How I use AI to Write Amazing PowerShell

Todd Klindt

23 Aug 2023

Todd Klindt

SYMPRAXIS CONSULTING

Todd Klindt

18 Year SharePoint MVP

Blogger, Speaker, Consultant at Sympraxis Consulting, Not a Developer

todd.klindt@sympraxisconsulting.com
@toddklindt
https://www.toddklindt.com/PoshPnP

MVP
Microsoft® Most Valuable Professional

Agenda

- What is AI?
- What is the focus [illegible] our discussion?
- How should I use it?
- Here are the tools I [illegible]?

Dazzle!

What is AI?

AI is many things

- NLP - Natural Language Processing
- ML - Machine Learning
- CV – Computer Vision

It learns like humans do, for better or worse

ChatGPT read the whole internet in September of 2021

Some Terms

- Neural Network – Like our brains
- Nodes – Line a neuron
- Parameter – Weights on connections
- Hallucinations – Stuff the AI imagined
- Temperature – Higher, more random, lower, more real
- Alignment – Are AI and humans' outcomes aligned?
- Large Language Model (LLM) – The AI's brain

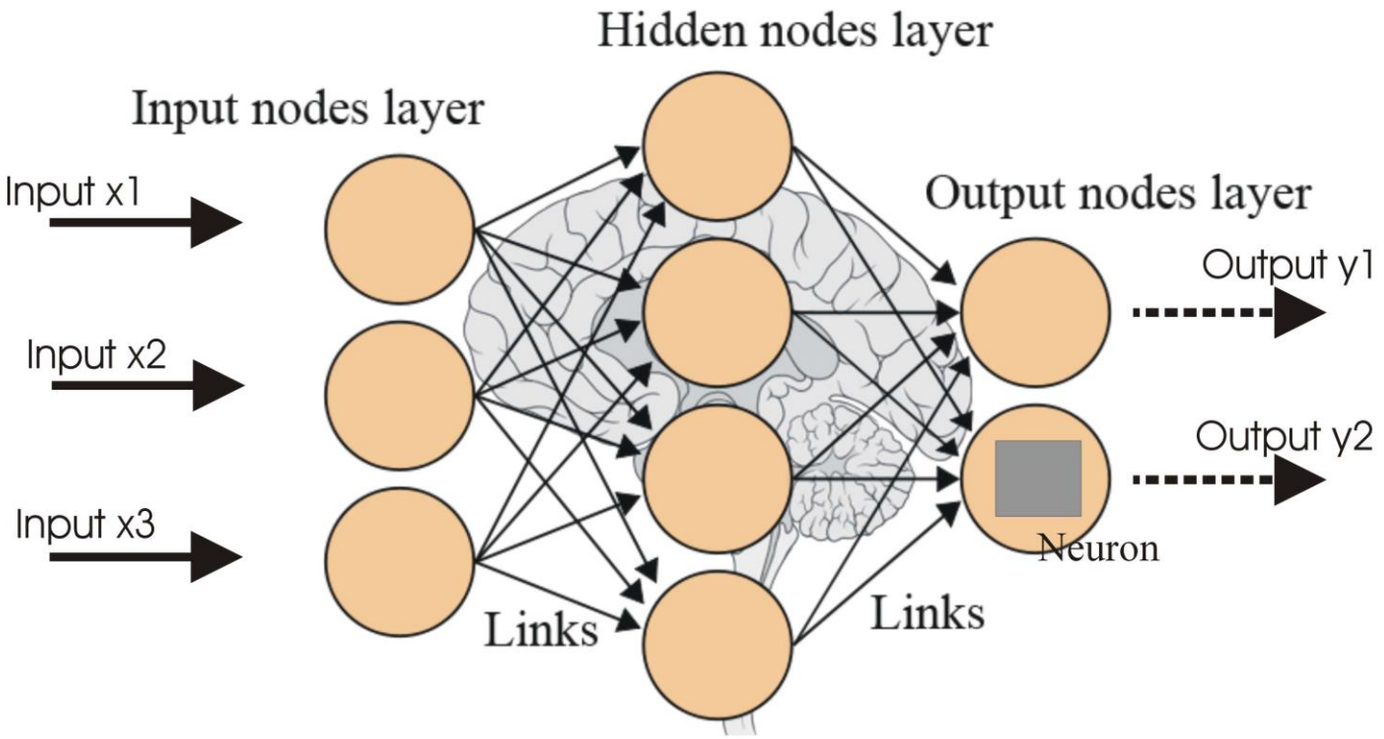
What the heck is a GPT, anyway?

