QRG

Table Of Contents

Computers

1. Get-ADComputer

The Get-ADComputer cmdlet gets a computer or performs a search to retrieve multiple computers.

The Identity parameter specifies the Active Directory computer to retrieve. You can identify a computer by its distinguished name, GUID, security identifier (SID) or Security Accounts Manager (SAM) account name. You can also set the parameter to a computer object variable, such as $ or pass a computer object through the pipeline to the Identity parameter.

To search for and retrieve more than one computer, use the Filter or LDAPFilter parameters. The Filter parameter uses the PowerShell Expression Language to write query strings for Active Directory. PowerShell Expression Language syntax provides rich type conversion support for value types received by the Filter parameter. For more information about the Filter parameter syntax, type Get-Help about_ActiveDirectory_Filter. If you have existing Lightweight Directory Access Protocol (LDAP) query strings, you can use the LDAPFilter parameter.

This cmdlet retrieves a default set of computer object properties. To retrieve additional properties use the Properties parameter. For more information about the how to determine the properties for computer objects, see the Properties parameter description.

1.1. Get a Single Computer properties

```
Get-ADComputer -Identity AustonMatthews-Laptop
```

This command gets all of the properties of the user with the SAM account name austonMatthews.

```
Get-ADComputer -Identity AustonMatthews-Laptop -Properties *
```

This command gets specified properties from the SAM account austonMatthews.

```
Get-ADComputer -Identity AustonMatthews-Laptop -Properties Name,Description,SID
```

This command retrieves specified properties and returns them in a table.

```
Get-ADComputer -Identity austonMatthews -Properties Name,Description,SID | Format-
Table Name,Desc,ID
```

1.2. Get Multiple Users' properties

Returns multiple Users based on Team property.

```
Get-ADComputer -Filter {Team -like "Toron*"} -Properties Name,Description,SID |
Format-Table Name,Desc,ID
```

---

2. New-ADComputer

The New-ADComputer cmdlet creates a new Active Directory computer object. This cmdlet does not join a computer to a domain. You can set commonly used computer property values by using the cmdlet parameters. Property values that are not associated with cmdlet parameters can be modified by using the OtherAttributes parameter.

You can use this cmdlet to provision a computer account before the computer is added to the domain. These pre-created computer objects can be used with offline domain join, unsecure domain join, and RODC domain join scenarios.

The Path parameter specifies the container or organizational unit (OU) for the new computer. When you do not specify the Path parameter, the cmdlet creates a computer account in the default container for computer objects in the domain.

```
New-ADComputer -Name "Auston-SRV2" -SamAccountName "Auston-SRV2" -Path
"OU=ApplicationServers,OU=ComputerAccounts,OU=Managed,DC=USERS02,DC=COM"
```

---

3. Remove-ADComputer

The Remove-ADComputer cmdlet removes an Active Directory computer.

The Identity parameter specifies the Active Directory computer to remove. You can identify a computer by its distinguished name, GUID, security identifier (SID), or Security Accounts Manager (SAM) account name. You can also set the Identity parameter to a computer object variable, such as $, or you can pass a computer object through the pipeline to the Identity parameter. For example, you can use the Get-ADComputer cmdlet to retrieve a computer object and then pass the object through the pipeline to the Remove-ADComputer cmdlet.

```
Remove-ADComputer -Identity "AustonMatthews-Laptop"
```

3.1. Remove Computer Recursively

This command removes a computer and all leaf objects that are located underneath it in the directory. Note that only a few computer objects create child objects, such as servers running the Clustering service. This example can be useful for removing those objects and any child objects owned by and associated with them.

```
Get-ADComputer -Identity "AustonMatthews-SRV4" | Remove-ADObject -Recursive
```

---

4. Set-ADComputer

The Set-ADComputer cmdlet modifies the properties of an Active Directory computer object. You can modify commonly used property values by using the cmdlet parameters. Property values that are not associated with

cmdlet parameters can be modified by using the Add, Replace, Clear, and Remove parameters.

The Identity parameter specifies the Active Directory computer to modify. You can identify a computer by its distinguished name, GUID, security identifier (SID) or Security Accounts Manager (SAM) account name. You can also set the Identity parameter to an object variable such as $, or you can pass an object through the pipeline to the Identity parameter. For example, you can use the Get-ADComputer cmdlet to retrieve a computer object and then pass the object through the pipeline to Set-ADComputer.

The Instance parameter provides a way to update a computer by applying the changes made to a copy of the computer object. When you set the Instance parameter to a copy of an Active Directory computer object that has been modified, the Set-ADComputer cmdlet makes the same changes to the original computer object. To get a copy of the object to modify, use the Get-ADComputer object. When you specify the Instance parameter you should not pass the Identity parameter. For more information about the Instance parameter, see the Instance parameter description.

4.1. Set property

```
Set-ADComputer -Identity "austonMatthews-Laptop" -Location "ScotiaBankArena-
Lockerroom"
```

4.2. Batch Set properties

This command gets all the computers in the directory that are located in the OU=Players,OU=UserAccounts,DC=LEAFS,DC=COM organizational unit. This will change the description property of all computers in that OU to "Player Laptop".

```
Get-ADComputer * -SearchBase 'OU=Players,OU=UserAccounts,DC=LEAFS,DC=COM' | Set-
ADComputer -Description "Player Laptop"
```

4.3. Search and Set properties

```
Get-ADComputer -Filter {Name -like "austonMatthews*"} | Set-ADComputer -
Description "Auston Matthew's Laptop"
```

DNS

5. Record Types

Record	Desc
A	FQDN => IPv4
AAAA	FQDN => IPv6
NS	DNS Servers
MX	Mail Exchange
CNAME	Canonical Name. FQDN => Alias

Record	Desc
PTR	Pointer Record. IP => FQDN
SRV	Server. In AD used to point the user to the local domain controller.
SOA	Start of authority. Provides general information about the DNS zone.

---

6. Add-DnsServerPrimaryZone

The Add-DnsServerPrimaryZone cmdlet adds a specified primary zone on a Domain Name System (DNS) server.

You can add an Active Directory-integrated forward lookup zone, an Active Directory-integrated reverse lookup zone, a file-backed forward lookup zone, or a file-backed reverse lookup zone.

6.1. Create an AD integrated forward lookup (FQDN => IP) zone.

```
Add-DnsServerPrimaryZone -Name "DNS-TWO.geoff.com" -ReplicationScope "Forest" -
PassThru
```

6.2. Create a file backed forward lookup (FQDN => IP) zone.

```
Add-DnsServerPrimaryZone -Name "DNS-TWO.geoff.com" -ZoneFile "DNS-TWO-
BACKUP.geoff.com.dns"
```

6.3. Create an AD integrated reverse lookup (IP => FQDN) zone.

```
Add-DnsServerPrimaryZone -NetworkID "10.1.0.0/24" -ReplicationScope "Forest"
```

6.4. Create a file backed reverse lookup (IP => FQDN) zone.

```
Add-DnsServerPrimaryZone -NetworkID 10.3.0.0/24 -ZoneFile "DNS-
BACKUP.evilcorp.com.dns"
```

---

7. Add-DnsServerResourceA

The Add-DnsServerResourceRecordA cmdlet adds a host address (A) record to a Domain Name System (DNS) zone. An A record specifies an IPv4 address.

```
Add-DnsServerResourceRecordA -Name "blog" -ZoneName "geoff.com" -IPv4Address
"192.168.0.12"
```

---

8. Add-DnsServerZoneDelegation

The Add-DnsServerZoneDelegation cmdlet adds a zone delegation to a Domain Name System (DNS) zone. For instance, you can add a child domain called west01 to your top level domain, contoso.com, and specify a DNS server for that delegated domain.

```
Add-DnsServerZoneDelegation -Name "evilcorp.com" -ChildZoneName "blog" -NameServer "blog.evilcorp.com" -IPAddress 172.23.90.136 -PassThru -Verbose
```

---

9. Get-DnsServerForwarder

The Get-DnsServerForwarder cmdlet gets configuration settings on a DNS server. A forwarder is a Domain Name System (DNS) server on a network that is used to forward DNS queries for external DNS names to DNS servers outside that network.

```
Get-DnsServerForwarder
```

---

10. Get-DnsServerResourceRecord

The Get-DnsServerResourceRecord cmdlet gets the following information for a specified resource record from a Domain Name System (DNS) zone:

- HostName
- RecordType
- RecordClass
- TimeToLive
- Timestamp
- RecordData

```
Get-DnsServerResourceRecord -ZoneName "geoff.com" -RRType "A"
```

```
Get-DnsServerResourceRecord -ZoneName "geoff.com" -RRType "CNAME"
```

The `-ExpandedProperty RecordData` flag can be set for more detailed output.

```
Get-DnsServerResourceRecord -ZoneName "geoff.com" -RRType "SRV" | Select-Object -ExpandedProperty RecordData
```

---

11. Remove-DnsServerResourceRecord

The Remove-DnsServerResourceRecord cmdlet removes resource record objects from a Domain Name System (DNS) zone.

