PowerShell Identities Tutorial: From Linux IDs to Windows

Introduction

If you're familiar with Linux user and group IDs (`uid`, `gid`), this tutorial will help you understand how identities work in Windows PowerShell. While the concepts are similar, the implementation and commands differ significantly.

Quick Comparison: Linux vs PowerShell

Concept	Linux	PowerShell
Current user	`whoami`	`whoami` or `$env:USERNAME`
User ID	`id -u`	`[System.Security.Principal.WindowsIdentity]::GetCurrent()`
Group membership	`id -G`	`whoami /groups`
All users	`cat /etc/passwd`	`Get-LocalUser`
All groups	`cat /etc/group`	`Get-LocalGroup`

Part 1: Understanding Windows Identity Basics

Security Identifiers (SIDs)

Windows uses Security Identifiers (SIDs) instead of numeric UIDs. SIDs are unique strings like `S-1-5-21-1234567890-987654321-1122334455-1001`.

Windows Identity Types

- **Local accounts**: Created on the local machine
- **Domain accounts**: Created in Active Directory
- **Built-in accounts**: System, Administrator, Guest, etc.
- **Service accounts**: For running services

Part 2: Getting Current User Information

Basic User Information

powershell

```powershell
# Get current username (like 'whoami' in Linux)
whoami
$env:USERNAME

# Get detailed current user info
[System.Security.Principal.WindowsIdentity]::GetCurrent()

# Get current user's SID
[System.Security.Principal.WindowsIdentity]::GetCurrent().User
```

Current User's Groups

powershell

```powershell
# List all groups for current user (similar to 'id -G' in Linux)
whoami /groups

# Get group membership using .NET
[System.Security.Principal.WindowsIdentity]::GetCurrent().Groups

# More readable group list
([System.Security.Principal.WindowsIdentity]::GetCurrent()).Groups | ForEach-Object {
    $_.Translate([System.Security.Principal.NTAccount])
}
```

Part 3: Working with Local Users

Listing Users

powershell

```powershell
# List all local users (similar to 'cat /etc/passwd')
Get-LocalUser

# Get specific user details
Get-LocalUser -Name "Administrator"

# List enabled users only
Get-LocalUser | Where-Object {$_.Enabled -eq $true}
```

User Properties

powershell

```powershell
# Get detailed user information
Get-LocalUser -Name "YourUsername" | Format-List *

# Common properties:
# - Name: Username
# - SID: Security Identifier
# - Enabled: Account status
# - LastLogon: Last login time
# - PasswordLastSet: When password was changed
```

Creating and Managing Users

powershell

```powershell
# Create a new local user
$Password = ConvertTo-SecureString "P@ssw0rd123" -AsPlainText -Force
New-LocalUser -Name "testuser" -Password $Password -Description "Test account"

# Enable/disable user
Enable-LocalUser -Name "testuser"
Disable-LocalUser -Name "testuser"

# Remove user
Remove-LocalUser -Name "testuser"
```

Part 4: Working with Groups

Listing Groups

powershell

```powershell
# List all local groups (similar to 'cat /etc/group')
Get-LocalGroup

# Get specific group
Get-LocalGroup -Name "Administrators"

# List group members
Get-LocalGroupMember -Group "Administrators"
```

Managing Group Membership

powershell

```powershell
# Add user to group (similar to 'usermod -aG group user' in Linux)
Add-LocalGroupMember -Group "Users" -Member "testuser"

# Remove user from group
Remove-LocalGroupMember -Group "Users" -Member "testuser"

# Check if user is in specific group
Get-LocalGroupMember -Group "Administrators" | Where-Object {$_.Name -like "*$env:USERNAME*"}
```

Part 5: Advanced Identity Operations

Working with SIDs

powershell

```powershell
# Convert username to SID
$user = New-Object System.Security.Principal.NTAccount("YourUsername")
$sid = $user.Translate([System.Security.Principal.SecurityIdentifier])
$sid.Value

# Convert SID back to username
$sidObject = New-Object System.Security.Principal.SecurityIdentifier("S-1-5-21-...")
$user = $sidObject.Translate([System.Security.Principal.NTAccount])
$user.Value
```

Checking Privileges

powershell

```powershell
# Check if running as administrator (similar to checking if uid=0 in Linux)
([Security.Principal.WindowsPrincipal] [Security.Principal.WindowsIdentity]::GetCurrent()).IsInRole([Security.Principal.Winc

# Get current user's privileges
whoami /priv
```