Generative – AI that makes new stuff

Pre-Trained – The model was trained ahead of time

Transformer – A new way to input and output data

Neural Network



What is the focus for our discussion?

- Now that we're AI experts…
- How can we use this tool?
- Write better code (PowerShell!)
  - Faster
  - Fewer errors
  - More functionality
  - Better documentation
  - Write better *anything*

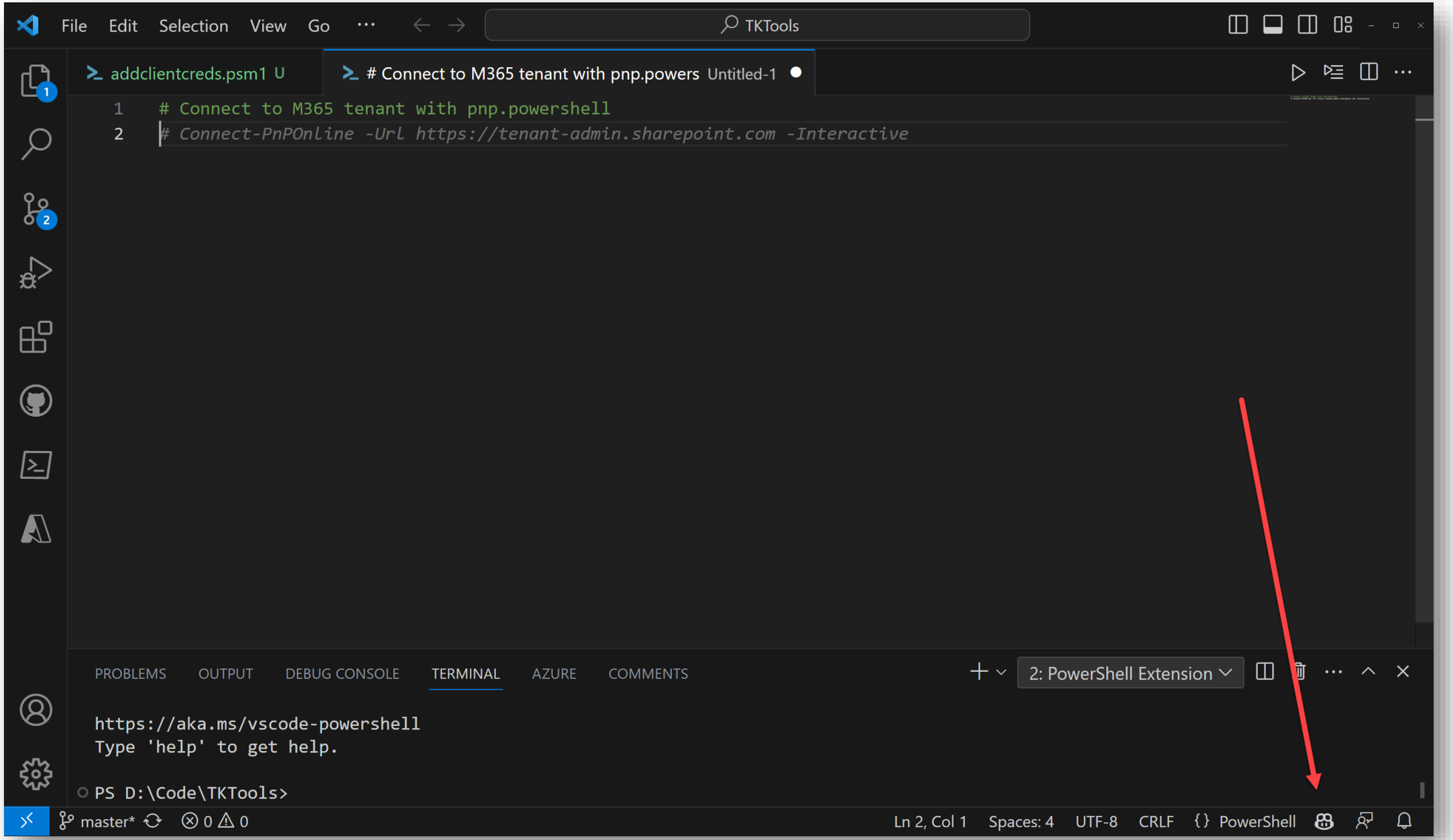
How?

