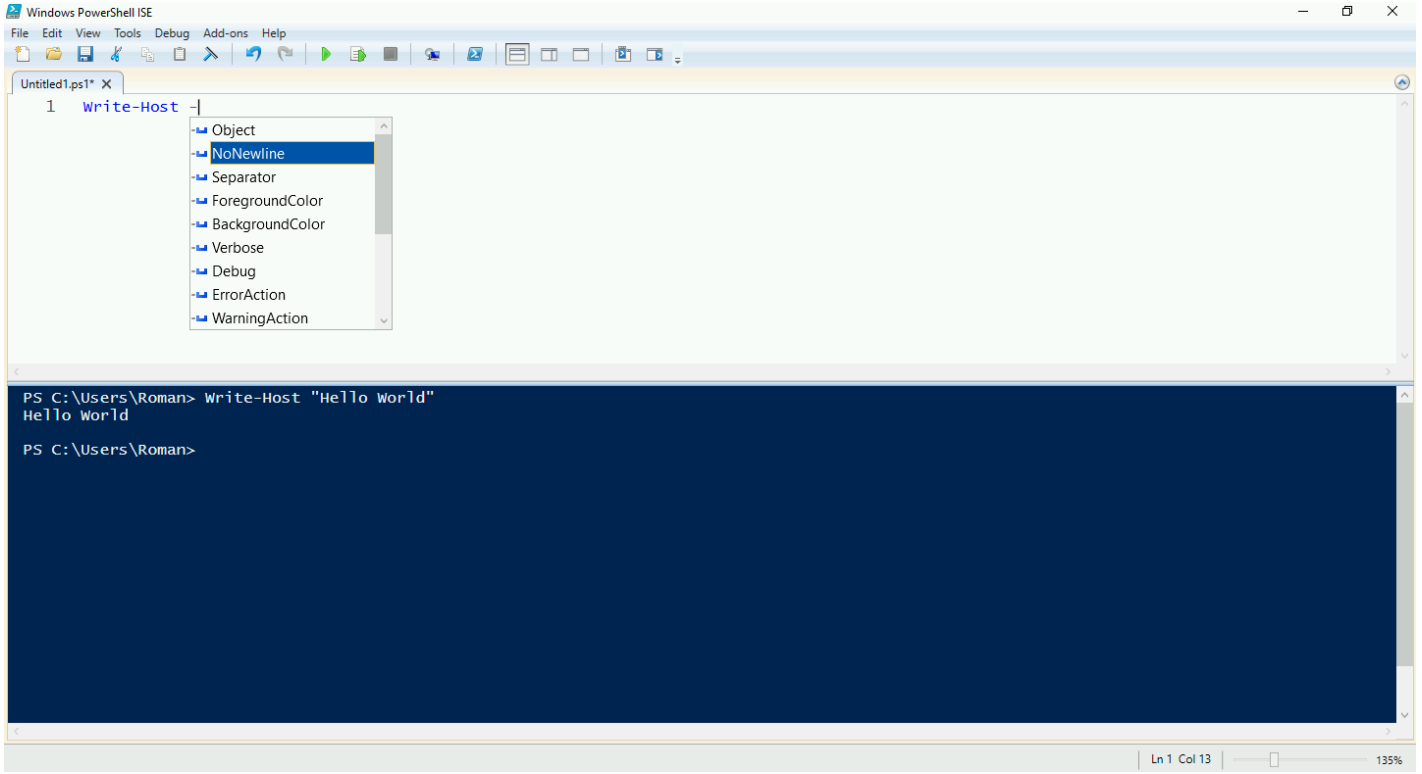
PowerShell für Office365-Admins

Mikel Münnekhoff, Roman Anasal

PowerShell-Grundlagen

Grundlagen

- Empfehlung: Verwendung von PowerShell ISE (Integrated Shell Environment)
  - Sehr gutes Autocomplete/Intellisense und integrierte Shell zum interaktiven Programmieren



Beispiel

```powershell
function Say-HelloWorld($name, $toUpper) {
    # Wenn kein Name angegeben wird
    # "World" nehmen
    if (!$name) {
        $name = "World"
    }

    if ($toUpper -eq $true) {
        $name = $name.ToUpper()
    } elseif ($toUpper -eq $false) {
        $name = $name.ToLower()
    }

    Write-Host "Hello $name"
}
```

```powershell
# Namen interaktiv einlesen z.B.
# Max
$username = Read-Host -Prompt "Name"

Say-HelloWorld
# Ausgabe: "Hello World"

Say-HelloWorld $username
# Ausgabe: "Hello Max"

Say-HelloWorld $username $true
# Ausgabe: "Hello MAX"

Say-HelloWorld $username $false
# Ausgabe: "Hello max"
```

