $_ + "'") -
ComputerName .} | Select-Object -Property Address,ResponseTime,StatusCode
```

위와 동일한 방법으로 전체 서브넷에 대해 ping 을 수행할 수도 있습니다. 예를 들어 네트워크 번호 192.168.1.0 을 사용하는 전용 네트워크를 확인하여 표준 클래스 C 서브넷 마스크(255.255.255.0)를 사용하는 경우, 192.168.1.1 에서 192.168.1.254 범위에 있는 주소만 유효한 로컬 주소입니다. 0 은 항상 네트워크 번호로 예약되어 있고 255 는 서브넷 브로드캐스트 주소입니다.

1..254 문을 사용하여 Windows PowerShell 에서 1 에서 254 까지의 번호 배열을 가져올 수 있으므로 배열을 생성하고 ping 문의 부분 주소에 값을 추가하여 서브넷 ping 을 완료할 수 있습니다.

```
1..254| ForEach-Object -Process {Get-WmiObject -Class Win32_PingStatus -Filter
("Address='192.168.1." + $_ + "'") -ComputerName .} | Select-Object -Property
Address,ResponseTime,StatusCode
```

참고:

또한 주소 범위를 생성하는 방법을 다른 작업에 적용할 수도 있습니다. 예를 들어 다음과 같이 전체 주소 집합을 생성할 수 있습니다.

```
$ips = 1..254 | ForEach-Object -Process {"192.168.1." + $_}
```

네트워크 어댑터 속성 검색

이 설명서 앞부분에서는 **Win32_NetworkAdapterConfiguration** 을 사용하여 일반적인 구성 속성을 검색할 수 있다고 설명했습니다. TCP/IP 정보에는 그대로 적용되지 않을 수 있지만 MAC 주소와 어댑터 유형 같은 네트워크 어댑터 정보는 컴퓨터의 상태를 파악하는 데 유용할 수 있습니다. 이 정보를 간단히 보려면 다음 명령을 입력하십시오.

```
Get-WmiObject -Class Win32_NetworkAdapter -ComputerName .
```

네트워크 어댑터의 **DNS** 도메인 할당

Win32_NetworkAdapterConfiguration SetDNSDomain 메서드를 사용하면 자동 이름 확인에 사용할 DNS 도메인을 자동으로 할당할 수 있습니다. 각 네트워크 어댑터 구성에 DNS 도메인을 독립적으로 할당하므로 다음과 같이 **ForEach-Object** 문을 사용하여 어댑터를 개별적으로 선택해야 합니다.

```
Get-WmiObject -Class Win32_NetworkAdapterConfiguration -Filter IPEnabled=true -
ComputerName . | ForEach-Object -Process { $_.InvokeMethod("SetDNSDomain",
"fabrikam.com")}
```

TCP/IP 만 사용하는 네트워크에 있는 컴퓨터의 몇 가지 네트워크 어댑터 구성이 실제 TCP/IP 어댑터가 아니므로 여기서는 필터링 문인 IPEnabled=true 를 사용합니다. 즉 이 구성은 모든 어댑터에 대한 RAS, PPTP, QoS 및 기타 서비스를 지원하는 일반 소프트웨어 요소이기 때문에 자체 주소를 가지고 있지 않습니다.

다음과 같이 **Get-WmiObject Filter** 대신 **Where-Object** 를 사용하여 명령을 필터링할 수 있습니다.

```
Get-WmiObject -Class Win32_NetworkAdapterConfiguration -ComputerName . | Where-
Object -FilterScript {$_.IPEnabled} | ForEach-Object -Process
{$_.InvokeMethod("SetDNSDomain", "fabrikam.com")}
```

DHCP 구성 작업 수행

DHCP 세부 정보는 DNS 구성과 같이 어댑터 집합을 사용하여 수정합니다. WMI 에서는 여러 가지 고유한 작업을 수행할 수 있는데, 이 설명서에서는 몇 가지 일반적인 작업을 단계별로 수행하는 방법을 보여 줍니다.

DHCP 사용 가능 어댑터 확인

다음 명령을 사용하여 컴퓨터에서 DHCP 사용 가능 어댑터를 찾을 수 있습니다.

```
Get-WmiObject -Class Win32_NetworkAdapterConfiguration -Filter "DHCPEnabled=true"
-ComputerName .
```

IP 구성 문제가 있는 어댑터를 찾지 않는 경우 다음과 같이 IPEnabled 어댑터만 찾도록 제한할 수 있습니다.

```
Get-WmiObject -Class Win32_NetworkAdapterConfiguration -Filter "IPEnabled=true and
DHCPEnabled=true" -ComputerName .
```

DHCP 속성 검색

일반적으로 어댑터의 DHCP 관련 속성은 DHCP 로 시작하므로 다음과 같이 **Select-Object -Property DHCP*** 파이프라인 요소를 추가하여 어댑터의 DHCP 요약 정보를 볼 수 있습니다.

```
Get-WmiObject -Class Win32_NetworkAdapterConfiguration -Filter "DHCPEnabled=true"
-ComputerName . | Select-Object -Property DHCP*
```

각 어댑터에 DHCP 설정

모든 어댑터에서 한꺼번에 DHCP 를 사용하도록 설정하려면 다음 명령을 사용하십시오.

