

# PowerShell FEATURES

**UI Features**

- Regular Expressions
- Exception Handling
- Array Support
- Functions
- Script Signing
- Tab Completion

**Execution Trust**

- [D] Do not run
- [R] Run once
- [A] Always run
- [?] Help

**Policies** → Set-Executionpolicy Unrestricted

- AllSigned
- RemoteSigned
- Unrestricted

**Command Nuggets**

help ... PowerShell g11  
 Get-Command ... get list of all cmdlets  
 Get-Help ... obtains details on a cmdlet  
 [cld] Clear-Host ... cleanup the working area  
 [dir] Get-ChildItem ... displays contents of current working directory  
 PS C:\> \$env:path ... see content of default path  
 PS C:\> .\ ... temporarily add current folder to search path

**Escape Characters**

'	Single quote	”	Double quote
`	Null	`a	Alert
`b	Backspace	`f	Form feed
`n	Newline	`r	Carriage return
`t	Horizontal tab	`v	Vertical tab

**Operator Character(s)**

=	Assignment
+	Add, append
-	Subtract
*	Multiply
/	Division
%	remainder of division (modulus)
+=	Add value to variable
-=	Subtract value from variable
*=	Multiply a variable
/=	Divide a variable
%=	Assign modulus

**Special Characters**

- \$\_ Current pipeline object when used in script blocks
- \$Error Recent error information
- \$HOME Home directory of user
- \$PSHome PowerShell home
- \$null Null object

**ErrorAction Arguments**

Script statement `$ErrorActionPreference = "..."`  
 Calling CmdLet args `-ErrorAction "..."`

**Continue** Generate error + continue  
**Stop** Generate error + stop  
**SilentContinue** Pretend nothing is wrong  
**Inquire** Generate error + ask user

**Trap Handling**

```
Trap [Exception] {
    statements
    Return [Value] | Continue | Break
}
```

Return Value  
 Continue execution after statement that caused exception  
 Terminate current execution

**Debugging Scripts**

Cmdlet to define debugging detail

Trace level detail argument

- 0 → No tracing
- 1 → Display each script statement
- 2 → Display (1) + var values and function calls