Grundlagen zur Syntax

- Variablen beginnen mit `$`
- Alles ist ein Objekt → Funktionen auf ein Objekt kann mit angehängtem Punkt aufgerufen werden: `$name.ToUpper()`
- Funktionsargumentewerden mit Leerzeichen getrennt:
```
Say-HelloWorld $username $true
```
- (optionale) Parameter und Flags/Switches werden mit einem Minus angegeben:
```
$username = Read-Host -Prompt "Name eingeben"
```

Grundlagen zur Syntax

- Vergleichsoperatoren für `if`
  - `$var1 -eq $var2`: “Equals” / beide Werte sind gleich
  - `$var1 -ne $var2` : “not equals” / Werte sind unterschiedlich
  - `$var1 -like "Prefix*"`: Stringvergleich mit Platzhaltern oder Regular Expressions
  - `-lt/-gt`: “Less than”/”Greater than” – Kleiner/größer als
  - `-lte/-gte`: Kleiner/größer oder gleich

- Logische Ausdrücke
  - `$var1 -and $var2`: Logisches Und
  - `$var1 -or $var2`: logische Oder
  - `-not $condition`: logische Negierung

Arbeiten mit Arrays

```powershell
$names = @("Alice"; "Bob"; "Charlie")
$colors = @(
    "Blue"
    "Red"
)

$names.Count # 3

$names[1] # Bob

$names.Contains("Alice") # True

"Bob" -in $names # True

$colors -join ", " # Blue, Red
```

```powershell
$upperNames = $names | ForEach-Object {
    # $_ enthält das aktuelle Schleifenobjekt
    $name = $_

    # wird Rückgabe nicht in eine Variable
    # eingelesen, wird der Wert ausgegeben
    # und hier dann als Element eines neuen
    # Arrays in $upperNames gespeichert
    $name.ToUpper()
}
$upperNames -join ", " # ALICE, BOB, CHARLIE

$longNames = $upperNames | Where-Object {
    # Nur Objekte für die hier $true zurück-
    # gegeben wird, werden von Where-Objekt
    # weitergegeben
    $_.Length -gt 3
}
$longNames -join ", " # ALICE, CHARLIE
```

Arbeiten mit Strings

```powershell
$name = "World"

$greeting = "Hello $name" # "Hello World"

$greeting = "Hello " + $name # "Hello World"

$greeting -replace "World","Universe" # "Hello Universe"

$parts = $greeting -split " " # @("Hello"; "World")

$parts -join "|" # "Hello|World"

$name.Length # 5
```

Arbeiten mit Hash-Maps

```powershell
$person = @{
    Firstname = "Max"
    Lastname = "Mustermann"
    Email = "max.mustermann@bdsu.it"
}

$person.Lastname = "Power"

$person
# Name                           Value
# ----                           -----
# Email                          max.mustermann@bdsu.it
# Firstname                      Max
# Lastname                       Power

$person.Keys # @("Firstname"; "Lastname"; "Email")
$person.Values
# @("Max"; "Power"; "max.mustermann@bdsu.it")
```

```powershell
$person = [psCustomObject]@{
    Firstname = "Max"
    Lastname = "Mustermann"
    Email = "max.mustermann@bdsu.it"
}

$person | ft # Format-Table
# Firstname Lastname   Email
# --------- --------   -----
# Max       Mustermann max.mustermann@bdsu.it

$person | fl # Format-List
# Firstname : Max
# Lastname  : Mustermann
# Email     : max.mustermann@bdsu.it
```

Weitere nützliche Cmdlets

```
$object | fl # Format-List: zeilenweise Keys/Values ausgeben

$objects | ft # Format-Table: (mehrere) Objekte in Tabellenformat formatieren

# nur ausgewählte Attribute ausgeben
$object | fl DisplayName,*Mail*
$objects | ft DisplayName,*Mail*

# Objekte zählen
$objects | measure
$count = $objects | measure | Select-Object -ExpandProperty Count

$objects | sort # Objekte sortieren
$objects | sort Lastname,Firstname # nach bestimmten Attributen sortieren
$objects | sort -Unique # Duplikate entfernen
```