```
Get-WmiObject -Class Win32_NetworkAdapterConfiguration -Filter IPEnabled=true -
ComputerName . | ForEach-Object -Process {$_.InvokeMethod("EnableDHCP", $null)}
```

또는 **Filter** 문 "IPEnabled=true 및 DHCPEnabled=false"를 사용할 수 있으므로 DHCP 가 이미 설정되어 있으면 다시 설정하지 않아도 오류가 발생하지 않습니다.

특정 어댑터의 DHCP 임대 해제 및 갱신

Win32_NetworkAdapterConfiguration 에는 **ReleaseDHCPLease** 및 **RenewDHCPLease** 메서드가 있는데, 두 메서드의 사용 방법은 동일합니다. 일반적으로 이러한 메서드는 특정 서브넷에 있는 어댑터 주소를 임대 해제 또는 갱신만 하면 되는 경우에 사용합니다. 서브넷에서 어댑터를 필터링하는 가장 쉬운 방법은 이 서브넷의 게이트웨이를 사용하는 어댑터 구성만 선택하는 것입니다. 예를 들어 다음 명령은 192.168.1.254 에서 DHCP 를 임대하는 로컬 컴퓨터에 있는 어댑터의 모든 DHCP 를 임대 해제합니다.

```
Get-WmiObject -Class Win32_NetworkAdapterConfiguration -Filter "IPEnabled=true and
DHCPEnabled=true" -ComputerName . | Where-Object -FilterScript {$_.DHCPServer -
contains "192.168.1.254"} | ForEach-Object -Process
{$_.InvokeMethod("ReleaseDHCPLease",$null)}
```

DHCP 임대를 갱신하려면 다음과 같이 **ReleaseDHCPLease** 대신 **RenewDHCPLease** 를 호출하기만 하면 됩니다.

```
Get-WmiObject -Class Win32_NetworkAdapterConfiguration -Filter "IPEnabled=true and
DHCPEnabled=true" -ComputerName . | Where-Object -FilterScript {$_.DHCPServer -
contains "192.168.1.254"} | ForEach-Object -Process
{$_.InvokeMethod("ReleaseDHCPLease",$null)}
```

참고:

원격 컴퓨터에 대해 이 메서드를 사용하면 임대 해제 또는 갱신된 어댑터를 통해 원격 시스템에 연결하는 경우 이 시스템에 액세스할 수 없습니다.

모든 어댑터의 **DHCP** 임대 해제 및 갱신

Win32_NetworkAdapterConfiguration 메서드, **ReleaseDHCPLeaseAll** 및 **RenewDHCPLeaseAll** 을 사용하여 모든 어댑터에서 DHCP 주소를 한꺼번에 임대 해제 또는 갱신할 수 있습니다. 그러나 특정 어댑터 대신 WMI 클래스에서 한꺼번에 임대 해제하고 갱신하므로 이 명령을 특정 어댑터가 아니라 WMI 클래스에 적용해야 합니다.

WMI 클래스를 모두 표시하고 원하는 클래스만 이름으로 선택하여 클래스 인스턴스 대신 WMI 클래스에 대한 참조를 사용할 수 있습니다. 예를 들어 다음 명령은 Win32_NetworkAdapterConfiguration 클래스를 반환합니다.

```
Get-WmiObject -List | Where-Object -FilterScript {$_.Name -eq
"Win32_NetworkAdapterConfiguration"}
```

전체 명령을 클래스로 처리한 다음 이 클래스에서 **ReleaseDHCPAdapterLease** 메서드를 호출할 수 있습니다. 다음 명령에서는 **Get-WmiObject** 및 **Where-Object** 파이프라인 요소가 괄호로 묶여 있기 때문에 먼저 평가됩니다.

```
( Get-WmiObject -List | Where-Object -FilterScript {$_.Name -eq
"Win32_NetworkAdapterConfiguration"} ) .InvokeMethod("ReleaseDHCPLeaseAll", $null)
```

다음과 같이 동일한 명령 형식을 사용하여 **RenewDHCPLeaseAll** 메서드를 호출할 수 있습니다.

```
( Get-WmiObject -List | Where-Object -FilterScript {$_.Name -eq
"Win32_NetworkAdapterConfiguration"} ) .InvokeMethod("RenewDHCPLeaseAll", $null)
```

네트워크 공유 만들기

다음과 같이 **Win32_Share Create** 메서드를 사용하여 네트워크 공유를 만들 수 있습니다.

```
(Get-WmiObject -List -ComputerName . | Where-Object -FilterScript {$_.Name -eq
"Win32_Share"}).InvokeMethod("Create",("C:\temp","TempShare",0,25,"test share of
the temp folder"))
```

또한 Windows PowerShell 의 **net share** 를 사용하여 공유를 만들 수도 있습니다.

```
net share tempshare=c:\temp /users:25 /remark:"test share of the temp folder"
```

네트워크 공유 제거

Win32_Share 와 함께 네트워크 공유를 제거할 수 있지만 **Win32_Share** 클래스 대신 제거할 특정 공유를 검색해야 하므로 제거 프로세스는 공유 만들기와 약간 다릅니다. 다음 문은 "TempShare" 공유를 제거합니다.

