PowerShell Permissions Tutorial

This comprehensive tutorial covers managing permissions in PowerShell across different systems including file system, registry, Active Directory, and services. You'll learn to view, modify, and troubleshoot permissions using PowerShell.

Prerequisites

- PowerShell 5.1 or later
- Administrative privileges for most permission operations
- Active Directory module for AD permissions (`Install-Module ActiveDirectory`)
- Understanding of Windows security principals (users, groups, SIDs)

Part 1: File System Permissions

Understanding Access Control Lists (ACLs)

```powershell
# Get ACL for a file or folder
$acl = Get-Acl -Path "C:\Example\Folder"
$acl | Format-List

# View access rules in detail
$acl.Access | Format-Table IdentityReference, FileSystemRights, AccessControlType, InheritanceFlags, PropagationFlags

# Get owner information
$acl.Owner
$acl.Group
```

Viewing File System Permissions

```powershell
# Simple permission view
Get-Acl "C:\Example\File.txt" | Select-Object Owner, Group, Access

# Detailed permission analysis
function Get-DetailedPermissions {
    param([string]$Path)

    $acl = Get-Acl -Path $Path

    Write-Host "Path: $Path" -ForegroundColor Yellow
    Write-Host "Owner: $($acl.Owner)" -ForegroundColor Green
    Write-Host "Group: $($acl.Group)" -ForegroundColor Green
    Write-Host "`nAccess Rules:" -ForegroundColor Cyan

    foreach ($access in $acl.Access) {
        Write-Host "  Identity: $($access.IdentityReference)" -ForegroundColor White
        Write-Host "  Rights: $($access.FileSystemRights)" -ForegroundColor Gray
        Write-Host "  Type: $($access.AccessControlType)" -ForegroundColor Gray
        Write-Host "  Inherited: $($access.IsInherited)" -ForegroundColor Gray
        Write-Host "  ---"
    }
}

# Usage
Get-DetailedPermissions -Path "C:\Example\Folder"
```

Setting File System Permissions

powershell

```powershell
# Create new access rule
$accessRule = New-Object System.Security.AccessControl.FileSystemAccessRule(
    "DOMAIN\username",          # Identity
    "FullControl",              # Rights
    "ContainerInherit,ObjectInherit", # Inheritance
    "None",                     # Propagation
    "Allow"                     # Access type
)

# Apply the rule
$acl = Get-Acl -Path "C:\Example\Folder"
$acl.SetAccessRule($accessRule)
Set-Acl -Path "C:\Example\Folder" -AclObject $acl

# Grant permissions (simpler method)
function Grant-FilePermission {
    param(
        [string]$Path,
        [string]$Identity,
        [string]$Rights = "ReadAndExecute",
        [switch]$Inherit
    )

    $acl = Get-Acl -Path $Path

    if ($Inherit) {
        $inheritanceFlags = "ContainerInherit,ObjectInherit"
    } else {
        $inheritanceFlags = "None"
    }

    $accessRule = New-Object System.Security.AccessControl.FileSystemAccessRule(
        $Identity, $Rights, $inheritanceFlags, "None", "Allow"
    )

    $acl.SetAccessRule($accessRule)
    Set-Acl -Path $Path -AclObject $acl

    Write-Host "Granted $Rights permissions to $Identity on $Path"
}

# Usage examples
Grant-FilePermission -Path "C:\Shared\Reports" -Identity "DOMAIN\ReportsGroup" -Rights "ReadAndExecute" -Inherit
Grant-FilePermission -Path "C:\Shared\Uploads" -Identity "Users" -Rights "Modify"
```

Removing File System Permissions

```powershell
# Remove specific permission
function Remove-FilePermission {
    param(
        [string]$Path,
        [string]$Identity,
        [string]$Rights = "FullControl"
    )

    $acl = Get-Acl -Path $Path

    $accessRule = New-Object System.Security.AccessControl.FileSystemAccessRule(
        $Identity, $Rights, "ContainerInherit,ObjectInherit", "None", "Allow"
    )

    $acl.RemoveAccessRule($accessRule)
    Set-Acl -Path $Path -AclObject $acl

    Write-Host "Removed $Rights permissions for $Identity from $Path"
}

# Remove all permissions for a user
function Remove-AllUserPermissions {
    param(
        [string]$Path,
        [string]$Identity
    )

    $acl = Get-Acl -Path $Path

    # Remove all access rules for the specified identity
    $acl.Access | Where-Object { $_.IdentityReference -eq $Identity } | ForEach-Object {
        $acl.RemoveAccessRule($_)
    }

    Set-Acl -Path $Path -AclObject $acl
    Write-Host "Removed all permissions for $Identity from $Path"
}
```

