CIS620 HW 10 (Last updated Apr 13, 6:15am)

PowerShell Tool-Building:
Microsoft Active Directory User Account, Activation, and Password Management

Due: 2016 Apr 17, before midnight (11:59pm)

Name: _____

Score: _____ / 42

Individual Assignment

This is an individual assignment, and you must write your scripts yourself and not get code or direct help on code from anyone else. If you find help online, in books, etc, you must give a reference to where you got it and what you got from there.

You may work together to discuss and understand the requirements, but you should work on your own to figure out how to do what is necessary for completing the assignment. Examples from class should point you very far in the right direction.

In short, the code you submit for this assignment is to be uniquely your own code, not simply copied from somewhere else, copied and modified from somewhere else, or created all or in part by someone else.

See syllabus and notes and class discussion about academic integrity and plagiarism.

If you have any questions about this, talk with the teacher right away, so you are sure that your work and submission is appropriate.

Recommendation/Caution

It is highly recommended that you judiciously use the VMware/vSphere "snapshot" capability so you can restore your Windows server if at some point you make a mess of Active Directory's user accounts, organizational units, etc. Possibly the best way to do this is to make a snapshot right before you start doing anything for this assignment so that you can revert to this original starting point if necessary.

As always, keep only one snapshot, and don't snapshot memory (RAM). I reserve the right to, without notice, delete any machines that have multiple snapshots and/or have snapshotted RAM, since doing so can lead to using more disk space than we can afford to be using on our system.

Of course, if you are creating your scripts on your Windows Server (as opposed to eLab or some other location), you would need to download them to another location before restoring a snapshot… or you would totally lose the scripts and have to recreate them from scratch.

Overview

Here are some *general requirements* for this assignment:

- You will create a set of scripts for managing Active Directory (AD) User Accounts on your Windows domain or another server, as specified in the scripts' parameters. From a high-level perspective, these scripts will manage user account creation/deletion, querying, activation/deactivation, and password management.
- You will write all these scripts on eLab or your home computer (if you choose to use a machine other than eLab, you will be responsible for making sure the Powershell Active Directory module is installed correctly on that machine; it is already available on eLab) and when you have completed the assignment you will ZIP up the files and post them as part of your submission in RamCT. Name the ZIP file HW10-LLLLFF.zip, where LLLLFF is the account name you have been assigned this semester. For example, for "Sally Brown", LLLLFF would be BrowSa, and the ZIP file would be named HW10-BrowSa.ZIP.
- Your scripts will be written to interact with whatever server is specified in the *server* parameter for each script, and to use the credentials of whatever user is specified in the *credential* parameter of each script. Every script you write will require these two parameters. They are mandatory and do not accept input from the pipeline.
- Your scripts will be able to manage Active Directory user accounts on any server, but I will test them with my server, c6TurkDa.TurkDom.Net, 10.6.20.51, when we are grading your submission. You will be given an account and organizational unit on that machine within which and through which you may manage accounts. For instance, a user by the name of "Sally Brown" would have an account on this server named "HW10_BrowSa". Notice that this is the normal LLLLFF account naming convention, but with "HW10_" added on the front. The password for your account will be the one we have received from you that you have been using for your VMs this semester. You cannot log in to my server; rather, you will perform all work remotely by issuing Powershell AD commands from the command-line, and, ultimately, from within your scripts. You can and should also ensure that your scripts work with your own Windows Server, the one you cloned/created earlier in the semester
- Because you will be managing accounts only in your specific Active Directory Organizational Unit (OU) on my machine, your scripts will also all need to be able to *optionally* specify a "path" or OU designation in which they should work. See the individual script specifications below for more details.
- You will name the scripts exactly as named in the detailed instructions below. Notice that these names have simple hyphens (dashes) in them. Be sure to name them correctly, including upper/lower-case / CamelCase style. They will, with some small variations as described below, follow the standard Powershell cmdlet "verb-noun" naming convention.
- (36 pts – 4 pts per script) You will create three (3) complete groups of scripts. The first group is A, the second, B, and the third C. The set of scripts within each group are designed to manage a specific aspect of user accounts in Active Directory, working together to provide the full functionality desired. The scripts in all of the groups also work together, some depending on others to provide functionality. Thus, for example, if your user account management scripts don't work, or not correctly, then it is likely that your user activation and/or password

management scripts may not work as expected/required and vice versa. However, each script will be graded and awarded points independently (as much as possible, given the interdependencies among them).