```
 (Get-WmiObject -Class Win32_Share -ComputerName . -Filter
"Name='TempShare'").InvokeMethod("Delete",$null)
```

또한 다음과 같이 **Net share** 를 사용할 수도 있습니다.

```
PS> net share tempshare /delete
Tempshare 가 삭제되었습니다.
```

액세스 가능한 **Windows** 네트워크 드라이브 연결

New-PSDrive 를 사용하면 Windows PowerShell 드라이브를 만들 수 있지만 이런 방식으로 만든 드라이브는 Windows PowerShell 에서만 사용할 수 있습니다. 새 네트워크 드라이브를 만들려면 **WScript.Network** COM 개체를 사용하면 됩니다. 다음 명령은 공유 ₩₩FPS01\users 를 로컬 드라이브 B:에 매핑합니다.

```
(New-Object -ComObject WScript.Network).MapNetworkDrive("B:", "\\FPS01\users")
```

또한 다음과 같이 **net use** 명령을 사용할 수도 있습니다.

```
net use B: \\FPS01\users
```

WScript.Network 나 net use 를 사용하여 매핑된 드라이브는 Windows PowerShell 에서 즉시 사용할 수 있습니다.

파일 및 폴더 작업 수행

Windows PowerShell 드라이브를 탐색하고 드라이브 항목을 조작하는 것은 Windows 의 실제 디스크 드라이브에 있는 파일 및 폴더를 조작하는 것과 유사합니다. 이 절에서는 특정 파일 및 폴더 조작 작업을 처리하는 방법을 설명합니다.

폴더 내의 모든 파일 및 폴더 표시

Get-ChildItem 을 사용하여 폴더 바로 아래에 있는 항목을 모두 볼 수 있습니다. 선택적 **Force** 매개 변수를 추가하면 숨겨진 항목이나 시스템 항목을 볼 수도 있습니다. 예를 들어 다음 명령은 Windows 의 실제 C 드라이브와 마찬가지로 Windows PowerShell C 드라이브 바로 아래에 있는 내용을 보여 줍니다.

```
Get-ChildItem -Force C:\
```

이 명령은 Cmd.exe 의 **DIR** 명령이나 UNIX 셸의 **ls** 를 사용하는 것과 매우 유사한 방법으로 바로 아래에 포함된 항목만 보여 줍니다. 포함된 항목을 모두 보려면 **-Recurse** 매개 변수를 지정해야 합니다. 다음과 같이 C: 드라이브에 있는 모든 항목을 표시하려면 작업을 완료하는 데 시간이 많이 걸릴 수 있습니다.

```
Get-ChildItem -Force C:\ -Recurse
```

Get-ChildItem 은 **Path**, **Filter**, **Include** 및 **Exclude** 매개 변수로 항목을 필터링할 수 있지만 이러한 변수는 일반적으로 이름을 기반으로 합니다. **Where-Object** 를 사용하여 항목의 다른 속성을 기반으로 복잡한 필터링을 수행할 수 있습니다.

다음 명령은 Program Files 폴더 내에서 2005 년 10 월 1 일 이후 마지막으로 수정되었고 1MB 보다 작거나 10MB 보다 크지 않은 모든 실행 파일을 찾습니다.

```
Get-ChildItem -Path $env:ProgramFiles -Recurse -Include *.exe | Where-Object -
FilterScript {($_.LastWriteTime -gt "2005-10-01") -and ($_.Length -ge 1m) -and
($_.Length -le 10m)}
```

파일 및 폴더 복사

Copy-Item 을 사용하여 항목을 복사할 수 있습니다. 다음 명령은 C:\boot.ini 를 C:\boot.bak 에 백업합니다.

```
Copy-Item -Path c:\boot.ini -Destination c:\boot.bak
```

대상 파일이 이미 있는 경우 복사가 실패합니다. 기존 파일을 덮어쓰려면 다음과 같이 Force 매개 변수를 사용하십시오.

```
Copy-Item -Path c:\boot.ini -Destination c:\boot.bak -Force
```

이 명령은 대상 파일이 읽기 전용인 경우에도 작동합니다.

폴더 복사도 마찬가지로 이 명령은 다음과 같이 C:\temp\test1 폴더를 c:\temp\DeleteMe 라는 새 폴더로 재귀적으로 복사합니다.

```
Copy-Item C:\temp\test1 -Recurse c:\temp\DeleteMe
```

또한 항목 선택을 복사할 수 있습니다. 다음 명령은 c:\data 의 임의 위치에 포함된 모든 .txt 파일을 c:\temp\text 로 복사합니다.

```
Copy-Item -Filter *.txt -Path c:\data -Recurse -Destination c:\temp\text
```

다른 도구를 사용하여 계속 파일 시스템 복사를 수행할 수 있습니다. Windows PowerShell 에서는 **Scripting.FileSystemObject** 와 같은 XCOPY, ROBOCOPY 및 COM 개체를 모두 사용할 수 있습니다. 예를 들어 다음과 같이 Windows 스크립트 호스트인 **Scripting.FileSystem COM** 클래스를 사용하여 C:\boot.ini 를 C:\boot.bak 에 백업할 수 있습니다.

