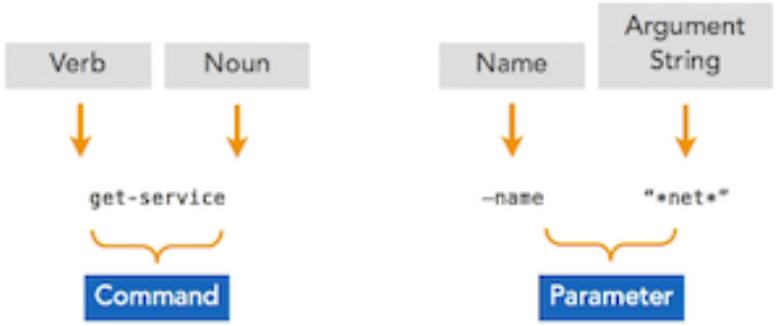
What version do I have? - check $PSVersiontable
Windows\System32\WindowsPowerShell\



Verbs: new get set stop start suspend restart resume format enable write use update export import
Noun: Objects like service
Combined noun-verb is commandlib (check this

Follow by -property (like name) to interrogate and then "argString"

Spits out:

```
Status      Name       DisplayName
------      ----       -----------
Stopped     NetLogon   NetLogon
Running     Netman     Network Connections
```

This isn't all of the property listings. Later is more on the .NET framework underneath.

Case-insensitive, but parameters and objects can some times be case-sensitive

Pipe is there
get-service I out-file c:\services.txt

Tab completion works

get-help is like man for commands. Options:
get-help get-service
get-help get-service -examples
get-help get-service -detailed
get-help get-service -full <----full is much like detailed but has 'related links' at bottom
get-help get-service -online <----Goes to the URL in IE/Edge

get-command -- shows all of the cmdlets AVAILABLE on the system, but not those that aren't loaded

ls and dir aliases to get-childitem or gci
The notes in get-help get-childitem mention it's built-in aliases

Also mentions about_Aliases; get-help about_Aliases
To work with aliases: new- get- set- import- export-alias
get- item
get-
get-process

The > in C:\> will turn red when your typed command isn't complete yet. If you hit return it will give you a prompt much like the Python shell ">>>" where it lets you continue the line

function add {$add = [int](2+2) write-output "$add"}
If you then type add it spis out 4.

-----> functions vs cmdlets. need more info
-----> functions vs methods. PS has methods too! Difference?

get-service | where-object {$_.status -eq "stopped"}

get-service | get-member ---shows all of the parameters you can query

```
PS C:\> get-service |get-member


   TypeName: System.ServiceProcess.ServiceController

Name                      MemberType    Definition
----                      ----------    ----------
Name                      AliasProperty Name = ServiceName
RequiredServices          AliasProperty RequiredServices = ServicesDependedOn
Disposed                  Event         System.EventHandler Disposed(Syste...
Close                     Method        void Close()
Continue                  Method        void Continue()
CreateObjRef              Method        System.Runtime.Remoting.ObjRef Cre...
Dispose                   Method        void Dispose(), void IDisposable.D...
Equals                    Method        bool Equals(System.Object obj)
ExecuteCommand            Method        void ExecuteCommand(int command)
GetHashCode               Method        int GetHashCode()
GetLifetimeService        Method        System.Object GetLifetimeService()
GetType                   Method        type GetType()
InitializeLifetimeService Method        System.Object InitializeLifetimeSe...
Pause                     Method        void Pause()
Refresh                   Method        void Refresh()
Start                     Method        void Start(), void Start(string[] ...
Stop                      Method        void Stop()
WaitForStatus             Method        void WaitForStatus(System.ServiceP...
CanPauseAndContinue       Property      bool CanPauseAndContinue {get;}
CanShutdown               Property      bool CanShutdown {get;}
CanStop                   Property      bool CanStop {get;}
Container                 Property      System.ComponentModel.IContainer C...
DependentServices         Property      System.ServiceProcess.ServiceContr...
DisplayName               Property      string DisplayName {get;set;}
MachineName               Property      string MachineName {get;set;}
ServiceHandle             Property      System.Runtime.InteropServices.Saf...
ServiceName               Property      string ServiceName {get;set;}
ServicesDependedOn        Property      System.ServiceProcess.ServiceContr...
ServiceType               Property      System.ServiceProcess.ServiceType ...
Site                      Property      System.ComponentModel.ISite Site {...
Status                    Property      System.ServiceProcess.ServiceContr...
ToString                  ScriptMethod  System.Object ToString();
```

get-service | stop-service -whatif
obviously, this would stop every service listed by get-service, but is an example of "-whatif" which will not actually execute it, but just shows what a command would do
If you instead use "-confirm" it will run normally, but for each action prompt with "Are you sure you want to perform this action?"

Typing ise brings ISE up, with shell at bottom and scripting pane at the top
Intellisense is what MS calls autocompletion suggestions (basically)
When you have lines of code you can hit the play button to run script or highlight some code and hit "run selection"
Command Addon window shows a list of commands to browse/chose from, lets you chose/input the command parameters and will paste the resulting proper syntax in for you

##Pipe Formating
get-service |format-list *
get-service |format-table displayname, status, requiredservices | sort-object -property status
get-service |sort-object -property status | format-list displayname, status, requiredservices

List view. Table is just like it sounds- column for name status, etc

```
DisplayName      : Windows Color System
Status           : Stopped
RequiredServices : {RpcSs}

DisplayName      : Diagnostic Service Host
Status           : Stopped
RequiredServices : {}
```

###Pipe OutPut
get-service |out-file c:\services.txt
get-service |export-csv c:\services.csv ####-Delimiter ";"

###Pipe Gridview --Gridview is more of a GUI table that can be sorted, filtered, searched, etc.
get-service |out-gridview
get-service |format-table displayname, status, requiredservices |out-gridview This WON'T work. use select-obj
get-service |select-object displayname, status, requiredservices | out-gridview
get-service |select-object * | out-gridview

#PowerShell Leveraging the ComputerName Parameter to get info from other systems
Get-Service
Get-Service -ComputerName dcdsc, webserver |Format-Table machinename, name,

status <--- dumps everything in a jumbled order
Get-Service -ComputerName dcdsc, webserver |Sort-Object -property machinename | Format-Table machinename, name, status
        <-- The sort-object directive tells it to order things by machine name as desired
Get-Service -ComputerName webserver, dcdsc | select-object * |Out-GridView

When you instead need to open a full shell on another system, File> New Remote PS Tab...
Asks for Computer name (or IP) and username to log in as. Prompt changes like below.

```
PS C:\Users\Administrator>
[webserver]: PS C:\Users\administrator.HALO\Documents>
```

Module: A collection of cmdlets for a particular application or function
get-module -listavailable
PS will do module autoloading when you call a command, rather than having to load a module manually
import-module -name applocker
get-command -module applocker <--- will show you all of that module's commands

Execution policy- restricted trust by PS to run scripts (signing etc)
Get-ExecutionPolicy
Unrestricted good for testing but can be dangerous if left on
Set-ExecutionPolicy
  Restricted - No scripts can be run. Windows PowerShell can be used only in interactive mode.
  AllSigned - Only scripts signed by a trusted publisher can be run.
  RemoteSigned - Downloaded scripts must be signed by a trusted publisher before they can be run.
  Unrestricted - No restrictions; all Windows PowerShell scripts can be run.

#Installing Window Roles and Features
Get-WindowsFeature <--what's been installed on this server?
Get-WindowsFeature -Name Web-Server
Get-WindowsFeature -Name Web-Server | Install-WindowsFeature <--choose this feature and install it
Remove-WindowsFeature

#Creating a Backup Policy for Windows Server 2012 R2
#Get-WindowsFeature -Name Windows-Server-Backup | Install-WindowsFeature
$policy = New-WBPolicy <--give the backup policy a name variable
$fileSpec = New-WBFileSpec -FileSpec C:\important <-- assign what to back up to a variable
Add-WBFileSpec -Policy $policy -FileSpec $filespec <-- plug these variables in to create policy
$backupLocation = New-WBBackupTarget -VolumePath E: <-- assign where to back up to a variable

Add-WBBackupTarget -Policy $policy -Target $backupLocation <-- plug that variables in to existing policy
Set-WBSchedule -Policy $policy 09:00 <--give it a time it should run
Set-WBPolicy -Policy $policy <--- enables the policy
$BUpolicy = Get-WBPolicy <--- assign the current policy to a variable so we can...
Start-WBBackup -Policy $BUpolicy <--- tell the backup policy to go ahead and run now

In this example, it is not quite clear why we need $BUPolicy when we possibly could have just said $policy

Use the -asynch flag to run this in the background (like & in BASH)

Using PS with Office365
- Install MS Online Service Sign-in Assistant for IT Professionals RTW
- Install the Azure AD Module for Win PS
    - Used to be MSOL and is sometimes still called the "MS Online Module"
    - In PS get- set- etc still named like Set-MsolAdministrativeUnit
- Import the Azure AD Module

Begin PluralSight Administering Active Directory Objects With PowerShell

<u>User Administration with *-ADUser Cmdlets</u>

Querying for single objects in AD with parameter -Identity
- Most AD cmdlets have this parameter (7 of 43 do not)
- To determine it's info or if it even exists

Scenario CatharineM calls and can't log in
Get-ADUser -identity CatharineM

```
DistinguishedName : CN=CatharineM,CN=Users,DC=Globomantics,DC=com
Enabled           : True
GivenName         : Catharine
Name              : CatharineM
ObjectClass       : user
ObjectGUID        : a246f65d-8f2b-4e91-b779-34d1ca206dc6
SamAccountName    : CatharineM
SID               : S-1-5-21-3160683373-2897392402-3108432669-1142
Surname           : Morissette
UserPrincipalName :
```

Get-ADUser -identity S-1-5-21-316(ETC) will give you the same information (SID, GUID, DN, and SAM Account Name all work)

PS TIP- Highlight the SID and hit Enter. Says it copies to the clipboard like Ctrl-C and just a right click to paste it in place

PS TIP- Finding the Help files
get-help Get-ADUser -Parameter identity <--This syntax will give just the help info on

this parameter
help Get-ADUser -Parameter filter

Get-ADgroup -identity administrators

Querying for multiple objects in AD with parameter -Filter

Scenario user needs account reactivated, name starts with "C" but we don't know who

Get-ADUser -filter {name -like "C*"} <--get all users whose name starts with a C
Get-ADUser -filter {name -like "C*" -and enabled -eq "FALSE"} <-- same but who's account isn't active?

