PowerShell 4.0

Grundlagen und Verwaltung des Active Directory

Andreas Dittfurth

1. Ausgabe, Juni 2014

WPOW40

HERDT

Befehlsketten sind in der Regel nach dem folgenden Prinzip aufgebaut:

Bearbeitungsschritt	Erläuterung	Häufige Cmdlets
1 Daten bereitstellen	Im ersten Schritt benötigen Sie Daten, mit denen Sie arbeiten wollen. Dafür setzen Sie in der Regel Cmdlets mit dem Verb *Get* ein.	`Get-…`
2 Ergebnis bearbeiten, bis es den Erwartungen entspricht	Hier werden Cmdlets eingesetzt, die mit Daten arbeiten, die sie über die Pipeline erhalten. Häufig finden Sie in diesem zweiten Schritt, der auch aus etlichen Teilschritten bestehen kann, Cmdlets mit dem Substantiv *Object*.	`…-Object`

3 Endergebnis ausgeben	Lassen Sie diesen Schritt weg, erfolgt eine Ausgabe im Konsolenfenster. Ansonsten haben Sie die Möglichkeit,
	✔ die Formatierung Ihren Wünschen anzupassen,
	✔ die Ausgabe umzuleiten, z. B. in eine Datei zu exportieren.
		`Export-…`
		`Format-…`
		`Out-…`

Eine Ausgabe erfolgt - sofern vorgesehen - erst im letzten Schritt, also mit dem letzten Cmdlet der Pipeline. Alle vorherigen Cmdlets leiten ihre Objekte direkt an das Cmdlet im nächsten Bearbeitungsschritt weiter. Soll das letzte Cmdlet z. B. das Ergebnis in einer Datei exportieren, erfolgt keine Ausgabe im Konsolenfenster.

Im letzten Kapitel haben Sie den ersten Bearbeitungsschritt kennengelernt und Informationen gesammelt. Die optionalen weiteren Schritte der Ergebnisbearbeitung und -ausgabe lernen Sie in den nachfolgenden Abschnitten kennen. Dabei lernen Sie in Beispielen, wie die PowerShell-Pipeline eingesetzt wird.

4.2 Verarbeitung vorliegender Daten: Object-Cmdlets

Wichtige Cmdlets, die die Pipeline benötigen

Fast alle *Object*-Cmdlets benötigen Daten aus der Pipeline, mit denen sie arbeiten können. Wenn Sie nach Cmdlets mit dem Substantiv *Object* suchen (`Get-Command –Noun object`), erhalten Sie die neun Cmdlets als Ergebnis. Während die Cmdlets `Compare-Object` (Vergleich zweier Sätze von Objekten) und `New-Object` (Erstellen eines neuen Objekts) an dieser Stelle keine Rolle spielen, werden die übrigen sieben Cmdlets in diesem Zusammenhang häufig verwendet.

Die folgende Übersicht zeigt Ihnen, um welche Cmdlets es sich handelt und welche Aufgabe sie erfüllen (jedes Cmdlet wird weiter unten ausführlich besprochen):

Cmdlet	Kurzbeschreibung
`ForEach-Object`	Erlaubt die Anwendung eines Skriptblocks auf jedes übergebene Objekt, funktioniert wie eine Schleife, die alle einzelnen Elemente durchläuft *Beispiel*: `3,4,5,6,7 | ForEach-Object -Process {$_*2}` Sie wollen alle vorliegenden Zahlenwerte verdoppeln.
`Group-Object`	Gruppiert übergebene Objekte nach den Werten ihrer Eigenschaften *Beispiel*: `Get-Service | Group-Object –Property Status` Die Liste der Dienste auf Ihrem Rechner sollen nach ihrem Status (gestartet oder beendet) gruppiert werden.
`Measure-Object`	Berechnet numerische Eigenschaften von Objekten wie Anzahl, Mittelwert, Summe etc. *Beispiel*: `Get-Command –CommandType cmdlet | Measure-Object` Sie wollen ermitteln, wie viele Cmdlets aktuell in der PowerShell verfügbar sind.

