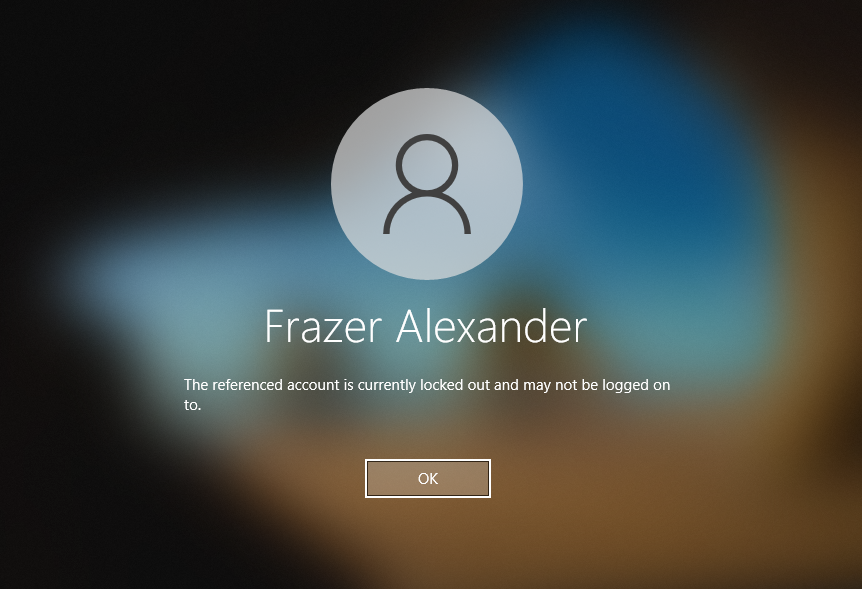
Powershell Password Reset Script

Powershell Scripts Tutorial

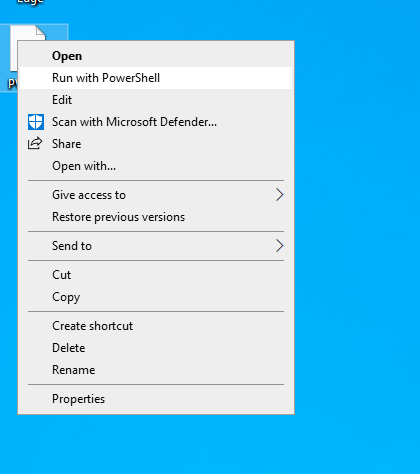
This PowerShell script will allow you to set user passwords from your desktop computer without needing to connect to the Domain Controller. It will also ensure that all users have their passwords reset to a good value that is in keeping with the company policy.

**Section One: Demonstration of the Script in Action**

Our test user Frazer calls to say that his account has been locked out.

****

On our own laptop computer, we simply go to the PowerShell script, right click it and choose Run with Powershell:

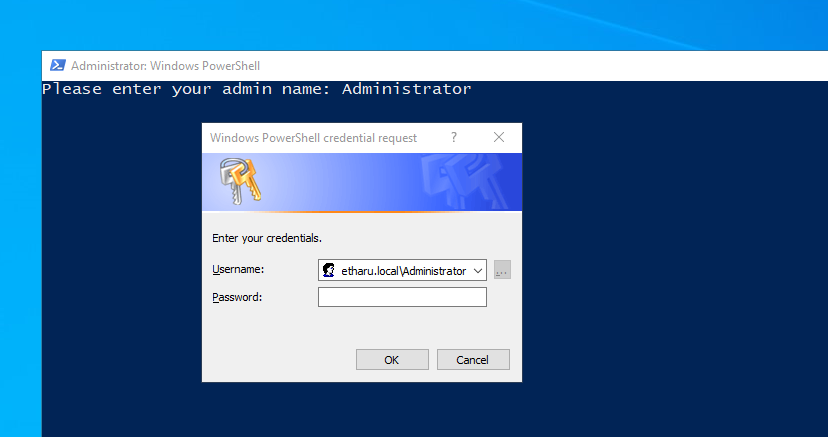


The script will take a moment to launch. This is because it begins by checking that you have a connection to the domain. In the event that it cannot gain a connection to the domain it will output the following error message:

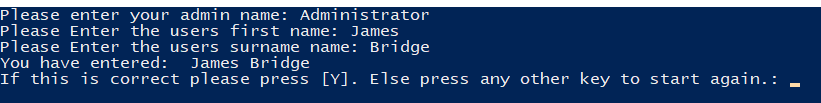


At this point you will need to check your local computers connectivity.

