



# 3장

필수적인 cmdlet 사용하기-1

# 전체 내용

별칭(Alias)  
사용하기

변수(Variable)  
사용하기

필수적인 cmdlet

쉬어가는 코너

# 1 – 별칭(Alias) 사용하기

- Alias를 사용하면 긴 명령어를 사용하지 않아도 된다
  - 자주 사용하는 cmdlet은 Built-in 및 Custom Alias를 만들어 이용한다
  - 긴 문장을 쓸 때 짧게 줄여 쓰는 효과가 있다
    - **Get-ChildItem -Path C:\Lab\\*.txt**  
| **Sort-Object -Property length**  
| **Select-Object -Last 5**
    - **dir C:\Lab\\*.txt | sort -Property length | select -Last 5**
    - **dir C:\Lab\\*.txt | sort -p length | select -l 5**  
\*\* 특히 -Property를 -p, -Last를 L로 줄여서 사용 가능하다
    - **dir C:\Lab\\*.txt | sort length | select -l 5**  
\*\* -Property는 생략 가능함
- **Function** 및 **긴 경로의 실행 파일**도 Alias로 만들어 사용할 수 있다
  - 간단한 하나의 cmdlet이 아닌 복잡한 script를 function으로 만들어, 그 복잡한 것을 다시 alias로 만들어 사용한다는 뜻
  - 긴 경로의 실행 파일도 alias로 만들어 사용하면 편리하다
- **Profile**에 **Alias**를 만들어 두면 PowerShell 콘솔을 실행할 때마다 자동으로 Alias를 사용할 수 있다

# 1 – 별칭(Alias) 사용하기

- 기본적으로 제공된 Alias 보기
  - Get-Alias
  - Get-Alias의 Alias는 **gal**
  - Get-Alias **-Name d\***
  - 자신이 사용하는 Alias의 원본 cmdlet 확인하기  
Get-Alias **-Name cp** (##강추)  
**Get-Alias cp**  
**gal cp**
- 특정한 Cmdlet의 Alias를 확인하기  
Get-Alias **-Definition Get-Process** (##강추)

# 1 – 별칭(Alias) 사용하기

## • 애용하는 Alias

- % -> ForEach-Object
- ? -> Where-Object (Where)
- type -> Get-Content
- cd -> Set-Location
- cls -> Clear-Host
- copy -> Copy-Item
- del -> Remove-Item
- diff -> Compare-Object
- dir -> Get-ChildItem
- fl -> Format-List
- ft -> Format-Table
- gcm -> Get-Command
- gm -> Get-Member
- group -> Group-Object

## • 애용하는 Alias

- gsv -> Get-Service
- ps -> Get-Process
- gwmi -> Get-WmiObject
- h -> Get-History
- icm -> Invoke-Command
- etsn -> Enter-PSSession
- ipmo -> Import-Module
- ise -> powershell\_ise.exe
- kill -> Stop-Process
- sls -> Select-String
- measure -> Measure-Object
- select -> Select-Object
- sleep -> Start-Sleep
- sort -> Sort-Object

# 1 – 별칭(Alias) 사용하기

- 새롭게 Alias 생성하여 사용하기
    - **Set-Alias** **-Name** d **-Value** Get-Date  
(줄여서 **Set-Alias** d Get-Date)
  - Function을 Alias로 만들어
    - **FindDefaultPrinter.ps1**
      - **function** FindDefaultPrinter  
    {Get-WMIObject -query "Select \* From Win32\_Printer Where Default = TRUE" }
    - **..FindDefaultPrinter.ps1**
    - **FindDefaultPrinter**
    - **FindDefaultPrinter | Select-Object -ExpandProperty Name**
- ```
PS C:\> FindDefaultPrinter | Select-Object -ExpandProperty Name
NP19168FB (HP LaserJet 400 M401dn)
PS C:\>
```
- **Set-Alias dp FindDefaultPrinter**
    - **dp**

# 1 – 별칭(Alias) 사용하기

- 긴 경로의 파일을 Alias로 사용
  - **Set-Alias excel**  
"C:\Program Files (x86)\Microsoft Office\root\Office16\EXCEL.EXE"
- Alias를 저장하여 다시 불러와서 사용하기
  - **Export-Alias** c:\Lab\AddedAlias.txt
  - c:\Lab\AddedAlias.txt을 열어서 추가된 것만 빼고 나머지는 삭제
  - **Import-Alias** c:\Lab\AddedAlias.txt (function은 실행 안 됨)
  - Alias는 session별로 실행되므로 다른 PowerShell console를 실행하면 이전에 만든 Alias가 사라진다는 것을 기억해 둔다
    - 이것을 해결하는 것은 Export-Alias한 파일을 Import-Alias를 하면 된다.
    - 또는 **PowerShell console 및 ISE의 Profile 파일에** 넣어 두면 항상 새롭게 생성한 Alias를 사용할 수 있다.

# 1 – 별칭(Alias) 사용하기

- 늘 사용할 Alias를 생성하여 profile에 저장하여 사용
  - **\$profile**
    - 이렇게 실행하면 **로그온 한 사용자의 프로파일**이 저장될 경로가 보인다
  - 다음과 같이 새로운 프로파일 파일(Microsoft.PowerShell\_profile.ps1)을 생성한다
    - **notepad \$profile**
  - Notepad 프로그램에서 아래 내용을 입력한 후 저장한다
    - **Set-Alias** d Get-Date
- function** FindDefaultPrinter  
{Get-WMIObject -query "Select \* From Win32\_Printer Where Default = TRUE"}  
**Set-Alias** dp FindDefaultPrinter
- Set-Alias excel** "C:\Program Files (x86)\Microsoft Office\root\Office16\EXCEL.EXE"
- 새로운 PowerShell 콘솔을 실행한다. 새롭게 생성한 alias를 사용할 수 있는지 테스트한다