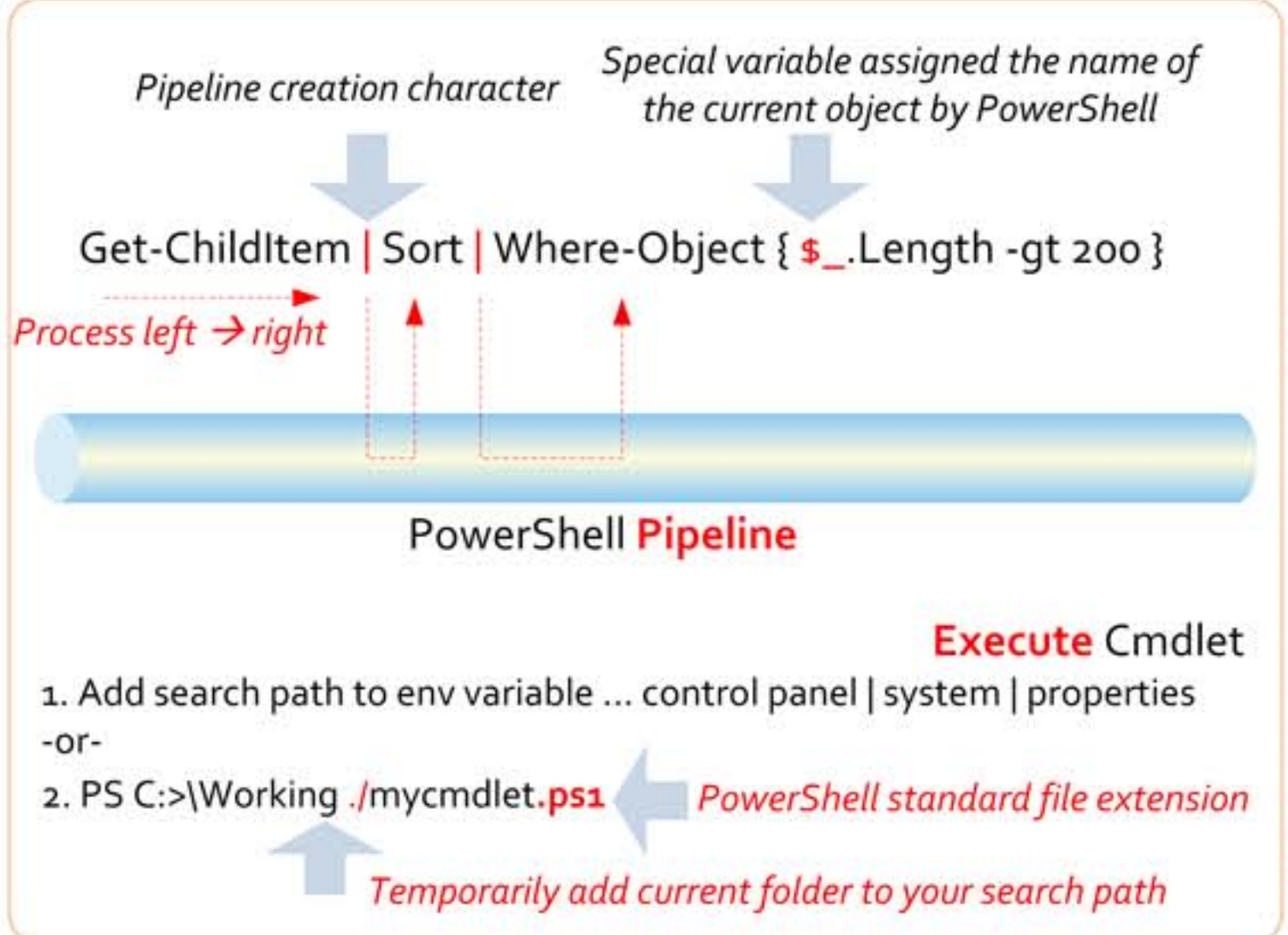
**Debugging Scripts**

```
PS C:\> .\eventest.ps1
DEBUG: 1* Set-PSDebug -Trace 1
DEBUG: 2* filter Is-Even 1
DEBUG: 11* write-host "Odd and even test program"
Odd and even test program
DEBUG: 13* $x = read-host "Enter a number please"
Enter a number please: 5
DEBUG: 15* $z = 0($x) | Is-Even
DEBUG: 15* $z = 0($x) | Is-Even
DEBUG: 3* write-host "$x + $y is + $z"
5 + 1 is + 5
DEBUG: 4* $y = $z * 2
DEBUG: 5* write-host "$y + $x is + $z"
11 + 5 is + 5
DEBUG: 6* if ( $y -eq 0 ) { $result = "true" }
else { $result = "false" }
DEBUG: 8* $result
DEBUG: 17* write-host "n Number " + $x + " even claim is " + $z
Number + 5 + even claim is + false
PS C:\>
```

Visit <http://www.powershell.com> for Analyzer, IDE, Community, ...

**PowerShell Data Types**

[array]	[hashtable]	[double]
[bool]	[int]	[float]
[byte]	[long]	[switch]
[char]	[single]	
[decimal]	[string]	



**Comparison operators**

-eq	Equal	-ne	Not equal
-lt	Less than	-le	Less than or equal
-gt	Greater than	-ge	Greater than or equal

**String comparison operators**

-not	Not	-and	And
!	Not	-or	Or

**Logical operators**

-ieq	Equal to	Case Insensitive
-ilt	Less than	Case Insensitive
-igt	Greater than	Case Insensitive
-ige	Greater than or equal to	Case Insensitive
-ile	Less than or equal to	Case Insensitive
-ine	Not Equal	Case Insensitive
-ceq	Equal to	Case Sensitive
-clt	Less than	Case Sensitive
-cgt	Greater than	Case Sensitive
-cge	Greater than or equal to	Case Sensitive
-cle	Less than or equal to	Case Sensitive
-cne	Not Equal	Case Sensitive

**variable SCOPE**

`$localVariable = $script:variableName`

Local Variable are restricted by scope

Script-level variables scope is established when script is executed and ends when script stops.