-not -and -or and some others like -gt -approx available

These commands purposely don't dump tons of info to keep bandwidth and possibly security concerns in mind

To get more info specify -property/-properties not shown in get-member

Get-ADUser -filter {city -eq "Chicago"}
Cmdlet communicates the filter statement to AD, even though we can't see that item in the limited property
list we got before

Get-ADUser -filter * l Select-Object name, city
This does NOT retrieve the city, because the default properties retrieved by Get-ADUser do not include the city property.
[name and department were in the example. Department didn't show in the table for the same reason]

Get-ADUser -filter * -property city l Select-Object name, city
This DOES work, because the default properties retrieved by Get-ADUser are supplemented by the requested
additional city property.

Here we can see that Get-Member doesn't return what we need- we have to specifically query with -property to grab what we need (whether it is department, city, or anything else the general info get-member/ select-object is going to grab). The "-filter {city -eq "Chicago"} query is specific, but the system is made so it doesn't deliver more than required (the security and bandwidth overprotection previously mentioned)

```
PS C:\Users\MikeH> get-aduser -Filter {city -eq "Chicago"} | get-member


   TypeName: Microsoft.ActiveDirectory.Management.ADUser

Name              MemberType            Definition
----              ----------            ----------
Contains          Method                bool Contains(string propertyName)
Equals            Method                bool Equals(System.Object obj)
GetEnumerator     Method                System.Collections.IDictionaryEnum
GetHashCode       Method                int GetHashCode()
GetType           Method                type GetType()
ToString          Method                string ToString()
Item              ParameterizedProperty Microsoft.ActiveDirectory.Manageme
DistinguishedName Property              System.String DistinguishedName {g
Enabled           Property              System.Boolean Enabled {get;set;}
GivenName         Property              System.String GivenName {get;set;}
Name              Property              System.String Name {get;}
ObjectClass       Property              System.String ObjectClass {get;set
ObjectGUID        Property              System.Nullable`1[[System.Guid, ms
SamAccountName    Property              System.String SamAccountName {get;
SID               Property              System.Security.Principal.Security
Surname           Property              System.String Surname {get;set;}
UserPrincipalName Property              System.String UserPrincipalName {g
```

City and department NOT mentioned here.

Get-ADUser -filter {city -eq "Chicago"} -properties city, department

THIS works. Better yet:

Get-ADUser -filter {city -eq "Chicago"} -properties city, department l format-table -property name, city, department

And don't forget using " l out-file c:\services.txt" or removing format-table and using " l out-gridview"

Get-ADOrganizationalUnit -Server parameter takes FQDNs or NetBIOS name - NO IP Addresses!!!

Get-ADOrganizationalUnit -filter * -server mydomainname.com

-searchbase [DistinguishedName]
-searchscope Onelevel (or 1) <---for just the specified level in the object hierarchy
-searchscope subtree (or 2) <--- default behavior - search levels below in the object hierarchy
-searchscope base (or 0) <--- Only validates that object exists. Nothing back means it's there, "Object not found" if not.

Scenario- new hire wants to list coworkers, knows they are in the OU IT
Get-ADUser -SearchBase "OU=Information Technology,DC=mydomainname,DC=Com" -filter *
Add " l select-object Givenname, distinguishedname" to see sub-DNs under IT like "ServerAdmin" for some of the people listed
Below shows the effect of applying "-searchscope Onelevel" (not specifying scope defaulted to "subtree" scope)

We get 3 instead of 11 entries returned

```
PS C:\Users\MikeH> Get-ADUser -SearchBase "OU=Information Technology,DC=Globomantics,
DC=Com" -Filter * | select-object Givenname, DistinguishedName

Givenname                                  DistinguishedName
---------                                  -----------------
Jessie                                     CN=JessieW,OU=Information Technology,D...
Julissa                                    CN=JulissaT,OU=Information Technology,...
Alexa                                      CN=AlexaP,OU=Information Technology,DC...
Audrey                                     CN=AudreyR,OU=ServerAdmins,OU=Informat...
Emma                                       CN=EmmaK,OU=ServerAdmins,OU=Informatio...
Bernita                                    CN=BernitaG,OU=ServerAdmins,OU=Informa...
Gita                                       CN=GitaM,OU=ServerAdmins,OU=Informatio...
Romeo                                      CN=RomeoE,OU=ClientAdmins,OU=Informati...
Janay                                      CN=JanayB,OU=ClientAdmins,OU=Informati...
Monique                                    CN=MoniqueV,OU=ClientAdmins,OU=Informa...
Vanna                                      CN=VannaB,OU=ClientAdmins,OU=Informati...


PS C:\Users\MikeH> Get-ADUser -SearchBase "OU=Information Technology,DC=Globomantics,
DC=Com" -Filter * -SearchScope OneLevel | select-object Givenname, DistinguishedName

Givenname                                  DistinguishedName
---------                                  -----------------
Alexa                                      CN=AlexaP,OU=Information Technology,DC...
Jessie                                     CN=JessieW,OU=Information Technology,D...
Julissa                                    CN=JulissaT,OU=Information Technology,...
```

-LDAPfilter
Leftover from PS 1.0, inherited from VBScript prior to AD modules
Prefix/Polish Notation (-and, -or, -not)
You will see this in old VBScript that haven't been updated to modern PS

Side note on reading output of help files (in this case "help New-ADUser") Using this cmdlet as an example, -Name is the only required parameter, and you know since the entire -OptionName <value> is not enclosed in square brackets like the others.

```
NAME
    New-ADUser
SYNOPSIS
    Creates a new Active Directory user.
SYNTAX
    New-ADUser [-Name] <String> [-AccountExpirationDate <DateTime>]
    [-AccountNotDelegated <Boolean>] [-AccountPassword <SecureString>]
```

New-ADUser JohnDoe <--- (note that -Name isn't necessary. This account will exist but be disabled until adding password))
Remove-ADUser -Identity JohnDoe

Adding a password needs ConvertTo-SecureString so you can send to the domain controller in a secure fashion (from sniffers)

```
PS C:\Users\MikeH> New-ADUser -name BobS -Department Sales -Title Manager -City Bosto
n -AccountPassword "Pass123!"
New-ADUser : Cannot bind parameter 'AccountPassword'. Cannot convert the "Pass123!"
value of type "System.String" to type "System.Security.SecureString".
At line:1 char:86
+ ... ccountPassword "Pass123!"
+                    ~~~~~~~~~~
    + CategoryInfo          : InvalidArgument: (:) [New-ADUser], ParameterBindingEx
   ception
    + FullyQualifiedErrorId : CannotConvertArgumentNoMessage,Microsoft.ActiveDirect
   ory.Management.Commands.NewADUser
```

Removed from this example is overworded discussion on help file: you need to use -Force with -AsPlainText for some reason

```
PS C:\Users\MikeH> $newPassword = ConvertTo-SecureString -String "Pass123!" -AsPlainT
ext -Force
PS C:\Users\MikeH> $newPassword
System.Security.SecureString
PS C:\Users\MikeH> New-ADUser -name BobS -Department Sales -Title Manager -City Bosto
n -AccountPassword $newPassword
```

This is missing the option " -Enabled $true " in order to make the account enabled

Can use CSV file of usernames, etc for input

Running " help -full New-ADUser " reveals that pipelining by Property is allowed, letting us dump input in en masse

```
-City <String>
    Specifies the user's town or city. This parameter sets the City property of
    a user. The LDAP display name (lDAPDisplayName) of this property is l.

    Required?                    false
    Position?                    named
    Default value
    Accept pipeline input?       True (ByPropertyName)
    Accept wildcard characters?  false
```

Here is what we get from HR:

```
"surname","givenname","title","department","city"
"Shryock","Lloyd","Associate","Marketing","Salt Lake City"
"Forsberg","Colene","Manager","Research","Salt Lake City"
"Gillett","Ute","Associate","Marketing","Salt Lake City"
"Vickrey","Kacey","Associate","Sales","Salt Lake City"
"Blauvelt","Tiffani","Associate","Sales","Salt Lake City"
"Maddocks","Rosalee","Associate","Research","Salt Lake City
"Tharpe","Wendi","Associate","Sales","Salt Lake City"
"Dahl","Alisa","Executive","Marketing","Salt Lake City"
"Weisz","James","Executive","Sales","Salt Lake City"
"Lovell","Randall","Associate","Research","Salt Lake City"
"Runner","Eulah","Manager","Sales","Salt Lake City"
"Cronkhite","Erma","Associate","Sales","Salt Lake City"
"Parada","Bret","Associate","Marketing","Salt Lake City"
"Vernon","Sandy","Executive","Sales","Salt Lake City"
```

We are going to need usernames as well.

First, run " Import-Csv .\SaltLakeNewHires.csv | Get-Member " to bring in what we have and display

```
PS C:\Users\MikeH> Import-Csv .\SaltLakeNewHires.csv | Get-Member
   TypeName: System.Management.Automation.PSCustomObject

Name        MemberType   Definition
----        ----------   ----------
Equals      Method       bool Equals(System.Object obj)
GetHashCode Method       int GetHashCode()
GetType     Method       type GetType()
ToString    Method       string ToString()
city        NoteProperty System.String city=Salt Lake City
department  NoteProperty System.String department=Marketing
givenname   NoteProperty System.String givenname=Lloyd
surname     NoteProperty System.String surname=Shryock
title       NoteProperty System.String title=Associate