## 2 - 변수(Variable) 사용하기

- **Variable은 Object의 임시 저장소**

- 변수를 사용하는 이유는 긴 명령어의 값을 임시로 저장하여 다음에 계속 사용하기 위함이다
- 변수를 확인하려면 **Get-Variable**

- **Variable 생성하기(New-Variable)**

- \$로 시작하고 그 다음에 [변수 이름]을 사용한다

**\$today = Get-Date**

\$today

- 변수 이름은 today이다
- 간단하게 변수를 생성할 때는 \$로 시작한 후 변수 이름을 그 다음에 쓰고 변수에 담아줄 내용을 = 다음에 입력하면 된다
- 변수를 실행할 때는 **\$변수이름**으로 하면 된다.
- **변수 생성하기: New-Variable 사용**  
New-Variable -Name server1 -Value "172.16.0.201"  
\$server1

## 2 - 변수(Variable) 사용하기

- Variable 수정하기(Set-Variable)
  - **Set-Variable** -Name server1 -Value "172.16.0.211"  
\$server1
  - **\$server1**="172.16.0.201"  
\$server1
- Variable 수정하지 못하게 하기(-Option ReadOnly)
  - **New-Variable** -Name server2 -Value "8.8.4.4" **-Option ReadOnly**  
\$server2
  - **\$server2**="8.8.8.8" (수정 못함)
  - **Set-Variable** -Name server2 -Value "8.8.8.8" (수정 못함)
- 수정 못하는 Variable을 다시 수정하기
  - Set-Variable -Name server2 **-Option none -Force**
  - **\$server2**="8.8.8.8" (수정 성공)

## 2 - 변수(Variable) 사용하기

- **Variable 삭제하기(Remove-Variable)**
  - **Remove-Variable** -Name server2 -**Force**  
\$server2 (삭제되어 없음)
- **기억해 두면 유용한 변수: Get-Variable**
  - \$profile
  - \$host
  - \$PSVersionTable
  - \$ConfirmPreference
  - \$WhatIfPreference

## 2 - 변수(Variable) 사용하기

- 특별한 변수

- **\$\_** (##강추)

- { }, -Filter, Where-Object를 사용할 때 앞에 나온 Object를 말한다
    - Get-Service | Where-Object {**\$\_**.name -like "app\*"}

- **\$Error**

- cmdlet을 처리할 때 발생한 error가 어떤 것들이 있었는지 확인하기
    - Test-Connection -Source student1 -Server student1000  
dir c:\jesuswithme -Filter \*.txt -Force  
**\$error**

- **\$Home**

- 사용자의 Home directory
    - cd **\$home**

- **\$PsHome** (##강추)

- Windows PowerShell이 설치된 위치
    - 새로 설치된 Module은 **\$psHOME\modules**에 있다

## 2 - 변수(Variable) 사용하기

- 변수(Variable) 사용 예

- 오늘 날짜를 이용한 것

- Get-Date | **Get-Member**

- **(Get-Date).DayofYear**

- \$today = Get-Date

- \$today

- \$today | **Get-Member**

- \$today.**DayofYear**

- \$today.**ToString()**

- Active Directory의 사용자 계정을 이용한 것

- \$aduser = Get-ADUser -Filter \* **-Properties \***

- \$aduser

- \$aduser | **Get-Member**

- \$aduser | Set-ADUser -Department "Sales"

- \$aduser | ft **name, department**

## 2 - 변수(Variable) 사용하기

### • 변수(Variable) 사용 예

- 동일한 성격의 데이터를 가지고 있는 **Array(配列)** 값(data)에 대한 위치에 대한 정보 불러오기: **[ ] 사용**
  - **\$var = 1,2,3,4,5**  
**\$var**
  - 제일 첫 번째 값: **[0]**
    - **(1,2,3,4,5)[0]**  
**\$var[0]**
  - 두 번째 값: **[1]**
    - **\$var[1]**
  - 변수 값 변경하기
    - **\$var[0]=100**  
**\$var**
  - 제일 마지막 값: **[-1]**
    - **\$var[-1]**  
**(1,2,3,4,5)[-1]**

### 3 – 필수적인 cmdlet

Get-Member

Select-Object

Sort-Object

Group-Object

Where-Object

ForEach-Object

Compare-Object

Tee-Object

Measure-Object

## 3 – 필수적인 cmdlet

- **Get-Member**

- **cmdlet의 Property 및 Method를 확인하는 것**
- 주로 Pipeline을 사용할 때 애용한다
- **Select-Object, Where-Object, Group-Object, Sort-Object, Format-Table, Format-List**를 사용하기 전에 **반드시 사용해 보기를 권장**
- Get-Process의 Property를 알기 위해서 **Get-Process | Get-Member**
- Get-Process | **Select-Object -Property** **name, vm**



## 3 – 필수적인 cmdlet

- **Get-Member의 Property와 Method의 정확한 뜻 알아보기**

- Format-List를 활용한다
  - Get-Process -Name notepad | fl \*
- Get-Member를 이용한다
  - Get-Process -Name notepad | **Get-Member**

**TypeName: System.Diagnostics.Process**에서 System.Diagnostics.Process을  
구글에서 검색하면 MSDN으로 연결되어 각 Property와 Method의 정의 및 설명을 알  
수 있다

## 3 – 필수적인 cmdlet

- **Pipeline 사용시 Get-Member 확인은 필수**

- Pipeline을 사용할 때 오류가 나는 이유 확인해 본다
  - Get-Process -Name notepad | Stop-Process (성공)  
Get-Service -Name bits | Stop-Process (실패)
- 왜 그런가?
  - Help Stop-Process를 하면 **-InputObject**는 **<Process[]>**이다. 즉, 앞에서 Process라는 Object Type을 사용해야지 뒤에서 ByValue를 처리할 수 있다는 것이다.
  - **Get-Process** -Name notepad | **Get-Member**를 하면  
TypeName: System.Diagnostics.**Process**이다
  - **Get-Service** -Name notepad | **Get-Member**를 하면  
TypeName: System.ServiceProcess.**ServiceController**이다.
- 결론은 Stop-Process가 Pipeline을 처리하기 위해서는 앞의 것이 Process인 것만 된다는 것이다.