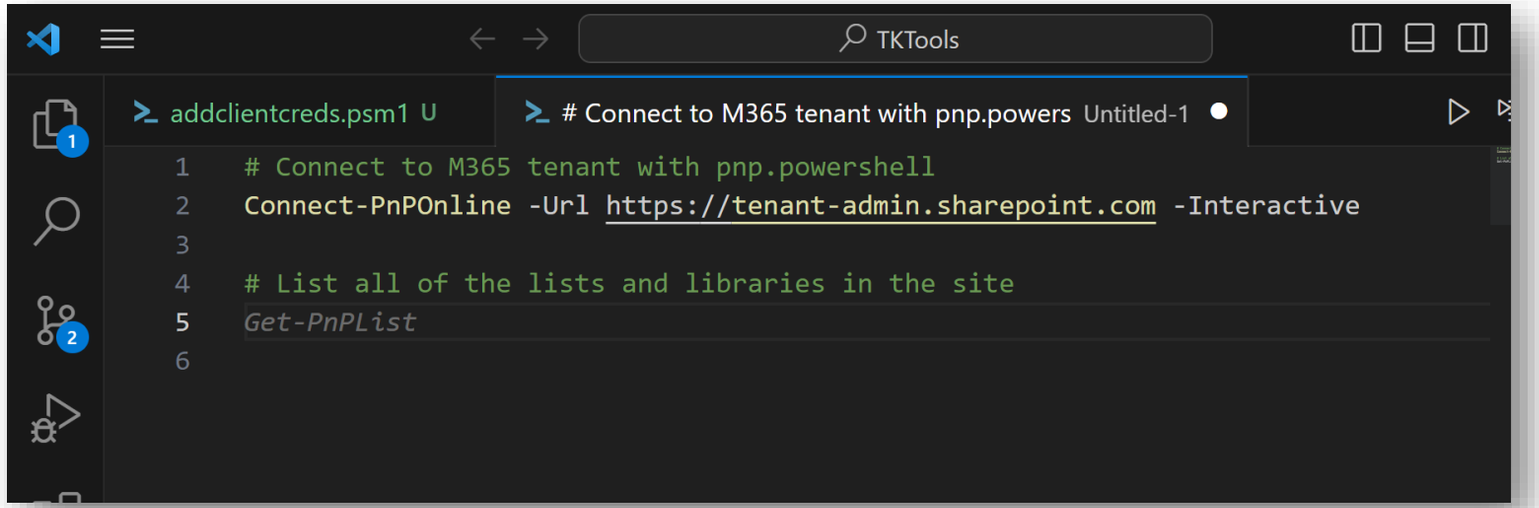
- Don’t think of it as a computer or software
- What role would you like it to play?

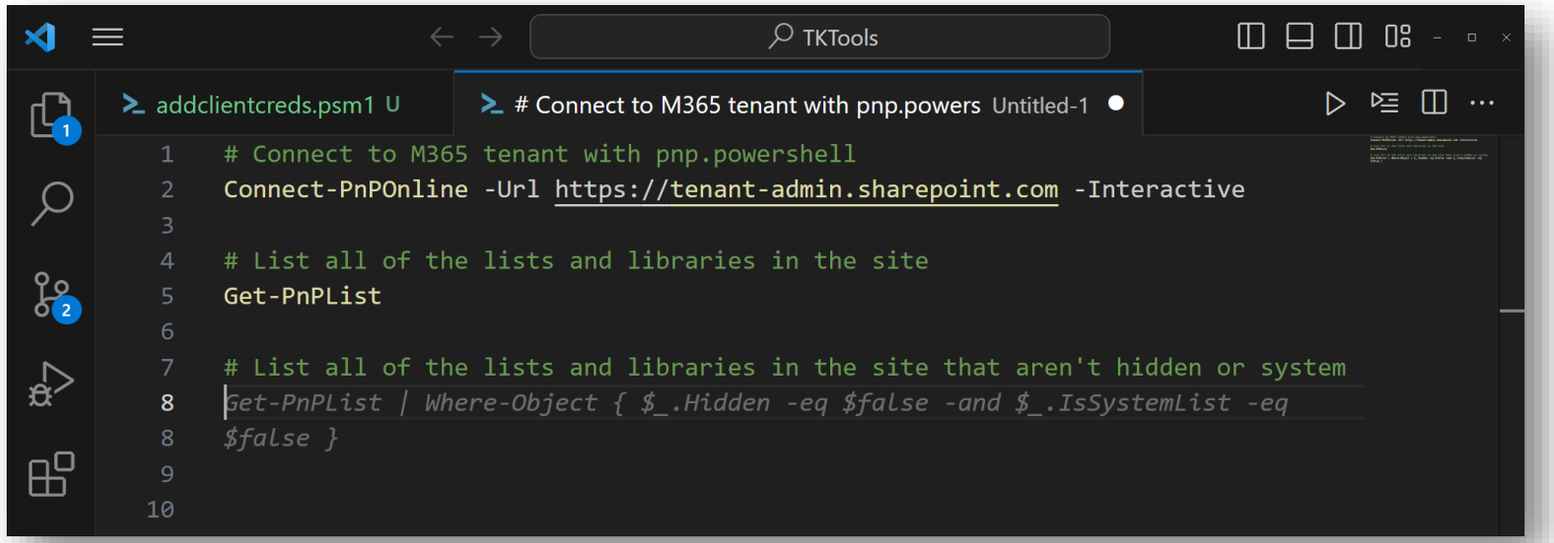
  Review? Creator? Collaborator?
- Initiate (proact?) or react?
  - Tyranny of the blank page
  - “I want to…”
  - “What’s the best way to…”
  - # Loop through $ListItemList and…”
- Decide which AI tool to use