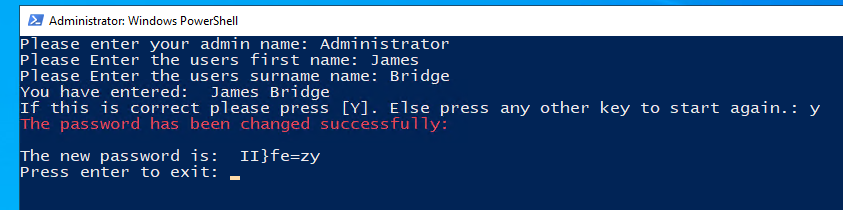
If the connection check passes, then the main body of the script will run. You will be prompted to enter your admin account name. And you will then be asked to enter your password:



Next, you will be asked to enter the name of the user whose account you wish to unlock. You will then be asked to confirm this by pressing ‘y’ and enter. Any other key will loop you back for a second chance to enter the user details.

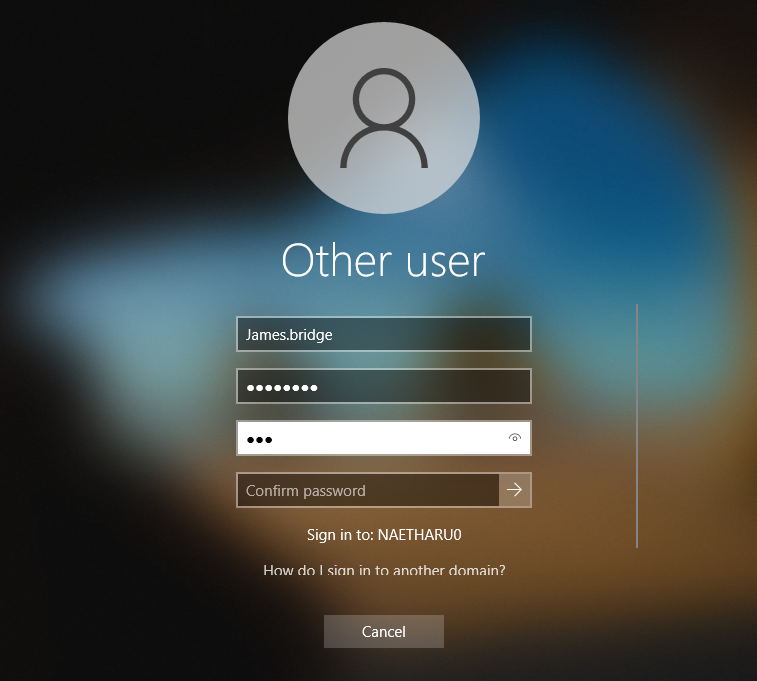


Once you press ‘y’ to confirm your selection the main body of the script will complete. The user account will be unlocked, enabled and the password will be reset to a random value using the .net framework Web.Security class. You will get the following result:



Give the user the new password.

The user will then be prompted to change their password on next login:



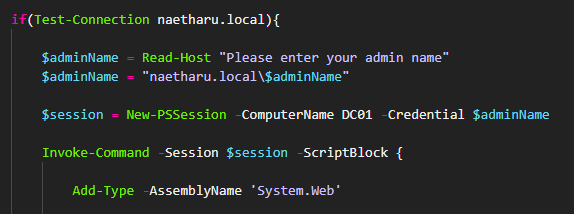
You have completed your password reset and never had to log into the DC! Happy days.

**How the inside of the script works:**

The script is broken down into three main areas:

1. Authentication with the domain
2. Acquisition of the user account details
3. Processing of the password reset

**Authentication with the domain**

****

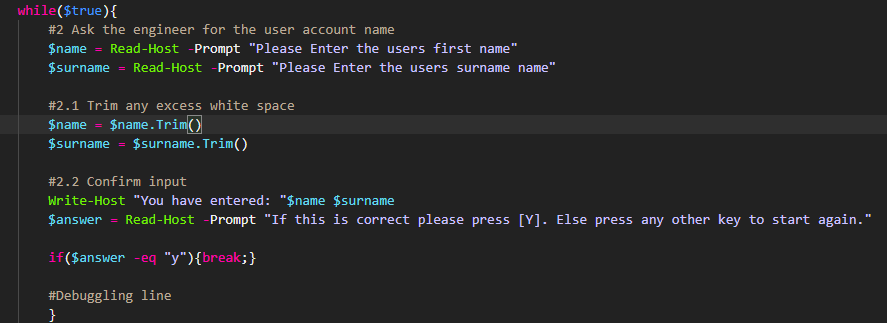
We start by opening a conditional statement (if) that tests a domain connection. If the connection succeeds then the script proceeds. If it fails, then the ‘else’ clause is processed, and the script closes gracefully with the error message that your computer is not on domain.