## 3 – 필수적인 cmdlet

- **Pipeline 사용시 Get-Member 확인은 필수**
  - 이제 **\*-Object, Format-\***의 것을 한 번 해보자
    - `Get-Process | Select-Object -Property name, vm`
    - `Get-Service | Sort-Object -Property name -Descending`
- **Help Select-Object** 및 **Help Sort-Object**를 하면 모두 **[-InputObject <PSObject>]**이다. 즉, 앞의 것이 PowerShell Object이면 된다는 것이다
  - 즉, `Get-Process`의 결과는 Object 값이기 때문에 Pipeline 처리가 가능하다
- **Help Format-Table**도 `Help Select-Object`와 동일하다
- 그러므로 **\*-Object, Format-\***의 모든 cmdlet를 사용할 때는 반드시 앞에 Pipeline을 사용할 것을 권장한다

# 3 – 필수적인 cmdlet

## • Collection vs Table

- **Table:** Column(열)과 Row(행)로 구성
- **Collection(=Array):** Property(속성)와 Object(개체)로 구성
  - Get-Service에서 **Property**란...  
Status, Name, DisplayName
  - Get-Service에서 **Object**란...  
Stopped - ALG - Application Layer Gateway Service
- Property 이름(Status, Name, DisplayName)을 사용하여 원하는 결과를 도출한다

```
PS C:\lab> Get-Service Property, not Column
```

| Status  | Name               | DisplayName                       |
|---------|--------------------|-----------------------------------|
| Running | AdobeARMservice    | Adobe Acrobat Update Service      |
| Stopped | AeLookupSvc        | Application Experience            |
| Running | AESTFilters        | Andrea ST Filters Service         |
| Stopped | ALG                | Application Layer Gateway Service |
| Stopped | AllUserInstallA... | Windows All-User Install Agent    |
| Stopped | AppIDSvc           | Application Identity              |
| Running | Appinfo            | Application Information           |
| Running | Apple Mobile De... | Apple Mobile Device               |

Object, not Row

Collection

## 3 – 필수적인 cmdlet

- **\*-Object들은 항상 Pipeline 다음에 사용한다**

- Noun에 Object가 포함된 모든 명령어 알아보기
  - Get-Command **\*-Object**
- **\*-Object**는 데이터를 마음대로 조작할 수 있다
- **Cmdlet | \*-Object** 형식으로 주로 사용되며, 여기에는 cmdlet의 Property를 주로 사용한다
  - Get-Service
  - Get-Service | Get-Member
  - Get-Service | Select-Object **-Property name**
  - Get-Service | Select name | **Sort-Object** -Property name -Descending
- 그래서 항상 cmdlet | Get-Member를 이용하여 사용 가능한 Property를 알아두어야 한다
- Pipeline을 할 때 다른 것의 input 값으로 사용할 수 있다(Measure-Object는 예외)

## 3 – 필수적인 cmdlet

- **Select-Object**

- 원하는 Property만을 선택하여 결과를 도출한다
- **-First, -Last, -Skip**을 주로 사용한다
- Format-Table과는 달리 결과를 다른 \*-Object의 Input으로 사용 가능
- 결과를 왼쪽으로 맞추기 위해 **Select-Object | Format-Table -AutoSize**

- **사용 예제**

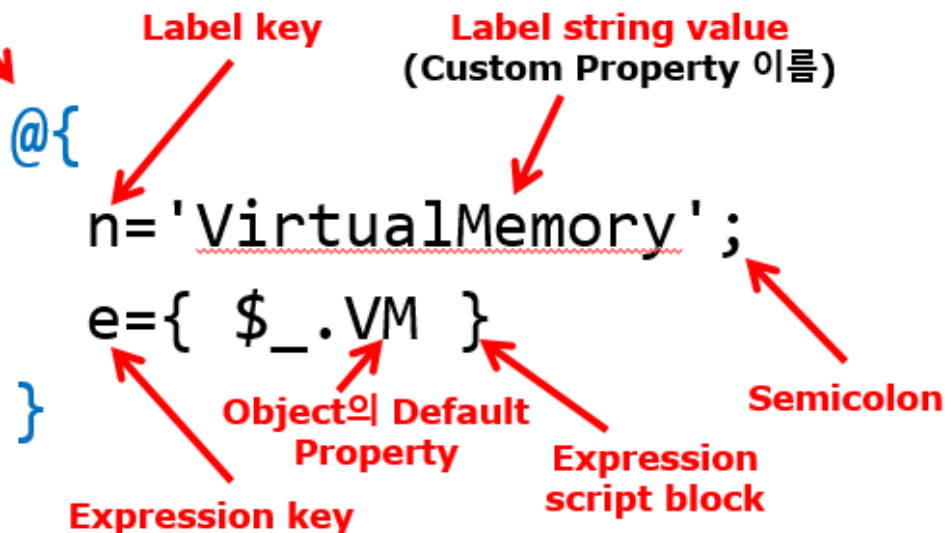
- Get-Service | **Select-Object -Property** Name,Status
- Get-Process | **Sort** VM -Descending | **Select -First 10**
- Gsv -Comp (Get-ADComputer -Filter \* | **Select -ExpandProperty** Name)
  - -ComputerName의 value에는 string을 사용해야 하므로 Select-Object에 -ExpandProperty를 사용했다. 그리고 value를 문장으로 가져 와서 사용 가능
- Get-Process | **Sort** PM -Descending | **Select** Name,ID,PM,VM **-Last 10**
- Get-EventLog **-Newest 10** -LogName Security | **Select-Object -Property** EventID,TimeWritten,Message
- GSV | **Select-Object** name, **RequiredServices, DependentServices**