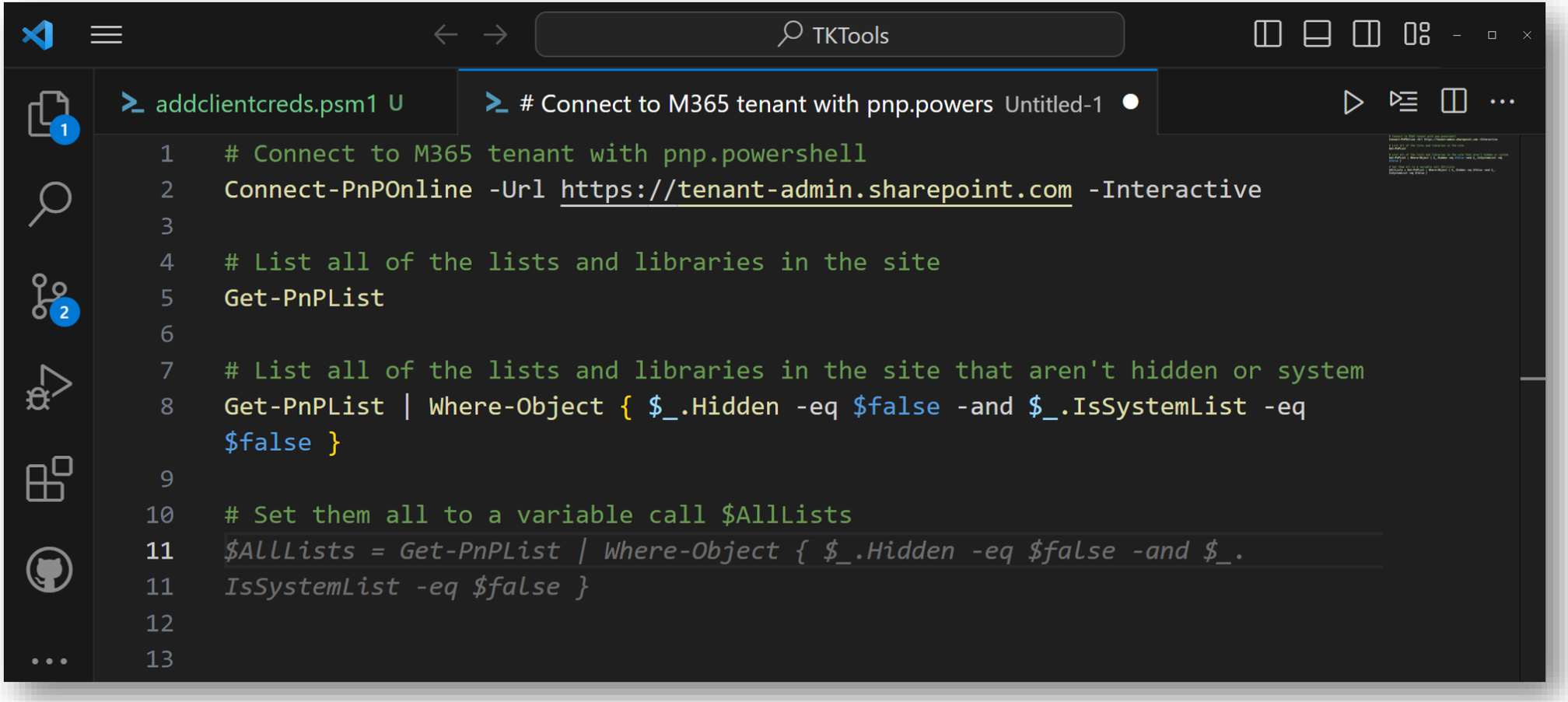
CoPilot

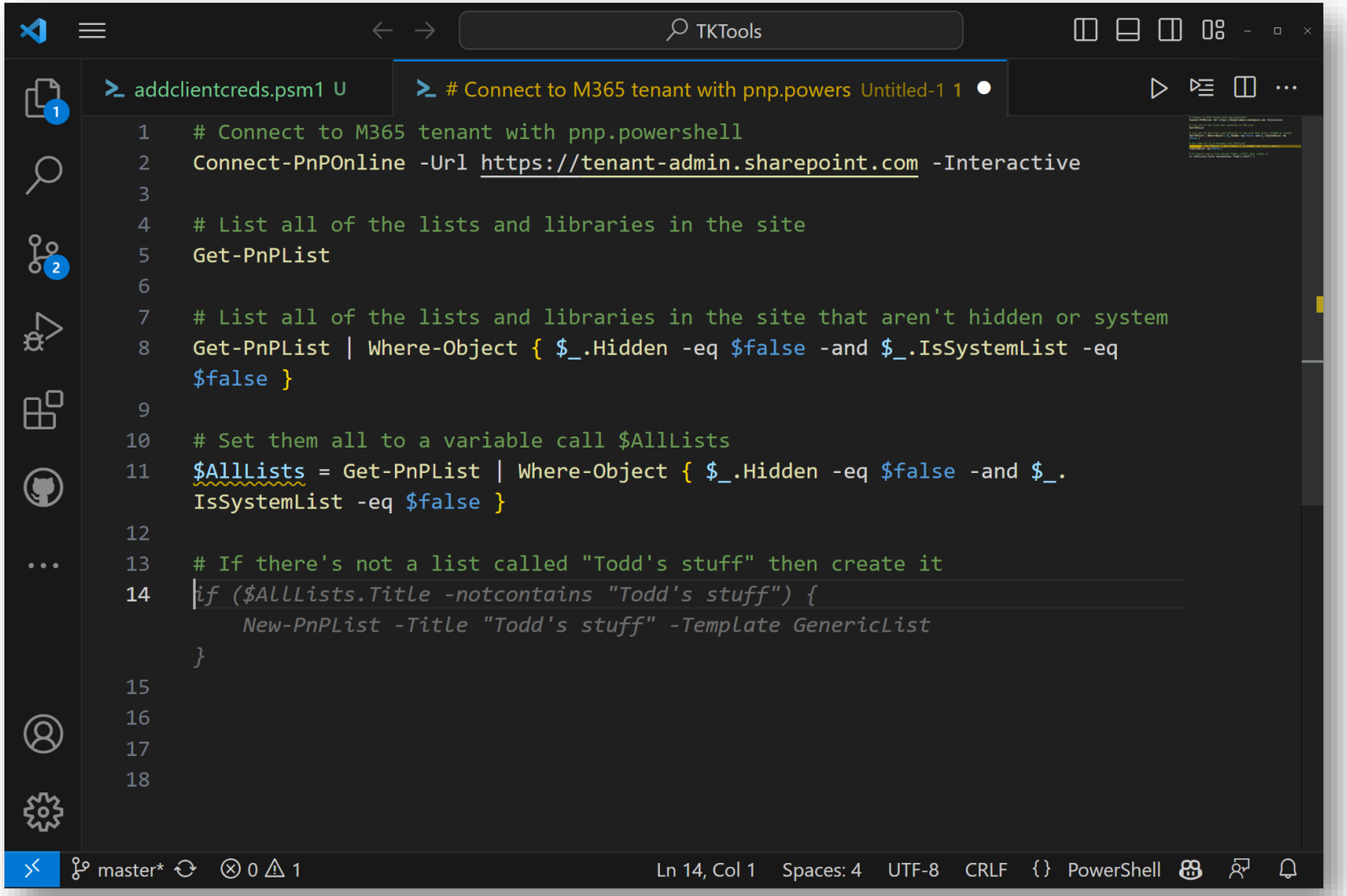
- GitHub Copilot
- Like pair programming
- Looks over your shoulder
- Uses its corpus, open tabs, and your code
- $20 a month