Once we have the connection made, we then collect the user’s admin name. We append this to the domain to give us our complete credentials.

Next, we open a remote session with the DC via the New-Session command. We store this in the $session variable so that we can draw on this later. It is this point that invokes the password request.

Finally, we run an invoke-command and feed it the pre-opened remote session before opening a script-block. All this does is wrap a script that we would like to run on the remote machine inside the invoke command scope. From here we can write our script as if we were running it locally on the DC.

**Acquisition of the user account details**



Here we invoke a while loop so that we can repeat the code until we have the results we want. We use a bit of a trick to do this. Normally we would give the while loop some condition to check against. But instead, we have used the in-build PowerShell variable $true. This always returns true. And so, our while loop will never satisfy the conditions necessary to exit and we will be stuck here forever! We resolve this by having a quick if statement at the end of the loop that checks to see if the user has accepted the input and if accepted, we then use a ***break*** command to end the loop.

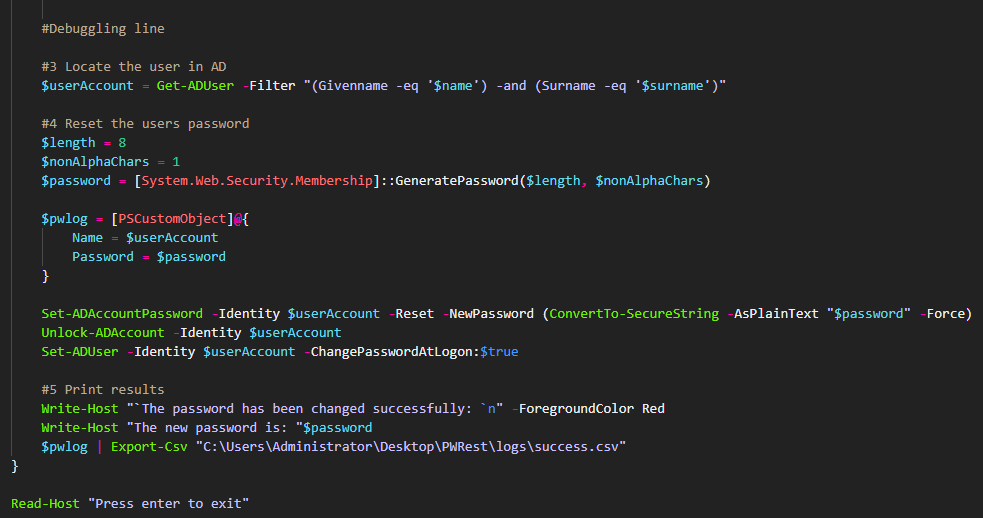
An alternative way to do this would have been to create a variable above the loop, and then to have swapped the value of that variable to $false once the condition was met. But honestly, I find that less readable. I am a big fan of clarity in code. And so, this is my preferred way to do things.

Inside our loop the code is simple. We start by reading some information from the console. The first name and the last name of our target user. We then use the .trim() method to strip away any extra white space. Just in case our engineer pushes space by accident – it can be easy to do on a command line and so little QOL changes of this kind are a welcome addition.

Finally, we write back the input to the command like and ask our engineer to confirm that they are happy with the results. If they press ‘y’ and enter we move forward. Any other key loops us back to the start of the while loop and off we go again. Notice that we use the -eq and not the -ceq so we are no fussed about case. Y or y will do just fine. Just another little QOL addition to avoid unnecessary pedantry in our code.

Now that we have our user details sorted on with the code!

**Processing of the password reset**



So here we start the business of actually getting the account fixed up.

First, we grab a reference to the account via the Get-ADUser command, using a conjunction of first and last name to ensure our reference is correct.

Next we use the .net framework class Web.Security to generate our new password. We could do this manually by some combination of pseudo-random number generation with conversion to ASCII but there’s no glory in re-inventing the wheel. We can choose the password length and the minimum number of non-alpha characters it must contain.

Note that the password length must comply with the domain policy elsewise it will fail at the point of application.

Next, we create a custom PowerShell object and we give it two properties. A name, which we set to be the account name, and a password which is where we store the plain-text password before running our encryption. This way we can report the password to the console to hand onto our user.

Finally, we get to the part where we do the heavy lifting. We start with using the Set-ADAccountPassword command and we feed in our newly created password using the ConvertTo-SecureString command to turn our password a special format that allows Windows handle it with extra care. For example, it avoids pushing the string to memory locations where it could be easily skimmed.

Next, we unlock the account and the finally we enforce a password change on next login. And then the final lines merely report the results of our process.