### 3 – 필수적인 cmdlet

- **Select-Object**에서는 **Custom Property**를 많이 사용

- Property를 다른 이름으로, 또는 가공(계산)하여 사용자가 직접 만들어서 사용하는 것을 말한다 (**Custom Property** 또는 **Calculated Property**)
- 만드는 방법은 ,@{ }로 하며, 이것을 **Hash Table**이라고 한다
- Label Key는 Name 또는 Label을 사용하며, 줄여서 L 또는 N을 사용한다
- **Label Key**의 값이 우리가 사용하고자 하는 **Custom Property**이다
- Expression key에서 `$_Property`를 사용하면 `$_`는 앞의 **Object**를 말한다

Hash table



The diagram illustrates the syntax of a PowerShell Hash Table used to define a custom property. It shows the following components with red arrows pointing to them:

- Label key**: Points to the `@{` symbol.
- Label string value (Custom Property 이름)**: Points to the string `n='VirtualMemory'`.
- Expression key**: Points to the opening curly brace `{` of the expression block.
- Object의 Default Property**: Points to the expression `$_ . VM`.
- Expression script block**: Points to the closing curly brace `}` of the expression block.
- Semicolon**: Points to the semicolon `;` at the end of the hash table definition.

```
@{  
  n='VirtualMemory';  
  e={ $_ . VM }  
}
```

### 3 – 필수적인 cmdlet

- **Select-Object에서 Custom Property 사용하는 예제**
  - **Get-Process | Select-Object -Property Name, ID, PM -First 5**
  - PM을 이해하기 쉽게 나타내고자 한다 (PM 대신 PM(KB), byte를 KB로)  
**Get-Process | Select-Object -Property Name, ID,**  
**@{Name='PM(KB)';Expression={\$\_.PM/1KB}} -First 5**
- **용량의 크기를 나타낼 때 사용하는 단위**
  - **KB, MB, GB, TB, PB**
  - 숫자의 소수점 자릿수를 나타낼 때는 **'{0:N2}' -f** 형식을 사용한다



## 3 – 필수적인 cmdlet

### • 사용 예제

- ps | Select name, @{N="Start Day"; E= {\$\_.**StartTime.DayOfWeek**}}
  - 참고: \$\_.StartTime.**DayOfWeek**를 어떻게 알아냈는가?
    - **Get-Process | Get-Member**에서 StartTime이라는 Property를 알아 냄
    - **(Get-Process).StartTime** | Get-Member에서 DayOfWeek라는 Property를 알아 냄
- Get-Volume | Select **DriveLetter**, @{n='**Size(GB)**'; e='{0:N1}' -f (\$\_.Size / 1GB)}, @{n='**FreeSpace(GB)**'; e='{0:N2}' -f (\$\_.SizeRemaining / 1GB)}
- Get-History | **Select \*, @{n='ExecutionTime';e={\$\_.EndExecutionTime - \$\_.StartExecutionTime}} | Sort-Object ExecutionTime -Descending**
- Get-Process| Select-Object -Property name,id,  
@{Name='VM(MB)';Expression={\$\_.VM/1MB -As [int]}},  
@{Label='PM(MB)' ; Expression={\$\_.PM/1MB -As [int]}}

## 3 – 필수적인 cmdlet

- **Sort-Object**

- 특정한 Parameter를 기준으로 **정렬(오름차순)**하여 결과를 보여준다
- 이것의 결과를 Pipeline 하여 다른 cmdlet의 Input 값으로 사용할 수 있다
- **내림차순**으로 보고자 하면 **-Descending**을 사용한다

- **사용 예제**

- Get-Process | Sort-Object -Property vm
- Ps | Sort vm **-Descending**
- Ps | Sort **name, vm**
  - 먼저 ProcessName으로 정렬한 후, 다시 한 번 더 vm을 기준으로 정렬한 것임
- ps -Computer Server2 | Sort-Object pm -Desc | Select -First 10 | FT name,vm,pm -AutoSize
- Dir c:\lab | **Group-Object -Property extension** | Sort-Object -Property extension | ft -auto
  - 파일 확장자별로 묶어서 수량을 보고자 할 때 유용하다
- Get-EventLog -LogName Security -Newest 10 | Sort **TimeWritten**

## 3 – 필수적인 cmdlet

- **Group-Object**

- 지정한 **Property** 값을 기준으로 묶어서 결과를 도출한다
- 항상 하나로 묶어진 것의 수량을 표시한다
- Group-Object를 한 다음 Pipeline을 하여 Sort-Object를 하는 것이 좋다

- **사용 예제**

- 각 Verb/Noun의 수량을 알기 위해(강추)
  - Get-Command | **Group-Object Verb** | Sort-Object Count -Desc (**##강추**)
  - 이렇게 하면 각 Verb별로 몇 개의 명령어가 있는지 알 수 있다

```
PS C:\> Get-Command | Group Verb | Sort Count -Desc | ft count, name
```

| Count | Name   |
|-------|--------|
| 902   | Get    |
| 498   | Set    |
| 425   | Remove |
| 378   | New    |
| 148   | Add    |

- 가장 많이 발생하는 Event를 확인하고 그 번호가 무엇인지 확인
  - Get-EventLog -LogName System | **Group-Object -Property EventID** | **Sort-Object -Property count -Desc**

# 3 – 필수적인 cmdlet

## • 사용 예제

- 현재 실행 중인 프로세스를 Company별로 묶어서 보기
  - **\*\* -NoElement은 Group 구성원을 표기하지 않도록 하는 것**
  - `Get-Process | Group-Object -Property Company -NoElement | Sort count`
  - `ps | Group-Object Company | Select-Object Count, Name | Sort count -Desc`
- 가장 많이 발생하는 이벤트(EventID 7036)를 확인하고 그것이 어떤 것인지 확인하기
  - `Get-Eventlog -LogName system -Newest 1000 | Group-Object -Property eventId | Sort-Object Count`
  - `Get-Eventlog -LogName system -Newest 1000 | ? {$_.eventId -eq 7036}`
- 어떤 프로세스가 가장 많이 실행되고 있는지 확인하기  
`Get-Process | Group-Object -Property Name -NoElement | Sort-Object Count`

