Module

Irgendwann wird jedes Skript groß genug, um unübersichtlich zu werden. Module helfen dabei, Funktionen, Variablen und andere Bestandteile sinnvoll zu organisieren und wiederzuverwenden. Kurz gesagt: weniger Chaos, mehr Struktur. Meistens jedenfalls.

Modules.psm1 – Erweiterungen für das Skript, die zusätzliche Variablen , Funktionen oder Objekte beinhalten. Es kann nicht allein ausgeführt werden und muss von einem **Skript** oder **Manifest** importiert werden .

Es gibt auch → **modulqualifizierter Aufruf**

- Manifest
- Import (Modul laden)

Manifest

Ein **PowerShell-Manifest** (`.psd1`) ist im Grunde nur eine Hashtable mit vordefinierten Keys. Es enthält die **Metadaten und Konfiguration** zu deinem Modul – also alles, was PowerShell über dein Modul wissen muss, ohne den eigentlichen Code auszuführen. Deshalb liegt es **bewusst in einer eigenen Datei**, getrennt vom Modul selbst.

□□ Grundlegende Metadaten

Das ist die Identität deines Moduls. Ohne das bist du einfach nur irgendein Skript mit Existenzkrise.

- `RootModule` – Hauptdatei (`.psm1` oder `.dll`)
- `ModuleVersion` – Version (z. B. `1.0.0`)
- `GUID` – Eindeutige ID
- `Author` – Ersteller/Entwickler des Moduls
- `CompanyName` – Unternehmen
- `Copyright`
- `Description` – Beschreibung des Moduls

□□ Abhängigkeiten

Hier sagst du: „Ich funktioniere nicht alleine, ich brauche andere.“

- `RequiredModules`
  → Externe Abhängigkeiten
  → Angabe über **Modulnamen** (PowerShell sucht es selbst)
  → Müssen im System vorhanden sein (installiert / auffindbar, z.B. über `$PSModulePath`)
  → Gehören **nicht** zu deinem Modul
- `RequiredAssemblies` – DLLs
- `ScriptsToProcess` – Skripte, die beim Laden ausgeführt werden
- `ModuleList` – Nur die Meta-Information welche andere Module zugehörig sind
- `NestedModules`
  → Angabe über **Dateien/Pfade** (du sagst konkret, was geladen wird)
  → Interne Bausteine deines Moduls
  → Werden zusammen mit deinem Modul geladen
  → Gehören **zu deinem Modul dazu**

□□ Export (was nach außen sichtbar ist)

Das ist der Teil, wo du entscheidest, was du der Welt zeigst. Oder versteckst.

- `FunctionsToExport`
- `CmdletsToExport`
- `VariablesToExport`
- `AliasesToExport`

□□ Klassiker:
`'*'` = alles exportieren
oder explizit = bessere Kontrolle

□□ Kompatibilität & Anforderungen

Hier wird’s picky.

- `PowerShellVersion`
- `PowerShellHostName`
- `PowerShellHostVersion`
- `DotNetFrameworkVersion`
- `CLRVersion`
- `ProcessorArchitecture`

□□ Private Daten (der geheime Keller)

Alles, was du selbst definierst.

- `PrivateData` – Hashtable für eigene Infos

Beispiel:

```
PrivateData = @{
    PSData = @{
        Tags = @('UI','Forms')
        ProjectUri = 'https://...'
    }
}
```

□□ PSData (wichtig für Gallery etc.)

Teil von `PrivateData`, aber so wichtig, dass es eigentlich sein eigenes Ding ist.

- `Tags`
- `LicenseUri`
- `ProjectUri`
- `IconUri`
- `ReleaseNotes`

□□ Sonstiges (der „Warum gibt’s das?“ Bereich)

Selten gebraucht, aber existiert halt:

- `FileList`
- `TypesToProcess`
- `FormatsToProcess`
- `CompatiblePSEditions` (`Desktop`, `Core`)
- `HelpInfoURI`

□□ Mini-Beispiel

Damit du nicht nur trockene Theorie hast:

```
@{
    RootModule        = 'MeinModul.psm1'
    ModuleVersion     = '1.0.0'
    GUID              = '12345678-abcd-1234-abcd-1234567890ab'
    Author            = 'Jonny'
    Description       = 'Mein erstes Modul'

    FunctionsToExport = @('Get-Thing', 'Set-Thing')

    PowerShellVersion = '5.1'

    PrivateData = @{
        PSData = @{
            Tags = @('Example','Test')
        }
    }
}
```

Der eigentliche Punkt (den viele übersehen)

Du brauchst vielleicht **30% davon wirklich**.