Cmdlet	Kurzbeschreibung
`Select-Object`	Wählt Eigenschaften eines Objekts gemäß Ihren Wünschen aus *Beispiel*: `Get-Process | Select-Object -Property Id, ProcessName` Sie möchten von der Liste der laufenden Prozesse nur die Eigenschaften *Id* und *ProcessName* sehen.
`Sort-Object`	Sortiert Objekte nach den Werten der Eigenschaften und entfernt bei Bedarf Mehrfachwerte *Beispiel*: `Get-ChildItem | Sort-Object -Property length` Sie wollen die Dateien bei einer Auflistung des aktuellen Verzeichnisses nach Größe sortieren.
`Tee-Object`	Die Ausgabe des Befehls wird in einer Datei oder Variablen gespeichert und zusätzlich in der Konsole angezeigt. *Beispiel*: `Get-Service | Tee-Object -FilePath .\dienste.txt` Eine Liste der Dienste auf Ihrem Rechner wird in der Konsole angezeigt und gleichzeitig in die Datei dienste.*txt* im aktuellen Verzeichnis gespeichert.
`Where-Object`	Filtert Objekte nach beliebigen Kriterien *Beispiel*: `Get-ChildItem C:\Windows\System32 | Where-Object {$_.length -gt 10mb}` Sie suchen im Verzeichnis *C:\Windows\System32* nach Dateien mit einer Dateigröße von mehr als 10 MB.

Sie sehen bereits an diesen Beispielen, wie effektiv eine solche Befehlskette funktionieren kann. Schauen Sie sich in den nächsten Abschnitten die vorgestellten Cmdlets nach ihrer Verwendungshäufigkeit noch etwas genauer an.

`Select-Object` - Auswahl nach Ihrem Wunsch

Das Cmdlet `Select-Object` ist eines der am häufigsten verwendeten Cmdlets in einer Pipeline. Mit seiner Hilfe wählen Sie die Eigenschaften eines Objekts aus, die Sie sehen wollen. Dabei stehen alle für das Objekt definierten Eigenschaften zur Verfügung. Es spielt keine Rolle, ob die gewünschte Eigenschaft in der Standardausgabe der PowerShell angezeigt wird.

Für das Cmdlet stehen u. a. folgende Parameter zur Verfügung:

`Select-Object`		
Parameter	**Typ**	**Beschreibung**
`-Property`	P (1)	Gibt die Eigenschaft(en) an, die ausgewählt werden soll(en). Dies kann auch eine Eigenschaft sein, die Ergebnis einer Berechnung ist.
`-First`	N	Gibt die ersten *x* Objekte des Ergebnisses an (x ist der angegebene Zahlenwert des Parameters)
`-Last`	N	Gibt die letzten *x* Objekte des Ergebnisses an (x ist der angegebene Zahlenwert des Parameters)
`-Skip`	N	Überspringt *x* Objekte des Ergebnisses (x ist der angegebene Zahlenwert des Parameters)
`-Unique`	S	Zeigt alle mehrfach vorkommenden Werte nur einmalig an
…		Vollständige Hilfe inklusive aller Parameter, Beispiele etc.: `Get-Help –Name Select-Object –Full`

In den folgenden praktischen Beispielen sehen Sie Anwendungsmöglichkeiten für das Cmdlet `Select-Object`:

1) Gefällt Ihnen die Standardausgabe eines PowerShell-Cmdlets nicht, weil sie zu viele Informationen liefert, filtern Sie die Ausgabe:

```
Get-Service -Name s* | Select-Object -Property Name, Status
```

- ✔ Für alle Prozesse, deren Name mit dem Buchstaben *s* beginnt, werden nur noch die Eigenschaften *Name* und *Status* angezeigt. Die Namen der Eigenschaften sehen Sie als Spaltenkopf in der Standardausgabe.

2) Vergleichen Sie die Ausgabe der PowerShell bei Eingabe der beiden folgenden Zeilen:

```
Get-Process –Name powershell
```

- ✔ Sie erhalten Informationen zum laufenden PowerShell-Prozess, standardmäßig formatiert als Tabelle mit acht vorgegebenen Spalten.

```
Get-Process –Name powershell | Select-Object –Property *
```

- ✔ PowerShell ändert die Formatierung und zeigt nunmehr eine Liste von mehr als 60 Eigenschaften des Prozesses an. Sie können gewünschte Eigenschaften direkt benennen oder Wildcards einsetzen.
- ✔ Über diesen Weg erhalten Sie als Nebenprodukt Kenntnis aller Namen der verfügbaren Eigenschaften. Alternativ finden Sie diese mit der folgenden Zeile heraus:

```
Get-Process –Name powershell | Get-Member
```

3) Sie haben eine Liste mit Namen von Siegern eines PowerShell-Wettbewerbs. Sie möchten aber nur herausfinden, welche Personen bereits den Wettbewerb gewonnen haben:

```
"Sam", "Hugo", "Sam", "Sam", "Karla", "Hugo" | Select-Object -Unique
```

- ✔ Als Ergebnis liefert die PowerShell drei Namen zurück, doppelte Werte werden ignoriert.
- ✔ Sie können dieses Beispiel gut mit den Parametern `–Skip`, `–First` oder `–Last` kombinieren. In der Praxis bietet sich an, vor der Auswahl von Eigenschaften die Objekte zu sortieren, so dass Sie eine Anzahl Objekte aus einer Art Rangliste wählen können.

