SharePoint konferenz 2019

Office 365 Groups and Microsoft Teams PowerShell Masterclass

Martina Grom
Office 365 MVP, Microsoft RD, atwork
@magrom | mg@atwork.at

Toni Pohl
Azure and Office Dev MVP, atwork
@atwork | tp@atwork.at

Agenda

New PowerShell Modules for Teams

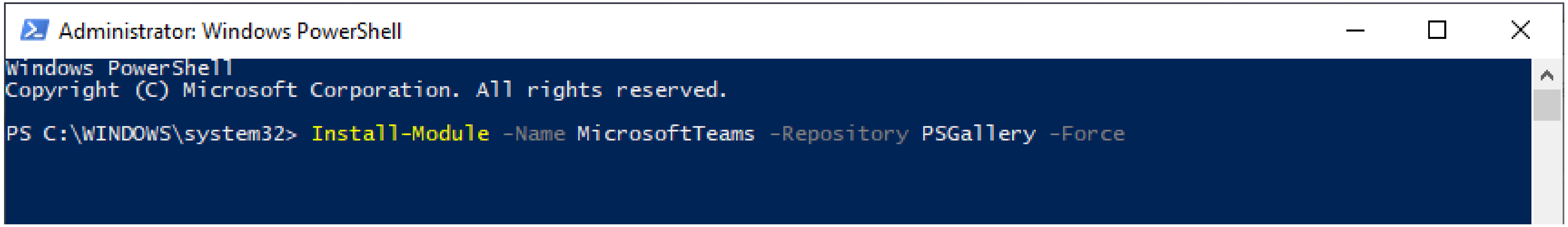
Use PowerShell at scale

PowerShell helps in Administration

Install the latest Teams PowerShell Module(s)

Microsoft Teams (1.0.0)
New: As Administrator, you can update the settings of any team.

Azure AD Preview (2.0.2.17)



Administrator: Windows PowerShell

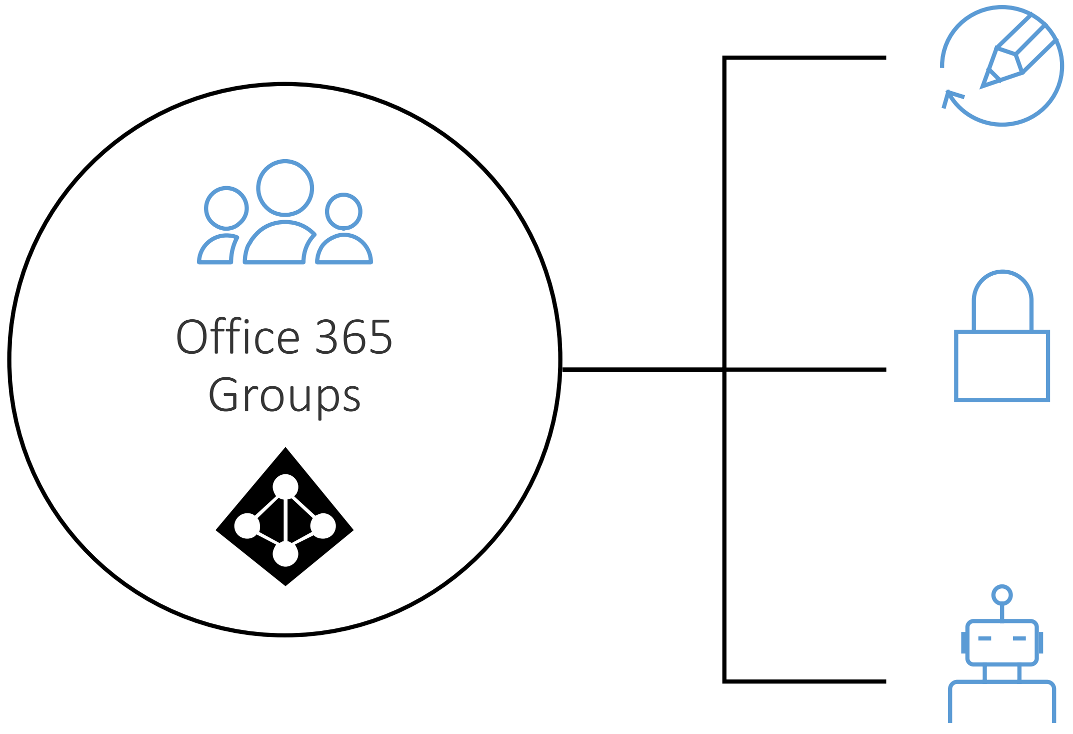
```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

PS C:\WINDOWS\system32> Install-Module -Name MicrosoftTeams -Repository PSGallery -Force
```

All Microsoft Teams PowerShells

All Office 365 Groups PowerShells

What is Office 365 Groups?