- These scripts will function as "cmdlets", not as "functions".
- You will use the BEGIN, PROCESS, and END block design when building your scripts.
- You will implement the specified named parameters and ability to pipe certain types of information into your scripts. You will implement parameters via the "CmdletBinding" and "param" / "Parameter" blocks in your script.
- You will generate a single consistent kind of output to the pipeline for each script, so that it can be used as a general-purpose tool.
- You will generate "Verbose" output via the Write-Verbose mechanism when the user specifies the "-Verbose" common parameter.
- You will generate "Error" output via the Write-Error mechanism.
- You will handle errors "gracefully" so that your scripts don't crash; rather, they continue processing additional data, but send error information to the error pipeline via Write-Error. Try-Catch and If statements are the standard ways that you will do this, depending on the type of error that needs to be handled.
- In addition to standard descriptive comment blocks and statements at the beginning of and throughout your scripts, you will also include comment blocks that will facilitate standard Powershell "Get-Help" functionality for your scripts.

Script Set A: User Account Management

This set of scripts provides functionality to manage user accounts. It allows accounts to be created, listed, and removed. The three scripts you will create in this group are:

1. HW10A1-Get-User.ps1
2. HW10A2-New-User.ps1
3. HW10A3-Remove-User.ps1

In addition to the general requirements detailed above, the specific design for each of these scripts is spelled out below.

HW10A1-Get-User.ps1

Gets Active Directory users.

Parameters *server* and *credential* as described in the Overview above.

Parameter *filter* – optional, cannot receive input from the pipeline, accepts filter specifications in the same format at the built-in Powershell cmtlet, Get-ADUser.

If no filter is given, it lists all AD users. If the filter is given, it lists only those matching the filter specification.

The output of this script is user account objects.

You will use the Get-ADUser cmdlet inside your script to implement this functionality.

Some Examples:

HW10A1-Get-User.ps1 -server 129.82.40.143 -credential (Get-Credential)

Lists all information about all AD Users.

HW10A1-Get-User.ps1 -server $server -credential $credAdmin -filter *
| Sort-Object
| Format-Table Name, SAMAccountName, DistinguishedName -AutoSize

Displays in table format (auto-sizing the column widths) the Name, SAMAccountName, and DistinguishedName of all AD users, in order by the default sort field.

(HW10A1-Get-User.ps1 -server 129.82.40.143 -credential (Get-Credential)
-filter 'Name -Like "HW10*"').name

Gets only the name of the accounts of the AD users whose account names start with "HW10".

HW10A1-Get-User.ps1 -server 129.82.40.143 -credential (Get-Credential)
-filter 'DistinguishedName -Like "*HW10_0tesT1"'

Gets only the accounts of the AD users who are in the “HW10_0tesT1” Organizational Unit.

HW10A1-Get-User.ps1 -server $server -credential $credAdmin -filter *
| Where-Object DistinguishedName -Like "*OU=HW10OU_0tesT1*"
| Sort-Object
| Format-Table Name, SAMAccountName, DistinguishedName

Displays in table format (*not* auto-sizing the column widths) the Name, SAMAccountName, and DistinguishedName of all AD users who are in an OU whose name starts with “HW10OU_0tesT1”, in order by the default SAMAccountName. You cannot “filter”, with Get-ADUser, by DistinguishedName, thus our use of Where-Object (?) to accomplish this.

HW10A2-New-User.ps1

Creates Active Directory users.

Parameters *server* and *credential* as described in the Overview above.