`Sort-Object` - Sortieren von Ergebnissen

Das Sortieren von Werten der Objekteigenschaften gehört zu den Standardaufgaben, wenn man eine Anzahl von Objekten weiter verarbeiten möchte. Die PowerShell stellt für diese Aufgabe das Cmdlet `Sort-Object` mit u. a. folgenden Parametern zur Verfügung:

`Sort-Object`		
Parameter	**Typ**	**Beschreibung**
`-Property`	P (1)	Gibt die Eigenschaft an, nach deren Werten sortiert werden soll. Kann weggelassen werden, woraufhin die Sortierung nach Standardeigenschaften des Objekttyps durchgeführt wird.
`-CaseSensitive`	S	Groß- und Kleinschreibung soll bei der Sortierung berücksichtigt werden.
`-Descending`	S	Die Sortierung wird in absteigender Reihenfolge durchgeführt.
`-Unique`	S	Entfernt Duplikate und zeigt nur eindeutige Werte an
…		Vollständige Hilfe inklusive aller Parameter, Beispiele etc.: `Get-Help –Name Sort-Object –Full`

Beispielanwendungen für die Sortierung von Objekten mithilfe des Cmdlets `Sort-Object` sind u. a.:

1) Sie lesen Einträge aus einem Ereignisprotokoll aus und möchten Sie nach der Eigenschaft *InstanceID* sortieren:

```
Get-EventLog -LogName System -Newest 20 | Sort-Object –Property InstanceID
```

- ✔ Sie erhalten die 20 jüngsten Einträge aus dem System-Ereignisprotokoll und geben Sie an das Cmdlet `Sort-Object` weiter, damit sie nach den Werten der Eigenschaft *InstanceID* aufsteigend sortiert werden.
- ✔ Eine absteigende Sortierung erreichen Sie, indem Sie den Parameter `–Descending` ergänzen.

2) Es bietet sich an, eine Dateiliste eines Verzeichnisses nach bestimmten Kriterien zu sortieren:

```
Get-ChildItem C:\Windows -File | Sort-Object –Property Extension, Length
```

- ✔ Das erste Cmdlet `Get-ChildItem` listet durch den Parameter `–File` ausschließlich Dateien des Windows-Verzeichnisses auf.
- ✔ Das folgende Cmdlet Sort-Object sortiert die übergebenen Objekte in erster Ebene nach ihrer Dateierweiterung, in zweiter Ebene zusätzlich aufsteigend nach Dateigröße.

`Where-Object` - Filtern von Ergebnissen

Wenn Sie über die Pipeline übergebene Objekte inhaltlich filtern wollen, kann das Cmdlet `Where-Object` nützlich sein. Folgende Parameter stehen Ihnen u. a. zur Verfügung:

`Where-Object`		
Parameter	**Typ**	**Beschreibung**
`-FilterScript`	P (1)	Als Wert wird ein Skriptblock in geschweiften Klammern erwartet, der auf jedes übergebene Objekt angewendet werden soll. Innerhalb des Skriptblocks wird das aktuelle Objekt durch die Variable `$_` angesprochen.
`-Property`	P (1)	Alternativ zu `–FilterScript`: Es wird kein Skriptblock angegeben, sondern direkt eine Objekteigenschaft.
`-Value`	P(2)	Wird der Parameter `–Property` verwendet, wird mit dem Parameter `–Value` der Wert für den angegebenen Parameter angegeben.
Vergleichsoperatoren	N	Filter machen häufig nur Sinn, wenn Sie mit Vergleichsoperatoren (wie *größer als*, *kleiner als*, *gleich* etc.) arbeiten. So können Sie die Werte der Objekteigenschaften mit einer Vorgabe abgleichen und auswählen (oder nicht). In der neuen Version der PowerShell sind viele Vergleichsoperatoren als Parameter integriert (vgl. Abschnitt 8.5). Wichtige Parameter sind `–EQ` (gleich), `-GT` (größer als), `-LT` (kleiner als), die in den folgenden Beispielen verwendet werden. Die komplette Liste der Parameter erhalten Sie durch die folgende Eingabe: `Get-Help Where-Object –Parameter *`
…		Vollständige Hilfe inklusive aller Parameter, Beispiele etc.: `Get-Help –Name Where-Object –Full`