You can either use the Get-DnsServerResourceRecord cmdlet to specify an object, or you can specify the RRtype, Name and RecordData of the resource record you want to remove. If you specify an RRtype or name and there are multiple resource records, you can specify the RecordData to delete a specific record. If you do not specify RecordData, the cmdlet deletes all records that match RRtype and Name for the specified zone.

```
Remove-DnsServerResourceRecord -ZoneName "geoff.com" -RRType "A" -Name "blog"
```

---

12. Set-DnsServerSecondaryZone

The Set-DnsServerSecondaryZone cmdlet changes settings for an existing secondary zone on a Domain Name System (DNS) server.

```
Set-DnsServerSecondaryZone "DNS-SECONDARY.geoff.com" -MasterServers
172.23.90.124,2001:4898:7020:f100:458f:e6a2:fcaf:698c -PassThru
```

FSMO

13. Schema Operations Master

Shows which domain controller holds the role of Schema Operations Master. The Schema Master is the only role that can update the AD Schema.

Limit one per forest.

This requires a user account on the DC that is a member of the Schema Admin Group.

```
Get-ADForest | select SchemaMaster
```

---

14. Domain-Naming Operations Master

Shows which domain controller holds the role of DomainNamingMaster.

This role is the only one that can add or remove domains.

Limit one per forest.

```
Get-ADForest | select DomainNamingMaster
```

---

15. Primary Domain Controller (PDC) Emulator Operations Master

Role in charge of syncing time, password change replications, locking accounts because of failed login, and stores a copy of the Group Policy Object.

Limit one per domain.

```
Get-ADDomain | select PDCEmulator
```

---

16. Relative ID (RID) Operations Master Role

The RID value is used to process of SID (unique ID in a domain) creation. It's a pool of IDs.

When a domain has multiple DCs, each DC is issued an initial pool of 500 RIDs. When 250 remain, the RID Role owner issues the DC sends another block.

Limit one per domain.

```
Get-ADDomain | select RIDMaster
```

---

17. Infastructure Operations Master

Replicates SID and Distinguished Name (DN) values changes to cross-domains, so that when a user moves domains, they have access to appropriate resources.

This role checks the SID and DN values against the global catalog. If the values are different, it updates the catalog and replicates the changes to other DCs.

Limit one per domain.

```
Get-ADDomain | select InfastructureMaster
```

---

18. Query to see the FSMO Roles

Displays which servers the roles reside on.

```
netdom query fsmo
```

---

19. Move-ADDirectoryServerOperationsMasterRole

The Move-ADDirectoryServerOperationMasterRole cmdlet moves one or more operation master roles to a directory server. You can move operation master roles to a directory server in a different domain if the credentials are the same in both domains.

The Identity parameter specifies the directory server that receives the roles. You can specify a directory server object by one of the following values:

```
Name of the server object (name)
The distinguished name of the NTDS Settings object
The distinguished name of the server object that represents the directory server
GUID (objectGUID) of server object under the configuration partition
GUID (objectGUID) of NTDS settings object under the configuration partition
```

For Active Directory Lightweight Directory Services (AD LDS) instances the syntax for the server object name is $. The following is an example of this syntax:

asia-w7-vm4$instance1

When you type this value in Windows PowerShell, you must use the backtick (`) as an escape character for the dollar sign ($). Therefore, for this example, you would type the following:

asia-w7-vm4`$instance1

You can also set the parameter to a directory server object variable, such as $.

The Move-ADDirectoryServerOperationMasterRole cmdlet provides two options for moving operation master roles:

Role transfer, which involves transferring roles to be moved by running the cmdlet using the Identity parameter to specify the current role holder and the OperationMasterRole parameter to specify the roles for

transfer. This is the recommended option.

Operation roles include PDCEmulator, RIDMaster, InfrastructureMaster, SchemaMaster, or DomainNamingMaster. To specify more than one role, use a comma-separated list.

Role seizure, which involves seizing roles you previously attempted to transfer by running the cmdlet a second time using the same parameters as the transfer operation, and adding the Force parameter. The Force parameter must be used as a switch to indicate that seizure, instead of transfer, of operation master roles is being performed. This operation still attempts graceful transfer first, then seizes if transfer is not possible.

Unlike using Ntdsutil.exe to move operation master roles, the Move-ADDirectoryServerOperationMasterRole cmdlet can be remotely executed from any domain joined computer where the Active Directory module for Windows PowerShell administration module is installed and available for use. This can make the process of moving roles simpler and easier to centrally administer as each of the two command operations required can be run remotely and do not have to be locally executed at each of the corresponding role holders involved in the movement of the roles, for instance, role transfer only allowed at the old role holder, role seizure only allowed at the new role holder.

19.1. Transfer roles to a specific domain controller

```
Move-ADDirectoryServerOperationMasterRole -Identity USER04-DC1 -
OperationMasterRole
SchemaMaster,DomainNamingMaster,PDCEmulator,RIDMaster,InfrastructureMaster
```

19.2. Seize roles using -Force

```
Move-ADDirectoryServerOperationMasterRole -Identity USER04-DC1 -
OperationMasterRole RIDMaster,InfrastructureMaster,DomainNamingMaster -Force
```

GPO

---

20. Get-GPUInheritance

The Get-GPInheritance cmdlet gets information about Group Policy inheritance for a specified domain or organizational unit (OU).

This information includes the following:

- A list of GPOs that are linked directly to the location (the GpoLinks property).
- A list of GPOs that are applied to the location when Group Policy is processed on a client (the InheritedGpoLinks property).
- Whether inheritance is blocked for the location (the GpoInheritanceBlocked property).

The InheritedGpoLinks property contains a list of the GPOs are applied to the OU or domain when Group Policy is processed on a client. The GPOs are listed according to the order of precedence with which they are applied. This list includes (in the following order):

- Inherited GPOs that are linked, enabled, and enforced at higher levels of the Group Policy hierarchy (for example, a site).

- GPOs that are linked and enabled directly at the specified location.
- If inheritance is not blocked for the specified location, inherited GPOs that are linked and enabled -- but not enforced -- at higher levels of the Group Policy hierarchy.

```
Get-GPInheritance -Target "ou=MyOU,dc=contoso,dc=com"
```

---

21. Get-GPO

The Get-GPO cmdlet gets one Group Policy Object (GPO) or all the GPOs in a domain. You can specify a GPO by its display name or by its globally unique identifier (GUID) to get a single GPO, or you can get all the GPOs in the domain through the All parameter.

This cmdlet returns one or more objects that represent the requested GPOs. By default, properties of the requested GPOs are printed to the display; however, you can also pipe the output of the Get-GPO cmdlet to other Group Policy cmdlets.

```
Get-GPO -Name "Hockey Rules"
```

---

22. New-GPLink

The New-GPLink cmdlet links a GPO to a site, domain, or organizational unit (OU). By default, the link is enabled, which means that the settings of the GPO are applied at the level of the target Active Directory container according to the rules of inheritance and precedence when Group Policy is processed.