Der Rest ist:

- entweder für Publishing
- oder für Leute, die Kontrolle lieben (oder Angst haben)

Wenn du lokal entwickelst:
→ `RootModule`, `ModuleVersion`, `FunctionsToExport`
→ fertig, läuft.

Import (Modul laden)

Hier passiert das Offensichtliche, das trotzdem erstaunlich oft falsch gemacht wird: Du lädst dein Modul in die aktuelle Session.

Ohne Import bist du einfach nur jemand, der Code hortet.

□□ Grundbefehl

```
Import-Module MeinModul
```

PowerShell sucht dein Modul automatisch in den bekannten Modulpfaden (`$env:PSModulePath`).

Wenn es da liegt: easy. Wenn nicht: dein Problem.

□□ Import über Pfad

Wenn dein Modul irgendwo rumliegt, wo PowerShell es nicht findet:

```
Import-Module "C:\MeinProjekt\MeinModul.psm1"
```

Oder direkt das Verzeichnis:

```
Import-Module "C:\MeinProjekt\MeinModul"
```

PowerShell nimmt dann automatisch das Manifest (`.psd1`) oder die Moduldatei (`.psm1`).

□□ Erneutes Laden (Reload)

Du änderst Code, aber PowerShell hängt noch am alten Stand fest? Überraschung.

```
Import-Module MeinModul -Force
```

`-Force` lädt das Modul neu, auch wenn es schon geladen ist.

Pflicht beim Entwickeln, außer du stehst auf Selbstverwirrung.

□□ Selektiver Import

Du musst nicht alles laden. Du kannst gezielt nur bestimmte Dinge importieren:

```
Import-Module MeinModul -Function Get-Thing
```

Oder mehrere:

```
Import-Module MeinModul -Function Get-Thing, Set-Thing
```

Funktioniert natürlich nur, wenn dein Manifest das auch erlaubt (`FunctionsToExport`).

□□ Automatischer Import

Seit neueren PowerShell-Versionen gilt:

Wenn du eine Funktion aus einem Modul aufrufst, wird es automatisch importiert.

```
Get-Thing
```

→ PowerShell denkt sich: „Kenn ich nicht… ach warte, da gibt’s ein Modul“

Klingt smart. Ist es auch.
Bis es nicht mehr funktioniert und du nicht weißt warum.

□□ Geladene Module anzeigen

```
Get-Module
```

Alle verfügbaren Module anzeigen:

```
Get-Module -ListAvailable
```

Wenn dein Modul hier nicht auftaucht, kannst du lange importieren. Wird nix.

□□ Modul entfernen

Falls du es wieder loswerden willst:

```
Remove-Module MeinModul
```

Hilfreich beim Entwickeln, wenn du nicht mit `-Force` arbeiten willst.

⚠ Bereits geladenes Modul

Wenn ein Modul bereits geladen ist und du erneut `Import-Module` aufrufst, passiert… nichts.

```
Import-Module MeinModul
```

PowerShell sagt sich: „Kenn ich schon“ und ignoriert dich stillschweigend.

Das bedeutet:

- Änderungen im Code werden **nicht** übernommen
- Du arbeitest weiter mit der alten Version

Wenn du sicherstellen willst, dass dein Modul wirklich neu geladen wird:

```
Import-Module MeinModul -Force
```

Alternativ kannst du es vorher entfernen:

```
Remove-Module MeinModul Import-Module MeinModul
```

Wichtig beim Entwickeln:
Wenn sich „irgendwas komisch anfühlt“, ist es erstaunlich oft genau das hier.

Du debugst dann nicht deinen Code.
Du debugst deine Vergangenheit.

Der eigentliche Punkt (den wieder alle ignorieren)

Import ist kein einmaliger Akt, sondern Teil deines Workflows.

- Entwicklung → ständig `-Force`
- Debugging → bewusst laden/entladen
- Produktion → einmal sauber laden und fertig

Wenn dein Modul sich beim Import komisch verhält, liegt es fast nie am Import.
Es liegt daran, was dein Modul beim Laden tut.

Und genau da trennt sich „läuft irgendwie“ von „ich hab verstanden, was ich da gebaut habe“.