A Quick PowerShell Reference Pocketbook

Statement Evaluation Order:
- Parenthesis `( )` are always evaluated first

Automatic Unit Recognition:
- Units like KB, MB, and GB can be used with numbers like `1MB + 2GB`. Results are in bytes by default.

Windows Batch & Other Commands:
- `cmd /c` command-name
- use `&` to execute commands in quotes, i.e. `& "c:\autoexec.bat"`
- To execute a program in current directory, prefix program name with `\` i.e. `.\editplus.exe` . Unless the folder is on the windows path `$env:path`
- Add folder to path: `$env:path += ";C:\programs\Windows NT\accessories"`

Cmdlet Convention: Use the forms of verbs and nouns i.e. `Get-Help`
`Add, Clear, Compare, Convert, Copy, Export, Format, Get, Group, Import, Measure, Move, New, Out, Read, Remove, Rename, Resolve, Restart, Resume, Select, Set, Sort, Split, Start, Stop, Suspend, Tee (Split up), Test, Trace, Update, Write.`

Cmdlet Parameters:
- Common Parameters i.e. `ErrorAction, WhatIf`
- Named Parameters i.e. `string, string[]`
- Switch Parameter i.e. `boolean ($true,$false)`
- Positional Parameters i.e. `Get-ChildItem -recurse, -name, c:\windows *.exe`
- *Disable Parameters:* Use `--` or `"-Parameter"` (quotes) to *disable* parameter recognition

Variables:
- Definition:
  - `$syntax` i.e. `$company = "Axiom Dynamics Corp."`
  - Typeless i.e. `$one = 1`
  - ${special variable} i.e. `${I am special} = "Special"`
  - `$assignment = value` or "`cmdlet` Result"
  - Multiple inline assignments are supported i.e. `$ten = $10 = 10`
  - separate statements by `;` i.e. `$one=1;$two=2`
  - `$_` - this reference
  - Swap variable values:
    ```
    $Value1 = 10; $Value2 = 20;
    $Value1, $Value2 = $Value2, $Value1
    $Value1
    $Value2
    ```
  - Virtual drive is a variable
  - Write-Protected variables
  - "backtick" character ’
  - "." before the script file path to turn off restrictions on visibility
  - Scope modifiers are `private, local, script,` and `global`
  - Type: `$variable.GetType()`
  - Type safety: `[oneoftypes] $variable` where,

[oneoftypes]: [array], [bool], [byte],[char],[datetime], [decimal], [double], [guid], [hashtable], [int16], [int32], [int], [int64],, [long], [nullable], [psobject], [regex], [sbyte], [scriptblock], [single], [float], [string], [switch], [timespan], [type], [uint16], [uint32], [uint64], [xml]
    - Variable Options: "None","ReadOnly","Constant","Private","AllScope"
    - Variable Validation Classes: ValidateNotNullOrEmptyAttribute,ValidatePatternAttribute,ValidateRangeAttribute,ValidateSetAttribute

Operators:
- **Arithmetic:**
    - Hex are prefixed with 0x
    - Operators: +, -, *, /, %
- **Conditional:** -eq, -ceq, -ieq, -ne, -cne, -ine, -gt, -cgt, -igt, -ge, -cge, -ige, -lt, -clt, -ilt, -le, -cle, -ile, -contains, -ccontains, icontains, -notcontains, -cnotcontains, -inotcontains, -not/!, -and, -or, -xor
- If/ElseIf/Else
- Switch w/ default
- strings (case sensitive)
- wildcards
- regex
- expressions,

Arrays:
- The comma always creates an array.
- Verify e.g. $a is [Array]
- Index starts at 0 and -1 is last element
- Force to return array by using @()
- Create New Array i.e.
    - $arr = 1..4
    - $arr = 1,2,3,4
    - $arr = "1","2"
    - $arr = 1
    - $arr = @()
- Reverse Arrays
- New elements Can be added by using += operator
- Strongly Typed Arrays: [[int]]$arr = 1,2,3
- Arrays (returns matching / unlatching members)