Sie können bei der Verwendung des Cmdlets aus zwei Syntaxtypen wählen: die neue *vereinfachte Syntax* (seit PowerShell 3.0) oder die *klassische Syntax* (Skriptblock, der in geschweiften Klammern anzugeben ist). In den folgenden Beispielen, die die Verwendung von `Where-Object` verdeutlichen sollen, werden beide Typen parallel verwendet:

1) Sie wollen aus der Liste der vorhandenen Dienste nur die momentan ausgeführten Dienste filtern:

```
Vereinfachte Syntax:  Get-Service | Where-Object Status -EQ Running
Klassische Syntax:    Get-Service | Where-Object { $_.Status -EQ "Running" }
```

- ✔ Mit dem angegebenen Befehl wird verglichen, ob der Wert der Eigenschaft *Status* der übergebenen Objekte mit *Running* („aktuell ausgeführt") übereinstimmt. Falls nicht, wird das aktuelle Objekt verworfen und das nächste betrachtet. Falls ja, wird es in der Ausgabe angezeigt.

2) Sie wollen aus dem Windows-Verzeichnis alle Dateien mit einer Dateigröße kleiner als 100 kB anzeigen:

```
Vereinfachte Syntax:  Get-ChildItem -Path C:\Windows -File | Where-Object Length
                      -LT 100KB
Klassische Syntax:    Get-ChildItem -Path C:\Windows -File | Where-Object { $_.Length
                      -LT 100KB }
```

- ✔ Der Wert der Eigenschaft *Length* wird mit dem angegebenen Wert 100 kB verglichen. Das Objekt wird angezeigt, wenn der Vergleich WAHR ergibt, ansonsten wird das Objekt übersprungen.

Mit beiden Schreibweisen erreichen Sie Ihr Ziel. Die klassische Schreibweise unterscheidet sich von der vereinfachten Syntax wie folgt:

- ✔ Die klassische Syntax arbeitet mit geschweiften Klammern. In diesem Skriptblock wird die Bedingung formuliert.
- ✔ Das aktuell verarbeitete Objekt wird durch die Zeichenfolge `$_` angesprochen.
- ✔ Zeichenketten sind in Anführungszeichen zu stellen.
- ✔ Im Gegensatz zur vereinfachten Syntax ist die klassische Syntax abwärtskompatibel.
- ✔ Im Skriptblock können mehrere Bedingungen miteinander verknüpft werden. Dies geschieht mit logischen Operatoren wie `–AND`, `-OR`, `-NOT` etc. Mit der vereinfachten Syntax geht dies nicht. Sie können nur in der Pipeline mehrere `Where-Object`-Cmdlets verwenden, was im Resultat einer `AND`-Verknüpfung entspricht.

Beispiel

`AND` (beide Bedingungen müssen zutreffen), auch mit vereinfachter Syntax möglich.

```
Get-ChildItem -Path C:\Windows -File | Where-Object Length -LT 100KB |
Where-Object Length -GT 50KB
```

OR (eine der beiden Bedingungen muss zutreffen), nur mit klassischer Syntax möglich.

```
Get-ChildItem -Path C:\Windows -File | Where-Object { $_.Length -LT 100KB -OR
$_.Length -GT 1MB }
```

Sollte ein in der Pipeline weiter vorn stehendes Cmdlet die Möglichkeit zur Filterung bieten, sollten Sie sie nutzen. `Where-Object` ist sicherlich praktisch, aber bedenken Sie, dass erst einmal alle Objekte beschafft und übertragen werden müssen, bevor sie nachträglich gefiltert werden. Beherrscht das Cmdlet, das die Daten beschafft, direkt die gewünschte Filterung, müssen nur die nötigen Daten übertragen werden. Die Filterung findet vor der Übertragung statt, ist damit auch häufig deutlich schneller. Schreiben Sie also z. B.:

```
Get-Process -Name powershell
```

… und nicht:

```
Get-Process | Where-Object Name -EQ powershell
```

`ForEach-Object` - dieselbe Aktion für alle Objekte

Das Cmdlet `ForEach-Object` führt dieselbe Aktion für alle Objekte aus, die über die Pipeline an das Cmdlet übergeben werden. Es wird also eine Schleife durchlaufen, die nacheinander die Objekte bearbeitet. Die Aktion, die dabei durchgeführt wird, ist frei programmierbar. Es kann sich durchaus um einen ganzen Skriptblock handeln.