PS C:\Users\MikeH> Import-Csv .\SaltLakeNewHires.csv | Select-Object -Property *, @{l
abel="name"; expression={$PSItem.givenname + $_.surname.substring(0,1)}}
```

Import-CSV .\SaltLakeNewHires.csv | select-object -property *, @{label="name"; expression={$PSItem.GivenName + $_.Surname.Substring(0,1)}}

There's no distinction to using $PSItem and $_ here. Substring specifies beginning and end (0,1)
Here the lines are broken for to make it more readable:

Import-CSV .\SaltLakeNewHires.csv | select-object -property *, @{label="name"; expression={$PSItem.GivenName + $_.Surname.Substring(0,1)}, @{label="SAMAccountName"; expression={$PSItem.GivenName + $_.Surname.Substring(0,1)}}

Adds a Name and SAMAccountName property to the objects created from the imported CSV data. Again, I was just being inconsistent with the $_ and $PSItem. For some reason, piping accounts to New-ADUser without the SAMAccountName was not working. The first account would get created, but the rest would complain that the name already existed.

Finished product piped to New-ADUser with account-enabling thrown on the end:

Import-CSV .\SaltLakeNewHires.csv | select-object -property *, @{label="name"; expression={$PSItem.GivenName + $_.Surname.Substring(0,1)}, @{label="SAMAccountName"; expression={$PSItem.GivenName + $_.Surname.Substring(0,1)}} | New-ADUser -AccountPassword $newPassword -Enabled $true

You might remember to test adding -Whatif on the end first. Here is what results:

```
PS C:\Users\MikeH> get-aduser -Filter {city -eq "Salt Lake City"} -Properties city, d
epartment, title | format-table name, city, department, title

name                 city                 department           title
----                 ----                 ----------           -----
LloydS               Salt Lake City       Marketing            Associate
ColeneF              Salt Lake City       Research             Manager
UteG                 Salt Lake City       Marketing            Associate
KaceyV               Salt Lake City       Sales                Associate
TiffaniB             Salt Lake City       Sales                Associate
RosaleeM             Salt Lake City       Research             Associate
WendiT               Salt Lake City       Sales                Associate
AlisaD               Salt Lake City       Marketing            Executive
JamesW               Salt Lake City       Sales                Executive
RandallL             Salt Lake City       Research             Associate
EulahR               Salt Lake City       Sales                Manager
ErmaC                Salt Lake City       Sales                Associate
BretP                Salt Lake City       Marketing            Associate
SandyV               Salt Lake City       Sales                Executive
```

Making Changes with Set-ADUser

Promotion
Set-ADUser -Title Manager -Identity KentonS

Martina moves to San Jose
Set-ADUser MartinaM -city "San Jose"
Note that the Identity parameter is positional, so it doesn't need to say Set-ADUser -Identity MartinaM.

Everyone in San Jose is moving to Santa Clara!
Get-ADUser -Filter {city -eq "San Jose"} | Set-ADUser -city "Santa Clara"

We need to record the user's license plates for the security guards to reference.
The -Add parameter is next because CarLicense is not a parameter of Set-ADUser all by itself.
The help file says "Property values that are not associated with cmdlet parameters can be modified by using the add, replace, clear, and remove parameters"
The section on -Add says it takes the form of a hashtable:

```
-Add @{Attribute1LDAPDisplayName=value1, value2, ...;
Attribute2LDAPDisplayName=value1, value2, ...;
AttributeNLDAPDisplayName=value1, value2, ...}
```

Set-ADUser -Identity GuiseppeC -add @{carLicense="GOHAWKS"}
Set-ADUser -Identity GuiseppeC -replace @{carLicense="STNLYCP"}
Set-ADUser -Identity GuiseppeC -clear "carLicense"
Clears the contents of the carLicense attribute of the user, replace updates.

Last name changing (marriages/divorces) - Stored in lots of places!
Name is a reference to CommonName, which is the first part of the DistinguishedName sequence

```
PS C:\Users\MikeH> get-aduser eleanoraP -Properties cn, displayname
CN                : EleanoraP
DisplayName       : Eleanora Peterson
DistinguishedName : CN=EleanoraP,CN=Users,DC=Globomantics,DC=com
Enabled           : True
GivenName         : Eleanora
Name              : EleanoraP
ObjectClass       : user
ObjectGUID        : 42aef9b6-8bb5-4722-807b-d3cad18c4fe9
SamAccountName    : EleanoraP
SID               : S-1-5-21-3160683373-2897392402-3108432669-1157
Surname           : Peterson
UserPrincipalName :
```

Rename-ADObject -Identity EleanoraP -NewName Bob

This fails: the Identity parameter requires a DN, not a SAMAccountName - note it's a -ADObject cmdlet, not a -ADUser cmdlet.

Rename-ADObject -Identity "CN=EleanoraP,CN=Users,DC=Globomantics,DC=com"-NewName EleanoraA

This sets the Name, CN, and DN properties.

Set-ADUser -Identity EleanoraP -Surname "Alanovna" -SamAccountName "EleanoraA" -DisplayName "Eleanora Alanovna"

DisplayName is "Full Name" in the AD GUI tools, but it is also for the Exchange Global Address List (GAL)

This command cleans up the name data that Rename-ADObject doesn't touch.

-Identity here is the SAMAccountName to specify which user to change

Disable/enable/unlock accounts

Disable-ADAccount RubenR
Enable-ADAccount RubenR
Unlock-ADAccount GaryE

You might need to specify -Identity there. The Get-ADUser command's default output doesn't show locked, and it is different than being enabled or not.

Use the -PassThrough option to show changes when a command changes something (so you don't have to (for example) type Get-ADUser. This also creates a .NET object of this information

All of the users in the Boston office need their passwords reset

Get-ADUser -filter {city -eq "Boston"} | Set-ADAccountPassword (ConvertTo-SecureString -String "Pass321!" -AsPlainText -Force) -Reset

Note how this does command insertion in parentheses for our securestring requirement (previously we assigned a variable)

Stale User accounts

Reason- bad user management like onexpiring passwords, not logged into for a long time

Get-ADUser -Filter {lastlogondate -gt "June 16"}
Get-ADUser -Filter {lastlogondate -lt "June 16"}
lastlogondate gets updated approx every 9-14 days
Since the domain controllers have to be polled to get this info, setting this global occasionally prevents forcing DC replication every time someone logs in somewhere to maintain the variable

Search-ADAccount -passwordNeverExpires | Select-Object name | set-ADUser -passwordNeverExpires $false
Fix accounts set to where the password never expires
Search-ADAccount -passwordExpired | Disable-ADAccount
If expired password, disable the account

Search-ADAccount -LockedOut

Search-ADAccount -AccountDisabled
Search-ADAccount -AccountDisabled | Move-ADObject "OU=Disabled Accounts,DC=Globomantics,DC=Com"
Moves accounts to the OU where we store accounts that are not being used.

Search-ADAccount -AccountInactive -DateTime "June 16"
Search-ADAccount -AccountInactive -TimeSpan 90.00:00:00
Retrieve user accounts that have been unused since before the specific date or within the specific span of past time.
Here 90 days and no HH:MM:SS

Search-ADAccount -AccountExpiring -DateTime "June 16"
Search-ADAccount -AccountExpiring -TimeSpan 5.00:00:00
Retrieve user accounts that will expire before the specific date passes or within the specific span of future time.
Search-ADAccount -AccountExpiring -TimeSpan 5
Whole number considered a day. In previous example, follow with a dot and time, or just time is good too

Computer Accounts (mostly the same as user accounts)

Remember NetBIOS name length restrictions.
"The maximum length of the host name and of the fully qualified domain name (FQDN) is 63 bytes per label and 255 bytes per FQDN. *Note Windows does not permit computer names that exceed **15 characters**, and you cannot specify a DNS host name that differs from the NETBIOS host name.*

New-ADComputer -Name SLCReceptionPC -SAMAccountName SaltLakeReceptionPC$ -OperatingSystem "Windows 8.1 Enterprise Edition"
SAMAccountName needs to end with a $
Creates a computer account - complete with a password, and the Enabled property set to TRUE

```
Get-ADComputer -Identity SLCReceptionPC

New-ADComputer -Name TempPC
```
This also works, automatically assigning a dollar sign on the SAMAccountName

```
Remove-ADComputer -Identity TempPC
```
Will ask for confirmation

New-ADComputer works almost just like New-ADUser where you can dump a CSV file in just like before. Using the same file as before:

Building gradually:

```
Import-csv .\SaltLakeNewHires.csv | Select-Object
@{name="Name";expression={$PSItem.GivenName+
$PSItem.Surname.Substring(0,1)+"LPT"}}
```
Creates new pipeline objects with a Name property given as <FirstName><LastInitial><LPT>

```
Import-csv .\SaltLakeNewHires.csv | Select-Object
@{name="Name";expression={$PSItem.GivenName+
$PSItem.Surname.Substring(0,1)+"LPT"}},
@{name="SAMAccountName";expression={$PSItem.GivenName+
$PSItem.Surname.Substring(0,1)+"LPT"}
```
Added line breaks here for readability

Creates new pipeline objects with a Name property given as <FirstName><LastInitial><LPT>, and a duplicate the expression for the SAMAccountName property. We need it to add a dollar sign to the SAMAccountName so fixed below and ready to proceed.

```
Import-csv .\SaltLakeNewHires.csv | Select-Object
@{name="Name";expression={$PSItem.GivenName+
$PSItem.Surname.Substring(0,1)+"LPT"}},@{name="SAMAccountName";expression={$
PSItem.GivenName+$PSItem.Surname.Substring(0,1)+"LPT$"} | New-ADComputer -
Location "Salt Lake City" -OperatingSystem "Windows 8.1 Enterprise Edition"
```
Creates the new Active Directory computer accounts

```
Get-ADComputer -filter {location -eq "Salt Lake City"}
Get-ADComputer -filter {operatingsystem -like "Windows 8*"}
```
Check the work

