PowerShell Cheatsheet

A concise reference for PowerShell commands, syntax, and best practices, designed to help both beginners and experienced users quickly find the information they need.

Core Cmdlets & Syntax

Basic Cmdlets

`Get-Command`	Retrieves all available cmdlets, functions, and aliases. **Example:** `Get-Command -Name Get-Process`
`Get-Help`	Displays help information about cmdlets, functions, and scripts. **Example:** `Get-Help Get-Process -Examples`
`Get-Member`	Inspects the properties and methods of an object. **Example:** `Get-Process | Get-Member`
`Set-Variable`	Creates and sets the value of a variable. **Example:** `Set-Variable -Name MyVar -Value "Hello"`
`Write-Host`	Displays output in the console. **Example:** `Write-Host "Hello, World!"`
`Write-Output`	Sends output as objects to the pipeline. **Example:** `Write-Output "Object data"`

Basic Syntax

$ - Variable declaration (`$var = 'value'`)
| - Pipeline (passes output of one command to another)
; - Command separator (allows multiple commands on one line)
() - Grouping commands for execution order or output capture
{} - Script block (used in loops, conditional statements, etc.)

Operators:
-eq (equals), -ne (not equals), -gt (greater than), -lt (less than), -ge (greater than or equals), -le (less than or equals), -and (logical AND), -or (logical OR), -not (logical NOT)

Redirection:
`>` (Output to file, overwrites), `>>` (Output to file, appends), `2>` (Error output), `2>&1` (Merge error and standard output)

Data Manipulation & Scripting

Working with Data

`ConvertFrom-Json`	Converts a JSON string to a PowerShell object. **Example:** `$json = '{"name":"John", "age":30}'; $obj = $json | ConvertFrom-Json`
`ConvertTo-Json`	Converts a PowerShell object to a JSON string. **Example:** `$obj | ConvertTo-Json`
`Import-Csv`	Imports data from a CSV file. **Example:** `$data = Import-Csv -Path 'data.csv'`
`Export-Csv`	Exports data to a CSV file. **Example:** `$data | Export-Csv -Path 'output.csv' -NoTypeInformation`
`ForEach-Object`	Iterates through a collection of objects. **Example:** `1,2,3 | ForEach-Object { $_ * 2 }`

Conditional Statements

If/Else:

```
if ($condition) {
 # Code to execute if condition is true
} elseif ($anotherCondition) {
 # Code to execute if another condition is true
} else {
 # Code to execute if all conditions are false
}
```

Looping

For Loop:

```
for ($i = 0; $i -lt 10; $i++) {
 # Code to execute
}
```

While Loop:

```
while ($condition) {
 # Code to execute while condition is true
}
```

Do-While Loop:

```
do {
 # Code to execute at least once
} while ($condition)
```

Functions and Modules

Functions

Defining a Function:

```
function Get-MyInfo {
  param (
    [string]$Name,
    [int]$Age
  )

  Write-Host "Name: $Name"
  Write-Host "Age: $Age"
}
```

Calling a Function:

```
Get-MyInfo -Name "Alice" -Age 30
```

`return`	Used to exit a function and optionally return a value. If no value is provided, nothing is returned.
	
	Example:
	`function Get-Value { return 10 }`
	`$result = Get-Value # $result will contain 10`

Modules

`Import-Module`	Imports a module into the current session. **Example:** `Import-Module -Name 'MyModule'`
`Get-Module`	Lists the modules that are imported in the current session. **Example:** `Get-Module -ListAvailable`
`Export-ModuleMember`	Specifies which module members (cmdlets, functions, variables, aliases) are exported from a script module. **Example:** `Export-ModuleMember -Function 'My-Function'`

Error Handling & Advanced Features

Error Handling

Try/Catch/Finally:

```
try {
  # Code that might throw an exception
  # Example: $result = Get-Content -Path 'nonexistent_file.txt'
} catch {
  # Code to handle the exception
  Write-Host "Error: $($_.Exception.Message)"
} finally {
  # Code that always executes, regardless of whether an exception occurred
  Write-Host "Finally block executed."
}
```

`$ErrorActionPreference`	Sets the default behavior when an error occurs. Common values: 'Stop', 'Continue', 'SilentlyContinue', 'Inquire'. **Example:** `$ErrorActionPreference = 'Stop'`
`Throw`	Throws a custom exception. **Example:** `Throw "Custom error message"`

Background Jobs

`Start-Job`	Starts a background job. **Example:** `Start-Job -ScriptBlock { Get-Process }`
`Get-Job`	Retrieves background jobs. **Example:** `Get-Job`
`Receive-Job`	Receives the output of a background job. **Example:** `Receive-Job -Id 1`
`Stop-Job`	Stops a running background job. **Example:** `Stop-Job -Id 1`
`Wait-Job`	Waits for a background job to complete. **Example:** `Wait-Job -Id 1`
`Remove-Job`	Removes a background job. **Example:** `Remove-Job -Id 1`

Remoting

`Enter-PSSession`	Starts an interactive session with a remote computer. **Example:** `Enter-PSSession -ComputerName 'RemoteServer'`
`Invoke-Command`	Runs commands on a remote computer. **Example:** `Invoke-Command -ComputerName 'RemoteServer' -ScriptBlock { Get-Process }`