You can specify the GPO by either its display name or its GUID; or the GPO can be piped into the cmdlet. You specify the site, domain, or organizational unit (OU) to link to by its Lightweight Directory Access Protocol (LDAP) distinguished name. You can use other parameters to specify whether the link is enabled, whether the link is enforced, and the order in which it is applied at the site, domain, or OU.

```
New-GPLink -Name GPO_NAME -Target "ou=MyOU,dc=contoso,dc=com"
```

22.1. Create GPO and Link it

```
New-GPO -Name "MyGPO" | New-GPLink -Target "ou=MyOU,dc=contoso,dc=com" -
LinkEnabled Yes
```

---

23. New-GPO

The New-GPO cmdlet creates a GPO with a specified name. By default, the newly created GPO is not linked to a site, domain, or organizational unit (OU).

You can use this cmdlet to create a GPO that is based on a starter GPO by specifying the GUID or the display name of the Starter GPO, or by piping a StarterGpo object into the cmdlet.

The cmdlet returns a GPO object, which represents the created GPO that you can pipe to other Group Policy cmdlets.

```
New-GPO -Name NEW_GPO_NAME
```

---

24. Remove-GPLink

The Remove-GPLink cmdlet removes the link between a Group Policy Object (GPO) and a specified site, domain, or OU. This cmdlet does not delete the actual GPO or any other links between the specified GPO and other sites, domains, or OUs.

```
Remove-GPLink -Name "MyGPO" -Target "OU=MyOU,dc=contoso,dc=com"
```

---

25. Remove-GPO

The Remove-GPO cmdlet removes the Group Policy Object (GPO) container and data from the directory service and the system volume folder (SysVol).

```
Remove-GPO -Name GPO_TO_REMOVE
```

---

26. Set-GPInheritance

The Set-GPInheritance cmdlet blocks or unblocks inheritance for a specified domain or organizational unit (OU).

GPOs are applied according to the Group Policy hierarchy in the following order: local GPO, GPOs linked to the site, GPOs linked to the domain, GPOs linked to OUs. By default, an Active Directory container inherits settings from GPOs that are applied at the next higher level in the hierarchy. Blocking inheritance prevents the settings in GPOs that are linked to higher-level sites, domains, or organizational units from being automatically inherited by the specified domain or OU, unless the link for a GPO is enforced.

You use the Target parameter to specify the Lightweight Directory Access Protocol (LDAP) distinguished name of the domain or OU, and use the IsBlocked parameter to specify whether to block or unblock inheritance.

26.1. Block inheritance in a domain

```
Set-GPInheritance -Target "ou=MyOU,dc=contoso,dc=com" -IsBlocked Yes
```

26.2. Unblock inheritance in a domain

```
Set-GPInheritance -Target "dc=northwest, dc=contoso, dc=com" -IsBlocked No
```

26.3. Unblock inheritance for a particular OU

```
Set-GPInheritance -Target "ou=MyOU,dc=contoso,dc=com" -IsBlocked No
```

---

27. Set-GPLink

The Set-GPLink cmdlet sets the properties of a Group Policy Object (GPO) link.

You can set the following properties:

- Enabled. If the GPO link is enabled, the settings of the GPO are applied when Group Policy is processed for the site, domain or OU.

- Enforced. If the GPO link is enforced, it cannot be blocked at a lower-level (in the Group Policy processing hierarchy) container.
- Order. The order specifies the precedence that the settings of the GPO take over conflicting settings in other GPOs that are linked, and enabled, to the same site, domain, or OU.

27.1. Enable the link between a GPO and OU

```
Set-GPLink -Name TestGPO -Target "ou=MyOU,dc=contoso,dc=com" -LinkEnabled Yes
```

Groups

28. Add-ADGroupMember

The Add-ADGroupMember cmdlet adds one or more users, groups, service accounts, or computers as new members of an Active Directory group.

The Identity parameter specifies the Active Directory group that receives the new members. You can identify a group by its distinguished name, GUID, security identifier, or Security Account Manager (SAM) account name. You can also specify group object variable, such as $, or pass a group object through the pipeline to the Identity parameter. For example, you can use the Get-ADGroup cmdlet to get a group object and then pass the object through the pipeline to the Add-ADGroupMember cmdlet.

The Members parameter specifies the new members to add to a group. You can identify a new member by its distinguished name, GUID, security identifier, or SAM account name. You can also specify user, computer, and group object variables, such as $. If you are specifying more than one new member, use a comma-separated list. You cannot pass user, computer, or group objects through the pipeline to this cmdlet. To add user, computer, or group objects to a group by using the pipeline, use the Add-ADPrincipalGroupMembership cmdlet.

```
Add-ADGroupMember -Identity playersGroup -Members austonMatthews,morganReilly
```

---

29. Get-ADGroup

The Get-ADGroup cmdlet gets a group or performs a search to retrieve multiple groups from an Active Directory.

The Identity parameter specifies the Active Directory group to get. You can identify a group by its distinguished name (DN), GUID, security identifier (SID), Security Accounts Manager (SAM) account name, or canonical name. You can also specify group object variable, such as $.

To search for and retrieve more than one group, use the Filter or LDAPFilter parameters. The Filter parameter uses the PowerShell Expression Language to write query strings for Active Directory. PowerShell Expression Language syntax provides rich type conversion support for value types received by the Filter parameter. For more information about the Filter parameter syntax, type Get-Help about_ActiveDirectory_Filter. If you have existing Lightweight Directory Access Protocol (LDAP) query strings, you can use the LDAPFilter parameter.

This cmdlet gets a default set of group object properties. To get additional properties use the Properties parameter. For more information about the how to determine the properties for group objects, see the Properties parameter description.

29.1. Get single group

```
Get-ADGroup -Identity playersGroup
```

This command gets the group with the SAM account name Administrators.

29.2. Filter for results

```
Get-ADGroup -Filter 'GroupCategory -eq "Security" -and GroupScope -ne
"DomainLocal"'
```

This command gets all groups that have a GroupCategory of Security but do not have a GroupScope of DomainLocal.

29.3. View specific properties from a group

```
Get-ADGroup -Identity playersGroup -Properties DistinguishedName,Members | fl
DN,Mems
```

---

30. New-ADGroup

The New-ADGroup cmdlet creates an Active Directory group object. Many object properties are defined by setting cmdlet parameters. Properties that cannot be set by cmdlet parameters can be set using the OtherAttributes parameter.

The Name and GroupScope parameters specify the name and scope of the group and are required to create a new group. You can define the new group as a security or distribution group by setting the GroupType parameter. The Path parameter specifies the container or organizational unit (OU) for the group.

```
New-ADGroup -Name "RODC Admins" -SamAccountName RODCAdmins -GroupCategory Security
-GroupScope Global -DisplayName "RODC Administrators" -Path
"CN=Users,DC=Fabrikam,DC=Com" -Description "Members of this group are RODC
Administrators"
```

---

31. Remove-ADGroup

The Remove-ADGroup cmdlet removes an Active Directory group object. You can use this cmdlet to remove security and distribution groups.

The Identity parameter specifies the Active Directory group to remove. You can identify a group by its distinguished name, GUID, security identifier, Security Account Manager (SAM) account name, or canonical name. You can also set the Identity parameter to an object variable such as $, or you can pass an object through the pipeline to the Identity parameter. For example, you can use the Get-ADGroup cmdlet to retrieve a group object and then pass the object through the pipeline to the Remove-ADGroup cmdlet.

If the ADGroup is being identified by its distinguished name, the Partition parameter is automatically determined.

```
Remove-ADGroup -Identity SanjaysReports
```

---

32. Remove-ADGroupMember

The Remove-ADGroupMember cmdlet removes one or more users, groups, service accounts, or computers from an Active Directory group.

The Identity parameter specifies the Active Directory group that contains the members to remove. You can identify a group by its distinguished name, GUID, security identifier, or Security Account Manager (SAM) account name. You can also specify a group object variable, such as $, or pass a group object through the pipeline to the Identity parameter. For example, you can use the Get-ADGroup cmdlet to retrieve a group object and then pass the object through the pipeline to the Remove-ADGroupMember cmdlet.