AzureAD Modul

Installation und Verbindung

- `Install-Module AzureAD`
  - Muss in einer PowerShell als Administrator ausgeführt warden

- `Connect-AzureAD`
  - Muss vor der Verwendung der anderen Cmdlets ausgeführt warden, optionale Parameter:
  - `-Credential $credentials`: Zugangsdaten, die mit Get-Credential in Variable eingelesen wurden
  - `-TenantId $tenantId`: um sich mit dem Account zu einem anderen Tenant zu verbinden (als Gast)

Arbeiten mit AzureADUser

- `$users = Get-AzureADUser -All $true`
- `$user = Get-AzureADUser -ObjectId 33868cff-57dc-4ad7-800e-1ba6563577ff`
- `$user = Get-AzureADUser -ObjectId demo@bdsu.it`
  - `-All $true`: ohne diesen Parameter warden nur die ersten 100 User zurückgegeben
  - `-ObjectId`: hier kann die ObjectID oder der Benutzername (UserPrincipalName) angegeben werden

- `$users | ft # User in lesbarer Tabelle anzeigen`
- `$user | fl # *alle* Attribute eines Users anzeigen`

- `Set-AzureADUser -ObjectId $id -DisplayName $newDisplayName # User bearbeiten`
- `New-AzureADUser -DisplayName "Max Mustermann" # neuen User erstellen`
- `# für weitere mögliche Attribute siehe jeweils Autocomplete`

- `Remove-AzureADUser -ObjectId $id # User löschen`

Wichtige AzureADUser Attribute

```
$user.ObjectId # eindeutige und feste ID
$user.UserPrincipalName # Benutzername (für Login)

$user.GivenName / $user.Surname # Vor-/Nachname
$user.DisplayName # Anzeigename

$user.Mail # Primäre E-Mail Adresse
$user.ProxyAddresses # Array mit allen E-Mail Adressen (Primär und Aliase)
$user.OtherMails # "Alternative E-Mails" – für Passwort Self-Service
```

Arbeiten mit AzureADGroup

```
$groups = Get-AzureADGroup -All $true
$group = Get-AzureADGroup -ObjectId $id

Set-AzureADGroup -ObjectId $id -DisplayName $newDisplayName # Gruppe bearbeiten
New-AzureADGroup -DisplayName $displayName # Gruppe erstellen
# für weitere mögliche Attribute siehe jeweils Autocomplete

Remove-AzureADGroup -ObjectId $groupId # Gruppe löschen

$members = Get-AzureADGroupMember -ObjectId $groupId -All $true # Gruppenmitglieder abrufen
```

- ohne `-All $true` würden nur die ersten 100 zurückgegeben werden

```
Add-AzureADGroupMember -ObjectId $groupId -RefObjectId $memberIds # Mitglieder hinzufügen
Remove-AzureADGroupMember -ObjectId $groupId -MemberId $memberId # Mitglied entfernen
```

Wichtige AzureADGroup Attribute

```
$group.ObjectId # eindeutige und unveränderbare ObjectId
$group.DisplayName # Anzeigename der Gruppe
$group.Description # optionale Beschreibung

$group.MailEnabled # ist die Gruppe E-Mail-aktiviert, d.h. ein Verteiler
$group.Mail # (primäre) E-Mail Adresse der Gruppe
$group.ProxyAddresses # alle E-Mail Adressen, primäre und Aliase

$group.SecurityEnabled # ist die Gruppe (auch) eine Sicherheitsgruppe

$members = Get-AzureADGroupMember -ObjectId $groupId
$members[0].ObjectType # "User", "Group" oder "Contact", da Gruppen nicht nur User enthalten
```

Exchange Online Remote Shell

Starten der Remote-Session

```
$credentials = Get-Credential
$session = New-PSSession -ConfigurationName Microsoft.Exchange -ConnectionUri
https://outlook.office365.com/powershell-liveid/ -Credential $credentials -Authentication
Basic -AllowRedirection
Import-PSSession $session

# Ab jetzt stehen die Exchange Cmdlets zur Verfügung:
# Get-Mailbox
# Get-DistributionGroup
# ...
```