```
PS C:\Users\MikeH> get-adcomputer lloydslpt


DistinguishedName : CN=LloydSLPT,CN=Computers,DC=Globomantics,DC=com
DNSHostName       :
Enabled           : True
Name              : LloydSLPT
ObjectClass       : computer
ObjectGUID        : af8d4f07-58e3-44f8-903f-440eeb2ece2c
SamAccountName    : LloydSLPT$
SID               : S-1-5-21-3160683373-2897392402-3108432669-2823
UserPrincipalName :
```

Computer accounts are automatically assigned a password on the domain controller (see below)

Note that these computers are not assigned an OU. They need that to apply Group Policy to them.

Get-ADComputer -filter {location -eq "Salt Lake City"} | Move-ADObject -TargetPath "OU=SLCClients,DC=Globomantics,DC=Com"

Moves all SLC computer accounts into the SLCClients OU

Get-ADComputer -filter {location -eq "Salt Lake City"} | select-object -property name, enabled, distinguishedname

---

Note on ComputerAccount Passwords - From Technet:

Computer accounts, by default, are created with a 240-character random password. If you provide a password, an attempt will be made to set that password however, this can fail due to password policy restrictions. The computer account will still be created and you can use Set-ADAccountPassword to set the password on that account. In order to ensure that accounts remain secure, computer accounts will never be enabled unless a valid password is set (either a randomly-generated or user-provided one) or PasswordNotRequired is set to true.

The account is created if the password fails for any reason.

The new ADComputer object will always either be disabled or have a user-requested or randomly-generated password. There is no way to create an enabled computer account object with a password that violates domain password policy, such as an empty password.

The following example shows how to set this parameter. This command will prompt you to enter the password.

-AccountPassword (Read-Host -AsSecureString "AccountPassword")

Disable/ Enable

Get-ADComputer -filter {location -eq "Salt Lake City"} | Disable-ADAccount

Get-ADComputer -filter {location -eq "Salt Lake City"} | Enable-ADAccount

Get-ADComputer -filter {location -eq "Salt Lake City"} | Set-ADComputer -Location

"SLC"
Changes the location property to "SLC" for all computer accounts whose location property currently reads 'Salt Lake City'.

Get-ADComputer -filter {location -eq "SLC"} | Set-ADComputer -OperatingSystem "Windows 8.1 Ultimate Edition"
Changes the OperatingSystem property of all computers with 'SLC' in their location property to "Windows 8.1 Ultimate Edition".

Get-ADComputer -filter {location -eq "SLC"} -property location, operatingsystem | select-object -property name, location, operatingsystem
Retrieves name, location property, and operatingsystem property of all SLC computer accounts.

Add department to all SLC Computer Accounts (the new laptops we just added) Matching the user's department
BEST CASE STUDY IN COMPLICATED FOREACH CONSTRUCTION:
This is video 04_03- Setting Computer Account Properties.mp4

```
PS C:\Users\MikeH> get-aduser -Filter {city -like "Salt*"} -Properties department | s
elect name, department
name                                    department
----                                    ----------
LloydS                                  Marketing
ColeneF                                 Research
UteG                                    Marketing
KaceyV                                  Sales
TiffaniB                                Sales
RosaleeM                                Research
WendiT                                  Sales
AlisaD                                  Marketing
JamesW                                  Sales
RandallL                                Research
EulahR                                  Sales
ErmaC                                   Sales
BretP                                   Marketing
SandyV                                  Sales
```

Syntax--- ForEach-Object { ___ ; ___ ; ___ }
Each part separated by semicolons can have functions

This won't work
Get-ADUser -Filter {city -like "Salt*"} -Properties department | ForEach-Object {Get-ADComputer -Filter {name -like "$($PSItem.GivenName)*"}}
Get the list of Computer accounts, whose name starts with the firstname of any of the Salt Lake users.
With PS pipelining the $PSItem (for ForEach-Object) doesn't contain anything inside of the scriptblock for the filter.

Building up the components
Get-ADUser -Filter {city -like "Salt*"} -Properties department |

```
        ForEach-Object {
                $CompName = "$($PSItem.GivenName)*" ;
                Get-ADComputer -Filter {name -like $CompName}
        }
```

We needed a local variable in the ForEach loop
This produces the list of computers that have names matching the usernames in Salt Lake so we can work with it

```
DistinguishedName : CN=BretPLPT,OU=SLCClients,DC=Globomantics,DC=com
DNSHostName       :
Enabled           : True
Name              : BretPLPT
ObjectClass       : computer
ObjectGUID        : 4777f9cd-f135-42dc-b683-50ca23de4f0d
SamAccountName    : BretPLPT$
SID               : S-1-5-21-3160683373-2897392402-3108432669-2835
UserPrincipalName :
```

Wrapping it up

Set another local variable Dept (just like CompName) inside the ForEach loop
Makes them available inside the filter parameter of Get-ADComputer and the Add parameter of Set-ADComputer.
Pulls the department property off of each user, assigns it to the department property of a computer with a similar first name. .
PSItem is grabbed and thrown in a variable before it can't be accessed anymore, and then the variable can provide the needed info later in the ForEach. Realistically, this should be a script, and definitely something that needs testing along the way with -whatif.

First, here is this again with line breaks and tabs for readability:

```
Get-ADUser -Filter {city -like "Salt*"} -Properties department |
        ForEach-Object {
                $CompName = "$($PSItem.GivenName)*" ;
                $Dept = "$($PSItem.Department) ;
                        Get-ADComputer -Filter {name -like $CompName} |
                                ForEach-Object {
                                        Set-ADComputer -Identity $PSItem -Add
@{department=$Dept}
                                }
                }
```

And here is how it looks on the CLI:

```
Get-ADUser -Filter {city -like "Salt*"} -Properties department | ForEach-Object
{$CompName = "$($PSItem.GivenName)*" ; $Dept = "$($PSItem.Department) ; Get-
ADComputer -Filter {name -like $CompName} | ForEach-Object {Set-ADComputer -
Identity $PSItem -Add @{department=$Dept}}}
```

Managing the Secure Channel

User tries logging in and gets the error
"The trust relationship between this workstation and the primary domain failed"
It means the secure channel failed- the workstation is giving faulty credentials
Reasons: recent backup restoration, machine was of during the maximum period needed to reset it's password (so it expired)

From MS without using PS on Win7:
To resolve this issue, remove the computer from the domain, and then connect the computer to the domain.
1 Use a local administrator account to log on to the computer.
2 Select Start, press and hold (or right-click) Computer > Properties.
3 Select Change settings next to the computer name.
4 On the Computer Name tab, select Change.
5 Under the Member of heading, select Workgroup, type a workgroup name, and then select OK.
6 When you are prompted to restart the computer, select OK.
7 On the Computer Name tab, select Change again.
8 Under the Member of heading, select Domain, and then type the domain name.
9 Select OK, and then type the credentials of the user who has permissions in the domain.
10 When you are prompted to restart the computer, select OK.
11 Restart the computer.

DONT tell the DC to delete the computerAccount and recreate it- it has a SID that is probably in permissions lists, even log subscriptions,
ETC, and will have to be re-added there, etc creating a huge mess. It also leaves dangling SIDs in the permissions list orphaned.

Test-ComputerSecureChannel
Reports if the connection between the DC and client seems to be working. Returns bool

Test-ComputerSecureChannel -Repair
Attempts to reset the secure channel if it has become compromised. Need admin permissions

Reset-ComputerMachinePassword
Attempts to generate a new machine password at the local computer Need admin permissions

Active Directory Groups
Computer accounts in the right groups
Services

Group Scope: Global, DomainLocal, and Universal
- calibrate avail membership and where it is visible, which domains their ACLs will recognized
Group Type/category: Security (SID, permissions), Distribution (email administrators)

PS terminology calls group types group category

New-ADGroup
Get-ADGroup

It is mentioned that converting group type/category from security to distribution drops the SID and status as a 'security principal
However, the image below shows the SID remains. NOT EXPLAINED.

```
DistinguishedName : CN=Boston Marketing,CN=Users,DC=Globomantics,DC=com
GroupCategory     : Security
GroupScope        : Universal
Name              : Boston Marketing
ObjectClass       : group
ObjectGUID        : 04369a98-9520-4368-babf-e5acaea7fb20
SamAccountName    : Boston Marketing
SID               : S-1-5-21-3160683373-2897392402-3108432669-3111
PS C:\Users\MikeH> Get-ADGroup "Boston Marketing" | Set-ADGroup -GroupCategory Distri
bution
PS C:\Users\MikeH> Get-ADGroup "Boston Marketing"
DistinguishedName : CN=Boston Marketing,CN=Users,DC=Globomantics,DC=com
GroupCategory     : Distribution
GroupScope        : Universal
Name              : Boston Marketing
ObjectClass       : group
ObjectGUID        : 04369a98-9520-4368-babf-e5acaea7fb20
SamAccountName    : Boston Marketing
SID               : S-1-5-21-3160683373-2897392402-3108432669-3111
```

New-ADGroup -Name "Research" -GroupScope Global
New-ADGroup -name "Boston Marketing" -GroupScope DomainLocal
New-ADGroup "Chicago Sales Executives" -GroupScope Global
    If you don't set groupscope it will ask you- (mandatory)

Get-ADGroup "Boston Marketing"

```
PS C:\Users\MikeH> Get-ADGroup "Boston Marketing"