Für das Cmdlet stehen u. a. folgende Parameter zur Verfügung:

ForEach-Object		
Parameter	**Typ**	**Beschreibung**
`-Process`	P (1)	Als Wert wird ein Skriptblock in geschweiften Klammern erwartet, der auf jedes übergebene Objekt angewendet werden soll. Innerhalb des Skriptblocks wird das aktuelle Objekt durch die Variable `$_` angesprochen.
`-MemberName`	P (1)	Alternativ zu `-Process`: Es wird kein Skriptblock angegeben, sondern direkt eine anzuzeigende Eigenschaft oder eine auszuführende Methode.
`-Begin`	N	In geschweiften Klammern wird ein Skriptblock angegeben, der vor Ausführung der Aktion für alle Objekte verarbeitet wird.
`-End`	N	Angabe eines Skriptblocks in geschweiften Klammern, der nach Beendigung der Aktion für alle Objekte ausgeführt wird
…		Vollständige Hilfe inklusive aller Parameter, Beispiele etc.: `Get-Help -Name ForEach-Object -Full`

Folgende Beispiele zeigen Ihnen mögliche Einsatzgebiete des Cmdlets `ForEach-Object`:

1) Es werden drei Zahlen an das Cmdlet `ForEach-Object` weitergereicht, um mit den unterschiedlichen Werten je ein Echoanforderungspaket („Ping“) an verschiedene IP-Adressen zu senden:

```
1..3 | ForEach-Object -Process { Test-Connection 127.0.0.$_ -Count 1}
```

- ✔ Da auf der linken Seite der Pipeline drei Zahlenwerte (1, 2, 3) vorhanden sind, wird das Cmdlet `ForEach-Object` drei Mal durchlaufen.
- ✔ Im Skriptblock, der den Wert für den Parameter `-Process` bildet, wird der Befehl mit jeweils nur einem Paket ausgeführt. Als Zieladresse werden die IP-Adressen 127.0.0.1, 127.0.0.2 und 127.0.0.3 verwendet. Im ersten Schleifendurchlauf repräsentiert `$_` das erste übergebene Objekt (also *1*), im zweiten Durchlauf *2*, im letzten schließlich *3*.

2) Beispiel 1) wird erweitert und mit Aktionen vor und nach der eigentlichen Ausführung des Cmdlets versehen:

```
1..3 | ForEach-Object  -Begin { Clear-Host; Write-Host "Ich beginne zu pingen :" }
                       -Process { Test-Connection 127.0.0.$_ -Count 1}
                       -End { Write-Host "Aktion beendet." }
```

- ✔ Hinweis: Die eine lange Eingabezeile wurde aus Gründen der Übersichtlichkeit umformatiert.
- ✔ Es wurden Aktionen zu Beginn und Ende hinzugefügt.
- ✔ Im Parameter `-Begin` sind mehrere Cmdlets zu finden. Als Trennzeichen zwischen den Cmdlets fungiert das Semikolon.

3) Mit der PowerShell können Sie auch Ihren Rechner herunterfahren. Das Cmdlet `Stop-Computer` - ohne weitere Parameter verwendet – führt dazu, dass Ihr lokaler Rechner herunterfährt. Aber als Administrator können Sie das Cmdlet auch anders einsetzen:

```
"Client1", "Server2", "Server3", "Client7" |
             ForEach-Object -Process { Stop-Computer -ComputerName $_ -WhatIf }
```

- ✔ Es liegt eine Anzahl von Computernamen aus dem lokalen Netzwerk vor. Die Namen können ebenso zeilenweise in einer Textdatei stehen. In dem Fall lesen Sie die Textdatei (im aktuellen Verzeichnis) mit dem Befehl `Get-Content .\dateiname.txt` ein.
- ✔ Auf jedes übergebene Objekt, also auf jeden Computernamen, wird ein Skriptblock angewendet. Dies ist das Cmdlet `Stop-Computer` mit dem Parameter `-ComputerName`. Folglich fährt jeder Rechner herunter, dessen Name als Wert (`$_`) an den genannten Parameter übergeben wird.
- ✔ Um das Beispiel zu entschärfen, wird zusätzlich der allgemeine Parameter `-WhatIf` verwendet, durch den eine Ausführung des Cmdlets nur simuliert wird.