Advanced File System Operations

```powershell
# Reset permissions to default
function Reset-FolderPermissions {
    param([string]$Path)

    # Remove explicit permissions and enable inheritance
    $acl = Get-Acl -Path $Path
    $acl.SetAccessRuleProtection($false, $false)
    Set-Acl -Path $Path -AclObject $acl

    Write-Host "Reset permissions and enabled inheritance for $Path"
}

# Copy permissions from one location to another
function Copy-Permissions {
    param(
        [string]$SourcePath,
        [string]$DestinationPath
    )

    $sourceAcl = Get-Acl -Path $SourcePath
    Set-Acl -Path $DestinationPath -AclObject $sourceAcl

    Write-Host "Copied permissions from $SourcePath to $DestinationPath"
}

# Bulk permission changes
function Set-BulkPermissions {
    param(
        [string[]]$Paths,
        [string]$Identity,
        [string]$Rights
    )

    foreach ($path in $Paths) {
        if (Test-Path $path) {
            Grant-FilePermission -Path $path -Identity $Identity -Rights $Rights
        } else {
            Write-Warning "Path not found: $path"
        }
    }
}
```

Part 2: Registry Permissions

Viewing Registry Permissions

```powershell
# Get registry key ACL
$regAcl = Get-Acl -Path "HKLM:\SOFTWARE\MyApp"
$regAcl.Access | Format-Table IdentityReference, RegistryRights, AccessControlType

# Function to analyze registry permissions
function Get-RegistryPermissions {
    param([string]$RegistryPath)

    try {
        $acl = Get-Acl -Path $RegistryPath

        Write-Host "Registry Path: $RegistryPath" -ForegroundColor Yellow
        Write-Host "Owner: $($acl.Owner)" -ForegroundColor Green

        foreach ($access in $acl.Access) {
            [PSCustomObject]@{
                Identity = $access.IdentityReference
                Rights = $access.RegistryRights
                AccessType = $access.AccessControlType
                Inherited = $access.IsInherited
            }
        }
    }
    catch {
        Write-Error "Failed to get permissions for $RegistryPath : $($_.Exception.Message)"
    }
}

# Usage
Get-RegistryPermissions -Path "HKLM:\SOFTWARE"
```

Setting Registry Permissions

powershell

```powershell
# Grant registry permissions
function Grant-RegistryPermission {
    param(
        [string]$RegistryPath,
        [string]$Identity,
        [string]$Rights = "ReadKey"
    )

    try {
        $acl = Get-Acl -Path $RegistryPath

        $accessRule = New-Object System.Security.AccessControl.RegistryAccessRule(
            $Identity, $Rights, "ContainerInherit,ObjectInherit", "None", "Allow"
        )

        $acl.SetAccessRule($accessRule)
        Set-Acl -Path $RegistryPath -AclObject $acl

        Write-Host "Granted $Rights permissions to $Identity on $RegistryPath"
    }
    catch {
        Write-Error "Failed to set registry permissions: $($_.Exception.Message)"
    }
}

# Usage
Grant-RegistryPermission -Path "HKLM:\SOFTWARE\MyApp" -Identity "DOMAIN\AppUsers" -Rights "ReadKey"
```