Parameter *path* – optional, cannot receive input from the pipeline, specifies the OU in which to create the new user. You will always need to use this for this assignment, even though it is “optional” in terms of how you define it in your script, because you only have permission to write to your OU on my Windows Domain Controller. The path is specified in the “DistinguishedName” format of “OU=<your OU Name>,OU=c6-2016-1,DC=c6TurkDa,DC=TurkDom,DC=Net”. Of course, your script should allow any path to be specified (or no path, since it is optional), not just your own, and not just on my Domain Controller.

Parameter *userName* – mandatory, can receive input from the pipeline, allows multiple items.

Parameter *prefix* – optional, cannot receive input from the pipeline, prefixes each account name to be created with this value.

Parameter *password* – mandatory, cannot receive input from the pipeline. This accepts a *SecureString* from the command-line. You can use the Powershell cmdlet *Read-Host* (with proper parameters) to get a *SecureString* from the user, while not showing it on the screen. You can use the Powershell cmdlet *ConvertTo-SecureString* (with proper parameters) to create a *SecureString* from a plaintext string.

Account names are created from the first four characters of the last name and the first two characters of the first name of each user name provided. (This is the same as how we have been creating account names this semester.) A prefix, if specified in its parameter, is added to the beginning of this to compose the complete account name.

Accounts should be automatically enabled by default as they are created.

The output of this script is account names.

You will use the New-ADUser cmdlet inside your script to implement this functionality.

Example:

```
HW10A2-New-User.ps1 -server $sv -credential $cr
-userName "Sally Smith","Jason Jones","Allison Smith Jones"
-prefix "HW10_" -Verbose
```

```
echo "Sally Smith","Jason Jones","Allison Smith Jones"
| HW10A2-New-User.ps1 -server $sv -credential $cr -prefix "HW10_"
```

Adds users HW10_smithsa, HW10_joneja, and HW10_joneal, if they do not already exist. First example also shows “verbose” output.

```
HW10A2-New-User.ps1 -server $sv -credential $cr
-userName "Sally Smith","Jason Jones","Allison Smith Jones"
-prefix "HW10_" -path "OU=HW10OU_0tesT1,DC=c0,DC=CIS620,DC=Net" -Verbose
```

Adds users HW10_smithsa, HW10_joneja, and HW10_joneal to the “HW10OU_0tesT1” OU, if they do not already exist. Displays “verbose” output.

HW10A3-Remove-User.ps1

Removes Active Directory users.

Parameters *server* and *credential* as described in the Overview above.

Parameter *accountName* – mandatory, can receive input from the pipeline, allows multiple items.

Parameter *filter* – mandatory, cannot receive input from the pipeline, accepts filter specifications in the same format at the built-in Powershell cmtlet, Get-ADUser.

accountName and *filter* parameters are mutually exclusive; one or the other may be specified, with *accountName* being the default. You can use *ParameterSetName* and *DefaultParameterSetName* CmdletBinding and Parameter values to implement this functionality.

Parameter *confirm* – optional, cannot receive input from the pipeline. Specifies whether confirmation is required or not. Default value is True, that confirmation is required. This is especially good for the case when someone runs the script with a filter of "*", as without *confirm* the default would be to blindly remove *all* AD users… ☺

Even with confirm:$true, your script is not to remove all users when the filter is *'*'* or *'Name -Like "*"'* (same as *{ Name -Like "*" }*).

The output of this script is user account names.

You will use the Remove-ADUser cmdlet inside your script to implement this functionality.

Some Examples:

```
HW10A3-Remove-User.ps1 -server $sv -credential $cr
        -accountName user1,user2 -Verbose -confirm:$false

HW10A3-Remove-User.ps1 -server $sv -credential $cr
        -accountName user1,user2 -Verbose -confirm:$true

echo user1,user2 | HW10A3-Remove-User.ps1 -server $sv -credential $cr

(HW10A1-Get-User.ps1 -server $sv -credential $cr -filter 'Name -Like "HW10_*"').Name
        | HW10A3-Remove-User.ps1 -server $sv -credential $cr -confirm:$false

HW10A3-Remove-User.ps1 -accountName (Get-Content fileNameOfUserNames)
        -Verbose -confirm:$false
```

Removes specified users. First and last examples do not require confirmation.

```
HW10A3-Remove-User.ps1 -server $sv -credential $cr -filter 'Name -Like "HW10*"'
```

Removes AD users whose account names start with "HW10". Requires confirmation.

