Windows PowerShell Cheat Sheet

APTILLON

Variable

Precede all variable names with `$`

```
$variableName = "variable value"
```

Automatic Variables

Variables that are created at runtime based on context.

Variable	Description
$true	A TRUE value.
$false	A FALSE value.
$null	A null value.
$()	Sub-expression.

Variable	Description
$_	The current object in a pipeline operation.
$?	Last operation execution status.
$Error	Array of error objects ($Error[0] is last error).
$LastExitCode	Contains the last executable program's exit code.

Operators

Traditional equality, comparison, and logical operators cannot be used (except for "!").

==	!=	<	<=	>	>=	&&	||	!	&	|	^
-eq	-ne	-lt	-le	-gt	-ge	-and	-or	-not (or !)	-band	-bor	-xor

Escape Character

Use the backward tick to escape special characters such as quotes and the dollar sign.

```
$text = "Tessa says `"hello!`""          >> Tessa says "hello!"
$pwd = "pa`$`$w0rd"                      >> pa$$w0rd
```

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!"
Write-Host "Storage = $($site.Usage.Storage/1MB)MB"
Write-Output $site
```

Types

Surround type name with square brackets. Some common data types are aliased for brevity.

```
[Microsoft.SharePoint.SPBasePermissions]
[xml], [int], [string], [bool], etc.
```

Statics

Call static members by separating the type and member by two colons.

```
[Microsoft.SharePoint.SPBasePermissions]::ManageWeb
[Microsoft.SharePoint.Administration.SPFarm]::Local
[Microsoft.SharePoint.Publishing.PublishingWeb]::GetPublishingWeb($web)
```

Type Cast

Precede variable name with type or use `-as` operator. PowerShell can also do a lot of implicit type casting.

```
[Microsoft.SharePoint.SPBasePermissions]"ManageWeb"
$perm = "ManageWeb" -as [Microsoft.SharePoint.SPBasePermissions]
[xml]$xml = "<Site Url='http://demo' />"
$roleDefinition.BasePermissions = "ViewListItems","AddListItems"
```

Arrays

Comma separate values. Declare using `@()`.

```
$perms = "ManageWeb", "ManageSubwebs"

$perms = @()
$perms += "ManageLists"
$perms += "ManageWeb", "ManageSubwebs"
```

Hash Tables

Declare using `@{}`. Separate key/value pairs with a semicolon. Values can include script blocks.

```
$values = @{Url="http://demo"; OwnerAlias="Aptillon\glapointe"}
$values += @{Template="STS#0}
```

Creating Objects

Use the `New-Object` cmdlet (pass constructor args as an array).
Pivot a hash table using the `PSObject` type.

```
$field = New-Object Microsoft.SharePoint.SPFieldText $fields, "Text", $fieldName

$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-SPWeb http://demo
try {
  $list = $web.GetList("Foo List")
} catch {
  Write-Warning "Could not find list. $($_.Exception.Message)"
} finally { $web.Dispose() }
```

Functions

Declare using the `function` keyword. Arguments are comma separated and wrapped in parenthesis. Function body is wrapped in curly braces.

```
function Get-SPGroup(
  [Microsoft.SharePoint.PowerShell.SPWebPipeBind]$web,[string]$group) {
    $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!

```
$group = Get-SPGroup "http://demo" "Demo Owners"
$group = Get-SPGroup -Web http://demo -Group "Demo Owners"
```

Loops

The `do/while`, `while`, `for`, and `foreach` loops are built-in constructs. `ForEach-Object` (aliased as `foreach` and `%`) is a cmdlet (use `$_` for the current object). `ForEach-Object` does not support break or continue statements.

```
do { Start-Sleep 2 } while (!(Get-SPSolution $name).Deployed)
while (!(Get-SPSolution $name).Deployed) { Start-Sleep 2 }
foreach ($site in (Get-SPSite -Limit All)) {$site.Url}
for ($i = 0; $i -lt 10; $i++) {Write-Host $i}

$web.Fields | ForEach-Object {$_.SchemaXml} | Out-File "C:\Fields.xml"
```

Conditionals

Use `if/elseif/else` statements or the `switch` statement to provide conditional logic. (Type `help about_switch` for information about the `switch` statement.)

```
Get-SPContentDatabase | ForEach-Object {
  if ($_.DiskSizeRequired -gt 100GB) {Write-Host "Over Limit: $($_.Name)" }
  elseif ($_.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

Get-SPContentDatabase | select
@{Expression={"$($_.DiskSizeRequired/1GB)GB"};Label="Size"}
```

Find Cmdlets and Members

Use `Get-Command` (aliased as `gcm`) to find cmdlets; use `Get-Member` (aliased as `gm`) to display object members.

```
Get-Command *service*
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 <dot><space>path\file.ps1 format to access functions in scripts

```
PS C:\> . C:\Scripts\Manage-SPGroup.ps1      >> Use the absolute path
PS C:\> . .\Scripts\Manage-SPGroup.ps1       >> Or the relative path
```