DistinguishedName : CN=Boston Marketing,CN=Users,DC=Globomantics,DC=com
GroupCategory     : Security
GroupScope        : DomainLocal
Name              : Boston Marketing
ObjectClass       : group
ObjectGUID        : 04369a98-9520-4368-babf-e5acaea7fb20
SamAccountName    : Boston Marketing
SID               : S-1-5-21-3160683373-2897392402-3108432669-3111
```

Get-ADGroup -filter {name -like "*sales*"}

Retrieves 10 standard properties groups with the word Sales in their names.

Set-ADGroup "Boston Marketing" -GroupScope Universal

Changes the Scope of the "Boston Marketing" group.

Set-ADGroup "Research" -GroupScope DomainLocal

FAILS

AD refuses to allow a group scope change from Global directly to Domain Local, or from Domain Local directly to Global.

Conversion can be made from either scope to Universal, and a Universal scope can be turned into either of the other two scopes.

But direct G -> DL or DL -> G is not permitted, either in PS or in the AD GUIs.

Get-ADGroup "Boston Marketing" | Set-ADGroup -GroupCategory Distribution

Changes the group type (known as a groupCategory) from Security to Distribution. Will DISCARD the SID from the security group, which will no longer be able to be assigned permissions.

Get-ADGroup "Boston Marketing" | Set-ADGroup -ManagedBy WendiT

Changes the managedBy attribute of the group, which is likened to only labeling some name on an org chart.

Does NOT automatically give permission to make changes to the group membership.

That requires changing the permissions on the Modify property of the group object - not easily performed in PS

(Checkbox "manager can change group membership list")

See TechNet article https://technet.microsoft.com/en-us/library/ff730951.aspx

"Research" | Get-ADGroupMember

Same as Get-ADGroupMember "Research"

Apparently it is inherited from older versions of Windows that user accounts don't know what groups they are members of.

It was "fixed" by adding a back-linked attribute in Windows 2000, but remains the case.
That listing is calculated by querying group membership lists to see if the user is listed in them.
Groups know who their members are, so they are queried with Get-ADPrincipalGroupMembership**.**

Add-ADGroupMember vs Add-ADPrincipalGroupMembership
Telling a group that it has a new member vs telling the user it has a new group membership

Remove-ADGroupMember -Identity "VolleyBall" -Members "AsleyE"
Removes Asley from "VolleyBall" - Tells the GROUP which members to remove

Remove-ADPrincipalGroupMembership
Tells the user object which group it is no longer going to be part of

Get-ADGroupMember "Chicago Sales"
Chicago Sales group: get membership list and info

Get-ADPrincipalGroupMembership "NellieP"
User object Nellie is a member of what groups?

Get-ADGroup "Chicago Sales"
Get general group info (it will ask for scope)

Adding people in the department "Research" to a new group "research"
Get-ADUser -filter {department -eq "research"} -Properties department | Add-ADGroupMember "Research"
This doesn't work. Add-ADGroupMember accepts a list of *groups* on the pipeline - not a list of users.
This cmd can be used for the rare circumstances with a list of groups that one user needs to become a member of.
This should be done using Add-ADPrincipalGroupMembership

Get-ADUser -filter {department -eq "research"} -Properties department | Add-ADPrincipalGroupMembership -memberOf "Research"
This works. Those users that have a department property of 'research' are now members of the AD group named 'Research'.

"Boston Marketing, Chicago Sales Executives" | Add-ADGroupMember -Members "NellieP"
Adds User to the membership list of two groups. (This is that rare situation referenced above).

Get-ADGroupMember "Research" | Get-Member
While you might expect that the objects produced by Get-ADGroupMember are

the same User accounts that were piped in, they aren't. They are ADPrincipal objects.

```
PS C:\Users\MikeH> Get-ADGroupMember "Research" | Get-Member
   TypeName: Microsoft.ActiveDirectory.Management.ADPrincipal

Name              MemberType            Definition
----              ----------            ----------
Contains          Method                bool Contains(string propertyName)
Equals            Method                bool Equals(System.Object obj)
GetEnumerator     Method                System.Collections.IDictionaryEnumerator ...
GetHashCode       Method                int GetHashCode()
GetType           Method                type GetType()
ToString          Method                string ToString()
Item              ParameterizedProperty Microsoft.ActiveDirectory.Management.ADPr...
distinguishedName Property              System.String distinguishedName {get;set;}
name              Property              System.String name {get;}
objectClass       Property              System.String objectClass {get;set;}
objectGUID        Property              System.Nullable`1[[System.Guid, mscorlib,...
SamAccountName    Property              System.String SamAccountName {get;set;}
SID               Property              System.Security.Principal.SecurityIdentif...
```

Get-ADGroupMember "Research" | Get-ADUser

This retrieves the User objects that are described in the membership list of the 'Research' group.

Get-ADUser -filter {department -eq "Marketing" -and city -eq "Boston"} | Add-ADPrincipalGroupMembership -MemberOf "Boston Marketing"

Adds users with these two attributes to the Boston Marketing group.

Get-ADGroupMember "Boston Marketing" | Add-ADPrincipalGroupMembership "Volleyball"

Adds all members of the Boston Marketing group to the VolleyBall group (VolleyBall group already exists.)

AD Organizational Units

To apply GPOs to objects

Get-ADOrganizationalUnit -filter *

New-ADOrganizationalUnit -name "Advertising"

New-ADOrganizationalUnit -path "ou=marketing,dc=mydomainname,dc=com" -name "Advertising"

More specific- add it to this existing OU

Get-ADOrganizationalUnit Advertising

Wont work and will say no object with that identity. Help says -Identity <ADOrganizationalUnit> type (not name type)

help Get-ADOrganizationalUnit -parameter identity says it needs to be a DN or objectGUID

Get-ADOrganizationalUnit -identity "ou=advertising,ou=marketing,dc=mydomainname,dc=com"

Yep- you have to type all that junk out.

Set-ADOrganizationalUnit

Remove-ADOrganizationalUnit "ou=advertising,ou=marketing,dc=mydomainname,dc=com"

ACCESS DENIED. Why? whoami and Get-ADPrincipalGroupMembership say we are an administrator! Do this:

Get-ADOrganizationalUnit -identity "ou=advertising,ou=marketing,dc=mydomainname,dc=com" properties *

Aha- check out "ProtectedFromAccidentalDeletion"

```
DistinguishedName               : OU=Advertising,DC=Globomantics,DC=com
dSCorePropagationData           : {9/18/2015 5:40:59 PM, 9/18/2015 5:38:52 PM,
                                  1/1/1601 12:00:00 AM}
instanceType                    : 4
isDeleted                       :
LastKnownParent                 :
LinkedGroupPolicyObjects        : {}
ManagedBy                       :
Modified                        : 9/18/2015 5:40:59 PM
modifyTimeStamp                 : 9/18/2015 5:40:59 PM
Name                            : Advertising
nTSecurityDescriptor            : System.DirectoryServices.ActiveDirectorySecurity
ObjectCategory                  : CN=Organizational-Unit,CN=Schema,CN=Configuration,
                                  DC=Globomantics,DC=com
ObjectClass                     : organizationalUnit
ObjectGUID                      : 5e09fea2-ad4e-45c2-a8de-8fb20a72a47d
ou                              : {Advertising}
PostalCode                      :
ProtectedFromAccidentalDeletion : True
sDRightsEffective               : 15
State                           :
StreetAddress                   :
uSNChanged                      : 68251
uSNCreated                      : 68248
whenChanged                     : 9/18/2015 5:40:59 PM
whenCreated                     : 9/18/2015 5:38:52 PM
```

Set-ADOrganizationalUnit -ProtectedFromAccidentalDeletion $false -identity "ou=advertising,ou=marketing,dc=mydomainname,dc=com"

That will fix it.

Remove-ADOrganizationalUnit "ou=advertising,ou=marketing,dc=mydomainname,dc=com"

Remove- Set- Get- and New ADOrganizationalUnit are the only standard tools specific to OUs

Other tools provide OU-related functionality

Move-ADObject -TargetPath like this:

```
PS C:\Users\MikeH> Get-ADUser -filter "department -eq 'marketing' -and city -eq 'Sant
a Clara'" -Properties department, city | Move-ADObject -TargetPath "ou=advertising,ou
=marketing,dc=globomantics,dc=com"
```

Get-ADOrganizationalUnit "ou=advertising,ou=marketing,dc=mydomainname,dc=com" | move-ADObject -targetpath "ou=sales,dc=mydomainname,dc=com"

It turns out the -ProtectedFromAccidentalDeletion also affects moving OUs around. You can pipe it through the fix above like this:

Get-ADOrganizationalUnit "ou=advertising,ou=marketing,dc=mydomainname,dc=com" |

Set-ADOrganizationalUnit -ProtectedFromAccidentalDeletion $false -passthrough | move-ADObject -targetpath "ou=sales,dc=mydomainname,dc=com"

In order for the piping to work properly passthrough is needed to hand the previously piped value over to move-ADObject

Get-ADObject -searchBase "ou=advertising,ou=marketing,dc=mydomainname,dc=com" *

Get-ADObject has searchBase and searchScope

Another way to do the above is to change directories into the AD drive (cd AD:)
Only present if the Active Directory module has already been imported
Get-ChildItem is sort of like dir here, and we can browse the domain like directories specifying the DN
Similarly Set-Location is the PS equivalent of cd
We can also use relative DN's (aka RDN) instead of writing the whole thing (such as with marketing below)

```
PS C:\Users\MikeH> cd AD:
PS AD:\> Get-ChildItem

Name                 ObjectClass          DistinguishedName
----                 -----------          -----------------
Globomantics         domainDNS            DC=Globomantics,DC=com
Configuration        configuration        CN=Configuration,DC=Globomantics,DC=com
Schema               dMD                  CN=Schema,CN=Configuration,DC=Globomant...
DomainDnsZones       domainDNS            DC=DomainDnsZones,DC=Globomantics,DC=com
ForestDnsZones       domainDNS            DC=ForestDnsZones,DC=Globomantics,DC=com

PS AD:\> cd globomantics
cd : Cannot find path 'AD:\globomantics' because it does not exist.
PS AD:\> cd "dc=globomantics,dc=com"
PS AD:\dc=globomantics,dc=com> dir

Name                 ObjectClass          DistinguishedName
----                 -----------          -----------------
Builtin              builtinDomain        CN=Builtin,DC=Globomantics,DC=com
Computers            container            CN=Computers,DC=Globomantics,DC=com
Domain Controllers   organizationalUnit   OU=Domain Controllers,DC=Globomantics,D...
ForeignSecurityPr... container            CN=ForeignSecurityPrincipals,DC=Globoma...
Information Techn... organizationalUnit   OU=Information Technology,DC=Globomanti...
Infrastructure       infrastructureUpdate CN=Infrastructure,DC=Globomantics,DC=com
LostAndFound         lostAndFound         CN=LostAndFound,DC=Globomantics,DC=com
Managed Service A... container            CN=Managed Service Accounts,DC=Globoman...
Marketing            organizationalUnit   OU=Marketing,DC=Globomantics,DC=com
NTDS Quotas          msDS-QuotaContainer  CN=NTDS Quotas,DC=Globomantics,DC=com
Program Data         container            CN=Program Data,DC=Globomantics,DC=com
SLCClients           organizationalUnit   OU=SLCClients,DC=Globomantics,DC=com
System               container            CN=System,DC=Globomantics,DC=com
TPM Devices          msTPM-Information... CN=TPM Devices,DC=Globomantics,DC=com
Users                container            CN=Users,DC=Globomantics,DC=com

PS AD:\dc=globomantics,dc=com> cd marketing
cd : Cannot find path 'AD:\marketing,dc=globomantics,dc=com' because it does not
PS AD:\dc=globomantics,dc=com> cd "ou=marketing"
PS AD:\ou=marketing,dc=globomantics,dc=com> dir

Name                 ObjectClass          DistinguishedName
----                 -----------          -----------------
Advertising          organizationalUnit   OU=Advertising,OU=Marketing,DC=Globoman...