## 3 – 필수적인 cmdlet

- **Where-Object**

- **지정한 Property를 기준으로 연산 작업을 하여 Filtering을 한다**
- 모든 cmdlet에 대하여 Pipeline을 사용하여 처리하는 장점이 있다
- 하지만 **-Filter**에 비하여는 처리 속도가 느리다는 단점이 있다
- 속도를 높이기 위해서는 -Filter를 지원하는 명령어에서는 -Filter를 사용하고, 어떤 경우에도 사용할 수 있게 호환성을 높이기 위해서는 | **Where-Object**를 사용한다
- -Filter를 사용하는 명령어들을 모두 확인하기
  - **Get-Command | Where-Object { \$\_.Parameters.Keys -Contains "Filter"}**
  - 또는 **Get-Command -ParameterName filter**
  - 또는 **Help \* -Parameter Filter**

# 3 – 필수적인 cmdlet

- **Where-Object**

- Where-Object의 Alias

- **Where**
- **?**

- Where-Object의 구문

- **Where-Object -FilterScript** {\$\_.<Property> <비교연산자> <Value>}

- 비교 연산자에 대하여 자세히 알아보기

- **Help About\_\***을 실행하여 Comparison이 들어 간 것을 찾는다
- **Help About\_Comparison\_Operators**
- 또는 **Help \*comparison\***
- 이곳을 통하여 비교 연산자에 대하여 자세히 공부해 둘 필요가 있다

### 3 – 필수적인 cmdlet

- **Where-Object**는 일반적이므로 다양하게 사용할 수 있지만 처리 속도는 느리다

- 다음 두 명령어의 처리 속도를 비교해 본다

**Measure-Command** { dir \$home -Recurse -Filter \*.txt }

**Measure-Command** { dir \$home -Recurse | Where { \$\_.name -like "\*.txt" } }

- 처리 속도는 -Filter가 빠르다. Where-Object는 모든 명령어에 다 사용할 수 있는 반면, -Filter는 지원하는 명령어에만 사용하는 제약 사항이 있다

- 다음 두 명령어의 처리 속도를 비교해 본다

**Measure-Command** { Get-Service -Name win\* }

**Measure-Command** { Get-Service | Where { \$\_.name -like 'win\*' } }

- 처리 속도는 Where-Object가 느리다. 하지만 Where-Object는 다양하게 응용할 수 있는 장점이 있다는데 유의한다
- **Get-Service -Name win\***은 하나의 Parameter에 대한 것만 결과를 볼 수 있지만, Comparison Operator를 사용하면 -And, -Or를 사용하여 여러 개의 Parameter를 동시에 결과를 얻을 수 있는 장점이 있다.  
Get-Service | Where { \$\_.name -like 'win\*' -And \$\_.status -eq 'running' }

## 3 – 필수적인 cmdlet

### • 사용 예제

- `Get-Process | Where-Object {$_ .WorkingSet -gt 500000000}`
- `Get-Process | Where {$_ .ProcessName -eq "calc"}`
- `Get-Process | ? { $_.name -match "ss$" } (##-match는 정규표현식 사용)`
- `Get-Service | Where {$_ .status -eq "Running"}`
- `Get-Process | Where {$_ .pm -gt 50} | Select-Object -Property name, id`

### • -Filter 문법

- 정확한 문법은 각 명령어의 도움말을 참고한다
- `Get-WmiObject -Class Win32_LogicalDisk -Filter "drivetype=3"`
  - \*\* -Filter {\$\_ .drivetype -eq 3} 은 틀림
- `Get-WmiObject -Class Win32_Service -Filter "name='WinRM'"`
- `Get-ADUser -Filter {department -eq "sales"}`
  - \*\* -Filter "department=sales" 틀림



## 3 – 필수적인 cmdlet

### • **ForEach-Object**

- 앞의 결과(object)를 하나씩 하나씩 반복적으로 가져와서 실행한다
- ForEach-Object의 Alias는 %이다
- 사용 구문: | **ForEach-Object** { 명령어 } 및 | **ForEach-Object** { \$\_.속성 }
- Help ForEach-Object

### • **사용 예제**

- 특정한 폴더의 파일 이름 및 프로세스 이름만 확인하고자 한다
  - dir c:\lab -Recurse | **ForEach-Object** { \$\_.name }
  - Get-Process | **ForEach-Object** { \$\_.name }
- 숫자의 사칙 연산을 한다
  - 1..13 | % { \$\_ \* 7 }
  - 30000, 56798, 12432 | **ForEach-Object -Process** { \$\_ / 1024 }
- 원격 컴퓨터를 강제로 GPupdate하기 (컴퓨터/사용자 모두 적용)
  - Get-ADComputer -Filter {Name -like "server\*"} | % { **Invoke-GPupdate** -Computer \$\_.name -Force -RandomDelayInMinutes 0 }

## 3 – 필수적인 cmdlet

### • 사용 예제

- 로그 파일에 Warning이라는 글자가 있는 것만 확인하기 **(IF 문장 사용)**
  - `$log = Get-Content C:\Windows\windowsupdate.log`  
`$log`  
`$log | ForEach-Object { if ($_ -match "WARNING:") { $_ } }`  
`$log | % { if ($_ -match "WARNING:") { $_ } } > c:\Lab\warning.txt` **(##강추)**
- 사용자의 Home folder에 어떤 파일이 있고, 파일 수량을 확인하기
  - `$files1 = Dir $home -Recurse | ForEach-Object { $_.name }` **(##강추)**  
`$files1`  
`$files1.count`
- 사용한 명령어만 뽑아서 파일로 저장하기
  - `Get-History | % {$_ .CommandLine} | Out-File c:\Lab\commandlist.ps1`  
`Get-History | % {$_ .CommandLine} | clip.exe`
- Mycmd.ps1 파일로 작성하여 실행하기
  - **Foreach (\$i in Get-History)**  
**{Add-Content -Path C:\Lab\Mycmd.ps1 -Value \$i.CommandLine}**