Hash Tables:
- Key-Value pair sep by ;
- Access/Modify using either [] or .dot

Note: Arrays and HT are references to data

Pipeline:
- Cmdlets To Pipe: Compare-Object, ConvertTo-Html, Export-Clixml,Export-Csv, ForEach-Object, Format-List, Format-Table, Format-Wide, Get-Unique, Group-Object, Import-Clixml, Measure-Object, more, Out-File, Out-Host, Out-Host, -paging, Out-Null (suppress output), Out-Printer, Out-String, Select-Object, Sort-Object, Tee-Object, Where-Object (?), Out-Default, Out-Host, * (wild card), Scriptblocks,
- Modes: Sequential/Streaming Mode

- ETS - Extended Type System: Emergency mode,
- `$$` - last token of pipeline

Loops:
- `ForEach-Object ("{indexes}") == %`
- `ForEach` - works with collection
- `Do/While, While, For, break, continue, :label`

Aliases:
- Example: PowerShell Virtual Drives
  ```
  $alias: Get-PSDrive
  dir alias:
  ```
- Group-Object definition
- Create new aliases using `Set-Alias`, i.e. `Set-Alias edit notepad.exe`
- Save Aliases using profiles or `Export-Alias/Import-Alias` cmdlets.
- *Life span of alias and function are only when PS console is open*.

Functions:
- Syntax: `myFunc { commands/code $args }`, multi line `(enter)` / single line `(;`
- Calling: `$function:tabexpansion | Out-File myscript.ps1/>file.ps1`
- Write-protected functions: `Set-Item function:{//code} -option constant`
- Delete function: `Del function:test`,
- Arguments: `Args: $args` - spaces ""
- Parameters: `function myFunc([type] $Param1=110, $Param2=$(Get-Date), [Switch] ParamN)`
- Returns are array or return keyword or omit returns
- `Write-Debug ,$ErrorActionPreference,`
- `Stop-Process -nameofFunction`
- `Dir function:`
- `$function: prompt`
- `Function {begin{}process{}end{}}`

Object:
- `New-Object:` Creates new object
- `Add-Member (NoteProperty, ScriptMethod, CodeMethod, Method)`
- Properties and Methods: `$objectName.PropertyName` or `$objectName.MethodName()`
- Versioning: Major, Minor, Build and Revision
- `-static`, constructors
- Can load assemblies and COM.

Scripts & Other Language Integration:
- Windows Batch Files
- VBScript `(Wscript //H:CScript - console, WScript //H:WScript - Windows)`
- `.\filename.ext`
- Own: `ps1.`
- Before executing scripted created on other machines you must `Set-ExecutionPolicy` to `RemoteSigned`
- `.\myscript.ps1`
- `Get-ExecutionPolicy, Set-ExecutionPolicy RemoteSigned`
- `md $env:appdata\PSScripts`
- `\myscript.ps1 Tobias,`
- `$($args[0])`
- `param ($path, $name), param ([string] $Name=$( ` Throw "Parameter missing: -name Name")`

- [int]$age=$(`,. .\calcfunctions.ps1,
- . $env:appdata\PSLib\calcfunctions.ps1,
- {begin{} process{} end{}}
- Four Different Profile Scripts: All users[private],Current user[private]

Debugging

- What-if
- Confirm:Low, Medium, and High, $ConfirmPreference = "Low"
- ErrorAction/$ErrorActionPreference:SilentlyContinue,Continue,Stop,Inquire
- $?
- Trap {}
- $_.Exception.Message
- ErrorVariable myError
- $Error
- Throw
- Write-Debug, $DebugPreference:SilentlyContinue,Continue,Stop,Inquire, Set-PSDebug –step
- @"
- {index[,alignment][:format]}