Script Set B: User Activation Management

This set of scripts provides functionality to manage user account activation/enabling and deactivation/disabling. It allows accounts to be enabled, disabled, and their status listed. The three scripts you will create in this group are:

1. HW10B1-Get-UserActivationStatus.ps1
2. HW10B2-Enable-User.ps1
3. HW10B3-Disable-User.ps1

In addition to the general requirements detailed above, the specific design for each of these scripts is spelled out below.

HW10B1-Get-UserActivationStatus.ps1

Gets the Activation Status of Active Directory users.

Parameters *server* and *credential* as described in the Overview above.

Parameter *filter* – optional, cannot receive input from the pipeline, accepts filter specifications in the same format at the built-in Powershell cmtlet, Get-ADUser.

If no filter is given, it lists all AD users. If the filter is given, it lists only those matching the filter specification.

The output of this script is a set of Powershell Objects, each of which has the properties SAMAccountName, Name, and Enabled, with values corresponding to those in the same-named attributes from the AD User Account Objects retrieved.

The parameters for the New-ADUser cmdlet let you specify what attributes of an account object you want, the Select-Object cmdlet lets you grab this information in a way that will let you output it as desired, and the Format-Table cmdlet lets you display this in the way desired. You will likely want to look at the help for these cmdlets to assist you in implementing this script as easily as possible.

You will use the Get-ADUser cmdlet inside your script to implement this functionality.

Some Examples:

HW10B1-Get-UserActivationStatus.ps1 -server $sv -credential $cr

Lists account name, real name, and activation status for all AD Users.

HW10B1-Get-UserActivationStatus.ps1 -server $sv -credential $cr
-filter 'Name -Like "HW10*"'

~~This script can also take piped output from HW10A1-Get-User.ps1, similar to how it is used in one of the HW10A3-Remove-User.ps1 examples. [This requirement was DELETED on 2016 Apr 11.] [This would actually be very easy to implement. In fact, HW10B2 and HW10B3 are implemented this way by making the *filter* and *accountName* parameters mutually exclusive…]~~

HW10B2-Enable-User.ps1

Enables Active Directory users.

Parameters *server* and *credential* as described in the Overview above.

Parameter *accountName* – mandatory, can receive input from the pipeline, allows multiple items.

Parameter *filter* – mandatory, cannot receive input from the pipeline, accepts filter specifications in the same format at the built-in Powershell cmtlet, Get-ADUser.

accountName and *filter* parameters are mutually exclusive; one or the other may be specified, with *accountName* being the default. You can use ParameterSetName and DefaultParameterSetName CmdletBinding and Parameter values to implement this functionality.

Parameter *confirm* – optional, cannot receive input from the pipeline. Specifies whether confirmation is required or not. Default value is True, that confirmation is required. This is especially good for the case when someone runs the script with a filter of "*", as without *confirm* the default would be to blindly enable *all* AD users… ☺

Even with confirm:$true, your script is not to enable all users when the filter is *'*'* or *'Name -Like "*"'* (same as *{ Name -Like "*" }*).

The output of this script is a set of SAMAccountName objects for the accounts that have been enabled.

Any list of AD User Objects or SAMAccountNames will be able to be piped into and processed by this script. At least one example of how this might be done is illustrated in the examples below.

You will use the Enable-ADAccount cmdlet inside your script to implement this functionality.