Part 6: Domain Operations (if applicable)

Domain User Information

powershell

```powershell
# Get domain users (requires appropriate permissions)
Get-ADUser -Filter * | Select-Object Name, SamAccountName, Enabled

# Get current domain
$env:USERDOMAIN

# Get domain groups
Get-ADGroup -Filter *

# Check domain group membership
Get-ADGroupMember -Identity "Domain Admins"
```

Part 7: Practical Examples

Example 1: User Audit Script

powershell

```powershell
# Create a user audit report
$users = Get-LocalUser
foreach ($user in $users) {
    Write-Host "User: $($user.Name)"
    Write-Host "  SID: $($user.SID)"
    Write-Host "  Enabled: $($user.Enabled)"
    Write-Host "  Last Logon: $($user.LastLogon)"
    Write-Host "  Groups:"

    # Get user's groups (this is complex for local users)
    $userSid = $user.SID
    $groups = Get-LocalGroup | Where-Object {
        (Get-LocalGroupMember -Group $_.Name -ErrorAction SilentlyContinue) |
        Where-Object {$_.SID -eq $userSid}
    }

    foreach ($group in $groups) {
        Write-Host "    - $($group.Name)"
    }
    Write-Host ""
}
```

Example 2: Find Users in Administrative Groups

powershell

```powershell
# Find all users with administrative privileges
$adminGroups = @("Administrators", "Power Users", "Backup Operators")

foreach ($group in $adminGroups) {
    Write-Host "=== $group ==="
    try {
        Get-LocalGroupMember -Group $group | ForEach-Object {
            Write-Host "  $($_.Name) ($($_.ObjectClass))"
        }
    }
    catch {
        Write-Host "  Group not found or access denied"
    }
    Write-Host ""
}
```

Part 8: Linux vs PowerShell Command Reference

User Information Commands

Task	Linux	PowerShell
Current user	`whoami`	`whoami` or `$env:USERNAME`
User ID	`id -u`	`(Get-LocalUser $env:USERNAME).SID`
User groups	`groups`	`whoami /groups`
All user info	`id`	`whoami /all`
Switch user	`su username`	`Start-Process powershell -Credential (Get-Credential)`

User Management Commands

Task	Linux	PowerShell
Add user	`useradd username`	`New-LocalUser -Name "username"`
Delete user	`userdel username`	`Remove-LocalUser -Name "username"`
Modify user	`usermod`	`Set-LocalUser`
Lock user	`usermod -L username`	`Disable-LocalUser -Name "username"`
Unlock user	`usermod -U username`	`Enable-LocalUser -Name "username"`

Group Management Commands

Task	Linux	PowerShell
Add to group	`usermod -aG group user`	`Add-LocalGroupMember -Group "group" -Member "user"`
Create group	`groupadd groupname`	`New-LocalGroup -Name "groupname"`
Delete group	`groupdel groupname`	`Remove-LocalGroup -Name "groupname"`
List groups	`cat /etc/group`	`Get-LocalGroup`

Key Differences to Remember

1. **SIDs vs UIDs**: Windows uses string-based SIDs instead of numeric UIDs
2. **Built-in accounts**: Windows has more built-in system accounts
3. **Domain integration**: Windows systems can be joined to Active Directory domains
4. **Privilege model**: Windows uses a different privilege escalation model (UAC)
5. **Group nesting**: Windows supports nested groups more extensively

Best Practices

1. Always use `-WhatIf` parameter when making changes to test first
2. Be careful with domain operations - they affect the entire domain
3. Use proper error handling when working with user/group cmdlets
4. Store passwords securely using `ConvertTo-SecureString`
5. Regular audits of user accounts and group memberships

Troubleshooting Common Issues

Access Denied Errors

```powershell
# Run PowerShell as Administrator for local user management
# For domain operations, ensure you have appropriate AD permissions
```

User Not Found

```powershell
# Check if user exists in local vs domain
Get-LocalUser -Name "username" -ErrorAction SilentlyContinue
Get-ADUser -Identity "username" -ErrorAction SilentlyContinue
```

This tutorial provides a foundation for understanding PowerShell identities coming from a Linux background. The concepts are similar, but the implementation details and available tools differ significantly between the two systems.