Part 3: Service Permissions

Viewing Service Permissions

powershell

```powershell
# Get service security descriptor
function Get-ServicePermissions {
    param([string]$ServiceName)

    $service = Get-WmiObject -Class Win32_Service -Filter "Name='$ServiceName'"
    if ($service) {
        $sddl = $service.GetSecurityDescriptor().Descriptor
        Write-Host "Service: $ServiceName"
        Write-Host "Security Descriptor: $sddl"

        # Convert SDDL to readable format (requires additional parsing)
        # This is a simplified version
        return $sddl
    } else {
        Write-Warning "Service '$ServiceName' not found"
    }
}

# More detailed service permissions using sc.exe
function Get-ServiceAcl {
    param([string]$ServiceName)

    $result = & sc.exe sdshow $ServiceName 2>&1
    if ($LASTEXITCODE -eq 0) {
        Write-Host "Service: $ServiceName"
        Write-Host "SDDL: $result"
        return $result
    } else {
        Write-Error "Failed to get ACL for service '$ServiceName': $result"
    }
}
```

Setting Service Permissions

powershell

```
# Set service permissions (requires administrative privileges)
function Set-ServicePermission {
    param(
        [string]$ServiceName,
        [string]$SDDL
    )

    $result = & sc.exe sdset $ServiceName $SDDL 2>&1
    if ($LASTEXITCODE -eq 0) {
        Write-Host "Successfully updated permissions for service '$ServiceName'"
    } else {
        Write-Error "Failed to set permissions for service '$ServiceName': $result"
    }
}
```

Part 4: Active Directory Permissions

Prerequisites for AD Permissions

powershell

```
# Import required modules
Import-Module ActiveDirectory
# For advanced AD operations, you might need:
# Import-Module Microsoft.ActiveDirectory.Management
```

Viewing AD Object Permissions

powershell

```
# Get AD object permissions
function Get-ADObjectPermissions {
    param(
        [string]$Identity,
        [string]$ObjectType = "User"
    )

    try {
        switch ($ObjectType) {
            "User" { $object = Get-ADUser -Identity $Identity }
            "Group" { $object = Get-ADGroup -Identity $Identity }
            "Computer" { $object = Get-ADComputer -Identity $Identity }
            "OU" { $object = Get-ADOrganizationalUnit -Identity $Identity }
        }

        $acl = Get-Acl -Path "AD:\$($object.DistinguishedName)"

        Write-Host "Object: $($object.Name)" -ForegroundColor Yellow
        Write-Host "Distinguished Name: $($object.DistinguishedName)" -ForegroundColor Green

        $acl.Access | Select-Object IdentityReference, ActiveDirectoryRights, AccessControlType, InheritanceType | Format-
    }
    catch {
        Write-Error "Failed to get AD permissions: $($_.Exception.Message)"
    }
}

# Usage
Get-ADObjectPermissions -Identity "john.doe" -ObjectType "User"
```

Delegating AD Permissions

powershell

```powershell
# Delegate permissions on OU
function Delegate-ADPermission {
    param(
        [string]$OUPath,
        [string]$Identity,
        [string]$Rights,
        [string]$ObjectType = "All"
    )

    try {
        $ou = Get-ADOrganizationalUnit -Identity $OUPath
        $acl = Get-Acl -Path "AD:\$($ou.DistinguishedName)"

        $accessRule = New-Object System.DirectoryServices.ActiveDirectoryAccessRule(
            [System.Security.Principal.NTAccount]$Identity,
            [System.DirectoryServices.ActiveDirectoryRights]$Rights,
            [System.Security.AccessControl.AccessControlType]::Allow
        )

        $acl.SetAccessRule($accessRule)
        Set-Acl -Path "AD:\$($ou.DistinguishedName)" -AclObject $acl

        Write-Host "Delegated $Rights permissions to $Identity on $OUPath"
    }
    catch {
        Write-Error "Failed to delegate AD permissions: $($_.Exception.Message)"
    }
}
```

Part 5: Share Permissions

Viewing Share Permissions

powershell

```powershell
# Get share permissions using WMI
function Get-SharePermissions {
    param([string]$ShareName)

    $share = Get-WmiObject -Class Win32_LogicalShareSecuritySetting -Filter "Name='$ShareName'"
    if ($share) {
        $securityDescriptor = $share.GetSecurityDescriptor()

        Write-Host "Share: $ShareName" -ForegroundColor Yellow

        foreach ($ace in $securityDescriptor.Descriptor.DACL) {
            [PSCustomObject]@{
                Trustee = $ace.Trustee.Name
                AccessMask = $ace.AccessMask
                AceType = $ace.AceType
            }
        }
    } else {
        Write-Warning "Share '$ShareName' not found"
    }
}

# Modern approach using Get-SmbShare (Windows 8/Server 2012+)
function Get-ModernSharePermissions {
    param([string]$ShareName)

    try {
        Get-SmbShareAccess -Name $ShareName | Format-Table Name, AccountName, AccessControlType, AccessRight
    }
    catch {
        Write-Error "Failed to get share permissions: $($_.Exception.Message)"
    }
}
```