Some Examples:

```
HW10B2-Enable-User.ps1 -server $sv -credential $cr -filter 'Name -Like "HW10*"'

HW10B2-Enable-User.ps1 -server $sv -credential $cr
        -filter 'Name -Like "HW10*"' -confirm:$False

(HW10A1-Get-User.ps1 -server $sv -credential $cr
        -filter 'Name -Like "HW10*"').SAMAccountName
        | HW10B2-Enable-User.ps1 -server $sv -credential $cr

(HW10A1-Get-User.ps1 -server $sv -credential $cr
        -filter 'Name -Like "HW10*"').SAMAccountName
        | HW10B2-Enable-User.ps1 -confirm:$False -server $sv -credential $cr
```

Enables all AD Users whose account names start with "HW10"

-confirm:$False does not require confirmation before carrying out the command.

HW10B3-Disable-User.ps1

Disables Active Directory users.

Parameters *server* and *credential* as described in the Overview above.

Parameter *accountName* – mandatory, can receive input from the pipeline, allows multiple items.

Parameter *filter* – mandatory, cannot receive input from the pipeline, accepts filter specifications in the same format at the built-in Powershell cmtlet, Get-ADUser.

accountName and *filter* parameters are mutually exclusive; one or the other may be specified, with *accountName* being the default. You can use ParameterSetName and DefaultParameterSetName CmdletBinding and Parameter values to implement this functionality.

Parameter *confirm* – optional, cannot receive input from the pipeline. Specifies whether confirmation is required or not. Default value is True, that confirmation is required. This is especially good for the case when someone runs the script with a filter of "*", as without *confirm* the default would be to blindly disable *all* AD users… ☺

Even with confirm:$true, your script is not to disable all users when the filter is *'*'* or *'Name -Like "*"'* (same as *{ Name -Like "*" }*).

The output of this script is a set of SAMAccountNames for the accounts that have been disabled.

Any list of AD User Objects or SAMAccountNames will be able to be piped into and processed by this script. At least one example of how this might be done is illustrated in the examples below.

You will use the Disable-ADAccount cmdlet inside your script to implement this functionality.

Some Examples:

same style as for HW10B2-Enable-User.ps1 .

Script Set C: User Password Management

This set of scripts provides functionality to manage user account passwords. It allows passwords to be set, rules associated with password changes specified, and password status listed. The three scripts you will create in this group are:

1. HW10C1-Get-PasswordRule.ps1
2. HW10C2-Set-Password.ps1
3. HW10C3-Set-PasswordRule.ps1

In addition to the general requirements detailed above, the specific design for each of these scripts is spelled out below.

HW10C1-Get-PasswordRule.ps1

Gets password "rules" for Active Directory users.

Password "Rules" for our purpose include whether the user must change their password the next time they logon, whether they can change their password at all, and whether their password ever expires. The parameters for setting each of these are described below.

Parameters *server* and *credential* as described in the Overview above.

Parameter *accountName* – mandatory, can receive input from the pipeline, allows multiple items.

Parameter *filter* – mandatory, cannot receive input from the pipeline, accepts filter specifications in the same format at the built-in Powershell cmtlet, Get-ADUser.

accountName and *filter* parameters are mutually exclusive; one or the other may be specified, with *accountName* being the default. You can use ParameterSetName and DefaultParameterSetName CmdletBinding and Parameter values to implement this functionality.

The output of this script is a set of Powershell Objects, each of which has the properties SAMAccountName, Name, MustChangePWAtNextLogon, CanChangePW, and PWNeverExpires, with appropriate values corresponding to those in the SAMAccountName, Name, PasswordExpired, CannotChangePassword (actually this one will be the inverse value), and PasswordNeverExpires attributes from the AD User Account Objects retrieved.

Any list of AD User Objects will be able to be piped into and processed by this script. At least one example of how this might be done is illustrated in the examples below.

You will use the Get-ADUser cmdlet inside your script to implement this functionality.