PS AD:\ou=marketing,dc=globomantics,dc=com> Set-Location "ou=advertising"
PS AD:\ou=advertising,ou=marketing,dc=globomantics,dc=com> Get-ChildItem

Name                 ObjectClass          DistinguishedName
----                 -----------          -----------------
AmeeB                user                 CN=AmeeB,OU=Advertising,OU=Marketing,DC...
AngelinaM            user                 CN=AngelinaM,OU=Advertising,OU=Marketin...
DaniaD               user                 CN=DaniaD,OU=Advertising,OU=Marketing,D...
EdmondM              user                 CN=EdmondM,OU=Advertising,OU=Marketing,...
ElvisG               user                 CN=ElvisG,OU=Advertising,OU=Marketing,D...
FidelaM              user                 CN=FidelaM,OU=Advertising,OU=Marketing,...
LarueA               user                 CN=LarueA,OU=Advertising,OU=Marketing,D...
LavadaR              user                 CN=LavadaR,OU=Advertising,OU=Marketing,...
MeldaG               user                 CN=MeldaG,OU=Advertising,OU=Marketing,D...
SuzannL              user                 CN=SuzannL,OU=Advertising,OU=Marketing,...

PS AD:\ou=advertising,ou=marketing,dc=globomantics,dc=com>
```

Optimizing and Measuring PS AD Cmdlets

Be selective/specific in your queries!

Get-ADUser -filter * | Measure-Object

Returns a measurement object that identifies the total number of users in AD

Get-ADUser -Filter * | where-object {PSItem.city -eq "Boston"} | Select-Object name,department,title

Returns no data - surprise! The objects coming from Get-ADUser have no city property by default - that would need to be added to the -Properties parameter of Get-ADUser

Get-ADUser **-Filter * - properties *** | where-object {PSItem.city -eq "Boston"} | Select-Object name,department,title

Works, but takes 18 seconds to complete! This also took a ton more CPU and RAM in the Performance Monitor

measure-command {Get-ADUser -Filter * | where-object {PSItem.city -eq "Boston"} | Select-Object name,department,title}

Returns an object that describes the time taken to perform the Get-ADUser retrieval

(measure-command {Get-ADUser -Filter * | where-object {PSItem.city -eq "Boston"} | Select-Object name,department,title}).TotalSeconds

Returns only the number of seconds needed to perform the Get-ADUser retrieval

Get-ADUser -Filter {city -eq "Boston"} **-Properties * | Select-Object name,department,title**

Returns same list of users, and faster!

(measure-command {Get-ADUser -Filter {city -eq "Boston"} -Properties * | Select-Object name,department,title}).TotalSeconds

Identifies the much faster retrieval of data - 69.23 times faster than the earlier retrieval!

Get-ADUser -filter {city -eq "Boston"} **-Properties department, title**

Retrieves just the basic 10 properties, plus department and title

Get-ADUser -filter {city -eq "Boston"} **-Properties name, department, title | select-object name, department, title**

Chiseling away most of the basic 10 properties, leaving behind name, department and title. (The boss just wanted those three properties, so I MUST chip away the others with Select-Object.

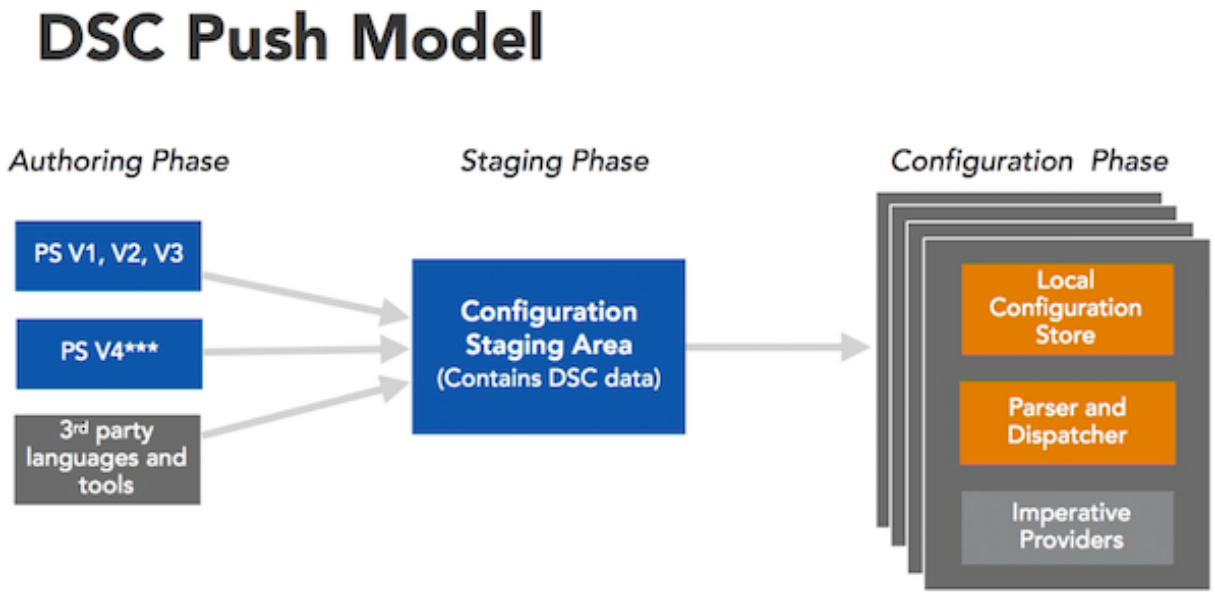
Measure-Object {Get-ADUser -filter {city -eq "Boston"} -Properties name, department, title | select-object name, department, title}

Nice fast retrieval! 6 one-hundredths of a second in this case! Sure beats 18 seconds, doesn't it?

---------- BEGIN POWERSHELL DESIRED STATE CONFIGURATION ESSENTIAL TRAINING

DSC Architecture: Push and Pull models

Authoring phase - describe config, outline websites or registry entries. Imperative or declarative code
Staging phase - converting those configs to MOF file so they can be deployed (MS Operational Framework)
https://technet.microsoft.com/en-us/library/dd320379.aspx
https://en.wikipedia.org/wiki/Microsoft_Operations_Framework
Configuration phase - (automated) Clients pull MOF file or DSC pushes it out to clients



Minimum requirements:
Original released w/ PS4 on Server 2012r2
Win Server 2008r2, Server2008 w/ .NET 4.5, Windows 7
Windows Management Framework (WMF) 4.0 for authoring/ managing
WMF5 w/ PS5 great
Win 8.1 and Server 2012r2 get KB2883200 for optimal DSC performance
Win Remote Mgmt (WinRM) Win Mgmt Instrumentation (WMI) updates usually in WMF latest version with latest PS.
When you download, you will also want to verify that you may need to get items for platforms that you will be managing out to also.

Module name: PSDesiredStateConfiguration

Test-DscConfiguration
Start-DscConfiguration -computername webserver -path ....
Get-DscConfiguration

Get-DscResource (like Get-Member for DSC)
Get-DscResource -Name file | Select-Object -ExpandProperty properties

Built-in resources include supporting local user, local group, registry keys and values, files and folders, managing env vars, archives like zip, managing services (stop start etc) conditions like waitforall, waitforsome, waitforany (check other servers are up before acting on directives- see Cross-Computer Synchronization below)

```
PS C:\> Get-DscResource

ImplementedAs   Name                    ModuleName                    Vers
                                                                      ion
-------------   ----                    ----------                    ----
Binary          File
PowerShell      Archive                 PSDesiredStateConfiguration   1.1
PowerShell      Environment             PSDesiredStateConfiguration   1.1
PowerShell      Group                   PSDesiredStateConfiguration   1.1
Binary          Log                     PSDesiredStateConfiguration   1.1
PowerShell      Package                 PSDesiredStateConfiguration   1.1
PowerShell      Registry                PSDesiredStateConfiguration   1.1
PowerShell      Script                  PSDesiredStateConfiguration   1.1
PowerShell      Service                 PSDesiredStateConfiguration   1.1
PowerShell      User                    PSDesiredStateConfiguration   1.1
PowerShell      WaitForAll              PSDesiredStateConfiguration   1.1
PowerShell      WaitForAny              PSDesiredStateConfiguration   1.1
PowerShell      WaitForSome             PSDesiredStateConfiguration   1.1
PowerShell      WindowsFeature          PSDesiredStateConfiguration   1.1
PowerShell      WindowsOptionalFeature  PSDesiredStateConfiguration   1.1
PowerShell      WindowsProcess          PSDesiredStateConfiguration   1.1


PS C:\> Get-DscResource -Name file |Select-Object -ExpandProperty properties

Name                 PropertyType   IsMandatory Values
----                 ------------   ----------- ------
DestinationPath      [string]              True {}
Attributes           [string[]]           False {Archive, Hidden, ReadOnly...
Checksum             [string]             False {CreatedDate, ModifiedDate...
Contents             [string]             False {}
Credential           [PSCredential]       False {}
DependsOn            [string[]]           False {}
Ensure               [string]             False {Absent, Present}
Force                [bool]               False {}
MatchSource          [bool]               False {}
PsDscRunAsCredential [PSCredential]       False {}
Recurse              [bool]               False {}
SourcePath           [string]             False {}
Type                 [string]             False {Directory, File}
```

Custom Resources
- provide schema definitions
- configuration implementation (code) can be MOF, C#, PS
- provisioned from the Resource Designer tool
- Tons of resources available as PS modules

Build Custom Windows PowerShell Desired State Configuration Resources
https://msdn.microsoft.com/en-us/powershell/dsc/authoringresource

Finding and using resources out there
PowershellGet module for PowerShell Gallery
Find-DscResource

Install-Module
Import-DscResource

http://www.nuget.org/packages">NuGet Packages</A>'

Find-DscResource
Find-DscResource -Repository PSGallery | install-module....

Get-WindowsFeature -computername webserver -name Web-A ....