Setting Share Permissions

powershell

```powershell
# Grant share permissions (modern method)
function Grant-SharePermission {
    param(
        [string]$ShareName,
        [string]$AccountName,
        [string]$AccessRight = "Read"
    )

    try {
        Grant-SmbShareAccess -Name $ShareName -AccountName $AccountName -AccessRight $AccessRight -Force
        Write-Host "Granted $AccessRight permissions to $AccountName on share $ShareName"
    }
    catch {
        Write-Error "Failed to grant share permission: $($_.Exception.Message)"
    }
}

# Remove share permissions
function Remove-SharePermission {
    param(
        [string]$ShareName,
        [string]$AccountName
    )

    try {
        Revoke-SmbShareAccess -Name $ShareName -AccountName $AccountName -Force
        Write-Host "Removed permissions for $AccountName from share $ShareName"
    }
    catch {
        Write-Error "Failed to remove share permission: $($_.Exception.Message)"
    }
}
```

Part 6: Practical Examples and Scripts

Permission Audit Script

powershell

```
function New-PermissionAuditReport {
    param(
        [string[]]$Paths,
        [string]$OutputPath = "PermissionAudit.csv"
    )

    $report = @()

    foreach ($path in $Paths) {
        if (Test-Path $path) {
            try {
                $acl = Get-Acl -Path $path

                foreach ($access in $acl.Access) {
                    $report += [PSCustomObject]@{
                        Path = $path
                        Owner = $acl.Owner
                        Identity = $access.IdentityReference
                        Rights = $access.FileSystemRights
                        AccessType = $access.AccessControlType
                        Inherited = $access.IsInherited
                        InheritanceFlags = $access.InheritanceFlags
                        PropagationFlags = $access.PropagationFlags
                        Timestamp = Get-Date
                    }
                }
            }
            catch {
                Write-Warning "Failed to process $path : $($_.Exception.Message)"
            }
        }
    }

    $report | Export-Csv -Path $OutputPath -NoTypeInformation
    Write-Host "Audit report saved to $OutputPath"
    return $report
}

# Usage
$pathsToAudit = @("C:\Shared", "C:\Data", "C:\Logs")
New-PermissionAuditReport -Paths $pathsToAudit
```

Permission Cleanup Script

powershell

```powershell
function Remove-OrphanedPermissions {
    param(
        [string]$Path,
        [switch]$WhatIf
    )

    $acl = Get-Acl -Path $Path
    $orphanedRules = @()

    foreach ($access in $acl.Access) {
        try {
            # Try to resolve the SID to a name
            $identity = [System.Security.Principal.SecurityIdentifier]$access.IdentityReference
            $account = $identity.Translate([System.Security.Principal.NTAccount])
        }
        catch {
            # If translation fails, it's likely an orphaned SID
            $orphanedRules += $access
            Write-Warning "Found orphaned permission: $($access.IdentityReference)"
        }
    }

    if ($orphanedRules.Count -gt 0) {
        if ($WhatIf) {
            Write-Host "Would remove $($orphanedRules.Count) orphaned permissions from $Path"
        } else {
            foreach ($rule in $orphanedRules) {
                $acl.RemoveAccessRule($rule)
            }
            Set-Acl -Path $Path -AclObject $acl
            Write-Host "Removed $($orphanedRules.Count) orphaned permissions from $Path"
        }
    } else {
        Write-Host "No orphaned permissions found in $Path"
    }
}
```

