ファイルとディレクトリを区別する方法

Get-ChildItem でファイルやフォルダを列挙したとき、ファイルだけを対象、またはディレクトリだけを対象としたい。

PSIsContainer プロパティを使う

```
Get-ChildItem -r C:\TargetFolder | where { $_.PSIsContainer }
```

PSIsContainer は PowerShell 上で参照できるプロパティ。
$True の場合はコンテナ（ディレクトリ）、$False の場合はコンテナ内のアイテム（ファイル）。

インスタンスの型チェックを使う

```
Get-ChildItem -r C:\TargetFolder | where { $_ -is [IO.FileInfo] }
```

-is 演算子で、オブジェクトが .NET Framework のある型である（またはその型を継承している）かどうか、を判定できる。
Get-ChildItem はファイルの場合は System.IO.FileInfo クラス、ディレクトリの場合は System.IO.DirectoryInfo クラスのインスタンスを返す。
Get-ChildItem がどんなオブジェクトを返してくるかは、Get-Member で調べることができる。
こちらの方が汎用性は高いと思われる。

.NET Framework の型名指定は角カッコ [] を使う（System 名前空間は省略可能とのこと)。文字列で指定してもよいようだが、この方が型名をしていしていることが明示的なので、記法としてはよいと思う。

ファイル IO

PowerShell には Get-Content があるので、テキストファイル処理のために .NET Framework クラスを使用する必要は通常ない。
しかし、たとえば System.Security.Cryptography で処理するため等で、生の FileStream オブジェクトインスタンスが欲しい場合がある。

```
$fs = New-Object IO.FileStream -ArgumentList "C:\target_file.txt", "Open", "Read"
```

ということで、コンストラクタへの引数を複数渡すときには -ArgumentList オプションが使える。
さらに、FileMode と FileAccess の列挙体の値を引数として渡すときに、それぞれの列挙体のメンバー名を文字列として渡せばよい。

これがわかれば以下の同義文も書ける。

```
$fs = [IO.System.File]::Open("C:\target_file.txt", "Open", "Read")
```

ところで、.NET Framework のクラスに渡すファイルパスが相対パスだった場合、それは $home を

基準に解釈されてしまう。
$pwd ではないので、注意が必要。
これを補正するためには、コマンドレットを経由してフルパスを得るようにすればよい。コマンドレットは相対パスを $pwd を基準に解釈してくれる。

その 1: Resolve-Path でフルパス文字列を得る。

```
param ($FilePath)

$fullPath = Resolve-Path $FilePath
$fs = [IO.System.File]::Open($fullPath, "Open", "Read")
```

その 2: Get-Item を使って FileInfo オブジェクトを取得する。FileInfo の他のプロパティも後で使いたいときなど。

```
param($FilePath)

$fi = Get-Item $FilePath
$fs = [IO.System.File]::Open($fi.FullName, "Open", "Read")
```

コマンドレットの標準エラー出力を捨てる

スクリプト中で try - catch を使ってエラー処理をしても、コマンドレットはエラー発生時には勝手に標準エラー出力に自分のエラーメッセージを吐き出してしまう。
これはかなりカッコ悪い。エラーメッセージを制御したいから try - catch を使っているのに、これでは意味がない。

そんなときには、コマンドレットの出力を $null にリダイレクトすることができる。
たとえば Resolve-Path は与えられたパスが存在しないとエラーを吐くが、これを catch してスクリプト独自のエラーメッセージを出したい場合。

```
param($Path)

try {
    $fullPath = Resolve-Path $Path 2> $null
} catch {
    Write-Output "My message is ... "
    ...
}
```

これで Revolve-Path のエラーメッセージは捨てられて、catch 節に制御が移る。
エラーが発生しなければ、$fullPath に Resolve-Path の結果が代入される。

スクリプトの実行停止

```
[System.Environment]::Exit(0)
```