Set-Acl

Reference

Feedback

Module: Microsoft.PowerShell.Security

In this article

Changes the security descriptor of a specified item, such as a file or a registry key.

Syntax

PowerShell Copy

```
Set-Acl
   [-Path] <String[]>
   [-AclObject] <Object>
   [[-CentralAccessPolicy] <String>]
   [-ClearCentralAccessPolicy]
   [-Passthru]
   [-Filter <String>]
   [-Include <String[]>]
   [-Exclude <String[]>]
   [-WhatIf]
   [-Confirm]
   [-UseTransaction]
   [<CommonParameters>]
```

PowerShell Copy

```
Set-Acl
   [-InputObject] <PSObject>
   [-AclObject] <Object>
   [-Passthru]
   [-Filter <String>]
   [-Include <String[]>]
   [-Exclude <String[]>]
   [-WhatIf]
   [-Confirm]
   [-UseTransaction]
   [<CommonParameters>]
```

PowerShell Copy

```
Set-Acl
   -LiteralPath <String[]>
   [-AclObject] <Object>
   [[-CentralAccessPolicy] <String>]
   [-ClearCentralAccessPolicy]
   [-Passthru]
   [-Filter <String>]
   [-Include <String[]>]
   [-Exclude <String[]>]
   [-WhatIf]
```

```
    [-Confirm]
    [-UseTransaction]
    [<CommonParameters>]
```

Description

The `Set-Acl` cmdlet changes the security descriptor of a specified item, such as a file or a registry key, to match the values in a security descriptor that you supply.

To use `Set-Acl`, use the **Path** or **InputObject** parameter to identify the item whose security descriptor you want to change. Then, use the **AclObject** or **SecurityDescriptor** parameters to supply a security descriptor that has the values you want to apply. `Set-Acl` applies the security descriptor that is supplied. It uses the value of the **AclObject** parameter as a model and changes the values in the item's security descriptor to match the values in the **AclObject** parameter.

Examples

Example 1: Copy a security descriptor from one file to another

PowerShell Copy

```
$DogACL = Get-Acl -Path "C:\Dog.txt"
Set-Acl -Path "C:\Cat.txt" -AclObject $DogACL
```

These commands copy the values from the security descriptor of the Dog.txt file to the security descriptor of the Cat.txt file. When the commands complete, the security descriptors of the Dog.txt and Cat.txt files are identical.

The first command uses the `Get-Acl` cmdlet to get the security descriptor of the Dog.txt file. The assignment operator (`=`) stores the security descriptor in the value of the $DogACL variable.

The second command uses `Set-Acl` to change the values in the ACL of Cat.txt to the values in `$DogACL`.

The value of the **Path** parameter is the path to the Cat.txt file. The value of the **AclObject** parameter is the model ACL, in this case, the ACL of Dog.txt as saved in the `$DogACL` variable.

Example 2: Use the pipeline operator to pass a descriptor

PowerShell Copy

```
Get-Acl -Path "C:\Dog.txt" | Set-Acl -Path "C:\Cat.txt"
```

This command is almost the same as the command in the previous example, except that it uses a pipeline operator (`|`) to send the security descriptor from a `Get-Acl` command to a `Set-Acl` command.

The first command uses the `Get-Acl` cmdlet to get the security descriptor of the Dog.txt file. The pipeline operator (`|`) passes an object that represents the Dog.txt security descriptor to the `Set-Acl` cmdlet.

The second command uses `Set-Acl` to apply the security descriptor of Dog.txt to Cat.txt. When the command completes, the ACLs of the Dog.txt and Cat.txt files are identical.

Example 3: Apply a security descriptor to multiple files

PowerShell Copy

```
$NewAcl = Get-Acl File0.txt
Get-ChildItem -Path "C:\temp" -Recurse -Include "*.txt" -Force | Set-Acl -AclOb
```

These commands apply the security descriptors in the File0.txt file to all text files in the `C:\Temp` directory and all of its subdirectories.

The first command gets the security descriptor of the File0.txt file in the current directory and uses the assignment operator (`=`) to store it in the `$NewACL` variable.

The first command in the pipeline uses the Get-ChildItem cmdlet to get all of the text files in the `C:\Temp` directory. The **Recurse** parameter extends the command to all subdirectories of `C:\temp`. The **Include** parameter limits the files retrieved to those with the `.txt` file name extension. The **Force** parameter gets hidden files, which would otherwise be excluded. (You cannot use `c:\temp\*.txt`, because the **Recurse** parameter works on directories, not on files.)