```
(New-Object -ComObject Scripting.FileSystemObject).CopyFile("c:\boot.ini",
"c:\boot.bak")
```

파일 및 폴더 만들기

새 항목 만들기는 Windows PowerShell 공급자에서 동일한 방식으로 작동합니다. Windows PowerShell 공급자에 한 가지 이상의 항목 유형이 있는 경우 항목 유형을 지정해야 합니다. 예를 들어 FileSystem Windows PowerShell 공급자는 디렉터리와 파일을 구별합니다.

이 명령은 다음과 같이 C:\temp\New Folder 라는 새 폴더를 만듭니다.

```
New-Item -Path 'C:\temp\New Folder' -ItemType "directory"
```

이 명령은 C:\temp\New Folder\file.txt 라는 새 빈 파일을 만듭니다.

```
New-Item -Path 'C:\temp\New Folder\file.txt' -ItemType "file"
```

폴더 내의 모든 파일 및 폴더 제거

Remove-Item 을 사용하면 포함된 항목을 제거할 수 있지만 이 항목에 다른 항목이 들어 있는 경우 제거를 확인하는 메시지가 나타납니다. 예를 들어 다른 항목이 들어 있는 C:\temp\DeleteMe 라는 폴더를 삭제하려는 경우 다음과 같이 삭제하기 전에 확인 메시지가 나타납니다.

```
Remove-Item C:\temp\DeleteMe

Confirm
C:\temp\DeleteMe 의 항목에는 하위 항목이 있으며 -recurse 매개 변수를 지정하지 않았습니다. 계속하면
해당 항목과 모든 하위 항목이 제거됩니다. 계속하시겠습니까?
[Y] 예  [A] 모두 예  [N] 아니요  [L] 모두 아니요  [S] 일시 중단  [?] 도움말
(기본값은 "Y"임):
```

폴더에 들어 있는 각 항목에 대해 이 메시지가 나타나지 않게 하려면 다음과 같이 **Recurse** 매개 변수를 지정하십시오.

```
Remove-Item C:\temp\DeleteMe -Recurse
```

Windows 액세스 드라이브로 로컬 폴더 매핑

또한 **subst** 명령을 사용하여 로컬 폴더를 매핑할 수 있습니다. 다음 명령은 로컬 Program Files 디렉터리에 로컬 드라이브 P:를 만듭니다.

```
subst p: $env:programfiles
```

그러면 네트워크 드라이브와 마찬가지로 **subst** 를 사용하여 Windows PowerShell 에 매핑된 드라이브가 Windows PowerShell 세션에 즉시 나타납니다.

텍스트 파일을 배열로 읽어오기

일반적으로 텍스트 데이터는 개별 데이터 요소로 취급되는 별도의 줄이 포함된 파일에 저장됩니다. **Get-Content** cmdlet 을 사용하여 다음과 같이 한 단계에서 전체 파일을 읽을 수 있습니다.

```
PS> Get-Content -Path C:\boot.ini
[boot loader]
timeout=5
default=multi(0)disk(0)rdisk(0)partition(1)\WINDOWS
[operating systems]
multi(0)disk(0)rdisk(0)partition(1)\WINDOWS="Microsoft Windows XP Professional"
 /noexecute=AlwaysOff /fastdetect
multi(0)disk(0)rdisk(0)partition(1)\WINDOWS=" Microsoft Windows XP Professional
with Data Execution Prevention" /noexecute=optin /fastdetect
```

Get-Content 는 파일에서 읽은 데이터를 한 줄에 하나의 요소가 표시된 배열로 취급합니다. 다음과 같이 반환된 내용의 **Length** 를 확인하면 이를 확인할 수 있습니다.

```
PS> (Get-Content -Path C:\boot.ini).Length
6
```

이 명령은 정보 목록을 Windows PowerShell 로 직접 가져오는 경우 가장 유용합니다. 예를 들어 파일의 각 줄에 하나의 이름을 사용하여 컴퓨터 이름 또는 IP 주소 목록을 C:\temp\domainMembers.txt 파일에 저장할 수 있습니다. 다음과 같이 **Get-Content** 를 사용하면 파일 내용을 검색하고 검색 내용을 **$Computers** 변수에 삽입할 수 있습니다.

```
$Computers = Get-Content -Path C:\temp\DomainMembers.txt
```

그러면 각 요소에 있는 컴퓨터 이름이 배열로 **$Computers** 에 포함됩니다.

레지스트리 키 수행

레지스트리 키는 Windows PowerShell 드라이브에 있는 항목이므로 레지스트리 키에 대한 작업 수행은 파일 및 폴더 작업과 매우 유사합니다. 한 가지 중요한 차이점은 레지스트리 기반 Windows PowerShell 드라이브의 모든 항목은 파일 시스템 드라이브의 폴더와 같이 컨테이너라는 점입니다. 그러나 레지스트리 항목 및 연결된 값은 고유한 항목이 아니라 항목의 속성입니다.

레지스트리 키의 모든 하위 키 표시

Get-ChildItem 을 사용하여 레지스트리 키 바로 아래에 있는 항목을 모두 볼 수 있습니다. 선택적 **Force** 매개 변수를 추가하면 숨겨진 항목이나 시스템 항목을 볼 수도 있습니다. 예를 들어 다음 명령은 HKEY_CURRENT_USER 레지스트리 하이브에 해당하는 Windows PowerShell 드라이브 HKCU: 바로 아래에 있는 항목을 보여 줍니다.