Bulk Permission Management

powershell

```powershell
function Set-StandardFolderPermissions {
    param(
        [string]$BasePath,
        [hashtable]$PermissionSet
    )

    <#
    Example PermissionSet:
    @{
        "DOMAIN\Administrators" = "FullControl"
        "DOMAIN\Users" = "ReadAndExecute"
        "DOMAIN\PowerUsers" = "Modify"
    }
    #>

    if (-not (Test-Path $BasePath)) {
        Write-Error "Path does not exist: $BasePath"
        return
    }

    try {
        # Get current ACL and remove inheritance
        $acl = Get-Acl -Path $BasePath
        $acl.SetAccessRuleProtection($true, $false)

        # Clear existing permissions
        $acl.Access | ForEach-Object { $acl.RemoveAccessRule($_) }

        # Apply new permissions
        foreach ($identity in $PermissionSet.Keys) {
            $rights = $PermissionSet[$identity]

            $accessRule = New-Object System.Security.AccessControl.FileSystemAccessRule(
                $identity, $rights, "ContainerInherit,ObjectInherit", "None", "Allow"
            )

            $acl.SetAccessRule($accessRule)
            Write-Host "Added $rights permissions for $identity"
        }

        # Apply the ACL
        Set-Acl -Path $BasePath -AclObject $acl
        Write-Host "Successfully applied standard permissions to $BasePath" -ForegroundColor Green
    }
```

```
        catch {
            Write-Error "Failed to set permissions: $($_.Exception.Message)"
        }
    }

    # Usage example
    $standardPermissions = @{
        "BUILTIN\Administrators" = "FullControl"
        "DOMAIN\Domain Admins" = "FullControl"
        "DOMAIN\FileServer Users" = "ReadAndExecute"
        "DOMAIN\FileServer Editors" = "Modify"
    }

    Set-StandardFolderPermissions -BasePath "C:\Shared\Documents" -PermissionSet $standardPermissions
```

Part 7: Advanced Topics

Working with SIDs and Security Principals

powershell

```powershell
# Convert between usernames and SIDs
function Convert-NameToSid {
    param([string]$AccountName)

    try {
        $account = New-Object System.Security.Principal.NTAccount($AccountName)
        $sid = $account.Translate([System.Security.Principal.SecurityIdentifier])
        return $sid.Value
    }
    catch {
        Write-Error "Failed to convert $AccountName to SID: $($_.Exception.Message)"
    }
}

function Convert-SidToName {
    param([string]$SID)

    try {
        $sid = New-Object System.Security.Principal.SecurityIdentifier($SID)
        $account = $sid.Translate([System.Security.Principal.NTAccount])
        return $account.Value
    }
    catch {
        Write-Error "Failed to convert $SID to name: $($_.Exception.Message)"
    }
}

# Usage
$sid = Convert-NameToSid -AccountName "DOMAIN\username"
$name = Convert-SidToName -SID $sid
```

Permission Inheritance Management

powershell

```powershell
function Set-InheritanceSettings {
    param(
        [string]$Path,
        [bool]$EnableInheritance = $true,
        [bool]$PreserveInheritedRules = $true
    )

    $acl = Get-Acl -Path $Path
    $acl.SetAccessRuleProtection(-not $EnableInheritance, $PreserveInheritedRules)
    Set-Acl -Path $Path -AclObject $acl

    $status = if ($EnableInheritance) { "enabled" } else { "disabled" }
    Write-Host "Inheritance $status for $Path"
}

# Disable inheritance and keep existing permissions
Set-InheritanceSettings -Path "C:\Secure\Folder" -EnableInheritance $false -PreserveInheritedRules $true

# Enable inheritance
Set-InheritanceSettings -Path "C:\Secure\Folder" -EnableInheritance $true
```

Part 8: Best Practices and Security

Security Best Practices

powershell

```powershell
# Function to check for common security issues
function Test-PermissionSecurity {
    param([string]$Path)

    $issues = @()
    $acl = Get-Acl -Path $Path

    # Check for Everyone group with excessive permissions
    $everyoneRules = $acl.Access | Where-Object { $_.IdentityReference -eq "Everyone" }
    foreach ($rule in $everyoneRules) {
        if ($rule.FileSystemRights -match "FullControl|Modify|Write") {
            $issues += "Everyone group has $($rule.FileSystemRights) permissions"
        }
    }

    # Check for Users group with write permissions
    $usersRules = $acl.Access | Where-Object { $_.IdentityReference -eq "BUILTIN\Users" }
    foreach ($rule in $usersRules) {
        if ($rule.FileSystemRights -match "FullControl|Modify|Write") {
            $issues += "Users group has $($rule.FileSystemRights) permissions"
        }
    }

    # Check for non-inherited administrative permissions
    $adminRules = $acl.Access | Where-Object {
        $_.IdentityReference -match "Administrator" -and -not $_.IsInherited
    }
    if ($adminRules.Count -eq 0) {
        $issues += "No explicit administrator permissions found"
    }

    return $issues
}

# Usage
$securityIssues = Test-PermissionSecurity -Path "C:\Sensitive\Data"
if ($securityIssues) {
    Write-Warning "Security issues found:"
    $securityIssues | ForEach-Object { Write-Host "  - $_" -ForegroundColor Red }
}
```