`Group-Object` - Eigenschaften mit gleichen Werten gruppieren

Das Cmdlet `Group-Object` dient zur Analyse von Häufigkeiten. Objekte werden auf Basis des Werts einer angegebenen Eigenschaft gruppiert. Standardmäßig erhalten Sie als Rückgabe eine Tabelle mit Angaben zur Häufigkeit, dem gruppierten Wert und den Mitgliedern der Gruppierung (Einzelwerten).

Geben Sie mehrere Eigenschaften an, wird zuerst nach der ersten Eigenschaft gruppiert und dann weiter verschachtelt, also Untergruppen für die weiteren Eigenschaften innerhalb der Hauptgruppierung gebildet.

Für das Cmdlet stehen u. a. folgende Parameter zur Verfügung:

`Group-Object`		
Parameter	**Typ**	**Beschreibung**
`-Property`	P (1)	Gibt die Eigenschaft für die Gruppierung an. Die Objekte werden auf Basis der Werte der Eigenschaft gruppiert.
`-NoElement`	S	Gibt die Einzelwerte nicht mit aus, sondern nur Anzahl und Wert, nach dem gruppiert wird
…		Vollständige Hilfe inklusive aller Parameter, Beispiele etc.: `Get-Help –Name Group-Object –Full`

In den folgenden praktischen Beispielen sehen Sie Anwendungsmöglichkeiten für das Cmdlet `Group-Object`:

1) Sie wollen sich informieren, welche Dateiendungen wie häufig im Windows-Verzeichnis zu finden sind:

```
Get-ChildItem C:\Windows –File | Group-Object -Property Extension
```

- ✔ `Get-ChildItem` mit dem Parameter `–File` liest alle Dateien im angegebenen Verzeichnis aus.
- ✔ Im zweiten Schritt werden die Dateien nach ihrer Endung gruppiert. Sie sehen in der Ausgabe alle vorkommenden Dateiendungen, die Häufigkeit ihres Vorkommens und die ersten vier Elemente der Gruppierungen (soweit es die Fensterbreite zulässt).

Sollten Sie nicht wissen, nach welchen Eigenschaften Sie hier im Beispiel gruppieren können, helfen Ihnen zum einen die Autovervollständigung in der PowerShell-Konsole sowie das IntelliSense in PowerShell ISE. Zum anderen erinnern Sie sich an das Cmdlet Get-Member, das die zur Verfügung stehenden Eigenschaften und Methoden anzeigt (z. B.: `Get-ChildItem C:\Windows –File | Get-Member`).

2) Sie wollen herausfinden, welche Verben in welcher Häufigkeit in allen Arten von PowerShell-Befehlen vorkommen:

```
Get-Command | Group-Object -Property Verb -NoElement | Sort-Object -Property Count
```

- ✔ `Get-Command` liefert die eine Liste der PowerShell-Befehle.
- ✔ Mit `Group-Object` werden sie nach ihrem Verb gruppiert. Auf die Ausgabe der Einzelelemente wird verzichtet.
- ✔ Letztlich sorgt `Sort-Object` für eine Sortierung nach der Häufigkeit der vorkommenden Verben. Oder genauer: `Group-Object` zeigt eine Tabelle mit den Eigenschaften *Count* (Häufigkeit des Vorkommens) und *Name* (Wert der Eigenschaft *Verb*) an. Es erfolgt eine Sortierung aufsteigend nach dem Zahlwert der Eigenschaft *Count*.

`Measure-Object` - Statistiken für die Objekte

Das Cmdlet `Measure-Object` berechnet numerische Eigenschaften bestimmter Objekttypen. Das Cmdlet kann Objekte zählen sowie Minimum, Maximum, Mittelwert und Summe numerischer Werte berechnen. Bei Textobjekten werden die Anzahl von Textzeilen, Wörtern und Zeichen gezählt.

Für das Cmdlet stehen u. a. folgende Parameter zur Verfügung:

Measure-Object		
Parameter	**Typ**	**Beschreibung**
`-Property`	P (1)	Gibt die Eigenschaft(en) für die Messung an. Ohne weitere Parameter wird die *Count*-Eigenschaft des Objekts angezeigt.
`-Average`	S	Für numerische Objekte: zeigt den Mittelwert der angegebenen Eigenschaft(en) an
`-Character`	S	Für Textobjekte: zählt die Anzahl der Zeichen
`-IgnoreWhiteSpace`	S	Für Textobjekte: Leerzeichen werden bei der Zählung von Wörtern und Zeichen ignoriert.
`-Line`	S	Für Textobjekte: zählt die Anzahl der Zeilen
`-Maximum`	S	Für numerische Objekte: zeigt den größten Wert der angegebenen Eigenschaft(en) an
`-Minimum`	S	Für numerische Objekte: zeigt den kleinsten Wert der angegebenen Eigenschaft(en) an
`-Sum`	S	Für numerische Objekte: zeigt die Summe der Werte der angegebenen Eigenschaft(en) an
`-Word`	S	Für Textobjekte: zählt die Anzahl der Wörter
…		Vollständige Hilfe inklusive aller Parameter, Beispiele etc.: `Get-Help –Name Measure-Object –Full`