```
PS> Get-ChildItem -Path hkcu:\


   Hive: Microsoft.PowerShell.Core\Registry::HKEY_CURRENT_USER

SKC  VC Name                           Property
---  -- ----                           --------
  2   0 AppEvents                     {}
  7  33 Console                       {ColorTable00, ColorTable01, ColorTab...
 25   1 Control Panel                 {Opened}
  0   5 Environment                   {APR_ICONV_PATH, INCLUDE, LIB, TEMP...}
  1   7 Identities                    {Last Username, Last User ...
  4   0 Keyboard Layout               {}
...
```

이러한 키는 레지스트리 편집기(Regedit.exe)의 HKEY_CURRENT_USER 에 표시되는 최상위 키입니다.

또한 레지스트리 공급자 이름(뒤에 "**::**"이 옴)을 지정하여 레지스트리 경로를 지정할 수도 있습니다. 레지스트리 공급자의 전체 이름은 **Microsoft.PowerShell.Core\Registry** 이지만 **Registry** 로 줄일 수 있습니다**.** 다음 명령은 HKCU: 바로 아래에 있는 내용을 보여 줍니다.

```
Get-ChildItem -Path Registry::HKEY_CURRENT_USER
Get-ChildItem -Path Microsoft.PowerShell.Core\Registry::HKEY_CURRENT_USER
Get-ChildItem -Path Registry::HKCU
```

```
Get-ChildItem -Path Microsoft.PowerShell.Core\Registry::HKCU
Get-ChildItem HKCU:
```

이러한 명령은 Cmd.exe 의 **DIR** 명령이나 UNIX 셸의 **ls** 를 사용하는 것과 매우 유사한 방법으로 바로 아래에 포함된 항목만 보여 줍니다. 포함된 항목을 모두 보려면 **-Recurse** 매개 변수를 지정해야 합니다. HKCU 에 있는 모든 레지스트리 키를 보려면 다음 명령을 사용하십시오. 이 작업은 완료하는 데 시간이 많이 걸릴 수 있습니다.

```
Get-ChildItem -Path hkcu:\ -Recurse
```

Get-ChildItem 은 **Path**, **Filter**, **Include** 및 **Exclude** 매개 변수로 복잡한 필터링 기능을 수행할 수 있지만 이러한 변수는 일반적으로 이름을 기반으로 합니다. **Where-Object** cmdlet 을 사용하여 항목의 다른 속성을 기반으로 복잡한 필터링을 수행할 수 있습니다. 다음 명령은 정확히 한 개의 하위 키와 네 개의 값을 갖는 HKCU:\Software 내에서 모든 키를 찾습니다.

```
Get-ChildItem -Path HKCU:\Software -Recurse | Where-Object -FilterScript
{($_.SubKeyCount -le 1) -and ($_.ValueCount -eq 4) }
```

키 복사

Copy-Item 을 사용하여 복사를 수행합니다. 다음 명령은 "CurrentVersion"라는 새 키를 만들 때 HKLM:\SOFTWARE\Microsoft\Windows\CurrentVersion 및 HKCU:\에 대한 모든 속성을 복사합니다.

```
Copy-Item -Path 'HKLM:\SOFTWARE\Microsoft\Windows\CurrentVersion' -Destination
hkcu:
```

새 키를 레지스트리 편집기나 **Get-ChildItem** 을 사용하여 검사하는 경우 포함된 하위 키가 새 위치에 복사되지 못할 수 있습니다. 컨테이너의 내용을 모두 복사하려면 **Recurse** 매개 변수를 지정해야 합니다. **Copy-Item** 복사 명령을 재귀적으로 실행하려면 다음 명령을 사용하십시오.

```
Copy-Item -Path 'HKLM:\SOFTWARE\Microsoft\Windows\CurrentVersion' -Destination
hkcu: -Recurse
```

파일 시스템 복사를 위해 이전부터 사용하던 다른 도구를 계속 사용할 수 있습니다. 예를 들어 reg.exe, regini.exe 및 regedit.exe 같은 레지스트리 편집 도구와 WScript.Shell 및 WMI 의 StdRegProv 클래스와 같이 레지스트리 편집을 지원하는 COM 개체를 Windows PowerShell 에서 사용할 수 있습니다.

키 만들기

레지스트리에서 새 키를 만드는 것은 파일 시스템에서 새 항목을 만드는 것보다 간단합니다. 모든 레지스트리 키가 컨테이너이므로 항목 유형을 지정할 필요 없이 다음과 같이 명시적 경로만 지정하면 됩니다.

```
New-Item -Path hkcu:\software\_DeleteMe
```

또한 다음과 같이 공급자 기반 경로를 사용하여 키를 지정할 수 있습니다.