File
Edit
Selection
View
Go
TKTools
addclientcreds.psm1 U
Connect to M365 tenant with pnp.powers
Untitled-1
1 # Connect to M365 tenant with pnp.powershell
2 # Connect-PnPOnline -Url https://tenant-admin.sharepoint.com -Interactive
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
AZURE
COMMENTS
2: PowerShell Extension
https://aka.ms/vscode-powershell
Type 'help' to get help.
PS D:\Code\TKTools>
master*
0 0
Ln 2, Col 1
Spaces: 4
UTF-8
CRLF
PowerShell


TKTools
addclientcreds.psm1 U
Connect to M365 tenant with pnp.powers
Untitled-1
Connect to M365 tenant with pnp.powershell
Connect-PnPOnline -Url https://tenant-admin.sharepoint.com -Interactive
List all of the lists and libraries in the site
Get-PnPList


TKTools
addclientcreds.psm1 U
Connect to M365 tenant with pnp.powers
Untitled-1
Connect to M365 tenant with pnp.powershell
Connect-PnPOnline -Url https://tenant-admin.sharepoint.com -Interactive
List all of the lists and libraries in the site
Get-PnPList
List all of the lists and libraries in the site that aren't hidden or system
Get-PnPList | Where-Object { $_.Hidden -eq $false -and $_.IsSystemList -eq $false }


TKTools
addclientcreds.psm1 U
Connect to M365 tenant with pnp.powers Untitled-1
Connect to M365 tenant with pnp.powershell
Connect-PnPOnline -Url https://tenant-admin.sharepoint.com -Interactive
List all of the lists and libraries in the site
Get-PnPList
List all of the lists and libraries in the site that aren't hidden or system
Get-PnPList | Where-Object { $_.Hidden -eq $false -and $_.IsSystemList -eq $false }
Set them all to a variable call $AllLists
$AllLists = Get-PnPList | Where-Object { $_.Hidden -eq $false -and $_.IsSystemList -eq $false }


TKTools
addclientcreds.psm1 U
Connect to M365 tenant with pnp.powers Untitled-1 1
Connect to M365 tenant with pnp.powershell
Connect-PnPOnline -Url https://tenant-admin.sharepoint.com -Interactive
List all of the lists and libraries in the site
Get-PnPList
List all of the lists and libraries in the site that aren't hidden or system
Get-PnPList | Where-Object { $_.Hidden -eq $false -and $_.IsSystemList -eq $false }
Set them all to a variable call $AllLists
$AllLists = Get-PnPList | Where-Object { $_.Hidden -eq $false -and $_.IsSystemList -eq $false }
# If there's not a list called "Todd's stuff" then create it
if ($AllLists.Title -notcontains "Todd's stuff") {
New-PnPList -Title "Todd's stuff" -Template GenericList
}
master*
0 1
Ln 14, Col 1
Spaces: 4
UTF-8
CRLF
PowerShell

ChatGPT

- That genie in a bottle
- "Write me some PowerShell code that…"
- Compliments GitHub Copilot
- Spring for the $20 a month

Plugins • Enabled plugins:

CH

Let's write a PowerShell function called Add-ClientCredential. It has 4 parameters, tenantname, username, password, and testcredential. It should use Add-PnpStoredCredential to store credentials for https://tenant.sharepoint.com, https://tenant.sharepoint.com/, and https://tenant-admin.sharepoint.com.

Certainly! Below is a PowerShell function called `Add-ClientCredential`. This function takes four parameters:

- `TenantName`: The name of the tenant (e.g., "contoso").
- `UserName`: The username for the credential (e.g., "user@contoso.com").
- `Password`: The password for the credential.
- `TestCredential`: A switch parameter that, when specified, will test the credentials after they are stored.

The function uses the `Add-PnPStoredCredential` cmdlet to store credentials for the following SharePoint URLs:

- `https://<TenantName>.sharepoint.com`
- `https://<TenantName>.sharepoint.com/`
- `https://<TenantName>-admin.sharepoint.com`

Here is the PowerShell function:

powershell

Copy code

Here is the PowerShell function:

powershell

Copy code

```powershell
function Add-ClientCredential {
    [CmdletBinding()]
    param (
        [Parameter(Mandatory = $true)]
        [string] $TenantName,

        [Parameter(Mandatory = $true)]
        [string] $UserName,

        [Parameter(Mandatory = $true)]
        [string] $Password,

        [Parameter(Mandatory = $false)]
        [switch] $TestCredential
    )

    $securePassword = ConvertTo-SecureString -String $Password -AsPlainText

    $urls = @(
        "https://$TenantName.sharepoint.com",
        "https://$TenantName.sharepoint.com/",
        "https://$TenantName-admin.sharepoint.com"
    )
```

Regenerate

Send a message

Certainly! Below is the updated `Add-ClientCredential` PowerShell function. Now, before adding a new credential, it checks if a credential with the same name already exists. If it does, the function prompts the user to confirm whether they want to replace the existing credential. If the user confirms, the function will replace the existing credential; otherwise, it will skip to the next URL.