Single identity for teamwork and beyond



Membership service for M365 teamwork apps

Teams, Outlook, Yammer, SharePoint, Planner, Stream, Forms, StaffHub, Dynamics CRM, Power BI

Centrally managed and governed

Information protection, governance, and compliance

Extensible

Microsoft Graph, connectors, 3rd party app development

Essential Office 365 Groups and Microsoft Teams PowerShell

- Connect
- Essentials
- Expiration policy
- Soft delete and restore
- Ownerless

Use an Automation Account in Azure

- Process Automation
- Configuration Management
- Shared Capabilities

DEMO Auotmation Account

Connect

- **Azure AD**
- **Microsoft Teams**

```
#Import-Module MicrosoftTeams
#Import-Module AzureAD

# Then, connect...
$cred = Get-Credential
$cred.password.MakeReadOnly()

# Connect to the Teams module
Connect-MicrosoftTeams -Credential $cred

# If required, connect to the AAD module
# $cred should still have a password
Connect-AzureAD -Credential $cred

OR

Connect-AzureADPreview -Credential $cred

#Disconnect
Remove-PSSession $Session
```

Groups Essentials

List all groups and teams

List all private groups

```
#List all groups in descending order
Get-UnifiedGroup | Select Id, DisplayName, ManagedBy, Alias,
AccessType, WhenCreated, @{Expression={([array](Get-
UnifiedGroupLinks -Identity $_.Id -LinkType Members)).Count };
Label='Members'} | Sort-Object whencreated | Format-Table
displayname, alias, managedby, Members, accesstype, whencreated

#List all private groups
Get-UnifiedGroup | Where-Object {$_.AccessType -eq 'Private'} |
Sort-Object whencreated | Format-Table displayname, alias,
managedby, accesstype, whencreated
```

Teams Essentials

- List all Teams
- Modify or Delete a specific Team
- Channels (add / remove)

```
#Get all Microsoft Teams
Get-Team

# Delete a specific team
$group = 'ID '
Remove-Team -GroupId $group

# See all channels of that group
Get-TeamChannel -GroupId $group

# Create a new team channel in that group
New-TeamChannel -GroupId $group -DisplayName 'My Team' -
Description 'My team'

# Delete a channel
Remove-TeamChannel -GroupId $group -DisplayName 'Updated the team'

# Add a new user as member to that group
Add-TeamUser -GroupId $group -User EnricoC@contoso.com

# Modify that group (Alias is the nickname without domain)
Set-Team -GroupId $group -DisplayName 'Updated the team'
-Visibility Private -Classification confidential -Alias 'My Team'
```

Groups Expiration policy

- Setup the expiry for groups
- Add an additional admin for ownerless groups
- Specify expiration for specific groups

```
#Gets current setting
Get-AzureADMSGroupLifecyclePolicy | Format-List

#Removes current policy
Remove-AzureADMSGroupLifecyclePolicy -Id "groupID"

# Setup of new Groups Lifecycle policy (None, All, Selected)
New-AzureADMSGroupLifecyclePolicy -GroupLifetimeInDays 31 -
ManagedGroupTypes All -AlternateNotificationEmails
admin@contoso.com

#Update of a policy
Set-AzureADMSGroupLifecyclePolicy -Id "groupID" -
GroupLifetimeInDays 32 -AlternateNotificationEmails
"admin@contoso.com" -ManagedGroupTypes "Selected"

#Retrieves Lifecyclepolicy of a selected group
Get-AzureADMSLifecyclePolicyGroup -Id

#Renews a group by updating the RenewedDateTime property on a
group to the current DateTime.
Reset-AzureADMSLifeCycleGroup -GroupId <String>

#Adds a group to a lifecycle policy
Add-AzureADMSLifecyclePolicyGroup -Id <String> -GroupId <String>
```

Soft delete and restore

- Soft-delete a group and keep it in that state for 30 days
- Restore soft-deleted groups and Teams
- Hard delete specific groups

```
# Get all the Groups
Get-AzureADGroup

# Soft Delete a specific group
Remove-AzureADGroup -ObjectId "obejctID"

# Show all Soft Deleted Groups in descending order
Get-AzureADMSDeletedGroup | Sort-Object DeletedDateTime -
Descending | Format-Table Id, DisplayName, Description,
Visibility, DeletedDateTime

# Restore a specific soft deleted group
Restore-AzureADMSDeletedDirectoryObject -Id "ObjectID"

# Hard Delete a Group
Remove-AzureADMSDeletedDirectoryObject – Id <objectId>
```

Classification, Blocked Words

- **Set Classification**
- **Add Blocked words**
- **Add Usage policy**

```
# Create the template
$Template = Get-AzureADDirectorySettingTemplate -Id 62375ab9-6b52-
47ed-826b-58e47e0e304b

$Setting = $template.CreateDirectorySetting()

$setting["UsageGuidelinesUrl"] = "http://atwork-it.com"

$setting["ClassificationList"] = "public, internal, confidential,
strictly confidential"

$setting["ClassificationDescriptions"] = "public:no
restrictions,internal:all internal users can
access,confidential:only special users can access,strictly
confidential:only selected users can access"

$setting["GuestUsageGuidelinesUrl"] = "http://atwork-it.com"

$setting["CustomBlockedWordsList"] = "boss,ceo,SPC"
```