```
New-Item -Path Registry::HKCU\_DeleteMe
```

키 삭제

기본적으로 항목 삭제는 모든 공급자에 대해 동일하게 수행됩니다. 다음 명령은 항목을 제거합니다.

```
Remove-Item -Path hkcu:\Software\_DeleteMe
Remove-Item -Path 'hkcu:\key with spaces in the name'
```

특정 키 아래에 있는 모든 키 제거

Remove-Item 을 사용하면 포함된 항목을 제거할 수 있지만 이 항목에 다른 항목이 들어 있는 경우 제거를 확인하는 메시지가 나타납니다. 예를 들어 앞에서 만든 HKCU:\CurrentVersion 하위 키를 삭제하려고 하면 다음과 같은 메시지가 나타납니다.

```
Remove-Item -Path hkcu:\CurrentVersion

Confirm
HKCU:\CurrentVersion\AdminDebug 의 항목에는 하위 항목이 있으며 -recurse 매개 변수를 지정하지
않았습니다. 계속하면 해당 항목과 모든 하위 항목이 제거됩니다. 계속하시겠습니까?
[Y] 예  [A] 모두 예  [N] 아니요  [L] 모두 아니요  [S] 일시 중단  [?] 도움말
(기본값은 "Y"임):
```

확인 메시지 없이 포함된 항목을 삭제하려면 다음과 같이 **-Recurse** 매개 변수를 지정하십시오.

```
Remove-Item -Path HKCU:\CurrentVersion -Recurse
```

또한 HKCU:\CurrentVersion 은 제거하지 않고 그 안에 들어 있는 항목만 모두 제거하려면 다음 명령을 사용하면 됩니다.

```
Remove-Item -Path HKCU:\CurrentVersion\* -Recurse
```

레지스트리 항목 작업 수행

레지스트리 항목은 키의 속성이고 직접 검색할 수 없으므로 레지스트리 항목에 대해 작업할 때는 조금 다른 방법을 사용해야 합니다.

레지스트리 항목 표시

다양한 방법을 사용하여 레지스트리 항목을 검사할 수 있습니다. 가장 간단한 방법은 키와 연결된 속성 이름을 가져오는 것입니다. 예를 들어 **HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion** 라는 레지스트리 키의 항목 이름을 보려면 **Get-Item** 을 사용합니다. 레지스트리 키에는 해당 레지스트리 키에 있는 레지스트리 항목의 목록인 "Property"라는 일반 이름의 속성이 있습니다. 다음 명령은 Property 속성을 선택하고 해당 항목을 확장하여 목록으로 보여 줍니다.

```
PS> Get-Item -Path
Registry::HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion | Select-
Object -ExpandProperty Property
```

```
DevicePath
MediaPathUnexpanded
ProgramFilesDir
CommonFilesDir
ProductId
```

레지스트리 항목을 보다 읽기 쉬운 형식으로 보려면 다음과 같이 **Get-ItemProperty** 를 사용하십시오.

```
PS> Get-ItemProperty -Path
Registry::HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion


PSPath              : Microsoft.PowerShell.Core\Registry::HKEY_LOCAL_MACHINE\SO
                      FTWARE\Microsoft\Windows\CurrentVersion
PSParentPath        : Microsoft.PowerShell.Core\Registry::HKEY_LOCAL_MACHINE\SO
                      FTWARE\Microsoft\Windows
PSChildName         : CurrentVersion
PSDrive             : HKLM
PSProvider          : Microsoft.PowerShell.Core\Registry
DevicePath          : C:\WINDOWS\inf
MediaPathUnexpanded : C:\WINDOWS\Media
ProgramFilesDir     : C:\Program Files
CommonFilesDir      : C:\Program Files\Common Files
ProductId           : 76487-338-1167776-22465
WallPaperDir        : C:\WINDOWS\Web\Wallpaper
MediaPath           : C:\WINDOWS\Media
ProgramFilesPath    : C:\Program Files
PF_AccessoriesName  : Accessories
(default)           :
```

키에 대한 Windows PowerShell 관련 속성은 **PSPath**, **PSParentPath**, **PSChildName** 및 **PSProvider** 와 같이 "PS"로 시작합니다.

"." 표기법을 사용하면 현재 위치를 참조할 수 있습니다. 다음과 같이 **Set-Location** 을 사용하여 **CurrentVersion** 레지스트리 컨테이너로 변경할 수 있습니다.

```
Set-Location -Path
Registry::HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion
```

또는 다음과 같이 **Set-Location** 과 함께 기본 제공 HKLM PSDrive 를 사용할 수 있습니다.

```
Set-Location -Path hklm:\SOFTWARE\Microsoft\Windows\CurrentVersion
```

그러면 다음과 같이 현재 위치에 **"."** 표기법을 사용하여 전체 경로를 지정하지 않고 속성을 표시할 수 있습니다.