Some Examples:

```
HW10C1-Get-PasswordRule.ps1 -server $sv -credential $cr
        -filter 'Name -Like "HW10*"'
```

```
HW10A1-Get-User.ps1 -server $sv -credential $cr -filter 'Name -Like "HW10*"'
        | HW10C1-Get-PasswordRule.ps1 -server $sv -credential $cr
        | Format-Table -AutoSize
```

Lists SAMAccountName, Name, MustChangePWAtNextLogon, CanChangePW, and PWNeverExpires for all AD Users whose account names start with “HW10”

```
HW10A1-Get-User.ps1 -server 129.82.40.143 -credential (Get-Credential)
        -filter 'DistinguishedName -Like "*HW10OU_0tesT1"'
        | HW10C1-Get-PasswordRule.ps1 -server $sv -credential $cr
        | Format-Table SAMAccountName, MustChangePWAtNextLogon
```

Lists SAMAccountName, MustChangePWAtNextLogon for all AD Users who are in the “HW10OU_0tesT1” Organizational Unit.

HW10C2-Set-Password.ps1

Sets passwords for Active Directory users.

Parameters *server* and *credential* as described in the Overview above.

Parameter *accountName* – mandatory, can receive input from the pipeline, allows multiple items.

Parameter *filter* – mandatory, cannot receive input from the pipeline, accepts filter specifications in the same format at the built-in Powershell cmtlet, Get-ADUser.

accountName and *filter* parameters are mutually exclusive; one or the other may be specified, with *accountName* being the default. You can use ParameterSetName and DefaultParameterSetName CmdletBinding and Parameter values to implement this functionality.

Parameter *password* – required, cannot receive input from the pipeline, this parameter is in the format of a Powershell SecureString.

Parameter *mustChangePWAtNextLogon* – optional, cannot receive input from the pipeline. Default value is True.

Parameter *confirm* – optional, cannot receive input from the pipeline. Specifies whether confirmation is required or not. Default value is True, that confirmation is required. This is especially good for the case when someone runs the script with a filter of "*", as without *confirm* the default would be to blindly reset the password for *all* AD users… ☺

Even with confirm:$true, your script is not to set the password for all users when the filter is *'*'* or *'Name -Like "*"'* (same as *{ Name -Like "*" }*).

The output of this script is account names.

Any list of AD User Objects will be able to be piped into and processed by this script. At least one example of how this might be done is illustrated in the examples below.

You will use the Set-ADAccountPassword cmdlet inside your script to implement this functionality.

Some Examples:

```
HW10C2-Set-Password.ps1 -server $sv -credential $cr
        -accountName HW10_BrowSa
        -password $securePassword

HW10C2-Set-Password.ps1 -server $sv -credential $cr
        -accountName HW10_BrowSa,HW10_SmitSa
        -password (Read-Host -Prompt "Enter password: " -AsSecureString)
```

Sets (resets) password for specified user(s) with accountName(s) to the value stored in the SecureString $securePassword or provided in the Read-Host interaction.

```
HW10C2-Set-Password.ps1 -server $sv -credential $cr
        -filter 'Name -Like "HW10_*"'
        -password $securePassword

HW10A1-Get-User.ps1 -server $sv -credential $cr
        -filter 'Name -Like "HW10_*"'
        | HW10C2-Set-Password.ps1 -server $sv -credential $cr
        -password ("ABCxyz123@#" | ConvertTo-SecureString -AsPlainText -Force)
```

Sets (resets) password for all AD Users whose account names start with "HW10_" to the value stored in the SecureString $securePassword or converted from plaintext to a SecurePassword while passing the password in.

HW10C3-Set-PasswordRule.ps1

Sets password "rules" for Active Directory users.

Parameters *server* and *credential* as described in the Overview above.

Parameter *accountName* – mandatory, can receive input from the pipeline, allows multiple items.

Parameter *filter* – mandatory, cannot receive input from the pipeline, accepts filter specifications in the same format at the built-in Powershell cmtlet, Get-ADUser.

accountName and *filter* parameters are mutually exclusive; one or the other may be specified, with *accountName* being the default. You can use ParameterSetName and DefaultParameterSetName CmdletBinding and Parameter values to implement this functionality.

Parameter *mustChangePWAtNextLogon* – optional, cannot receive input from the pipeline. Switch. Default value is False if parameter is used. If parameter is not used, this attribute of the specified user account is not changed.