Die folgenden Beispiele zeigen das Cmdlet im Praxiseinsatz:

1) Sie möchten Statistikwerte für eine Zahlenreihe ermitteln:

```
7, 3, 5, 22, -5, 4, 11, 17, 2 | Measure-Object -Average -Sum -Maximum –Minimum
```

✔ Numerische Werte erhalten Sie im Beispiel aus einer Eingabe. Mithilfe von `Measure-Object` und den entsprechenden Switch-Parametern werden nicht nur die Anzahl der numerischen Werte, sondern auch Mittelwert, Summe, Maximal- und Minimalwert ausgegeben.

2) Sie möchten die Statistiken einer Textdatei auslesen:

```
Get-Content –Path .\beispiel.txt | Measure-Object -Line –Word -Character
```

✔ Über die Pipeline wird der Inhalt einer Textdatei übergeben und statistisch ausgewertet. In diesem Fall werden Zeilen, Wörter und Zeichen gezählt.

✔ Sollten Sie in diesem Beispiel einen Switch-Parameter für numerische Objekte einsetzen, weisen Fehlermeldungen auf den unpassenden Objekttyp hin.

3) Sie möchten erfahren, wie groß die Dateien im Windows-Verzeichnis sind:

```
Get-ChildItem -Path C:\Windows | Measure-Object –Property Length -Sum
```

✔ Im ersten Cmdlet erfolgt der Zugriff auf die Dateien im Windows-Verzeichnis. Measure-Object zeigt für die übergebenen Objekte die Summe der Eigenschaft *Length* (Dateigröße) an.

`Tee-Object` - Variable und Ausgabe

Das Cmdlet `Tee-Object` ist ein Sonderfall. Während ein Objekt normalerweise entweder angezeigt oder verarbeitet wird, sendet `Tee-Object` die Ausgabe eines Befehls in beide Richtungen. Zum einen erfolgt die Speicherung in einer Variablen oder einer Datei, zum anderen erfolgt die Ausgabe in der Konsole bzw. Weitergabe über die Pipeline.

Für dieses Cmdlet stehen u. a. folgende Parameter zur Verfügung:

`Tee-Object`		
Parameter	**Typ**	**Beschreibung**
`-FilePath`	P (1)	Speichert das Objekt in der angegebenen Datei
`-Variable`	N	Das Objekt wird in der angegebenen Variablen gespeichert. Als Wert ist hier nur der Name erwartet, nicht das übliche führende $, das Variablen in vielen Programmiersprachen kennzeichnet.
`-Append`	S	Wird der Parameter angegeben, werden Daten an eine Datei angehängt. Ohne diesen Parameter wird eine bestehende Datei ersetzt.
…		Vollständige Hilfe inklusive aller Parameter, Beispiele etc.: `Get-Help –Name Tee-Object –Full`

Zwei Beispiele zeigen das Cmdlet im Einsatz:

1) Alle Prozesse auf Ihrem Rechner sollen ausgegeben und zusätzlich in einer Datei gespeichert werden:

```
Get-Process | Tee-Object -FilePath .\beispiel.txt
```

- ✔ Die Liste der Prozesse wird in der Datei *beispiel.txt* im aktuellen Verzeichnis gespeichert. `Tee-Object` ist das letzte Cmdlet in der Pipeline, damit erfolgt eine Ausgabe und keine Weiterleitung an ein folgendes Cmdlet.

2) Sie wollen das aktuelle Datum und die aktuelle Uhrzeit in einer Variablen speichern und zusätzlich die Werte einiger Eigenschaften in der Konsole anzeigen.

```
Get-Date | Tee-Object -Variable beispiel | Select-Object -Property Month, DayOfWeek
```

- ✔ `Tee-Object` sorgt dafür, dass das übergebene Objekt in der Variablen `$beispiel` gespeichert wird. Da das Cmdlet in diesem Beispiel nicht am Ende der Pipeline steht, wird das verarbeitete Objekt an das folgende Cmdlet weitergegeben.
- ✔ Achten Sie darauf, dass der Variablenname als Wert der Eigenschaft `–Variable` ohne das führende $ einzugeben ist.