```
PS> Get-ItemProperty -Path .
...
DevicePath          : C:\WINDOWS\inf
MediaPathUnexpanded : C:\WINDOWS\Media
ProgramFilesDir     : C:\Program Files
...
```

파일 시스템과 마찬가지로 경로를 확장하여 이 위치에서 **Get-ItemProperty -Path ..\Help** 를 사용하여 **HKLM:\SOFTWARE\Microsoft\Windows\Help** 에 대한 **ItemProperty** 목록을 가져올 수 있습니다.

단일 레지스트리 항목 보기

다양한 방법으로 레지스트리 키에서 특정 항목을 검색할 수 있습니다. 다음 예제에서는 **HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion** 에서 **DevicePath** 값을 검색합니다.

Get-ItemProperty 와 함께 **Path** 매개 변수를 사용하여 키 이름을 지정하고 **Name** 매개 변수를 사용하여 **DevicePath** 항목 이름을 지정하십시오.

```
PS> Get-ItemProperty -Path HKLM:\Software\Microsoft\Windows\CurrentVersion -Name
DevicePath


PSPath       : Microsoft.PowerShell.Core\Registry::HKEY_LOCAL_MACHINE\Software\
               Microsoft\Windows\CurrentVersion
PSParentPath : Microsoft.PowerShell.Core\Registry::HKEY_LOCAL_MACHINE\Software\
               Microsoft\Windows
PSChildName  : CurrentVersion
PSDrive      : HKLM
PSProvider   : Microsoft.PowerShell.Core\Registry
DevicePath   : C:\WINDOWS\inf
```

이 명령은 표준 Windows PowerShell 속성 및 **DevicePath** 속성을 반환합니다.

참고:

Get-ItemProperty 에는 **Filter**, **Include** 및 **Exclude** 매개 변수가 있지만 이러한 매개 변수는 항목 속성인 레지스트리 항목 대신 항목 경로인 레지스트리 키를 참조하므로 속성 이름을 기준으로 필터링하는 데 사용할 수 없습니다.

또 다른 옵션은 Reg.exe 명령줄 도구를 사용하는 것입니다. reg.exe 에 대한 자세한 내용을 보려면 명령 프롬프트에서 **reg.exe /?**를 입력하십시오. DevicePath 항목을 찾으려면 다음 명령과 같이 reg.exe 를 사용하십시오.

```
PS> reg query HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion /v DevicePath

! REG.EXE VERSION 3.0

HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion
    DevicePath  REG_EXPAND_SZ   %SystemRoot%\inf
```

또한 **WshShell COM** 개체를 사용하여 일부 레지스트리 항목을 찾을 수 있지만 이러한 방법으로는 "\") 같은 문자를 포함한 큰 이진 데이터나 레지스트리 항목 이름을 처리할 수 없습니다. 다음과 같이 ₩ 구분 기호를 사용하여 항목 경로에 속성 이름을 추가하십시오.

```
PS> (New-Object -ComObject
WScript.Shell).RegRead("HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\DevicePath")
```

```
%SystemRoot%\inf
```

새 레지스트리 항목 만들기

CurrentVersion 키에 "PowerShellPath"라는 새로운 항목을 추가하려면 키, 항목 이름 및 항목 값에 대한 경로와 함께 **New-ItemProperty** 를 사용하십시오. 이 예제에서는 **$PSHome** 이라는 Windows PowerShell 변수 값을 사용합니다. 이 변수는 Windows PowerShell 의 설치 디렉터리에 경로를 저장합니다.

다음 명령을 사용하여 키에 새 항목을 추가하고 새 항목에 대한 정보를 표시할 수 있습니다.

```
PS> New-ItemProperty -Path HKLM:\SOFTWARE\Microsoft\Windows\CurrentVersion -Name
PowerShellPath -PropertyType String -Value $PSHome


PSPath         : Microsoft.PowerShell.Core\Registry::HKEY_LOCAL_MACHINE\SOFTWAR
                 E\Microsoft\Windows\CurrentVersion
PSParentPath   : Microsoft.PowerShell.Core\Registry::HKEY_LOCAL_MACHINE\SOFTWAR
                 E\Microsoft\Windows
PSChildName    : CurrentVersion
PSDrive        : HKLM
PSProvider     : Microsoft.PowerShell.Core\Registry
PowerShellPath : C:\Program Files\Windows PowerShell\v1.0
```

PropertyType 은 다음 표에서 **Microsoft.Win32.RegistryValueKind** 라는 열거 멤버의 이름이어야 합니다.

PropertyType 값	의미
Binary	이진 데이터
DWORD	유효한 UInt32 숫자
ExpandString	동적으로 확장된 환경 변수를 포함할 수 있는 문자열
MultiString	여러 줄 문자열
String	모든 문자열 값
QWord	8 바이트의 이진 데이터

참고:

다음과 같이 **Path** 매개 변수 값의 배열을 지정하여 레지스트리 항목을 여러 위치에 추가할 수 있습니다.

```
New-ItemProperty -Path HKLM:\SOFTWARE\Microsoft\Windows\CurrentVersion,
HKCU:\SOFTWARE\Microsoft\Windows\CurrentVersion -Name PowerShellPath -PropertyType
String -Value $PSHome
```

또한 **Force** 매개 변수를 **New-ItemProperty** 명령에 추가하여 기존 레지스트리 항목 값을 덮어쓸 수 있습니다.

레지스트리 항목의 이름 바꾸기

다음과 같이 **PowerShellPath** 항목의 이름을 "PSHome"로 바꾸려면 **Rename-ItemProperty 를** 사용하십시오.

