Programmierrichtlinien PowerShell

Bildungsgang:	Berufsschule Fachinformatiker/innen Systemintegration
Lernfeldnummer:	6
Lernfeldname:	Bereitstellen und Entwickeln von Anwendungssystemen
Version:	1.2
Stand:	25.09.19
Autor:	Markus Breuer
Organisation:	BK GuT

Ziel der Programmierrichtlinien ist die Förderung eines einheitlichen Programmierstils. Hierdurch werden die Programme einfacher zu lesen und verstehen. Dies ist insbesondere bei kooperativem Arbeiten mehrerer Entwickler hilfreich. Ebenso hilft es bei der Einarbeitung in fremde Programme.

Die hier beschriebenen Programmierrichtlinien sind am BK GuT bei allen PowerShell Aufgaben verbindlich. In begründeten Fällen kann abgewichen werden, sofern dies die Erreichung der obigen Ziele fördert.

Inhaltsverzeichnis

1 Namensgebung

- Es sollen immer die vollständigen Commandlet-Namen in der üblichen Schreibweise genutzt werden. Von PowerShell vorgegebene Alias-Namen können genutzt werden. Auf die Verwendung selbstdefinierten Alias-Namen soll verzichtet werden.

```
☺
Write-Output "Hallo Welt"
```

```
😐
echo "Hallo Welt"
```

```
☹
write-output "Hallo Welt"
ausgabe "Hallo Welt"
```

- Für Variablen und Funktionen sind beschreibende Namen zu verwenden. Bei zusammengesetzten Namen ist die camelCase-Notation zu verwenden. Bei Funktionsnamen soll die Verb-Substantiv-Form genommen werden, allerdings ohne Bindestrich.

```
☺
$prozessListe = ...
```

```
☹
$pl = ...
$x = $y
```

- Befehle zur Ablaufsteuerung (if, else, switch, while, …) werden klein geschrieben.

```
☺
if( $ueberlauf -eq $true) {
   Write-Output "Zu viele Elemente"
   }
```

```
☹
If( $ueberlauf -eq $true) { ...
IF( $ueberlauf -eq $true) { ...
```

- Bei Kommentaren, Variablen- und Funktionsnamen, Ausgabetexten, u.s.w. soll einheitlich die deutsche Sprache verwendet werden.

2 Formatierung

- Strukturblöcke werden 4 Zeichen eingerückt.
- Die öffnende Klammer eines Strukturblocks ist in der selben Zeile. Die schließende Klammer eines Strukturblocks ist immer in einer eigenen Zeile.

```
☺
    if( $ueberlauf -eq $true) {
       Write-Output "Zu viele Elemente"
       }
```

- Jede Funktion soll mit 2 Leerzeilen umgeben sein. Logisch zusammenhängende Zeilen können mit einer Leerzeile gekennzeichnet werden.
- Keine Leerzeichen am Ende einer Zeile.

3 Kommentare

- Jedes PowerShell-Skript fängt mit einem Standard-Kommentar an, der kurz über Inhalt, Autor, Erstellungsdatum, u.s.w. informiert.

```
    ☺
    <#

    .DESCRIPTION

        Das Skript begrüßt den Benutzer mit Namen und gibt das
        aktuelle Datum und die aktuelle Zeit aus.

    .NOTES

        Einordnung:          FISI-LF6-LS1-AB1-A2a
        Aufgabe:             aufgabe_01

        Name:                Markus Breuer
        Organisaion:         BK-GuT

        Erstellt:            07.07.2018
        Letzte Änderung:     23.10.2018

#>
```

- Der obige Minimalkommentar kann bei Bedarf um weitere Abschnitte, z.B. .PARAMETERS ergänzt werden. Er ist so gestaltet, dass er mit dem Get-Help – Commandlet zusammenarbeitet.

- Bei größeren Skripten, sollen die Kommentare helfen, sich zu orientieren.
- Sofern es nicht aus dem Variablennamen hervorgeht, wofür die Variable genutzt wird, soll ein erklärender Kommentar eingefügt werden.

4 Funktionen

- Für Funktionen sind beschreibende Namen zu verwenden. Bei zusammengesetzten Namen ist die camelCase-Notation zu verwenden. Bei Funktionsnamen soll die Verb-Substantiv-Form genommen werden, allerdings ohne Bindestrich.
- Jede Funktion soll mit 2 Leerzeilen umgeben sein.
- Verwenden Sie Parameter, um Probleme mit lokalen und globalen Variablenbereichen zu vermeiden.
- Für jede Funktion soll es einen Kommentar geben, der beschreibt, was diese Funktion macht. Bei Bedarf sind auch die Funktionsparameter zu beschreiben.
- Eine Funktion soll genau eine Aufgabe erfüllen.

5 Gesamtstruktur

- Alle Skripte sollen eine einheitliche Struktur besitzen.
- Alle Skripte starten mit einem einheitlichen Kommentar (siehe oben).
- Werden Funktionen genutzt, so sind diese in einem eigenen Bereich zu bündeln. Zusammengehörige Funktionen sollen möglichst zusammen stehen.
- Variablendeklarationen sollen in einem eigenen geschlossenen Bereich vorgenommen werden. Dies gilt insbesondere für Konfigurationsvariablen.

6 Ein- und Ausgabe

- Write-Output und Write-Host können zur Ausgabe genutzt werden. Außer in begründeten Sondersituationen sollen sie nicht gemischt eingesetzt werden.
- Wenn ein Skript vom Benutzer mehrere Informationen abfragt, ist ein einheitlicher Stil zu verwenden.
- Die Ausgabe von Zeit angaben soll einheitlich im Format HH:MM:SS erfolgen, aud Sekunden kann verzichtet werden. Die Stundenangabe soll im 24-Stunden-Modus erfolgen.
- Die Datumsangabe soll einheitlich im Format TT.MM.JJJJ erfolgen.
- Alle Ein- und Ausgaben sollen in deutscher Sprache erfolgen.

7 Sonstiges

- Semikolon am Zeilenende sollen vermieden werden.

```
☺
$prozessListe = Get-Process
```

```
☹
$prozessListe = Get-Process;
```

- Jede Zeile enhält nur einen Befehl

```
☺
Get-Process
Get-Service
```

```
☹
Get-Process; Get-Service
```

- Zeilen sollen maximal 90 Zeichen lang sein. Versuchen Sie ohne Folgezeilen mit dem Hochkomma (`)auszukommen.

```
☹
Get-WmiObject -Class win32_service `
              -Property name, state `
              -ComputerName server2
```

- Pipes sollen nach Möglichkeit in einer Zeile geschrieben werden. Ist ein Umbruch notwendig, so startet die nächste Zeile eingerückt mit dem Pipe-Symbol.

```
☺
Get-Process | Sort-Object Processname | Format-List

Get...sehr lang `
    | weiter geht es
```

- Fest codierte Zahlenwerte sollen vermieden werden. Statt dessen sind Konstanten mit aussagefähigen Namen zu verwenden.

- Benutze volle Parameternamen.

```
☺
Get-WmiObject -Class win32_service -Property name, state
```

```
☹
Get-WmiObject win32_service name, state
```