Error Handling and Logging

powershell

```powershell
function Set-PermissionWithLogging {
    param(
        [string]$Path,
        [string]$Identity,
        [string]$Rights,
        [string]$LogPath = "PermissionChanges.log"
    )

    $timestamp = Get-Date -Format "yyyy-MM-dd HH:mm:ss"
    $logEntry = "$timestamp - Attempting to grant $Rights to $Identity on $Path"

    try {
        # Log the attempt
        Add-Content -Path $LogPath -Value $logEntry

        # Apply the permission
        Grant-FilePermission -Path $Path -Identity $Identity -Rights $Rights

        # Log success
        $successEntry = "$timestamp - SUCCESS: Granted $Rights to $Identity on $Path"
        Add-Content -Path $LogPath -Value $successEntry

        Write-Host "Permission granted successfully" -ForegroundColor Green
    }
    catch {
        # Log the error
        $errorEntry = "$timestamp - ERROR: Failed to grant permission - $($_.Exception.Message)"
        Add-Content -Path $LogPath -Value $errorEntry

        Write-Error "Failed to set permission: $($_.Exception.Message)"
        throw
    }
}
```

Troubleshooting Common Issues

Access Denied Errors

```powershell
# Check if current user has permission to modify ACL
function Test-AclModifyPermission {
    param([string]$Path)

    try {
        $acl = Get-Acl -Path $Path
        $testRule = New-Object System.Security.AccessControl.FileSystemAccessRule(
            [System.Security.Principal.WindowsIdentity]::GetCurrent().Name,
            "ReadPermissions", "None", "None", "Allow"
        )

        # Try to add and immediately remove a test rule
        $acl.SetAccessRule($testRule)
        $acl.RemoveAccessRule($testRule)

        return $true
    }
    catch {
        return $false
    }
}

# Take ownership if needed
function Take-Ownership {
    param([string]$Path)

    try {
        $acl = Get-Acl -Path $Path
        $currentUser = [System.Security.Principal.WindowsIdentity]::GetCurrent().Name
        $acl.SetOwner([System.Security.Principal.NTAccount]$currentUser)
        Set-Acl -Path $Path -AclObject $acl

        Write-Host "Took ownership of $Path"
    }
    catch {
        Write-Error "Failed to take ownership: $($_.Exception.Message)"
    }
}
```

Module and Cmdlet Availability

powershell

```powershell
# Check for required modules and cmdlets
function Test-PermissionModules {
    $requiredCmdlets = @(
        "Get-Acl",
        "Set-Acl",
        "Get-SmbShare",
        "Grant-SmbShareAccess"
    )

    foreach ($cmdlet in $requiredCmdlets) {
        if (Get-Command $cmdlet -ErrorAction SilentlyContinue) {
            Write-Host "✓ $cmdlet available" -ForegroundColor Green
        } else {
            Write-Host "✗ $cmdlet not available" -ForegroundColor Red
        }
    }

    # Check for Active Directory module
    if (Get-Module -ListAvailable -Name ActiveDirectory) {
        Write-Host "✓ Active Directory module available" -ForegroundColor Green
    } else {
        Write-Host "✗ Active Directory module not available" -ForegroundColor Red
        Write-Host "  Install with: Install-WindowsFeature RSAT-AD-PowerShell"
    }
}

Test-PermissionModules
```

This tutorial provides a comprehensive foundation for managing permissions in PowerShell. Always test permission changes in a development environment first, maintain proper backups, and document all permission modifications for audit purposes.