## 3 – 필수적인 cmdlet

### • 사용 예제

- 설치된 프로그램 중에서 삭제 가능한 프로그램 목록 확인하기

<레지스트리를 PSDrive로 생성하기>

- **New-PSDrive -Name Uninstall -PSProvider Registry -Root HKEY\_LOCAL\_MACHINE\Software\Microsoft\Windows\CurrentVersion\Uninstall**
- **Get-ChildItem -Path Uninstall:**

- <삭제 가능한 프로그램의 수량 확인>

- **(Get-ChildItem -Path Uninstall:).Count**

- <삭제 가능한 프로그램 확인하기>

- **Get-ChildItem -Path Uninstall: | Get-Member**  
\*\* 여기에 보니 **GetValue**라는 Method가 있다

- 우리가 원하는 정답은 다음과 같다

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

## 3 – 필수적인 cmdlet

### • **ForEach-Object vs Foreach**

- ForEach-Object보다는 Foreach가 훨씬 처리 속도가 빠르다
- ForEach-Object는 cmdlet | ForEach-Object { } 형식으로 사용하여 일일이 Object를 하나씩 가져와서 처리한다
- Foreach는 Array로 한꺼번에 처리한다.
- **Foreach ( \$item in \$Array/Collection ) { \$item.속성 }** 형식이다
- 속도를 비교할 때는 Measure-Command {명령어}를 사용한다
- 다음 내용을 비교해 본다. TotalMilliseconds 항목을 참고한다  
**Measure-Command** { dir 'C:\Program Files' -Recurse | % {\$\_name} }

**Measure-Command** { **Foreach** (\$item in dir 'c:\program files' -Recurse) { \$item.name } }

\*\* Foreach가 ForEach-Object 보다 2배 정도 빠르게 처리한다

## 3 – 필수적인 cmdlet

- **Measure-Object**

- 명령어 개수, 파일 개수, 컴퓨터 계정 개수, 프로세스 개수 등등의 수량(數量)을 알아볼 때 사용한다

- **사용 예제**

- 사용 가능한 명령어 수량 확인하기
  - **Get-Command | Measure-Object**
  - Help Get-Command -Full  
(##많은 Parameter 중에서 -ParameterName이 있다)
  - Get-Command **-ParameterName** *parameter* (중요)  
(cmdlet 중에서 parameter라는 매개변수를 사용하는 명령어가 Get-Help)
    - Get-Command -ParameterName **Filter**
    - Get-Command -Parametername **AsString**
  - **Help** Get-Command **-Parameter** *CommandType* (중요)  
(특정한 cmdlet의 특정한 Parameter의 내용만 확인하기)
  - Get-Command **-CommandType** cmdlet | Measure
  - Get-Command **-CommandType** application | Measure

## 3 – 필수적인 cmdlet

### • 사용 예제

- AD 전체 사용자/컴퓨터(Get-ADComputer) 계정 개수 확인
  - **Get-ADUser -Filter \*** | Measure-Object
- Lab OU에 소속된 사용자 개수
  - Get-AdUser -Filter \* -SearchBase "**OU=Lab, DC=MCTconnected, DC=com**" | Measure-Object
- 특정한 폴더 및 그 하위 디렉터리의 파일 수량 확인하기
  - Dir c:\lab | Measure
  - dir c:\lab **-Recurse** | Measure
- PM을 기준으로 개수를 구한 것. PM의 합, 최대, 최소값, 평균값 보기
  - Get-Process | **Measure-Object**
  - Get-Process | **Measure-Object -Property pm**
  - Get-Process | **Measure-Object -Property pm -sum -min -max -average**

## 3 – 필수적인 cmdlet

- **Compare-Object**

- 2개의 Object를 비교하여 변경된 것을 확인한다
- 구문: **Compare-Object** (기준 값) (변한 값)  
**Compare-Object [-ReferenceObject] <PSObject[]> [-DifferenceObject] <PSObject[]>**
- Compare-Object 대신 Alias로서 **diff**를 사용해도 된다
- **Export-CliXml** 명령으로 현재 상태를 xml 형식으로 저장하고, **Import-CliXml**로 저장된 값을 불러 와서 비교한다

# 3 – 필수적인 cmdlet

## • 사용 예제

- 로그인 했을 때랑 일정 시간 후에 실행되고 있는 프로그램을 비교한다

- Get-Process | **Export-CliXml** c:\lab\baseline.xml

- xml 파일의 내용을 보기

- Import-Clixml** C:\Lab\baseline.xml

- Import-Clixml C:\Lab\baseline.xml | **Out-GridView**

- Notepad, calc, mspaint 프로그램을 실행한다

- **Compare-Object -Property name** (Get-Process) (**Import-CliXml** c:\Lab\baseline.xml) (**##강추**)

- 어떤 파일이 추가로 생성되었는지 확인한다

- Dir c:\Lab | **Export-CliXml** c:\recentFiles.xml

- Fsutil file createnew** c:\Lab\GodisLove.txt 1024000

- Fsutil file createnew** c:\Lab\JesusisLove.txt 1024000

- **Compare-Object** (Import-CLIXML c:\recentFiles.xml) (Get-ChildItem c:\Lab) **-Property name**