The Members parameter specifies the users, computers and groups to remove from the group specified by the Identity parameter. You can identify a user, computer or group by its distinguished name, GUID, security identifier, or SAM account name. You can also specify user, computer, and group object variables, such as $. If you are specifying more than one new member, use a comma-separated list. You cannot pass user, computer, or group objects through the pipeline to this cmdlet. To remove user, computer, or group objects from a group by using the pipeline, use the Remove-ADPrincipalGroupMembership cmdlet.

```
Remove-ADGroupMember -Identity playersGroup -Members austonMatthews
```

---

33. Set-ADGroup

The Set-ADGroup cmdlet modifies the properties of an Active Directory group. You can modify commonly used property values by using the cmdlet parameters. Property values that are not associated with cmdlet parameters can be modified by using the Add, Replace, Clear, and Remove parameters.

The Identity parameter specifies the Active Directory group to modify. You can identify a group by its distinguished name, GUID, security identifier, or Security Account Manager (SAM) account name. You can also set the Identity parameter to an object variable such as $, or you can pass a group object through the pipeline to the Identity parameter. For example, you can use the Get-ADGroup cmdlet to get a group object and then pass the object through the pipeline to the Set-ADGroup cmdlet.

The Instance parameter provides a way to update a group object by applying the changes made to a copy of the object. When you set the Instance parameter to a copy of an Active Directory group object that has been modified, the Set-ADGroup cmdlet makes the same changes to the original group object. To get a copy of the object to modify, use the Get-ADGroup cmdlet. The Identity parameter is not allowed when you use the Instance parameter. For more information about the Instance parameter, see the Instance parameter description.

```
Set-ADGroup -Server localhost:60000 -Identity "CN=AccessControl,DC=AppNC" -
Description "Access Group" -Passthru
```

Group Policies

34. Local Group Policy

```
gpedit.msc
```

Object Management

35. Disable-ADAccount

The Disable-ADAccount cmdlet disables an Active Directory user, computer, or service account.

The Identity parameter specifies the Active Directory user, computer service account, or other service account that you want to disable. You can identify an account by its distinguished name, GUID, security identifier (SID), or Security Accounts Manager (SAM) account name. You can also set the Identity parameter to an object variable such as $, or you can pass an account object through the pipeline to the Identity parameter. For example, you can use the Get-ADUser cmdlet to retrieve a user account object and then pass the object through the pipeline to the Disable-ADAccount cmdlet. Similarly, you can use Get-ADComputer and Search-ADAccount to retrieve account objects.

```
Disable-ADAcount -Identity austonMatthews
```

---

36. Enable-ADAccount

The Enable-ADAccount cmdlet enables an Active Directory user, computer, or service account.

The Identity parameter specifies the Active Directory user, computer, or service account that you want to enable. You can identify an account by its distinguished name, GUID, security identifier (SID) or Security Accounts Manager (SAM) account name. You can also set the Identity parameter to an object variable such as $, or you can pass an account object through the pipeline to the Identity parameter. For example, you can use the Get-ADUser cmdlet to retrieve an account object and then pass the object through the pipeline to the Enable-ADAccount cmdlet. Similarly, you can use Get-ADComputer and Search-ADAccount to retrieve account objects.

```
Enable-ADAccount -Identity austonMatthews
```

---

37. Move-ADObject

The Move-ADObject cmdlet moves an object or a container of objects from one container to another or from one domain to another.

When an object is moved between domains, both the source DC and the target DC need to be the RID Master of their domains. If a different DC is being used, you will receive the following error:

move-adobject : The requested operation could not be performed because the directory service is not the master for that type of operation

The Identity parameter specifies the Active Directory object or container to move. You can identify an object or container by its distinguished name or GUID. You can also set the Identity parameter to an object variable such as $, or you can pass an object through the pipeline to the Identity parameter. For example, you can use the Get-ADObject cmdlet to retrieve an object and then pass the object through the pipeline to the Move-ADObject cmdlet. You can also use the Get-ADGroup, Get-ADUser, Get-ADComputer, Get-ADServiceAccount, Get-ADOrganizationalUnit, and Get-ADFineGrainedPasswordPolicy cmdlets to get an object that you can pass through the pipeline to this cmdlet.

The TargetPath parameter must be specified. This parameter identifies the new location for the object or container.

If you have ProtectedFromAccidentalDeletion enabled you cannot move objects. It must be disabled first.

37.1. Move single User

```
Get-ADUser "austonMatthews" | Move-ADObject -TargetPath
"OU=Users,OU=Toronto,DC=leafs,DC=com"
```

37.2. Move multiple objects at once

```
Get-ADUser -Filter {Name -like "aust*"} -SearchBase
"OU=users,OU=Washington,DC=leafs,DC=com" | Move-ADObject -TargetPath
"OU=Users,OU=Toronto,DC=leafs,DC=com"
```

---

38. Search-ADAccount

The Search-ADAccount cmdlet retrieves one or more user, computer, or service accounts that meet the criteria specified by the parameters. Search criteria include account and password status. For example, you can search for all accounts that have expired by specifying the AccountExpired parameter. Similarly, you can search for all accounts with an expired password by specifying the PasswordExpired parameter. You can limit the search to user accounts by specifying the UsersOnly parameter. Similarly, when you specify the ComputersOnly parameter, the cmdlet only retrieves computer accounts.

Some search parameters, such as AccountExpiring and AccountInactive use a default time that you can change by specifying the DateTime or TimeSpan parameter. The DateTime parameter specifies a distinct time. The TimeSpan parameter specifies a time range from the current time. For example, to search for all accounts that expire in 10 days, specify the AccountExpiring and TimeSpan parameter and set the value of TimeSpan to 10.00:00:00. To search for all accounts that expire before December 31, 2012, set the DateTime parameter to 12/31/2012.

https://docs.microsoft.com/en-us/powershell/module/activedirectory/search-adaccount?view=windowsserver2019-ps#parameters

38.1. Get all Disabled Accounts

```
Search-ADAccount -AccountDisabled | FT Name, ObjectClass -A
```

Gets all disabled accounts and returns them in a table with the name and Object Class (user, computer, etc).

38.2. Get only users with disabled accounts

```
Search-ADAccount -AccountDisabled -UsersOnly | FT Name,ObjectClass -A
```

Gets all disabled users and returns them in a table with the name and Object Class (user, computer, etc).

38.3. Get all accounts that have expired passwords

```
Search-ADAccount -AccountExpired | FT Name,ObjectClass -A
```

38.4. Get all accounts that are locked out.

```
Search-ADAccount -LockedOut | FT Name,ObjectClass -A
```

---

39. Set-ADAccountPassword

The Set-ADAccountPassword cmdlet sets the password for a user, computer, or service account.

The Identity parameter specifies the Active Directory account to modify.

You can identify an account by its distinguished name, GUID, security identifier (SID) or security accounts manager (SAM) account name. You can also set the Identity parameter to an object variable such as $, or you can pass an object through the pipeline to the Identity parameter. For example, you can use the Search-ADAccount cmdlet to retrieve an account object and then pass the object through the pipeline to the Set-ADAccountPassword cmdlet. Similarly, you can use Get-ADUser, Get-ADComputer, or Get-ADServiceAccount, for standalone MSAs, cmdlets to retrieve account objects that you can pass through the pipeline to this cmdlet.

39.1. Set/Reset Password

```
Set-ADAccountPassword -Identity austonMatthews -Reset -NewPassword (ConvertTo-
SecureString -AsPlainText "Eye1theR0kitRich@rd" -Force)
```

39.2. Set/Reset Password from CLI

This command prompts the user for a new password that is stored in a temporary variable named $NewPassword, then uses it to reset the password for the user account with the matching SamAccountName.

```
$NewPassword = (Read-Host -Prompt "Provide New Password" -AsSecureString)
Set-ADAccountPassword -Identity austonMatthews -NewPassword $NewPassword -Reset
```

---

40. Unlock-ADAccount

The Unlock-ADAccount cmdlet restores Active Directory Domain Services (AD DS) access for an account that is locked. AD DS access is suspended or locked for an account when the number of incorrect password entries exceeds the maximum number allowed by the account password policy.

the Identity parameter specifies the Active Directory account to unlock. You can identify an account by its distinguished name, GUID, security identifier (SID) or Security Accounts Manager (SAM) account name. You can also set the Identity parameter to an account object variable such as $, or you can pass an object through

the pipeline to the Identity parameter. For example, you can use the Search-ADAccount cmdlet to get an account object and then pass the object through the pipeline to the Unlock-ADAccount cmdlet to unlock the account. Similarly, you can use Get-ADUser and Get-ADComputer to get objects to pass through the pipeline.

```
Unlock-ADAccount SAMACCOUNTNAME
```

OU

41. Get-ADOrganizationalUnit

The Get-ADOrganizationalUnit cmdlet gets an organizational unit (OU) object or performs a search to get multiple OUs.