The pipeline operator (`|`) sends the objects representing the retrieved files to the `Set-Acl` cmdlet, which applies the security descriptor in the **AclObject** parameter to all of the files in the pipeline.

In practice, it is best to use the **WhatIf** parameter with all `Set-Acl` commands that can affect more than one item. In this case, the second command in the pipeline would be `Set-Acl -AclObject $NewAcl -WhatIf`. This command lists the files that would be affected by the command. After reviewing the result, you can run the command again without the **WhatIf** parameter.

Example 4: Disable inheritance and preserve inherited access rules

PowerShell Copy

```
$NewAcl = Get-Acl -Path "C:\Pets\Dog.txt"
$isProtected = $true
$preserveInheritance = $true
$NewAcl.SetAccessRuleProtection($isProtected, $preserveInheritance)
Set-Acl -Path "C:\Pets\Dog.txt" -AclObject $NewAcl
```

These commands disable access inheritance from parent folders, while still preserving the existing inherited access rules.

The first command uses the `Get-Acl` cmdlet to get the security descriptor of the Dog.txt file.

Next, variables are created to convert the inherited access rules to explicit access rules. To protect the access rules associated with this from inheritance, set the `$isProtected` variable to `$true`. To allow inheritance, set `$isProtected` to `$false`. For more information, see set access rule protection.

Set the `$preserveInheritance` variable to `$true` to preserve inherited access rules or `$false` to remove inherited access rules. Then the access rule protection is updated using the **SetAccessRuleProtection()** method.

The last command uses `Set-Acl` to apply the security descriptor of to Dog.txt. When the command completes, the ACLs of the Dog.txt that were inherited from the Pets folder will be

applied directly to Dog.txt, and new access policies added to Pets will not change the access to Dog.txt.

Example 5: Grant Administrators Full Control of the file

PowerShell Copy

```
$NewAcl = Get-Acl -Path "C:\Pets\Dog.txt"
# Set properties
$identity = "BUILTIN\Administrators"
$fileSystemRights = "FullControl"
$type = "Allow"
# Create new rule
$fileSystemAccessRuleArgumentList = $identity, $fileSystemRights, $type
$fileSystemAccessRule = New-Object -TypeName System.Security.AccessControl.FileS
# Apply new rule
$NewAcl.SetAccessRule($fileSystemAccessRule)
Set-Acl -Path "C:\Pets\Dog.txt" -AclObject $NewAcl
```

This command will grant the **BUILTIN\Administrators** group Full control of the Dog.txt file.

The first command uses the `Get-Acl` cmdlet to get the security descriptor of the Dog.txt file.

Next variables are created to grant the **BUILTIN\Administrators** group full control of the Dog.txt file. The `$identity` variable set to the name of a user account. The `$fileSystemRights` variable set to FullControl, and can be any one of the FileSystemRights values that specifies the type of operation associated with the access rule. The `$type` variable set to "Allow" to specifies whether to allow or deny the operation. The `$fileSystemAccessRuleArgumentList` variable is an argument list is to be passed when making the new **FileSystemAccessRule** object. Then a new **FileSystemAccessRule** object is created, and the **FileSystemAccessRule** object is passed to the **SetAccessRule()** method, adds the new access rule.

The last command uses `Set-Acl` to apply the security descriptor of to Dog.txt. When the command completes, the **BUILTIN\Administrators** group will have full control of the Dog.txt.

Parameters

-AclObject

Specifies an ACL with the desired property values. `Set-Acl` changes the ACL of item specified by the **Path** or **InputObject** parameter to match the values in the specified security object.

You can save the output of a `Get-Acl` command in a variable and then use the **AclObject** parameter to pass the variable, or type a `Get-Acl` command.

Expand table

Type:	Object
Position:	1
Default value:	None
Required:	True
Accept pipeline input:	True
Accept wildcard characters:	False

-CentralAccessPolicy

Establishes or changes the central access policy of the item. Enter the CAP ID or friendly name of a central access policy on the computer.

Beginning in Windows Server 2012, administrators can use Active Directory and Group Policy to set central access policies for users and groups. For more information, see Dynamic Access Control: Scenario Overview.

This parameter was introduced in Windows PowerShell 3.0.

Expand table

Type:	String
Position:	2
Default value:	None
Required:	False
Accept pipeline input:	True
Accept wildcard characters:	False

-ClearCentralAccessPolicy

Removes the central access policy from the specified item.

Beginning in Windows Server 2012, administrators can use Active Directory and Group Policy to set central access policies for users and groups. For more information, see Dynamic Access Control: Scenario Overview.