Here is the updated PowerShell function:

powershell Copy code

```powershell
function Add-ClientCredential {
    [CmdletBinding()]
    param (
```

```powershell
    )

    foreach ($url in $urls) {
        $existingCredential = Get-PnPStoredCredential -Name $url -ErrorActio

        if ($null -ne $existingCredential) {
            $replace = $false
            $replace = Read-Host "Credential for $url already exists. Do you
            if ($replace -eq 'Y' -or $replace -eq 'y') {
                $replace = $true
            } else {
                $replace = $false
            }

            if (-not $replace) {
                Write-Output "Skipping credential for $url."
                continue
            }
        }

        Add-PnPStoredCredential -Name $url -Username $UserName -Password $se
```

CH

Looks good. Let's have it prompt the user if they don't enter tenantname, username, or password. And let's have the password take a securestrong instead of a string.

Certainly! Below is the updated `Add-ClientCredential` PowerShell function. Now, if the `TenantName`, `UserName`, or `Password` parameters are not provided when the function is called, it will prompt the user to enter them. Additionally, the `Password` parameter now accepts a `SecureString` object instead of a plain text string.

Here is the updated PowerShell function:

powershell Copy code

```powershell
function Add-ClientCredential {
    [CmdletBinding()]
    param (
        [Parameter(Mandatory = $false)]
        [string] $TenantName,

        [Parameter(Mandatory = $false)]
        [string] $UserName,

        [Parameter(Mandatory = $false)]
        [SecureString] $Password,

        [Parameter(Mandatory = $false)]
        [switch] $TestCredential
    )
```

```powershell
    if (-not $TenantName) {
        $TenantName = Read-Host "Please enter the Tenant Name"
    }

    if (-not $UserName) {
        $UserName = Read-Host "Please enter the User Name"
    }

    if (-not $Password) {
        $Password = Read-Host "Please enter the Password" -AsSecureString
    }

    $urls = @(
```

CH

Looks good. Add a check to make sure the pnp.powershell is installed. If it's not, tell the user and give them the line to install it.

Certainly! Below is the final version of the `Add-ClientCredential` PowerShell function. Now, before attempting to use any PnP cmdlets, the function checks if the `PnP.PowerShell` module is installed. If it is not installed, the function will inform the user and provide the command line to install the module.

Here is the updated PowerShell function:

powershell Copy code

```powershell
function Add-ClientCredential {
    [CmdletBinding()]
    param (
        [Parameter(Mandatory = $false)]
        [switch] $TestCredential
    )

    # Check if PnP.PowerShell module is installed
    $pnpModule = Get-Module -ListAvailable -Name PnP.PowerShell -ErrorAction
    if ($null -eq $pnpModule) {
        Write-Warning "The PnP.PowerShell module is not installed."
        Write-Output "To install the PnP.PowerShell module, run the followin
        Write-Output "Install-Module -Name PnP.PowerShell -Scope CurrentUser
        return
    }

    if (-not $TenantName) {
        $TenantName = Read-Host "Please enter the Tenant Name"
    }
```

CH

If the script finds an existing credential, show the username for that credential.

Certainly! Below is the updated **`Add-ClientCredential`** PowerShell function. Now, when the function detects an existing credential for a given URL, it will display the username associated with that credential and then prompt the user to confirm whether they want to replace the existing credential.

Here is the updated PowerShell function:

powershell Copy code

```
function Add-ClientCredential {
    [CmdletBinding()]
```

```
foreach ($url in $urls) {
    $existingCredential = Get-PnPStoredCredential -Name $url -ErrorAction
    SilentlyContinue

    if ($null -ne $existingCredential) {
        Write-Output "Existing credential found for $url with username: $
        ($existingCredential.UserName)"
        $replace = Read-Host "Do you want to replace it? (Y/N)"
        if ($replace -eq 'Y' -or $replace -eq 'y') {
            $replace = $true
        } else {
            $replace = $false
```

CH

Looks great! Please add a description and provide a few examples.