The Identity parameter specifies the Active Directory OU to get. You can identify an OU by its distinguished name or GUID. You can also set the parameter to an OU object variable, such as $ or pass an OU object through the pipeline to the Identity parameter.

To search for and retrieve more than one OU, use the Filter or LDAPFilter parameters. The Filter parameter uses the PowerShell Expression Language to write query strings for Active Directory. PowerShell Expression Language syntax provides rich type conversion support for value types received by the Filter parameter. For more information about the Filter parameter syntax, type Get-Help about_ActiveDirectory_Filter. If you have existing Lightweight Directory Access Protocol (LDAP) query strings, you can use the LDAPFilter parameter.

This cmdlet gets a default set of OU object properties. To get additional properties, use the Properties parameter. For more information about the how to determine the properties for computer objects, see the Properties parameter description.

```
Get-ADOrganizationalUnit -Identity "OU=Toronto,DC=leafs,DC=com"
```

---

42. New-ADOrganizationUnit

The New-ADOrganizationalUnit cmdlet creates an Active Directory organizational unit (OU). You can set commonly used OU property values by using the cmdlet parameters. Property values that are not associated with cmdlet parameters can be set by using the* OtherAttributes* parameter.

You must set the Name parameter to create a new OU. If you do not specify the Path parameter, the cmdlet creates an OU under the default NC head for the domain.

42.1. Create new OU

```
New-ADOrganizationUnit -Name "Toronto" -Description "Toronto Branch"
```

When no "Path" is defined, then the OU is created in root.

42.2. Create new OU with path

```
New-ADOrganizationalUnit -Name "Toronto" -Path "DC=leafs,DC=COM"
```

---

43. Remove-ADOrganizationalUnit

The Remove-ADOrganizationalUnit cmdlet removes an Active Directory organizational unit (OU).

The Identity parameter specifies the organizational unit to remove. You can identify an organizational unit by its distinguished name or GUID. You can also set the parameter to an organizational unit object variable, such as $ or you can pass an object through the pipeline to the Identity parameter. For example, you can use the Get-ADOrganizationalUnit cmdlet to retrieve the object and then pass the object through the pipeline to the Remove-ADOrganizationalUnit cmdlet.

If the object you want to remove has child objects, you must specify the Recursive parameter.

If the ProtectedFromAccidentalDeletion property of the organizational unit object is set to true, the cmdlet returns a terminating error.

```
Remove-ADOrganizationalUnit "OU=users,OU=Toronto,DC=leafs,DC=com"
```

---

44. Set-ADOrganizationalUnit

The Set-ADOrganizationalUnit cmdlet modifies the properties of an Active Directory organizational unit (OU). You can modify commonly used property values by using the cmdlet parameters. Property values that are not associated with cmdlet parameters can be modified by using the Add, Remove, Replace, and Clear parameters.

The Identity parameter specifies the Active Directory organizational unit to modify. You can identify an organizational unit by its distinguished name or GUID.

You can also set the Identity parameter to an object variable such as $, or you can pass an object through the pipeline to the Identity parameter. For example, you can use the Get-ADOrganizationalUnit cmdlet to retrieve an organizational unit object and then pass the object through the pipeline to the Set-ADOrganizationalUnit cmdlet.

```
Set-ADOrganizationalUnit -Identity "OU=UserAccounts,DC=leafs,DC=COM" -Description
"This Organizational Unit holds all of the users accounts of leafs.COM"
```

44.1. Set Managed By Attribute

```
Set-ADOrganizationalUnit -Identity "OU=UserAccounts,DC=leafs,DC=COM" -ManagedBy
"Toronto IT Team"
```

Make sure that the managed by value is set to an existing user or group. Otherwise the command will fail.

44.2. Set Protected From Accidental Deletion Attribute

```
Set-ADOrganizationalUnit -Identity "OU=UserAccounts,DC=leafs,DC=COM" -
ProtectedFromAccidentalDeletion $true
```

Powershell

45. Commands

45.1. cd

```
Changes the current working directory.
```

Command	Desc
`cd [PARAMS]`	Base command.
`cd ..`	Backs up one directory.
`cd Test`	Enters into the 'Test' subdirectory.

45.2. cls

```
Clears the PS console screen.
```

Command	Desc
`cls`	Base command.

45.3. dir

```
Displays the contents of the current working directory.
```

Command	Desc
`dir`	Base command.

45.4. Get-Command

```
Displays the commands that PS is aware of.
```

Command	Desc
`Get-Command`	Base command.
`Get-Command -Name <NAME OF COMMAND>`	Displays the command with the name provided.

Command	Desc
`Get-Command -Verb <NAME OF VERB> [Get, Add, Clear, Set]`	Displays all the known commands with the given verb.
`Get-Command -Noun <NAME OF NOUN> [Job, Process, List, Host, etc]`	Displays all the known commands with the given noun.
`Get-Command -Verb <NAME OF VERB> -Noun <NAME OF NOUN>`	Displays all commands with the given verb and noun.

45.5. Get-Content

```
Gets content from a resource.
```

Command	Desc
`Get-Content -Path <PATH TO FILE>`	Retrieves the content of the file.

45.6. Get-Help

```
Shows the documentation for a given command.
```

Command	Desc
`Get-Help <NAME OF COMMAND>`	Base command.
`Get-Help <NAME OF COMMAND> -Full`	Displays the full documentation for that command.
`Get-Help -Name <NAME OF COMMAND>*`	Displays documentation for commands that begin with the supplied name.

45.7. Get-Member

```
Returns a list of all methods and properties of an object.
```

Command	Desc
`Get-Member -InputObject <OBJECT NAME>`	Returns a list of all methods and properties of the supplied object.

45.8. Get-Service

```
Returns a list of all services running on the OS.
```

Command	Desc
`Get-Service`	Returns a list of all services
`Get-Service <NAME>`	Returns the service by name.

45.9. Get-Variable

```
Returns a list of all variables, both user defined and built in.
```

Command	Desc
`Get-Variable`	Base command.
`Get-Variable -Name <NAME OF VARIABLE>`	Returns the name and value for a given variable.
`Get-Variable -Name Preference*`	Returns a list of predefined preferences built into PS.

45.10. Resolve-DnsName

```
Resolve DNS information for a given IP address.
```

Command	Desc
`Resolve-DnsName -Name <IP ADDRESS>`	Base command.

45.11. Test-Connection

```
Pings a host and returns information.
```

Command	Desc
`Test-Connection <HOST>`	Base command.
`Test-Connection -ComputerName <NAME>`	Tests the connection to a computer using the host name.
`Test-Connection <HOST> -Count <NUMBER>`	Only sends one ICMP packet.

Command	Desc
`Test-Connection <HOST> -Quiet`	Forces the command to return a simple bool for connection status.

45.12. Update-Help

```
Compares local documentation for commands against an online repository and
downloads new docs if needed.

May Require Admin.
```

Command	Desc
`Update-Help`	Base command.

46. Config

46.1. Set-ExecutionPolicy

```
Sets the execution policy for scripts.
```

Command	Desc
`Set-ExecutionPolicy -ExecutionPolicy <POLICY> [Restricted, AllSigned, RemoteSigned, Unrestricted]`	Base command.

46.2. Set-StrictMode

```
Turns on many errors that are off by default.
```

Command	Desc
`Set-StrictMode -Version Latest`	Base command.

47. Errors

47.1. Try, Catch, Finally

```
Flow control and error handling.
Note that finally is optional. If it is set then it will run, even after an error
```

```
was caught.
Can only find terminating errors.

Use $_.Exception.Message in the catch block return just the message.

try {
    # Something Dangerous
} catch {
    # Error Occured
} finally {
    # Final code
}
```

47.1.1. $Error

```
Stores a long history of errors. Newest errors are added to the front of the
array.
```

Command	Desc
`$Error`	Base command.
`$Error[0]`	Get the last error.