This parameter was introduced in Windows PowerShell 3.0.

Expand table

Type:	SwitchParameter
Position:	Named
Default value:	False
Required:	False
Accept pipeline input:	False
Accept wildcard characters:	False

-Confirm

Prompts you for confirmation before running the cmdlet.

Expand table

Type:	SwitchParameter
Aliases:	cf
Position:	Named
Default value:	False
Required:	False
Accept pipeline input:	False
Accept wildcard characters:	False

-Exclude

Omits the specified items. The value of this parameter qualifies the **Path** parameter. Enter a path element or pattern, such as `*.txt`. Wildcards are permitted.

Expand table

Type:	String[]
Position:	Named
Default value:	None
Required:	False
Accept pipeline input:	False
Accept wildcard characters:	True

-Filter

Specifies a filter in the provider's format or language. The value of this parameter qualifies the **Path** parameter. The syntax of the filter, including the use of wildcards, depends on the provider. Filters are more efficient than other parameters, because the provider applies them when retrieving the objects, rather than having PowerShell filter the objects after they are retrieved.

Expand table

Type:	String
Position:	Named
Default value:	None
Required:	False
Accept pipeline input:	False
Accept wildcard characters:	True

-Include

Changes only the specified items. The value of this parameter qualifies the **Path** parameter. Enter a path element or pattern, such as `*.txt`. Wildcards are permitted.

Expand table

Type:	String[]
Position:	Named
Default value:	None
Required:	False
Accept pipeline input:	False
Accept wildcard characters:	True

-InputObject

Changes the security descriptor of the specified object. Enter a variable that contains the object or a command that gets the object.

You cannot pipe the object to be changed to `Set-Acl`. Instead, use the **InputObject** parameter explicitly in the command.

This parameter was introduced in Windows PowerShell 3.0.

Expand table

Type:	PSObject
Position:	0
Default value:	None
Required:	True
Accept pipeline input:	True
Accept wildcard characters:	False

-LiteralPath

Changes the security descriptor of the specified item. Unlike **Path**, the value of the **LiteralPath** parameter is used exactly as it is typed. No characters are interpreted as wildcards. If the path includes escape characters, enclose it in single quotation marks (`'`). Single quotation marks tell PowerShell not to interpret any characters as escape sequences.

This parameter was introduced in Windows PowerShell 3.0.

Expand table

Type:	String[]
Aliases:	PSPath
Position:	Named
Default value:	None
Required:	True
Accept pipeline input:	True
Accept wildcard characters:	False

-Passthru

Returns an object that represents the security descriptor that was changed. By default, this cmdlet does not generate any output.

Expand table

Type:	SwitchParameter
Position:	Named
Default value:	None
Required:	False
Accept pipeline input:	False
Accept wildcard characters:	False

-Path

Changes the security descriptor of the specified item. Enter the path to an item, such as a path to a file or registry key. Wildcards are permitted.

If you pass a security object to `Set-Acl` (either by using the **AclObject** or **SecurityDescriptor** parameters or by passing a security object from Get-Acl to `Set-Acl`),

and you omit the **Path** parameter (name and value), `Set-Acl` uses the path that is included in the security object.

Expand table

Type:	String[]
Position:	0
Default value:	None
Required:	True
Accept pipeline input:	True
Accept wildcard characters:	True

-UseTransaction

Includes the command in the active transaction. This parameter is valid only when a transaction is in progress. For more information, see about_Transactions.

Expand table

Type:	SwitchParameter
Aliases:	usetx
Position:	Named
Default value:	False
Required:	False
Accept pipeline input:	False
Accept wildcard characters:	False

-WhatIf

Shows what would happen if the cmdlet runs. The cmdlet is not run.

Expand table

Type:	SwitchParameter
Aliases:	wi
Position:	Named
Default value:	False
Required:	False
Accept pipeline input:	False
Accept wildcard characters:	False

Inputs

ObjectSecurity

You can pipe an ACL object to this cmdlet.

CommonSecurityDescriptor

You can pipe a security descriptor to this cmdlet.

Outputs

None

By default, this cmdlet returns no output.

FileSecurity

When you use the **PassThru** parameter, this cmdlet returns a security object. The type of the security object depends on the type of the item.

Notes

The `Set-Acl` cmdlet is supported by the PowerShell file system and registry providers. As such, you can use it to change the security descriptors of files, directories, and registry keys.

Related Links

- Get-Acl
- FileSystemAccessRule
- ObjectSecurity.SetAccessRuleProtection
- FileSystemRights