- ✔ Durch Eingabe von `$beispiel` in der Konsole (nach Ausführung der obigen Beispielzeile) können Sie sich den Inhalt der Variablen anzeigen lassen.
- ✔ `Select-Object` zeigt die Werte der beiden ausgewählten Eigenschaften in der Konsole an.

4.3 Formatierung und Ausgabe

Wenn die gewünschten Daten die Pipeline durchlaufen haben, bleibt als optionaler letzter Arbeitsschritt die Aufarbeitung der Daten. Sie können die Daten nach Ihren Wünschen für die Ausgabe formatieren oder die Ausgabe in eine Datei umleiten. Zudem stehen Ihnen Werkzeuge zur Verfügung, die Daten in verschiedene Formate zu konvertieren.

Für diese Aufgaben stehen Ihnen Cmdlet-Familien zur Verfügung, die an ihrem Verb erkennbar sind: *Out* (für Ausgabe), *Export* (für den Export) und *ConvertTo* (für die Konvertierung in ein anderes Format).

Pipeline zur Absicherung verwenden

Wenn Sie z. B. einen Prozess beenden wollen, verwenden Sie das Cmdlet `Stop-Process`. Falls Sie sich nicht ganz sicher sind, dass Sie den gewünschten Prozess adressieren, lesen Sie erst über ein Get-Cmdlet die gesuchte Information aus.

```
a) Stop-Process –Name powershell    oder
b) Get-Process –Name powershell
c) Get-Process –Name powershell | Stop-Process
```

Bedienen Sie sich immer der Methode b) + c), wenn Sie nicht ganz sicher sind. Methode a) allein beendet alle Prozesse mit Namen *powershell*. Gab es wirklich nur den einen, den Sie beenden wollen? Schauen Sie erst, welches Ergebnis Sie erhalten, und wenden Sie dann die beabsichtigte Aktion auf dieses Ergebnis an.

4.6 Kurz zusammengefasst

Mithilfe der Pipeline sind Sie in der Lage, komplexe Aufgaben zu erledigen. Die Pipeline arbeitet objektorientiert. Es wird also keine Zeichenfolge übergeben, sondern jeweils komplette Objekte, die mit all ihren Eigenschaften und Methoden dem nächsten Cmdlet zur Verfügung stehen.

Im ersten Schritt werden Informationen gesammelt, die in den optionalen weiteren Arbeitsschritten bearbeitet und den eigenen Wünschen angepasst werden. Im letzten optionalen Arbeitsschritt erfolgt dann die Umwandlung oder der Export des Ergebnisses - oder einfach die Ausgabe in der Konsole.

Setzen Sie die Cmdlets in der Pipeline sinnvoll ein. Filtern Sie z. B. Daten möglichst, bevor sie über die Pipeline an das nächste Cmdlet übergeben werden. Dies erhöht die Geschwindigkeit, da nur die gewünschten Daten gesammelt und weitergeleitet werden müssen.

4.7 Übung

Arbeit mit der Pipeline

Übungsdatei: – **Ergebnisdatei: *4_ergebnisse.txt***

① Zeigen Sie alle Dienste an, die mit dem Buchstaben s beginnen, aber beendet (*stopped*) sind.

② Sortieren Sie den Verzeichnisinhalt des Windows-Verzeichnisses (nur Dateien) absteigend nach Größe und exportieren das Ergebnis in die CSV-Datei `dateien.csv` im aktuellen Verzeichnis.

③ Es liegen drei Dateien vor: `prozesse1.csv`, `prozesse2.csv` und `prozesse3.csv`. Die Dateien wurden mit folgenden Befehlen erstellt:

```
-1- Get-Process | Export-Csv -Path .\prozesse1.csv
-2- Get-Process | Export-Csv -Path .\prozesse2.csv –Delimiter ";"
-3- Get-Process | Export-Csv -Path .\prozesse3.csv –UseCulture
```

a) Welche Unterschiede weisen die Dateien -1-, -2- und -3- auf?
b) Was ist der Unterschied zwischen den Dateien -2- und -3-?

④ Wie viele Dateien mit der Erweiterung `.exe` (extension) weist Ihr Windows-Verzeichnis auf? Wie groß sind diese Dateien insgesamt und durchschnittlich?

⑤ a) Rufen Sie - mit Mitteln, die in diesem Kapitel besprochen wurden - zehnmal die Anwendung `notepad.exe` auf.
b) Lassen Sie sich die zehn laufenden Prozesse anzeigen und beenden sie dann.