```
#WebServer Configuration with Content
Configuration IISWebsite
{
   Node webserver              <---hardcoded name of the machine
   {
      WindowsFeature IIS
      {
         Ensure = "Present"        <--- Can be either "present" or "absent"
         Name = "Web-Server"     <--- feature name on the machine you will need to call
on (listed in Get-WindowsFeature)
      }
      WindowsFeature ASP
      {
         Ensure = "Present"
         Name = "Web-ASP-Net45"
      }
       File WebContent
      {
         Ensure = "Present"
         Type = "Directory"
         SourcePath = "C:\Bakery"
         DestinationPath = "C:\inetpub\wwwroot"
         Recurse = $true
      }
   }
}
IISWebSite
```

Running this spits out webserver.mof
Open a remote shell connection to the machine hit play and it deploys/ pushes there

Here we are calling built-in WindowsFeature and File resources
To get the names of the available features, run Get-WindowsFeature

```
PS C:\> Get-WindowsFeature

Display Name                                            Name
------------                                            ----
[ ] Active Directory Certificate Services               AD-Certificate
    [ ] Certification Authority                         ADCS-Cert-Authority
    [ ] Certificate Enrollment Policy Web Service       ADCS-Enroll-Web-Pol
    [ ] Certificate Enrollment Web Service              ADCS-Enroll-Web-Svc
    [ ] Certification Authority Web Enrollment          ADCS-Web-Enrollment
    [ ] Network Device Enrollment Service               ADCS-Device-Enrollment
    [ ] Online Responder                                ADCS-Online-Cert
[X] Active Directory Domain Services                    AD-Domain-Services
[ ] Active Directory Federation Services                ADFS-Federation
[ ] Active Directory Lightweight Directory Services     ADLDS
[ ] Active Directory Rights Management Services         ADRMS
    [ ] Active Directory Rights Management Server       ADRMS-Server
    [ ] Identity Federation Support                     ADRMS-Identity
[ ] Application Server                                  Application-Server
    [ ] .NET Framework 4.5                              AS-NET-Framework
    [ ] COM+ Network Access                             AS-Ent-Services
    [ ] Distributed Transactions                        AS-Dist-Transaction
        [ ] WS-Atomic Transactions                      AS-WS-Atomic
        [ ] Incoming Network Transactions               AS-Incoming-Trans
        [ ] Outgoing Network Transactions               AS-Outgoing-Trans
    [ ] TCP Port Sharing                                AS-TCP-Port-Sharing
```

Cross-Computer Synchronization
waitforall
waitforany
waitforsome
Wait for desired state on all, any, or specific # of target nodes.

```
Configuration wait
{
   Node Web2                    <---------Machine being managed here.  Wait for Web1 to
install first
   {
      WaitForAll WebSite
      {
         ResourceName      = '[WindowsFeature]IIS'
         NodeName          = 'Web1'
         RetryIntervalSec  = 15
         RetryCount        = 30
      }
      WindowsFeature IIS               <---------What to do after wait here
      {
         Ensure = "Present"
         Name = "Web-Server"
         DependsOn = '[WaitForAll]Website'
      }
   }
}
wait
```

Useful for having the computers wait for the DC to be up before attempting to join a domain, for example.

DependsOn. Obviously this shows UserExample has to wait to do anything until GroupExample is set up on Test-PC1:

```
Configuration DependsOnExample {
    Node Test-PC1 {
        Group GroupExample {
            Ensure = "Present"
            GroupName = "TestGroup"
        }

        User UserExample {
            Ensure = "Present"
            FullName = "TestUser"
            DependsOn = "GroupExample"
        }
    }
}
```

In the previous IIS example, we could add this line to the block "File WebContent" to ensure stuff doesn't get loaded in until IIS is confirmed installed.

```
            DependsOn ='[WindowsFeature]IIS'
```

Parameterized Configurations

```
Configuration IISWebsite
{
    Param
    (
        #Node Names
        [parameter(mandatory)]
        [validateNullorEmpty()]
        [string[]]$Nodename,

        #Name of Website
        [parameter(mandatory)]
        [validateNullorEmpty()]
        [string[]]$Websitename
    )

    Node $NodeName
    {
        WindowsFeature IIS
        {
            Ensure = "Present"
```

```
            Name = "Web-Server"
        }
        WindowsFeature ASP
        {
            Ensure = "Present"
            Name = "Web-ASP-Net45"
        }
        File WebContent
        {
            Ensure = "Present"
            Type = "Directory"
            SourcePath = "C:\Bakery"
            DestinationPath = "C:\inetpub\wwwroot"
            Recurse = $true
        }
    }
}
IISWebSite


-----------------------------------------


$MyData =
@{
    AllNodes =
    @(
        @{
            NodeName          = "*"             <-- the asterisk here means for all nodes
regardless of nodename
            LogPath           = "C:\Logs"             <-- and for all nodes apply this
        },
        @{
            NodeName = "VM-1";
            Role     = "WebServer"
            SiteContents = "C:\Site1"
            SiteName = "Website1"
        },
        @{
            NodeName = "VM-2";
            Role     = "SQLServer"
        },
        @{
            NodeName = "VM-3";
            Role     = "WebServer";
            SiteContents = "C:\Site2"
            SiteName = "Website3"
```

```
        }
    );

    NonNodeData =
    @{
        ConfigFileContents = (Get-Content C:\Template\Config.xml)
     }
}

configuration MyConfiguration
{
    Import-DscResource -ModuleName xWebAdministration -Name MSFT_xWebsite

    node $AllNodes.Where{$_.Role -eq "WebServer"}.NodeName
    {
        xWebsite Site
        {
            Name         = $Node.SiteName
            PhysicalPath = $Node.SiteContents
            Ensure       = "Present"
        }

        File ConfigFile
        {
            DestinationPath = $Node.SiteContents + "\\config.xml"
            Contents = $ConfigurationData.NonNodeData.ConfigFileContents
        }
    }
}


Get-WindowsFeature -computername webserver -Name Web-ASP*

Start-DscConfiguration -computername webserver -path iiswebsite -wait -verbose

Start-DscConfiguration -computername webserver -path IISWebsiteProd -wait -verbose

Start-DscConfiguration -computername webserver -path IISWebsiteTest -wait -verbose -
force

--- If you highlight one of these and hit play, in the case it nukes what you had, you can
--- run Restore-configuration and it will undo the change (for example, if a configuration
said Ensure = "absent" instead of "present"

Get-WindowsFeature -computername webserver -Name Web-ASP*

Get-Service -Name W3SVC -ComputerName webserver
```

Get-DscConfiguration

Test-DscConfiguration <---- this checks to see if the configuration matches what it should be (boolean)

LCM is Local Configuration manager- processes the MOF file. Is the "engine of DSC"
It is what reads the configs and determines refresh mode (push or pull) and how often to poll resources

Configuring http/https pull server - needs IIS, DSC service, and the xPSDesiredStateConfiguration module

```
Configuration PullServer {
Import-DscResource -ModuleName xPSDesiredStateConfiguration

        # Load the Windows Server DSC Service feature
        WindowsFeature DSCServiceFeature
        {
          Ensure = 'Present'
          Name = 'DSC-Service'
        }

        # Use the DSC Resource to simplify deployment of the web service
        xDSCWebService PSDSCPullServer
        {
          Ensure = 'Present'
          EndpointName = 'PSDSCPullServer'
          Port = 8080
          PhysicalPath = "$env:SYSTEMDRIVE\inetpub\wwwroot\PSDSCPullServer"
          CertificateThumbPrint = 'AllowUnencryptedTraffic'
          ModulePath =
"$env:PROGRAMFILES\WindowsPowerShell\DscService\Modules"
          ConfigurationPath =
"$env:PROGRAMFILES\WindowsPowerShell\DscService\Configuration"
          State = 'Started'
          DependsOn = '[WindowsFeature]DSCServiceFeature'
        }
}
PullServer -OutputPath 'C:\PullServerConfig\'
Start-DscConfiguration -Wait -Force -Verbose -Path 'C:\PullServerConfig\'
```

Runing this will spit out the MOF file and a corresponding checksum file
This goes in a separate script:

```
configuration Sample_xDscWebService
{
```

```
    param
    (
        [string[]]$NodeName = 'localhost',

        [ValidateNotNullOrEmpty()]
        [string] $certificateThumbPrint,

        [Parameter(Mandatory)]
        [ValidateNotNullOrEmpty()]
        [string] $RegistrationKey
    )


    Import-DSCResource -ModuleName xPSDesiredStateConfiguration

    Node $NodeName
    {
       WindowsFeature DSCServiceFeature
       {
          Ensure = "Present"
          Name   = "DSC-Service"
       }


       xDscWebService PSDSCPullServer
       {
          Ensure                 = "Present"
          EndpointName           = "PSDSCPullServer"
          Port                   = 8080
          PhysicalPath           = "$env:SystemDrive\inetpub\PSDSCPullServer"
          CertificateThumbPrint  = $certificateThumbPrint
          ModulePath             =
"$env:PROGRAMFILES\WindowsPowerShell\DscService\Modules"
          ConfigurationPath      =
"$env:PROGRAMFILES\WindowsPowerShell\DscService\Configuration"
          State                  = "Started"
          DependsOn              = "[WindowsFeature]DSCServiceFeature"
       }

      File RegistrationKeyFile
      {
         Ensure          ='Present'
         Type            = 'File'
         DestinationPath =
"$env:ProgramFiles\WindowsPowerShell\DscService\RegistrationKeys.txt"
         Contents        = $RegistrationKey
      }
```

```
    }
}
```

For CertificateThumbPrint, this will dump out a table for you and then you can assign it to a variable
GUID for the registry key
dir Cert:\LocalMachine\my

Sample_xDSCService -certificateThumbprint 'A7000024B753FA6FFF88E966FD6E19301FAE9CCC' -RegistrationKey '140a952b-b9d6-406b-b416-e0f759c9c0e4' -OutpuPath c:\Configs\PullServer

So then you have your actual configuration file:

```
[DSCLocalConfigurationManager()]
configuration PullClientConfigID
{
    Node localhost
    {
        Settings
        {
            RefreshMode = 'Pull'
            RefreshFrequencyMins = 30
            ConfigurationID = 'fc0b764b-0263-4c0f-afa3-c9c69b243781'
            RebootNodeIfNeeded = $true
        }

        ConfigurationRepositoryWeb webserver
        {
            ServerURL = 'https://webserver:8080/PSDSCPullServer.svc'

        }

    }
}

PullClientConfigID
Set-DscLocalConfigurationManager -path C:\Users\administrator\PullClientConfigID