---

48. Flow Control

48.1. not

```
-not (condition)
```

48.2. if elseif else

```
if(statement) {
    # Something
} elseif {
    # Something
} else {
    # Something
}
```

48.3. Switch

```
switch (statement) {
    caseOne {
        # Something
    }
    caseTwo {
        # Something
    }
    default {
        # Something
    }
}
```

48.4. foreach

```
foreach($something in $array) {
    # Something

    ** Modifying an array element inside here does not modify the original array.
}
```

48.5. ForEach-Object

```
$array | ForEach-Object - Process {
    # Something

    ** Modifying an array element inside here does not modify the original array.
}
```

48.6. for

```
for($i = 0; $i -lt 10; $i++) {
    # Something
}
```

48.7. while

```
while (condition) {
    # Something
}
```

48.8. do while

```
do {
    # Something
} while (condition)
```

48.9. do until

```
do {
    # Something
} until (condition)
```

---

49. Powershell Functions

```
function Write-Name {

    param (

        # Mandatory and allow piping
        [Parameter(Mandatory, ValueFromPipeline)]
        [string] $Name,

        # Optional named parameter w/ validation
        [Parameter()]
        [ValidateSet(33, 44, 55)]
        [int] $Age

    )

    # Runs once before process block
    begin {
        Write-Host "Beginning"
    }

    # Runs once for every value piped in. Required if you are piping in an array
of values.
    process {
```

```
        Write-Host "My name is $Name and I am $Age years old."
    }

    # Runs once after the process block is complete.
    end {
        Write-Host "Ending"
    }

}
```

$Names = ('Geoff', 'Kelly', 'Ellie')

$Names | Write-Name -Age 33

---

50. Modules

50.1. Module Save Locations

```
System - C:\Windows\System32\WindowsPowershell\1.0\Modules
All Users - C:\Program Files\WindowsPowershell\Modules
Current User - C:\Users\<Current User>\Documents\WindowsPowerShell\Modules
```

50.2. Add a Module folder

```
$CurrentPath = [Environment]::GetEnvironmentVariable("PSModulePath", "Machine")
[Environment]::SetEnvironmentVariable("PSModulePath", $CurrentValue + ";C:\
<DIRECTORY PATH>", "Machine")
```

50.3. Get-Module

```
Shows the documentation for a given command.
```

Command	Desc
`Get-Module <NAME OF COMMAND>`	Modules from this session only.
`Get-Module -ListAvailable`	Modules that are installed on the system (and all user folders) and available for import.

50.4. Import-Module

```
Imports a module manually, if auto-import fails.
```

Command	Desc
`Import-Module -Name <NAME OF MODULE>`	Imports Module
`Import-Module -Name <NAME OF MODULE> -Force`	Unload and then reimport module. Needs to be run if module code changes since last import.

50.5. Remove-Module

```
Removes a module from the current session. It does not uninstall it.
```

Command	Desc
`Remove-Module -Name <NAME OF MODULE>`	Removes Module

50.6. Find-Module

```
Searches the Powershell Gallery for modules with the name or partial name.
```

Command	Desc
`Remove-Module -Name <NAME OF MODULE>*`	Searches for the Module
`Remove-Module -Name <NAME OF MODULE> | Install-Module`	Searches for the Module and then installs it.

50.7. Install-Module

```
Installs a module from the Powershell Gallery.
```

Command	Desc
`Install-Module -Name <NAME OF MODULE>`	Install the Module. By default it's installed for all users.

50.8. Uninstall-Module

```
Uninstalls a module from the system.
```

Command	Desc
`Uninstall-Module -Name <NAME OF MODULE>`	Uninstalls the Module.

50.9. Create Custom Module

1. Create a folder with the module name. The folder must be the same name as the module.
2. Create custom .psm1 file.
3. Cretae a custom module manifest:

```
New-ModuleManifest -Path '<PATH TO MANIFEST>.psd1' -Author '<AUTHOR>' -RootModule <PSM1 FILE NAME> -Description '<DESCRIPTION>'
```

51. Operators

Operator	JS
-eq	===
-ne	!==
-gt	>
-ge	>=
-lt	<
-le	<=
-contains	array.contains('value')

52. Parsing Files

52.1. CSV

52.1.1. Reading CSV Files

```
Import-Csv -Path <FILE PATH>
```

52.1.2. Querying CSV Files

```
Import Csv -Path <FILE PATH> | Where-Object {$_.'<HEADER KEY>' -eq '<VALUE>'}
```

52.1.3. Renaming Header

```
Import Csv -Path <FILE PATH> -Header '<KEY ONE>','<KEY TWO>','<KEY THREE>'
```

52.1.4. Creating CSV Files

> You can add the -NoTypeInformation to remove the line at the top of the CSV file saying what powershell type this file came from.
>
> Add the -Append flag to prevent this from overwriting the file with each new line.

```
<OBJECT> | Export-Csv -Path <OUTPUT FILE PATH>.csv
```

52.2. Excel Sheets

> Requires the third party module ImportExcel to be installed from the gallery.

52.2.1. Creating an Excel Sheet

```
<OBJECT> | Export-Excel <FILE PATH>.xlsx
```

52.2.2. Creating a worksheet

```
<OBJECT> | Export-Excel <FILE PATH>.xlsx -WorksheetName '<WORKSHEET NAME>'
```

52.2.3. Querying Worksheet Available

> Returns a list of the worksheets. Use a for loop to iterate through them.

```
Get-ExcelSheetInfo -Path <FILE PATH>.xlsx
```

52.2.4. Importing Excel Sheets

> By defualt only the first worksheet will be imported.

```
Import-Excel -Path <FILE PATH> -WorksheetName '<WORKSHEET NAME>'
```

52.2.5. Dynamic Fields

```
Fields whos value is calculated on the fly, then added to the object/excel sheet.

Usually done in a for loop so that each value is set for each row.
```

```
<OBJECT> | Select-Object -Property @{Name = '<KEY NAME>'; Expression = {<EXPRESSION CALCULATED>}}
```

Example: Adds a timestamp column to each entry.

```
<OBJECT> | Select-Object -Property @{Name = 'Timestamp'; Expression = {Get-Date -Format 'MM-dd-yy hh:mm:ss'}}}
```

52.3. JSON

52.3.1. Reading JSON

```
Raw returns a plain string from the JSON file.
```

```
Get-Content -Path <PATH TO JSON FILE>.json -Raw | ConvertFromJson
```

52.3.2. Creating JSON

```
Use the -Compress flag to obfuscate the json output.
```

```
<OBJECT> | ConvertToJson -Compress
```

52.3.3. JSON and Web Requests

```
Use the latter to ignore the request status code and just show the result.
```

```
Invoke-RestMethod -Uri <URI>
(Invoke-RestMethod -Uri <URI>).result
```

---

53. Remote Execution

53.1. Invoke-Command

```
Sends a command to a remote server.
```

```
If no Session variable is set then this command establishes connection, executes
the command then closes the session. Think one off. Not very fast.

Computers must be on the same AD domain and the executing computer must have
priviledges on the host.

Notice that local variables don't always work on the host.

** The 'using' method can cause problems when testing with Pester. You may have to
use the ArgumentList.
```

Command	Desc
`Invoke-Command -ComputerName <HOST COMPUTER NAME> -ScriptBlock {<SCRIPT TO EXECUTE}`	Base command.
`Invoke-Command -ComputerName <HOST COMPUTER NAME> -FilePath <SCRIPT FILE PATH>`	Executes the commands from a file.
`Invoke-Command -ComputerName <HOST COMPUTER NAME> { <SCRIPT TO RUN> $($args[0]) } -ArgumentList <LOCAL VARIABLE>`	Uses the argument list to pass local variables to the host.
`Invoke-Command -ComputerName <HOST COMPUTER NAME> { <SCRIPT TO RUN> $using:<LOCAL VARIABLE> }`	Uses the 'using' command to pass local variables to the host.
`Invoke-Command -Session <SESSION> { <SCRIPT TO RUN> }`	Executes the command using a pre-established session.