## 3 – 필수적인 cmdlet

- 사용 예제

- 복사된 파일 내용의 수정 여부 확인하기

- C:\Lab\files.txt를 생성하여 복사한다(C:\Lab\CopyOfFiles.txt)  
C:\Lab\files.txt의 내용을 업데이트한다

- **Compare-Object -ReferenceObject \$(Get-Content C:\Lab\files.txt) -DifferenceObject \$(Get-Content C:\Lab\CopyOfFiles.txt)**

- = **Compare-Object \$(Get-Content C:\Lab\files.txt) \$(Get-Content C:\Lab\CopyOfFiles.txt)**

- = **Compare-Object (Get-Content C:\Lab\files.txt) (Get-Content C:\Lab\CopyOfFiles.txt)**  
**(##강추)**

# 3 – 필수적인 cmdlet

- 사용 예제

- 두 컴퓨터의 서비스 상태를 비교한다

- \$s1 = Get-Service -ComputerName Dc1  
\$s2 = Get-Service -ComputerName Server1

- **Diff \$s1 \$s2 -Property Name, Status -PassThru | Sort Name | Select MachineName, Name, Status | FT -AutoSize**  
\*\* 여기서는 PSObject를 변수(\$s1, \$s2)로 사용했다

| MachineName | Name           | Status  |
|-------------|----------------|---------|
| dc1         | ADWS           | Running |
| server1     | aspnet_state   | Stopped |
| dc1         | Dfs            | Running |
| dc1         | DFSR           | Running |
| dc1         | DNS            | Running |
| dc1         | DsRoleSvc      | Stopped |
| dc1         | IsmServ        | Running |
| dc1         | Kdc            | Running |
| dc1         | KdsSvc         | Stopped |
| dc1         | NTDS           | Running |
| dc1         | NtFrs          | Stopped |
| dc1         | RemoteRegistry | Stopped |
| server1     | RemoteRegistry | Running |
| dc1         | TrkWks         | Stopped |
| server1     | TrkWks         | Running |
| dc1         | vds            | Running |
| server1     | vds            | Stopped |

## 3 – 필수적인 cmdlet

- **Parameter의 Value를 어떻게 표현하는가?**

- 먼저 각 명령어의 도움말을 참고한다
- Parameter의 value가 **<string>** 이면 1)단순 **string**, 2)(명령어 결과 **string**)
  - Get-Service -ComputerName **Server1, Server2**
  - Get-Service -ComputerName (**Get-Content c:\computers.txt**)
  - Get-Service -ComputerName (**Get-ADComputer -Filter \* | Select-Object -ExpandProperty name**)
- Parameter의 value가 **<PSObject>** 이면 1)(명령어), 2)\$ (명령어 결과 **string 값**), 3)변수
  - Compare-Object (**Get-Process**) (**Import-CliXml c:\baseline.xml**) -Property name
  - Compare-Object -ReferenceObject \$(**Get-Content C:\Lab\files.txt**) -DifferenceObject \$(**Get-Content C:\Lab\CopyOfFiles.txt**)
  - Compare-Object **\$s1 \$s2** -Property Name, Status -**PassThru**

## 3 – 필수적인 cmdlet

- **Tee-Object**

- 동시에 2가지 작업을 한다
- 명령어의 결과를 1)파일로 저장하고, 동시에 2)콘솔 화면에 출력
- 명령어의 결과를 2개의 다른 파일로 저장할 수 있다

- **사용 예제**

- Get-EventLog System -Newest 100 -EntryType Error | **Tee-Object** -FilePath C:\Lab\event.txt | Ft TimeGenerated, EventId, Source -AutoSize  
**Type** C:\Lab\event.txt
- Get-ChildItem "C:\ProgramData\" | **Tee-Object** "C:\Lab\TeeProgDat.txt" | Format-Table Name  
**Get-Content** "C:\Lab\TeeProgDat.txt"
- Get-EventLog System -Newest 25 -EntryType Error | **Tee-Object** -FilePath "c:\Lab\NewOnly.txt" | Out-File "c:\Lab\Continuous.txt" -Append  
**Invoke-Item** "c:\Lab\Continuous.txt"

## 4 – 쉬어가는 코너

하루 동안 변경(생성)된 파일 확인하기

파일의 내용이 변경된 것 확인하기

## 4 – 쉬어가는 코너

- 하루 동안 변경(생성)된 파일 확인하기

- 사용자 프로파일 위치에 저장된 파일 중에서 변경/생성된 것 확인하기
  - `$NewFiles = Get-ChildItem $Home -Recurse -Force | Where-Object {$_.LastWriteTime -gt (Get-Date).date}`

**\$NewFiles**

- 이 목록을 파일로 저장하기
  - `$NewFiles | Export-Csv c:\Lab\newfiles.csv`
- 하루 동안 변경된 파일 자체를 다른 곳(FileServer)으로 몽땅 복사하기
  - `$NewFiles | Copy-Item -Destination \\Server1\Lab\newfiles`
- [특정한 날 이후]로 지금까지 변경된 파일 확인하기 (**##강추**)
  - `Get-ChildItem $Home -Recurse -Force | Where-Object {$_.LastWriteTime -gt "04/30/2014"}`

## 4 – 쉬어가는 코너

- 하루 동안 변경(생성)된 파일 확인하기

- 하루 동안 변경된 파일 목록을 파일로 저장하여 확인하기

- `Dir $home -Recurse -Force | Export-Clixml \\server1\Lab\InitialFiles.xml`

- ## 이 문장을 logon.ps1 파일로 작성하여 Logon Script로 사용하면 좋다

- **Diff** (Dir \$home -Recurse -Force) (**Import-Clixml** \\server1\Lab\InitialFiles.xml) -Property Name | ft -A | Out-File \\server1\Lab\CorDFilesToday.txt

- ## 이것을 logoff.ps1 파일로 작성하여 Shutdown Script로 사용한다

- 이렇게 하면 File Server(Server1)에 CorDFilesToday.txt 파일을 보면 오늘 하루 동안 생성 및 삭제(변경된 것 모두)의 파일/폴더 이름만 알 수 있다.