Certainly! Below is the final version of the `Add-ClientCredential` PowerShell function, complete with a description and examples.

Description:

This PowerShell function is designed to manage SharePoint Online credentials for a given tenant. It allows the user to securely store credentials for various SharePoint Online URLs using the `PnP.PowerShell` module. If credentials for a given URL already exist, the function will display the associated username and prompt the user to confirm whether they want to replace the existing credentials. The function also includes an option to test the

```
<#
.SYNOPSIS
This function stores SharePoint Online credentials for a given tenant using

.DESCRIPTION
The Add-ClientCredential function securely stores credentials for various Sh
If credentials for a given URL already exist, the function will display the
prompt the user to confirm whether they want to replace the existing credent

.PARAMETER TenantName
The name of the SharePoint Online tenant.

.PARAMETER UserName
The username for the SharePoint Online credentials.
```

CH

add support if the user enters contoso.sharepoint.com, contoso.onmicrosoft.com, https or http versions of them.

Certainly! The updated `Add-ClientCredential` PowerShell function now includes support for various formats of the tenant name. It can handle inputs like `contoso`, `contoso.sharepoint.com`, `contoso.onmicrosoft.com`, `https://contoso.sharepoint.com`, or `http://contoso.sharepoint.com`. The function

```
# Normalize the tenant name to extract the base tenant name
$TenantName = $TenantName -replace 'https://|http://|\.sharepoint\.com|\.onmicrosoft\.com', ''
```

CH

This all works great. Is there anything else I should add, or any other errors I should trap for?

Great to hear that it works well! Here are a few additional considerations for making your script more robust and user-friendly:

1. **Validate Tenant Name Format:**
   - You might want to add a regular expression check to ensure that the tenant name entered by the user is in a valid format after normalization. This can help to catch typos or invalid characters early.
2. **Check for Null or Empty Password:**
   - Before proceeding, you might want to check that the user didn't enter an empty password. If the password is empty, you can prompt the user to enter it again.
3. **Error Handling for PnP Cmdlets:**
   - Consider adding `` `try`/`catch` `` blocks around the PnP cmdlets to handle any exceptions that they might throw. This way, if an error occurs (e.g., network issues, server unresponsive), your script can catch that error and handle it gracefully, such as by logging the error and continuing with the next iteration of the loop.
4. **Confirm Impact with User:**
   - Before making changes (especially when replacing existing credentials), consider using the `` `ShouldProcess` `` method to confirm the impact with the user. This is a best practice for PowerShell functions that make changes to a system.
5. **Support for `` `-WhatIf` `` and `` `-Confirm` `` Parameters:**

Here is an example of some of these improvements:

powershell — Copy code

```powershell
function Add-ClientCredential {
    [CmdletBinding(SupportsShouldProcess=$true, ConfirmImpact='Medium')]
    param (
        [Parameter(Mandatory = $false)]
        [ValidateNotNullOrEmpty()]
        [string] $TenantName,

        [Parameter(Mandatory = $false)]
        [ValidateNotNullOrEmpty()]
        [string] $UserName,
```

CH

Good ideas. Add validation for tenant name format and error handling for pnp cmdlets.

Certainly! Below is the updated `Add-ClientCredential` PowerShell function with added validation for the tenant name format and error handling for PnP cmdlets.

The tenant name validation ensures that after normalization, the tenant name contains only valid characters (letters, numbers, and hyphens). The error handling includes `try`/`catch` blocks around the PnP cmdlets to gracefully handle any exceptions that they might throw, log the error, and continue with the next iteration of the loop.

Here is the updated PowerShell function:

powershell Copy code

```
<#
.SYNOPSIS
This function stores SharePoint Online credentials for a given tenant using
```

```powershell
# Normalize the tenant name to extract the base tenant name
$TenantName = $TenantName -replace 'https://|http://|\.sharepoint\.com|\.onmicrosoft\.com', ''

# Validate tenant name format
if (-not ($TenantName -match '^[a-zA-Z0-9\-]+$')) {
    Write-Warning "Invalid tenant name format. Please enter a valid tenant name."
    return
}
```

Resources

- Shawn Wang on AI
- ChatGPT
- GitHub Copilot
- The 12 minute version of this with demos
- Why AI Will Save the World - Marc Andreessen

SYMPRAXIS CONSULTING

Questions?

todd.klindt@sympraxisconsulting.com
@toddklindt
www.toddklindt.com

www.toddklindt.com/sef2023

MVP Microsoft® Most Valuable Professional