53.2. New-PSSession

```
Creates a new session with a host.
```

Command	Desc
`New-PSSession -ComputerName <COMPUTER NAME>`	Base command.

53.3. Get-PSSession

```
Returns a list of past sessions. Can be saved as a variable.
```

Command	Desc
`Get-PSSession`	Base command.

Command	Desc
`Get-PSSession -Id <SESSION ID>`	Return a session by Id.

53.4. Disconnect-PSSession

```
Disconnects the session but allows you to connect back later. You pipe in the
session from Get-Session.
```

Command	Desc
`Get-PSSession | Disconnect-PSSession`	Base command.
`Get-PSSession -Id <SESSION ID> | Disconnect-PSSession`	Disconnects a session by Id.

53.5. Connect-PSSession

```
Connects to previouslly disconnected hosts, that you have already connected to.
```

Command	Desc
`Connect-PSSession -ComputerName <COMPUTER NAME>`	Base command.

53.6. Remove-PSSession

```
Removes a previouslly connected session. You will not be allowed to connect again
without creating a new session.
```

Command	Desc
`Get-PSSession | Remove-PSSession`	Base command.
`Get-PSSession -Id <SESSION ID> | Remove-PSSession`	Removes a session by Id.

54. Types

54.1. $null

```
The non-value assignment.
```

Command	Desc
`$name = $null`	The variable 'name' is created but it's value is null.

54.2. $True and $False

```
The true and false values.
```

Command	Desc
`$isAlive = $True`	Value is true.
`$wantsToBeAtWork = $False`	Value is false.

54.3. Arrays

```
Creates a fixed size array.

Not efficient for larger data sets.
```

Command	Desc
`$array = (<ONE>, <TWO>, <THREE>)`	Creates a fixed size array.
`$array += <four>`	Adds element 'four'.
`$array[0]`	Returns first value of the array.
`$array[-1]`	Returns the last value of the array.
`$array[1..2]`	Returns the second and third values of the array.
`$array = $array.Remove('four')`	Removes element 'four'.

54.4. Array Lists

```
Creates a variable size array. Better for larger data sets. When adding or
removing there is no need for overwrite.
```

Command	Desc
`$arrayList = [System.Collections.ArrayList](<ONE>, <TWO>, <THREE>)`	Base command.
`$arrayList.Add(<FOUR>)`	Adds a fourth element.

Command	Desc
`$arrayList.Remove(<FOUR>)`	Removes the 'fourth' element.

54.5. Hash Tables

```
Creates a hash table dictionary. Notice the '@'
```

Command	Desc
`$hashTable = @{ name = <NAME>; age = <AGE>;}`	Base command.
`$hashTable.Keys`	Returns a list of all of the keys.
`$hashTable.Values`	Returns a list of all of the values.
`$hashTable['<KEY>'] = <VALUE>`	Overwrites or creates a new entry into the hash table.
`$hashTable.Remove('<KEY>')`	Removes a key and value pair.
`$hashTable.ContainsKey('<KEY>')`	Returns a bool if a key exists in the hash table.
`$hashTable.ContainsValue('<Value>')`	Returns a bool if a value exists in the hash table.

54.6. Custom Objects

```
Creates a custom object.
```

Command	Desc
`$customObject = [PSCustomObject]@{<KEY> = <VALUE>;}`	Creates a custom object.

Service Accounts

55. Add-ADComputerServiceAccount

The Add-ADComputerServiceAccount cmdlet adds one or more computer service accounts to an Active Directory computer.

The Computer parameter specifies the Active Directory computer that will host the new service accounts. You can identify a computer by its distinguished name, GUID, security identifier (SID) or Security Accounts Manager (SAM) account name. You can also set the Computer parameter to a computer object variable, such

as $, or pass a computer object through the pipeline to the Computer parameter. For example, you can use the Get-ADComputer cmdlet to retrieve a computer object and then pass the object through the pipeline to the Add-ADComputerServiceAccount cmdlet.

The ServiceAccount parameter specifies the service accounts to add. You can identify a service account by its distinguished name, GUID, Security Identifier (SID) or Security Accounts Manager (SAM) account name. You can also specify service account object variables, such as $. If you are specifying more than one account, use a comma-separated list.

Note: Adding a service account is a different operation than installing the service account locally.

55.1. Add single account

```
Add-ADComputerServiceAccount -Computer ComputerAcct1 -ServiceAccount SvcAcct1
```

55.2. Add multiple accounts

```
Add-ADComputerServiceAccount -Computer ComputerAcct1 -ServiceAccount
SvcAcct1,SvcAcct2
```

---

56. Install-ADServiceAccount

The Install-ADServiceAccount cmdlet installs an existing Active Directory managed service account on the computer on which the cmdlet is run. This cmdlet verifies that the computer is eligible to host the managed service account. The cmdlet also makes the required changes locally so that the managed service account password can be managed without requiring any user action.

The Identity parameter specifies the Active Directory managed service account to install. You can identify a managed service account by its distinguished name, GUID, security identifier (SID), or security accounts manager (SAM) account name. You can also set the parameter to a managed service account object variable, such as $ or pass a managed service account object through the pipeline to the Identity parameter. For example, you can use Get-ADServiceAccount to get a managed service account object and then pass the object through the pipeline to the Install-ADServiceAccount.

The AccountPassword parameter allows you to pass a secure string that contains the password of a standalone managed service account and is ignored for group managed service accounts. Alternatively, you can use PromptForPassword parameter to prompt for the standalone managed service account password. You must enter the password of a standalone managed service account if you want to install an account that you have provisioned. This is required when you are installing a standalone managed service account on a server located on a segmented network (site) with read-only domain controllers (for example, a perimeter network or DMZ). In this case you should create the standalone managed service account, link it with the appropriate computer account, and assign a well-known password that must be passed when installing the standalone managed service account on the server on the read-only domain controller site. If you pass both AccountPassword and PromptForPassword parameters, the AccountPassword parameter takes precedence.

```
Install-ADServiceAccount -Identity 'SQL-HR-svc-01'
```

---

57. New-ADServiceAccount

The New-ADServiceAccount cmdlet creates a new Active Directory managed service account. By default, the cmdlet creates a group managed service account. To create a standalone managed service account which is linked to a specific computer, use the RestrictToSingleComputer parameter. To create a group managed service account which can only be used in client roles, use the RestrictToOutboundAuthenticationOnly parameter. This creates a group managed service account that can be used for outbound connections only and any attempts to connect to services using this account will fail because the account does not have enough information for authentication. You can set commonly used managed service account property values by using the cmdlet parameters. Property values that are not associated with cmdlet parameters can be set by using the OtherAttributes parameter.

The Path parameter specifies the container or organizational unit (OU) for the new managed service account object. When you do not specify the Path parameter, the cmdlet creates an object in the default managed service accounts container for managed service account objects in the domain.

```
New-ADServiceAccount -Name "Service01" -RestrictToSingleComputer
```

---

58. Remove-ADServiceAccount

The Remove-ADServiceAccount cmdlet removes an Active Directory managed service account. This cmdlet does not make changes to any computers that use the managed service account. After this operation, the managed service account no longer exists in the directory, but computers are configured to use the managed service account.

The Identity parameter specifies the Active Directory managed service account to remove. You can identify a managed service account by its distinguished name (DN), GUID, security identifier (SID), or Security Account Manager (SAM) account name. You can also set the Identity parameter to a managed service account object variable, such as $, or you can pass a managed service account object through the pipeline to the Identity parameter. For example, you can use the Get-ADServiceAccount cmdlet to retrieve a managed service account object and then pass the object through the pipeline to the Remove-ADServiceAccount cmdlet.

Note: Removing the service account is a different operation than uninstalling the service account locally.

```
Remove-ADServiceAccount -Identity SQL-SRV1
```

---

59. Test-ADServiceAccount

The Test-ADServiceAccount cmdlet tests a managed service account (MSA) from a local computer.

the Identity parameter specifies the Active Directory MSA account to test. You can identify a MSA by its distinguished name (DN), GUID, security identifier (SID), or Security Account Manager (SAM) account name. You can also set the parameter to a MSA object variable, such as $ or pass a MSA object through the pipeline to the Identity parameter. For example, you can use the Get-ADServiceAccount to get a MSA object and then pass that object through the pipeline to the Test-ADServiceAccount cmdlet.

```
Test-ADServiceAccount -Identity MSA1
```

This command tests the specified service account, MSA1, from the local computer. The test indicates whether the account is ready for use, which means it can be authenticated and that it can access the domain using its

current credentials.