Audit Ownerless Groups

- **Identify the set of groups which do not have owners.**
- **Add an *admin* as owner of the ownerless group**
- **Add *another user* as the owner to ownerless groups.**

```
#Get Ownerless groups

$OwnerlessGroups =[array](Get-UnifiedGroup | Where-Object
{([array](Get-UnifiedGroupLinks -Identity $_.Id -LinkType
Owners)).Count -eq 0}) | Select Id, DisplayName, ManagedBy,
WhenCreated, SMTPAddress



#Assign owner to the ownerless group

for($i=0; $i -lt $OwnerlessGroups.Count; $i++)

    {
    Add-UnifiedGroupLinks $ OwnerlessGroups.Alias -LinkType
    member -Links admin@contoso.com
    Add-UnifiedGroupLinks $ OwnerlessGroups.Alias -LinkType Owner
    -Links admin@contoso.com
    }
```

Transfer of Group Ownership

- Identify the groups of which a user is an owner.
- Assign a new owner to the groups.
- Remove previous owner from the groups identified.

```
#Get the groups of which a user is an owner

     $UserOwnedObjects = Get-AzureADUser -SearchString adelev|
     Get-AzureADUserOwnedObject |  Where-Object {$_.ObjectType -
     eq "Group"}
     $UserOwnedGroups = @()
     for($i=0; $i -lt $UserOwnedObjects.Count; $i++)
     {
          $mbx = Get-UnifiedGroup $UserOwnedObjects[$i].ObjectId
          -ErrorAction SilentlyContinue
          if ( $mbx -ne $null )
          {
          $UserOwnedGroups += $mbx}
     }

#Assign new owner to the groups

     for($i=0; $i -lt $ UserOwnedGroups.Count; $i++)
     {
          Add-UnifiedGroupLinks $ UserOwnedGroups.Alias -LinkType
          member -Links admin@contoso.com
          Add-UnifiedGroupLinks $ UserOwnedGroups.Alias -LinkType
          Owner -Links admin@contoso.com
     }
```

Audit Groups Without Classification

- Identify the set of groups which don’t have classification assigned.
- Update groups to assign a classification.
- Configure additional group properties based on classification.

```
#Retrieve all groups where classification is blank or null

$GroupsWithNoClassification = Get-UnifiedGroup | Where-Object
{$_.Classification -Eq $Null -or $_.Classification -Eq ""} | Sort-
Object DisplayName | Select DisplayName, Classification,
ExternalDirectoryObjectId



#Set the classification of each group to “Medium"

ForEach ($G in $GroupsWithNoClassification) {

    If ($G.Classification -Eq $Null -or $G.Classification -Eq "")
    {

        Set-UnifiedGroup -Identity $G.DisplayName -Classification
        “High”

        Write-Host “The group classification setting for"
        $G.DisplayName "was updated to medium.“

    }

}
```

Audit Groups With Conflicting Properties

- Identify set of groups which have conflicting properties, such as a classification of “High” but Privacy of “Public”.
- Update the set of groups to assign correct values.
- Verify properties were updated successfully.

```
#Identify which groups have a mismatch of properties

$GroupsWithClassificationMismatch = Get-UnifiedGroup | Where-
Object {$_.AccessType -eq ‘Public’ -and $_.Classification -eq
“High”} | Select DisplayName, Classification, AccessType,
ExternalDirectoryObjectId



#Set the Access Type to Private for all "High" groups

ForEach ($G in $GroupsWithClassificationMismatch)
{

     Set-UnifiedGroup -Identity $G.DisplayName -AccessType
     ‘Private’

     Write-Host “The following Group privacy setting was
     updated:” $G.DisplayName

}
```

Disable / Hide Groups in Outlook

- Disable Groups in Outlook
- Hide from Address list
- Verify properties were updated successfully.

```
Set-UnifiedGroup -Identity <guid> -HiddenFromAddressListsEnabled
$true


$Groups = Get-UnifiedGroup -ResultSize Unlimited | ? {$_.Name -
like "Section_*"}

Foreach ($Group in $Groups) { Set-UnifiedGroup $Group.Guid -
HiddenFromAddressListsEnabled $true

}


Set-UnifiedGroup -Identity [Group] -
HiddenFromExchangeClientsEnabled:$True
```

Key session takeaways

New PowerShell Modules for Teams

Use PowerShell at scale

PowerShell helps in Administration

Fragen?

SharePoint
konferenz 2019

Ich freue mich auf Ihr Feedback!

Vielen Dank!

@magrom & @atwork