Parameter *canChangePW* – optional, cannot receive input from the pipeline. Switch. Default value is True if parameter is used. If parameter is not used, this attribute of the specified user account is not changed.

Parameter *pwNeverExpires* – optional, cannot receive input from the pipeline. Switch. Default value is True if parameter is used. If parameter is not used, this attribute of the specified user account is not changed.

If *none* of the three parameters above (*mustChangePWAtNextLogon*, *canChangePW*, *pwNeverExpires*) are specified, the script provides an error-message indicating that at least one of these parameters much be used.

Parameter *confirm* – optional, cannot receive input from the pipeline. Specifies whether confirmation is required or not. Default value is True, that confirmation is required. This is especially good for the case when someone runs the script with no filter, as without *confirm* the default would be to blindly set the password rules for *all* AD users… ☺

Even with confirm:$true, your script is not to set the password rule for all users when the filter is *'*'* or *'Name -Like "*"'* (same as *{ Name -Like "*" }*).

The output of this script is a set of Powershell Objects, each of which has the properties SAMAccountName, Name, MustChangePWAtNextLogon, CanChangePW, and PWNeverExpires, with values corresponding to those in the same-named attributes from the AD User Account Objects processed.

Any list of AD User Objects will be able to be piped into and processed by this script. At least one example of how this might be done is illustrated in the examples below.

You will use the Set-ADUser cmdlet inside your script to implement this functionality.

Some Examples:

HW10C3-Set-PasswordRule.ps1 -server $sv -credential $cr
-filter 'Name -Like "HW10*"' -mustChangePWAtNextLogon -canChangePW -pwNeverExpires

HW10C3-Set-PasswordRule.ps1 -server $sv -credential $cr
-filter 'Name -Like "HW10*"' -mustChangePWAtNextLogon

HW10C3-Set-PasswordRule.ps1 -server $sv -credential $cr
-filter 'Name -Like "HW10*"' -canChangePW$false

HW10C3-Set-PasswordRule.ps1 -server $sv -credential $cr
-filter 'Name -Like "HW10*"' -pwNeverExpires

HW10A1-Get-User.ps1 -server $sv -credential $cr
-filter 'Name -Like "HW10*"'
| HW10C3-Set-PasswordRule.ps1 -server $sv -credential $cr
-mustChangePWAtNextLogon

HW10A1-Get-User.ps1 -server $sv -credential $cr
-filter 'Name -Like "HW10*"'
| HW10C3-Set-PasswordRule.ps1 -server $sv -credential $cr
-canChangePW:$false

HW10A1-Get-User.ps1 -server $sv -credential $cr
-filter 'Name -Like "HW10*"'
| HW10C3-Set-PasswordRule.ps1 -server $sv -credential $cr
-pwNeverExpires

Sets the specified password "rule" to True for all AD Users who account names start with "HW10"

HW10A1-Get-User.ps1 -server $sv -credential $cr
-filter 'Name -Like "HW10*"'
| HW10C3-Set-PasswordRule.ps1 -server $sv -credential $cr
-pwNeverExpires

Displays error message indicating that at least one of the three password "rule" parameters must be specified, as shown below:
ERROR: At least one of the password "rule" parameters must be specified.
You can use Powershell ParameterSet's to enforce this.

Submit

Submit the following on Canvas:

1. Make sure all your scripts are named correctly as specified in this assignment, and ZIP them all up and post them as your submission in RamCT for this assignment. Name the ZIP file HW10-LLLLFF.zip, where LLLLFF is the account name you have been assigned this semester. For example, for "Sally Brown", LLLLFF would be BrowSa, and the ZIP file would be named HW10-BrowSa.ZIP.

Notes about Appropriate Use

It is assumed that you will use these machines responsibly. That means not using them to do anything illegal, and not using them to try to create havoc or create problems (intentionally, in fun, or anything else) for other teams / team members. It means that you will not log into other team's machines, and will not disturb them or their connections in any way. The class where you can do these types of things is the "hacking" class... You can take it after you have finished this class! :-) I will take any infractions against these rules very seriously.