**processing INCOMING data**

**sinput** → delay execution of function until all data is collected

For example:

```
function Get-FileName {
    $input | Where-Object { $_.Name -ne "WINDOWS" } | Sort-Object
}
```

- Get-FileName collects all data passed to it in sinput variable
- Pass this data down the pipeline to the "Where-Object" cmdlet
- Pass remaining data to Sort-Object cmdlet

# PowerShell SYNTAX

**OBJECTS**

**Object concatenation**

```
$x = "PowerShell is: "
$y = "Cool"
$z = $x + $y
```

**Object repetition**

```
$x = "Ha " * 5
```

**Object replacing**

```
$x = "HaHaWeepHa "
$y = $x -replace "Weep" "Ha"
```

**CONDITIONAL logic**

**Comparisons**

```
5 -eq 10
false
```

**Combining Pipelines and Operators**

```
PS: C:\> Get-Process | Where-Object { $_.Processname -eq "Outlook" }
```

**If statement**

```
if (condition) {code block}
elseif (condition) {code block}
else {code block}
```

**if** (\$energy -gt 100) {\$energyLevel = 0}  
 elseif (\$energy -ge 50) {\$energyLevel = 1}  
 else {\$energyLevel = 2}

**switch statement**

```
switch (expression) {
    value {codeblock}
    default {codeblock}
}
```

**switch** (\$energyLevel)

```
{
    0 {Write-Host "Excellent"}
    1 {Write-Host "Reasonable"}
    2 {Write-Host "Caution"}
    default {Write-Host "SOS"}
}
```

**FUNCTIONS**

**function Structure**

```
function name (params) {
    [$result=] code block
    [$result]
}
```

**function name** {  
 params ([datatype]name,...)  
 [\$result=] code block  
 [\$result]  
}

**function name** {  
 [\$result=] \$args[0] + \$args[1]  
 [\$result]  
}

**function Add-Numbers** ( \$x, \$y ) {  
 \$result = \$x + \$y  
 \$result  
}

**Function Calling**

```
$z = Add-Numbers 2, 3
$z = Add-Numbers -x 2, -y 3
```

Positional  
 Named

Parameters are positional and named

**ARRAYS**

**Array Creation and Management**

```
# -- simple array
$x = $()
$x[0] = "amazing"

# -- drops second '5'
$numbers = @(0,1,2,3,4,5,5,6,7,8,9)
$numbers = $numbers[0..5 + 7..9]
```

**Associative Array**

```
@ids = {}
$ids[1962] = "Carola"
$ids[1999] = Thorsten

$nicknames = {}
$nicknames = @{Alexander="Tiger"}
$x = $nicknames["Alexander"]
```

**LOOPS**

**do statement**

```
do {
    code block
} while ( condition )
```

**do** {  
 \$i++  
} while ( \$i -le 10 )

**while statement**

```
while ( condition ) {
    code block }
```

**while** ( \$i -le 10 ) {  
 \$i++  
}

**for statement**

```
for (initialization;condition;step) {
    code block }
```

**for** (\$i=0; \$i -le 10; \$i++)  
 { j++ }

**foreach statement**

```
foreach ( element in collection ) {
    code block }
```

**foreach** ( \$i in numbers )  
 { Write-Host \$i }

**Altering Loop Execution statements**

**break** - terminate loop  
**continue** - skip current, continue

**FILTERS**

**filter Structure**

```
filter name { code block }
#filter has access to input pipeline
```

**filter Is-Even** {  
 write-host "\$\_" + " is " + \$\_  
 \$y = \$\_ % 2  
 write-host "\$y" + " is " + \$\_  
 if ( \$y -eq 0 ) {\$result = "true"}  
 else {\$result = "false"}  
 \$result  
}

```
write-host "Odd and even test program"
$x = read-host "Enter a number please"
$z = @($x) | Is-Even
write-host "`n Number even claim is " + $z
```