Users

60. Basic User Object

Param	Desc
Name	Full Name.
GivenName	First Name.
Surname	Last Name.
SamAccountName	Username.
UserPrincipalName	UPN.
Path	Defines the OU path.
Account Password	Password. Can be set as an input to immediately ask for a typed password.
Enabled	Start the account as enabled?

61. Get-ADUser

The Get-ADUser cmdlet gets a specified user object or performs a search to get multiple user objects.

The Identity parameter specifies the Active Directory user to get. You can identify a user by its distinguished name (DN), GUID, security identifier (SID), Security Account Manager (SAM) account name, or name. You can also set the parameter to a user object variable such as $ or pass a user object through the pipeline to the Identity parameter.

To search for and retrieve more than one user, use the Filter or LDAPFilter parameters. The Filter parameter uses the PowerShell Expression Language to write query strings for Active Directory. PowerShell Expression Language syntax provides rich type-conversion support for value types received by the Filter parameter. For more information about the Filter parameter syntax, type Get-Help about_ActiveDirectory_Filter. If you have existing Lightweight Directory Access Protocol (LDAP) query strings, you can use the LDAPFilter parameter.

This cmdlet retrieves a default set of user object properties. To retrieve additional properties use the Properties parameter. For more information about how to determine the properties for user objects, see the Properties parameter description.

61.1. Get a Single Users properties

```
Get-ADUser -Identity austonMatthews
```

This command gets all of the properties of the user with the SAM account name austonMatthews.

```
Get-ADUser -Identity austonMatthews -Properties *
```

This command gets specified properties from the SAM account austonMatthews.

```
Get-ADUser -Identity austonMatthews -Properties Name,Position,Number
```

This command retrieves specified properties and returns them in a table.

```
Get-ADUser -Identity austonMatthews -Properties Name,Position,Number | Format-
Table Name,Pos,Num
```

61.2. Get Multiple Users' properties

Returns multiple Users based on Team property.

```
Get-ADUser -Filter {Team -like "Toron*"} -Properties Name,Position,Number |
Format-Table Name,Pos,Num
```

---

62. New-ADUser

The New-ADUser cmdlet creates an Active Directory user. You can set commonly used user property values by using the cmdlet parameters.

View the syntax:

```
Get-Command NewADUser -Syntax
```

You can set property values that are not associated with cmdlet parameters by using the OtherAttributes parameter. When using this parameter, be sure to place single quotes around the attribute name.

You must specify the SamAccountName parameter to create a user.

You can use the New-ADUser cmdlet to create different types of user accounts such as iNetOrgPerson accounts. To do this in Active Directory Domain Services (AD DS), set the Type parameter to the Lightweight Directory Access Protocol (LDAP) display name for the type of account you want to create. This type can be any class in the Active Directory schema that is a subclass of user and that has an object category of person.

The Path parameter specifies the container or organizational unit (OU) for the new user. When you do not specify the Path parameter, the cmdlet creates a user object in the default container for user objects in the domain.

The following methods explain different ways to create an object by using this cmdlet.

62.1. Create an enabled user with password input on the CLI

```
New-ADUser -Name "Auston Matthews" -GivenName "Auston" -Surname "Matthews" -
SamAccountName "austonMatthews" -UserPrincipalName "auston@leafs.com" -Path
"OU=Users,OU=CAN,DC=leafs,DC=com" -AccountPassword(Read-Host -AsSecureString "Type
Password for User") -Enabled $true
```

62.2. Create Multiple Users from CSV file

```
Import-Csv "<FILE PATH>.csv" | ForEach-Object {
```

```
$upn = $_."SamAccountName" + "@evilcorp.com"

New-ADUser -Name $_."Name" -GivenName $_."GivenName" -Surname $_."Surname" -
SamAccountName $_."SamAccountName" -UserPrincipalName $upn -Path $_."Path" -
AccountPassword (ConvertTo-SecureString "Pa$$w0rd" -AsPlainText -force) -Enabled
$true

}
```

62.3. Create a disabled user with minimal details

This account cannot be enabled until a password is set.

```
New-ADUser -Name "Auston Matthews" -GivenName "Auston" -Surname "Matthews" -
SamAccountName "austonMatthews" -UserPrincipalName "auston@leafs.com" -Path
"OU=Users,OU=CAN,DC=leafs,DC=com"
```

---

63. Remove-ADUser

The Remove-ADUser cmdlet removes an Active Directory user.

The Identity parameter specifies the Active Directory user to remove. You can identify a user by its distinguished name (DN), GUID, security identifier (SID), or Security Account Manager (SAM) account name. You can also set the Identity parameter to a user object variable, such as $, or you can pass a user object through the pipeline to the Identity parameter. For example, you can use the Get-ADUser cmdlet to retrieve a user object and then pass the object through the pipeline to the Remove-ADUser cmdlet.

```
Remove-ADUser -Identity AustonMatthews
```

63.1. Search and Remove

```
Get-ADUser -Filter {Name -like "Austo*"} | Remove-ADUser
```

---

64. Set-ADUser

The Set-ADUser cmdlet modifies the properties of an Active Directory user. You can modify commonly used property values by using the cmdlet parameters. You can set property values that are not associated with cmdlet parameters by using the Add, Remove, Replace, and Clear parameters.

The Identity parameter specifies the Active Directory user to modify. You can identify a user by its distinguished name, GUID, security identifier (SID), or Security Account Manager (SAM) account name. You can also set the Identity parameter to an object variable such as $, or you can pass an object through the pipeline to the Identity parameter. For example, you can use the Get-ADUser cmdlet to retrieve a user object and then pass the object through the pipeline to the Set-ADUser cmdlet.

The Instance parameter provides a way to update a user object by applying the changes made to a copy of the object. When you set the Instance parameter to a copy of an Active Directory user object that has been modified, the Set-ADUser cmdlet makes the same changes to the original user object. To get a copy of the

object to modify, use the Get-ADUser object. The Identity parameter is not allowed when you use the Instance parameter. For more information about the Instance parameter, see the Instance parameter description.

64.1. Set properties

```
Set-ADUser -Identity austonMatthews -HomePage 'http://www.goleafsgo.com' -
LogonWorkstations 'AustonMatthews-LPTOP'
```

64.2. Set, Replace, and Clear properties

```
Set-ADUser -Identity austonMatthews -Remove @{otherMailbox="auston.matthews"} -Add
@{url="goleafsgo.com"} -Replace @{title="center"} -Clear description
```

64.3. Set properties using AD information

```
$Manager = Get-ADUser -Identity KyleDubas -Server Corp-DC01

Set-ADUser -Identity austonMatthews -Manager $Manager -Server Scotia-Bank-Arena-
DC02
```

64.4. Batch Set properties

This command gets all the users in the directory that are located in the OU=Players,OU=UserAccounts,DC=LEAFS,DC=COM organizational unit. This will change the city property of all users in that OU to Toronto.

```
Get-ADUser * -SearchBase 'OU=Players,OU=UserAccounts,DC=LEAFS,DC=COM' | Set-ADUser
-City "Toronto"
```

64.5. Filter for Users and then set properties

Filters for all users in that OU with a name begining with "Aust".

```
Get-ADUser -Filter 'Name -like "Aust*"' -SearchBase
'OU=Players,OU=UserAccounts,DC=Leafs,DC=COM' | Set-ADUser -City "Toronto"
```

Utils

65. gpresult

Displays the Resultant Set of Policy (RSoP) information for a remote user and computer. To use RSoP reporting for remotely targeted computers through the firewall, you must have firewall rules that enable inbound network traffic on the ports.

```
gpresult /r
```

65.1. For a remote user

```
gpresult /s COMPUTERNAME /r
```

---

66. gpupdate

Updates Group Policy settings.

```
gpupdate /force
```