For SMB share, resources needed, modules xsmshare and cntfsaccesscontrol

Find-DscResource
Find-DscResource -Repository PSGallery
#Set-PSRepository -Name PSGallery -InstallationPolicy Trusted
Find-DscResource -moduleName "xSmbShare" -Repository psgallery IInstall-Module
Find-DscResource -moduleName "cNtfsAccessControl" -Repository psgallery IInstall-
```

Module
Find-DscResource -moduleName "xPSDesiredStateConfiguration" -Repository psgallery IInstall-Module

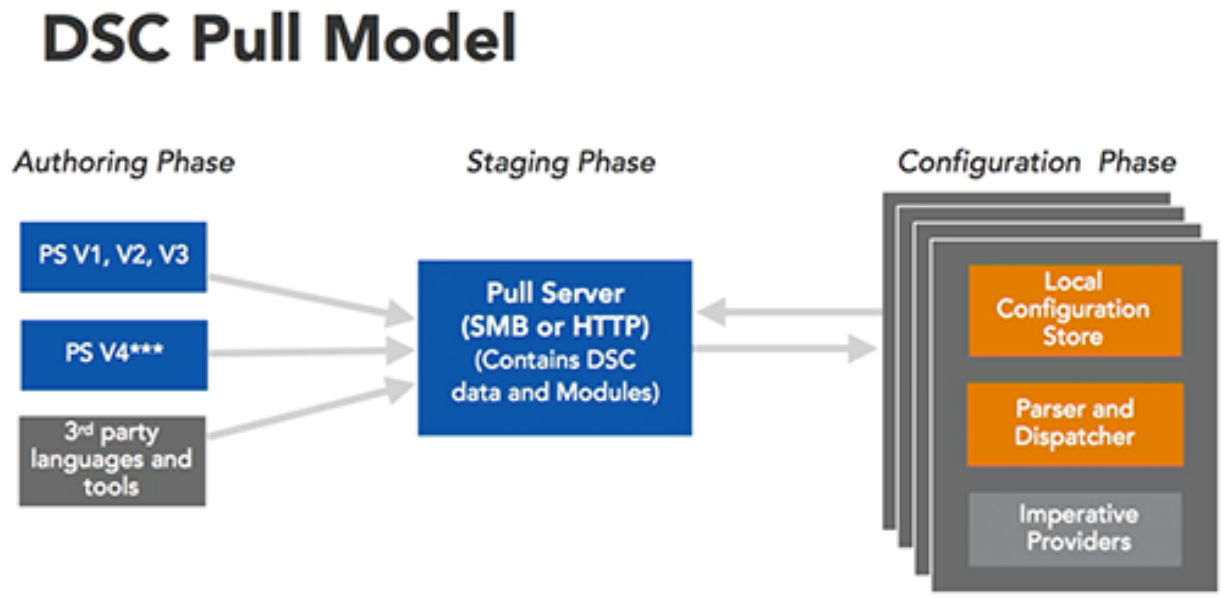
---

```
Configuration DSCSMB {

Import-DscResource -ModuleName PSDesiredStateConfiguration
Import-DscResource -ModuleName xSmbShare
Import-DscResource -ModuleName cNtfsAccessControl

   Node localhost {

      File CreateFolder {

         DestinationPath = 'c:\DscSmbShare'
         Type = 'Directory'
         Ensure = 'Present'

      }

      xSMBShare CreateShare {

         Name = 'DscSmbShare'
         Path = 'c:\DscSmbShare'
         FullAccess = 'administrator','halo\dcdsc$','halo\webserver$'    ---- this line is for convenience in the example
         #ReadAccess = 'halo\dcdsc$,'halo\webserver$'      ---- Real-world use, readaccess is sufficient for just these
         FolderEnumerationMode = 'AccessBased'
         Ensure = 'Present'
         DependsOn = '[File]CreateFolder'

      }

      cNtfsPermissionEntry PermissionSet1 {

      Ensure = 'Present'
      Path = 'C:\DSCSMBshare'
      Principal = 'halo\dcdsc$'
      AccessControlInformation = @(
         cNtfsAccessControlInformation
         {
            AccessControlType = 'Allow'
            FileSystemRights = 'ReadAndExecute'
            Inheritance = 'ThisFolderSubfoldersAndFiles'
```

```
            NoPropagateInherit = $false
        }
    )
    DependsOn = '[File]CreateFolder'

    }
    cNtfsPermissionEntry PermissionSet2 {

    Ensure = 'Present'
    Path = 'C:\DSCSMBshare'
    Principal = 'halo\dscweb$'
    AccessControlInformation = @(
        cNtfsAccessControlInformation
        {
            AccessControlType = 'Allow'
            FileSystemRights = 'ReadAndExecute'
            Inheritance = 'ThisFolderSubfoldersAndFiles'
            NoPropagateInherit = $false
        }
    )
    DependsOn = '[File]CreateFolder'

    }
    cNtfsPermissionEntry PermissionSet3 {

    Ensure = 'Present'
    Path = 'C:\DSCSMBshare'
    Principal = 'halo\webserver$'
    AccessControlInformation = @(
        cNtfsAccessControlInformation
        {
            AccessControlType = 'Allow'
            FileSystemRights = 'ReadAndExecute'
            Inheritance = 'ThisFolderSubfoldersAndFiles'
            NoPropagateInherit = $false
        }
    )
    DependsOn = '[File]CreateFolder'

    }

  }

}
DSCSMB

----------------------------
```

Start-DscConfiguration -ComputerName localhost -path DSCSMB -wait -verbose -Force
  -- using a push configuration to make a pull server
  -- the MOF file that is spit out needs to have the filename start with the GUID to link with target machine
  -- type "New-guid" to spit one out to use
New-DscChecksum -Path $env:PROGRAMFILES\WindowsPowerShell\DscService\Configuration
  -- allows checking that it is a legitimate config file - spits out the same checksum file as mentioned before to go with the MOF

Pull servers



In addition to push and pull mode, there is also configuration mode

```
[DSCLocalConfigurationManager()]
configuration PullClientConfigID
{
   Node localhost
   {
      Settings
      {
         ConfigurationMode = 'ApplyandAutoCorrect' #or can use ApplyOnly or ApplyandMonitor
         RefreshMode = 'Pull'
         ConfigurationID = 'fc0b764b-0263-4c0f-afa3-c9c69b243781'
         RefreshFrequencyMins = 30
         RebootNodeIfNeeded = $true
      }
```

```
        ConfigurationRepositoryShare DSCWeb
        {
            SourcePath = '\\DSCWeb\DscSmbShare'
        }
    }
}
PullClientConfigID

New-guid
Get-DscLocalConfigurationManager
Set-DscLocalConfigurationManager -Path C:
\Users\administrator.HALO\Documents\PullClientConfigID -verbose

Update-DscConfiguration -Verbose

[DSCLocalConfigurationManager()]
configuration PullClientConfigID
{
    Node localhost
    {
        Settings
        {
            RefreshMode = 'Pull'
            ConfigurationID = 'fc0b764b-0263-4c0f-afa3-c9c69b243781'
            RefreshFrequencyMins = 30
            RebootNodeIfNeeded = $true
        }
        ConfigurationRepositoryShare DSCWeb
        {
            SourcePath = '\\DSCWeb\DscSmbShare'

        }
    }
}
PullClientConfigID
```

DSC for Linux

Packages:
nxArchive - syncs a tar or zip with a directory
nxFile - manage files and directories
nxFileLine - manage lines inside a file
nxPackage - manage packages
nxUser, nxGroup
nxScript - run scripts
nxEnvironment - define environmental variables
nxSshAuthorizedKeys - manage public keys

Dependencies:

Required Package	Description	Minimum Version
glibc	GNU Library	2.4–31.30
python	Python	2.4–3.4
omiserver	Open Management Infrastructure	1.0.8.1
openssl	OpenSSL library	0.9.8 or 1.0
ctypes	Python ctypes library	Must match Python version
libcurl	cURL HTTP client library	7.15.1

Get OMI from theopengroup.org
LCM for Linux written in Python w/ C wrapper

This example just drops a file in

```
Configuration ExampleConfiguration{

    Import-DscResource -Name nxFile

    Node  "centosdsc"{
    nxFile ExampleFile {

        DestinationPath = "/tmp/test"
        Contents = "hello world `n"
        Ensure = "Present"
        Type = "File"
        force = $true
    }

    }
}
ExampleConfiguration -OutputPath:"C:\temp"
```

Environmental variables to set
$Node = "CentosDSC"

$Credential = Get-Credential -UserName:"root" -Message:"Enter Password:"

$opt = New-CimSessionOption -UseSsl:$true -SkipCACheck:$true -SkipCNCheck:$true -SkipRevocationCheck:$true

--- this is skipping SSL validation checks, which you wouldn't want in a production environment

$Sess=New-CimSession -Credential:$credential -ComputerName:$Node -Port:5986 -Authentication:basic -SessionOption:$opt -OperationTimeoutSec:90
-- highlight and run to start session

Start-DscConfiguration -Path:"c:\temp" -CimSession:$Sess -Wait -Verbose

Get-DscConfiguration -CimSession:$Sess -Verbose

https://github.com/Microsoft/PowerShell-DSC-for-Linux

Save-Module -Name xDscDiagnostics -Path c:\xdiags

Install-Module -Name xDscDiagnostics

Get-xDscOperation

Get-XDscOperation -ComputerName localhost

Trace-xDscOperation -JobId 339ef6a2-01c9-11e6-80be-00155d199b02

#Remote Note: New-NetFirewallRule -Name "Service RemoteAdmin" -Action Allow

Get-Eventlog
Desired State in Event Viewer

https://en.wikipedia.org/wiki/System_Center_Configuration_Manager
https://technet.microsoft.com/en-us/library/dn958404(v=sc.20).aspx
http://powershelldistrict.com/top-6-things-you-need-to-know-when-scripting-with-sccm/
http://www.dexterposh.com/2014/06/powershell-sccm-2012-getting-started.html
https://github.com/PowerShellMafia/PowerSCCM
https://community.spiceworks.com/topic/1369218-how-use-sccm-to-run-powershell-script-on-user-s-pc
http://gerryhampsoncm.blogspot.com/2013/11/basic-powershell-cmdlets-for-configmgr.html