## 4 – 쉬어가는 코너

### • 파일의 내용이 변경된 것 확인하기

- 파일이 교체되었거나 동일한 파일 이름인데 내용이 수정되었는지를 확인하기 위해서는 **사전에 만들어 놓은 Hash 값을 비교하면 알 수 있다**
- 해쉬 값을 알기 위해서는 **Get-FileHash**를 사용하고, 두 개를 비교하여 차이를 알기 위해서는 **Compare-Object**를 사용하면 된다
- Get-FileHash는 PowerShell 4.0부터 지원하고 파일만 가능하다
- 파일의 해쉬 값을 만들어 csv 파일에 저장한다
  - `Dir c:\Lab -File | Get-FileHash -Algorithm MD5 | Export-Csv c:\Lab\LabfolderHash.csv`
- C:\Lab의 파일을 C:\Lab1으로 모두 복사하여 하나의 파일의 내용을 수정한 후, 각각의 파일에 대하여 Hash 값을 만들어 확인한다
  - `$lab = dir c:\lab -Recurse -File | Get-FileHash -Algorithm md5`  
`$lab1 = dir c:\lab1 -Recurse -File | Get-FileHash -Algorithm md5`
- 어떤 파일이 수정되었는지 Hash 값으로 알아 본다
  - `Compare-Object $lab $lab1 -Property hash -PassThru | ft -AutoSize`
- 파일의 어느 부분이 수정되었는지 확인하기
  - `Diff (Get-Content C:\lab\vslee.txt) (Get-Content C:\lab1\vslee.txt)`



## 4 – 쉬어가는 코너

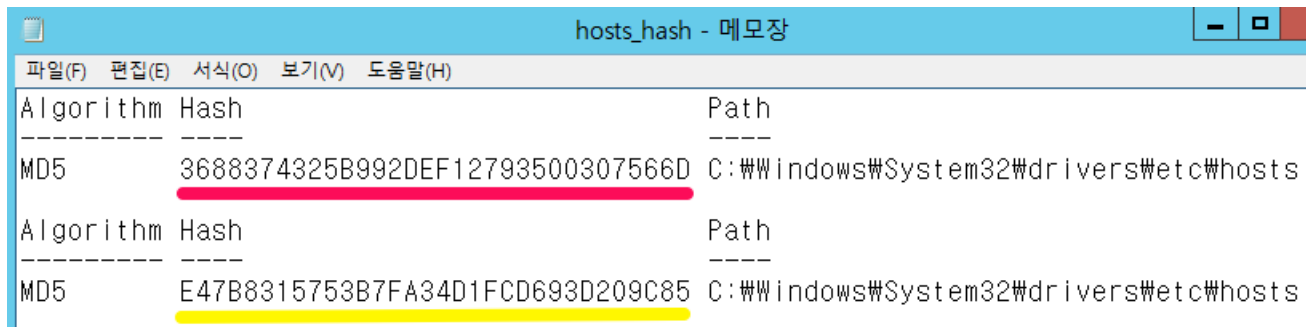
- **파일의 내용이 변경된 것 확인하기**

- 로컬 컴퓨터에 저장된 파일들을 모두 File Server에 복사한 후 이 두 위치에 저장된 파일들이 변경되었는지 확인하기
  - **Copy-Item** -Path **C:\Lab\\*** -Destination **\\Server1\Lab -PassThru**
  - **\$before** = dir C:\Lab -Recurse -Force | Get-FileHash -Algorithm MD5
  - **\$after** = dir \\server1\Lab -Recurse -Force | Get-FileHash -Algorithm MD5
  - Compare-Object \$before \$after **-Property hash -PassThru** | ft -Auto  
\*\* 파일 수량이 차이가 나도 화면에 알려준다
- 내용의 차이가 나는 파일의 내용을 확인한다
  - **Diff (Gc C:\lab\files1.txt) (GC \\server1\lab\files1.txt)**

## 4 – 쉬어가는 코너

### • 파일의 내용이 변경된 것 확인하기

- Hosts 파일이 변경되었는지 확인하려면 평소에 이 파일에 대한 Hash 값을 생성하여 파일로 저장한 후 나중에 다시 Hash 값을 생성하여 비교해 본다
- Server1의 c:\Windows\System32\Drivers\Etc\Hosts 파일에 대하여 Hash 값을 생성하여 저장한다
  - **Get-FileHash** c:\Windows\System32\Drivers\Etc\hosts -Algorithm md5 | ft -AutoSize | Out-File c:\Lab\Hosts\_hash.txt
- DC1의 hosts 파일을 수정하여 Server1의 Hosts 파일과 교체한다
- Server1에서 다시 Hash를 생성한다
  - **Get-FileHash** c:\Windows\System32\Drivers\Etc\hosts -Algorithm md5 | ft -AutoSize | Out-File c:\Lab\Hosts\_hash.txt **-Append**



| hosts_hash - 메모장 |                                  |                                       |
|------------------|----------------------------------|---------------------------------------|
| 파일(F)            | 편집(E)                            | 서식(O) 보기(V) 도움말(H)                    |
| Algorithm        | Hash                             | Path                                  |
| -----            | -----                            | -----                                 |
| MD5              | 3688374325B992DEF12793500307566D | C:\Windows\System32\drivers\etc\hosts |
| Algorithm        | Hash                             | Path                                  |
| -----            | -----                            | -----                                 |
| MD5              | E47B8315753B7FA34D1FCD693D209C85 | C:\Windows\System32\drivers\etc\hosts |

- Alias를 만들기
- Alias를 Profile에 넣기
- 가장 많이 사용하는 명령어-1
- 변경된 파일 확인하기

# 정리

- 별칭(Alias) 사용하기
- 변수(Variable) 사용하기
- 필수적인 cmdlet