### Windows PowerShell Cheat Sheet

**Category** | **Description** | **Examples**
---|---|---
**Variable** | Precede all variable names with `$` | `$variableName = "variable value"`
**Automatic Variables** | Variables that are created at runtime based on context. | `$true`, `$false`, `$null`, `@{}`
**Operators** | Support a variety of operators for arithmetic, comparison, logic, assignments, and more. | `-eq`, `!=`, `<`, `>`, `==`, `|`, `&`, `&|`,
**Escape Character** | Use the backward tick to escape special characters such as quotes and the dollar sign. | `$text = "Tessa says "hello!""`
**Write Output** | Use `Write-Host` to dump to the console. Use `Write-Output` to dump to the pipeline. When accessing variable members wrap in `{}`. | `Write-Host "It's a great day to learn PowerShell!"`
**Types** | Surround type name with square brackets. Some common data types are aliased for brevity. | `[Microsoft.SharePoint.SPBasePermissions]`
**Static** | Call static members by separating the type and member by two colons. | `[Microsoft.SharePoint.SPBasePermissions]:..Local`
**Creating Objects** | Use the New-Object cmdlet (pass constructor args as an array). Pivot a hash table using the PSObject type. | `$obj = New-Object PSObject -Property "hash"`
**Throw Errors** | Use the throw keyword. | `throw "An unknown error occurred."`
**Catch Errors** | Use the try/catch/finally keywords. `$` represents the error object in the catch block. Add an optional type after the catch keyword to catch a specific exception (you can have multiple catch blocks). | `$web = Get-SWeb http://demo`
**Functions** | Declare using the `function` keyword. Arguments are comma separated and wrapped in parenthesis. Function body is wrapped in curly braces. | `function Get-SPGroup {
        $spWeb = $web.Read()
        $spGroup = $spWeb.SiteGroups[$group]
        $spWeb.Dispose() } return $spGroup }`
**Passing Script / Function Args** | No commas or parenthesis. Positional or named. PowerShell script and function parameters only! | `Get-SPContentDatabase | Get-SPObject { if ($_.DiskSizeRequired -gt 100GB) { Write-Host "Over Limit: 
    $($_.Name)" } else if ($_.DiskSizeRequired -gt 80GB) { Write-Host "Close to Limit: 
    $($_.Name)" } else {Write-Host "Good: $($_.Name)"} } }`
**Filter Results** | Use Where-Object (aliased as where and ?) to filter pipeline objects; use Select-Object (aliased as select) to display specific properties. | `Get-SPContentDatabase | where { [$_.DiskSizeRequired -gt 80GB] | select Name, Server, DiskSizeRequired | sort DiskSizeRequired -Descending }`
**Find Cmdlets and Members** | Use Get-Command (aliased as gccmd) to find cmdlets; use Get-Member (aliased as gm) to display object members. | `Get-Cmdlet *command` `Get-SPSite http://demo | Get-Member`
**Define Script Parameters** | Use the param keyword to define one or more parameters (wrap in parenthesis). Comma-separate parameters. (Works with function parameters too). | `param(
    [Microsoft.SharePoint.PowerShell.SPWebPipeBind]$web = ($throw -Web is required.),
    [switch] $force, [string] $backupPath = "C:\Backups")`
**Dot Source** | Load scripts using `<dtd><space>path|file.ps1 format` to access functions in scripts | `PS C:\> . C:\Scripts\Manage-SPGroup.ps1` `PS C:\> .\Scripts\Manage-SPGroup.ps1`