- # ggf. müssen vorher die lokalen Sicherheitseinstellungen gelockert werden
- # dafür muss dieser Befehl in einer PowerShell als Administrator ausgeführt werden
- Set-ExecutionPolicy -ExecutionPolicy RemoteSigned
- # oder
- Set-ExecutionPolicy -ExecutionPolicy Bypass

Wichtige Exchange-Cmdlets

- New-Mailbox
- Get-Mailbox
- Set-Mailbox
- Remove-Mailbox

- New-DistributionGroup
- Get-DistributionGroup
- Set-DistributionGroup
- Remove-DistributionGroup

- Add-DistributionGroupMember
- Get-DistributionGroupMember
- Remove-DistributionGroupMember

- New-MailContact
- Get-MailContact
- Set-MailContact
- Remove-MailContact

Arbeiten mit Mailboxes

```powershell
$mailboxes = Get-Mailbox -ResultSize Unlimited
$mailbox = Get-Mailbox -Identity demo@bdsu.it

Set-Mailbox -Identity $mailbox.Identity -DisplayName "Demo"

# wichtige Attribute

$mailbox.Identity # eindeutige ID
$mailbox.DisplayName # Anzeigename des Postfachs
$mailbox.PrimarySmtpAddress # Primäre/Absenderadresse
$mailbox.EmailAddresses # Liste primärer und sekundärer (alias) Adressen
$mailbox.HiddenFromAddressListsEnabled # aus Adressbuch ausblenden
$mailbox.ForwardingSmtpAddress # Adresse für E-Mail-Weiterleitung
$mailbox.DeliverToMailboxAndForward # weitergeleitete E-Mails in Postfach speichern
```

Arbeiten mit Verteilern

```powershell
$groups = Get-DistributionGroup -ResultSize Unlimited
$group = Get-DistributionGroup -Identity qm@bdsu.it

Set-DistributionGroup -Identity $group.Identity -DisplayName "QM"

# wichtige Attribute

$group.Identity # eindeutige ID
$group.DisplayName # Anzeigename des Postfachs
$group.PrimarySmtpAddress # Primäre/Absenderadresse
$group.EmailAddresses # Liste primärer und sekundärer (alias) Adressen
$group.HiddenFromAddressListsEnabled # aus Adressbuch ausblenden
$group.ManagedBy # Besitzer der Gruppe
$group.ModeratedBy # Moderatoren der Gruppe
$group.ModerationEnabled # Nachrichten müssen durch Modertor freigegeben werden
```

Arbeiten mit Verteilermitgliedern

```powershell
$members = Get-DistributionGroupMember -Identity qm@bdsu.it

$members | ForEach-Object {
    Remove-DistributionGroupMember -Identity qm@bdsu.it -Member $_.Identity
}

$members | ForEach-Object {
    Add-DistributionGroupMember -Identity qm@bdsu.it -Member $_.Identity
}

# Wenn eingestellt ist, dass nur Beseitzer die Gruppenmitglieder bearbeiten können
# muss auch als globaler Administrator jeweils angegeben werden:
# -BypassSecurityGroupManagerCheck $true
```

Heutige Aufgabe

Szenario: Neue Trainees

- Eure JE nimmt neue Trainees auf. Gleichzeitig werden alle alten Trainees zu vollwertigen Mitgliedern.
- Die Accounts für die neuen Trainees wurden bereits per CSV-Import angelegt.
- Die Verteiler für Trainees und Mitglieder müssen umgestellt werden.
- Optional: Auch die Vorstände wechseln. Daher muss die Moderation der o.g. Verteiler umgestellt werden.

- Ihr benötigt:
  - Zugangsdaten für demo@bdsu.it
  - demo_users.csv
  - Einen zugeteilten Demo-Slot

  - Trainees.1@bdsu.it
  - Mitglieder.1@bdsu.it

Arbeiten mit CSV

```powershell
$data = Import-Csv -Delimiter ";" -Encoding UTF8 -Path ./demo_users.csv

$users = @(
    # casten zu psCustomObject damit Keys zu CSV Spalten werden
    [psCustomObject]@{Firstname = "Max"; Lastname = "Mustermann"}
    [psCustomObject]@{Firstname = "John"; Lastname = "Doe"}
)

$users | ConvertTo-Csv -Delimiter "," -NoTypeInformation
# "Firstname","Lastname"
# "Max","Mustermann"
# "John","Doe"

$users | Export-Csv -Delimiter ";" -Encoding UTF8 -NoTypeInformation -Path ./users.csv
```