```
Rename-ItemProperty -Path HKLM:\SOFTWARE\Microsoft\Windows\CurrentVersion -Name
PowerShellPath -NewName PSHome
```

이름이 바뀐 값을 표시하려면 명령에 **PassThru** 매개 변수를 추가하십시오.

```
Rename-ItemProperty -Path HKLM:\SOFTWARE\Microsoft\Windows\CurrentVersion -Name
PowerShellPath -NewName PSHome -passthru
```

레지스트리 항목 삭제

PSHome 및 PowerShellPath 레지스트리 항목을 둘 다 삭제하려면 다음과 같이 **Remove-ItemProperty** 를 사용하십시오.

```
Remove-ItemProperty -Path HKLM:\SOFTWARE\Microsoft\Windows\CurrentVersion -Name
PSHome
Remove-ItemProperty -Path HKCU:\SOFTWARE\Microsoft\Windows\CurrentVersion -Name
PowerShellPath
```

부록 1 – 호환성 별칭

Windows PowerShell 에서는 다양하게 변형된 별칭을 통해 Unix 및 Cmd 사용자가 친숙한 명령 이름을 사용할 수 있습니다. 다음 표에는 별칭이 나타내는 Windows PowerShell 명령과 표준 Windows PowerShell 별칭(있는 경우)과 함께 가장 일반적인 별칭이 나와 있습니다.

참고:

다음과 같이 Get-Alias 를 사용하여 PowerShell 에서 별칭이 가리키는 PowerShell 명령을 찾을 수 있습니다.

PS> Get-Alias cls

```
CommandType     Name                           Definition
-----------     ----                           ----------
Alias           cls                            Clear-Host
```

CMD 명령	Unix 명령	PS 명령	PS 별칭

CMD 명령	Unix 명령	PS 명령	PS 별칭
dir	**ls**	**Get-ChildItem**	**gci**
cls	**clear**	**Clear-Host** (함수)	해당 없음
del, erase, rmdir	**rm**	**Remove-Item**	**ri**
copy	**cp**	**Copy-Item**	**ci**
move	**mv**	**Move-Item**	**mi**
rename	**mv**	**Rename-Item**	**rni**
type	**cat**	**Get-Content**	**gc**
cd	**cd**	**Set-Location**	**sl**
md	**mkdir**	**New-Item**	**ni**
해당 없음	**pushd**	**Push-Location**	해당 없음
해당 없음	**popd**	**Pop-Location**	해당 없음

부록 **2 -** 사용자 지정 **PowerShell** 바로 가기 만들기

다음 절차에 따라 PowerShell 에 대한 바로 가기 만들기를 단계별로 수행하여 여러 가지 편리한 옵션을 사용자 지정합니다.

1. powershell.exe 를 가리키는 바로 가기를 만듭니다.
2. 바로 가기를 마우스 오른쪽 단추로 클릭하여 속성을 선택합니다.
3. 옵션 탭을 클릭합니다.
4. 마우스 기반 선택 및 복사를 설정하려면 편집 옵션에서 QuickEdit 확인란을 선택합니다. 그러면 PowerShell 콘솔 창에서 마우스 왼쪽 단추로 텍스트를 선택하고 Enter 키를 누르거나 마우스 오른쪽 단추로 클릭하여 클립보드에 텍스트를 복사할 수 있습니다.
5. 편집 옵션에서 삽입 모드 확인란을 선택합니다. 그러면 콘솔 창을 마우스 오른쪽 단추로 클릭하여 클립보드에서 텍스트를 자동으로 붙여 넣을 수 있습니다.
6. 명령 기록의 버퍼 크기 스핀 상자에서 1~999 사이의 숫자를 입력하거나 선택합니다. 그러면 통해 콘솔 버퍼에 유지할 입력 명령 수가 설정됩니다.
7. 명령 기록에서 중복 시 이전 항목 삭제 확인란을 선택하여 콘솔 버퍼에서 반복 명령을 제거합니다.
8. 레이아웃 탭을 클릭합니다.
9. 화면 버퍼 그룹 상자에서 높이 스크롤 상자에 1~9999 사이의 숫자를 입력합니다. 높이는 버퍼링된 출력 결과의 줄 수를 나타냅니다. 이 값은 콘솔 창을 스크롤할 때 볼 수 있는 최대 줄 수입니다. 이 값이 창 크기 상자에 나타난 높이보다 작으면 창 크기 높이는 자동으로 동일한 값으로 줄어듭니다.

10. 창 크기 그룹 상자에서 너비에 1~9999 사이의 숫자를 입력합니다. 이 값은 콘솔 창에 표시된 문자 수를 나타냅니다. 기본 너비는 80 이고 PowerShell 의 출력 형식은 이 너비를 기반으로 설계됩니다.
11. 데스크톱이 열린 상태에서 데스크톱의 특정 지점에 콘솔을 배치하려면 창 위치 그룹 상자에서 Let 시스템 위치 창 확인란의 선택을 취소하고 창 위치에서 왼쪽 및 위쪽 값을 변경합니다.
12. 작업이 끝나면 확인 단추를 클릭합니다.