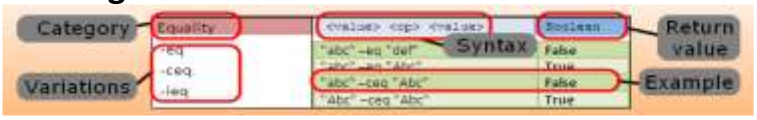
PowerShell String Comparison and List Filtering

This reference brings together relevant operators plus key language constructs to compare strings in either scalar or array context. (Available online at Simple-Talk.com at http://bit.ly/l7g6Fj.)



Operator [1]	String		Array	
Equality	`<value> <op> <value>`	Boolean	`<array> <op> <value>`	Sub-list
–eq	"abc" –eq "def"	**False**	"dog","dogwood","cat","Dog" –eq "dog"	**("dog","Dog")**
–ceq	"abc" –eq "Abc"	**True**	"dog","dogwood","cat","Dog" –ceq "Cat"	**( )**
–ieq	"abc" –ceq "Abc"	**False**	@() –eq "dog"	**( )**
	"Abc" –ceq "Abc"	**True**		
Equality/negated	`<value> <op> <value>`	Boolean	`<array> <op> <value>`	Sub-list
–ne	"abc" –ne "def"	**True**	"dog","cat","Dog" –ne "dog"	**("cat")**
–cne	"abc" –ne "Abc"	**False**	"dog","cat","Dog" –cne "dog"	**("cat","Dog")**
–ine	"abc" –cne "Abc"	**True**	@() –ne "dog"	**( )**
	"Abc" –cne "Abc"	**False**		
Wildcard (glob) [2]	`<target> <op> <glob>`	Boolean	`<array> <op> <glob>`	Sub-list
–like	"dog" –like "dog*"	**True**	"f42e","12a8","a000","948f" –like "[a-f]*"	**("f42e","a000")**
–clike	"kookaburra" –like "k??k*burra"	**True**	"f42e","12a8","a000","948f" – like "[a-f]"	**( )**
–ilike	"kookaburra" –like "k?k*burra"	**False**	"dove","wren","Warbler" –like "w*"	**("wren","Warbler")**
	"kookaburra" –clike "K*"	**False**	"dove","wren","Warbler" –clike "w*"	**("wren")**
	"kookaburra" –clike "[kK]*"	**True**		
Wildcard/negated [2]	`<target> <op> <glob>`	Boolean	`<array> <op> <glob>`	Sub-list
–notlike	"coelacanth" –notlike "cat"	**True**	"dove","wren","Warbler" –notlike "w*"	**("dove")**
–cnotlike	"dog" –notlike "D?g"	**False**	"dove","wren","Warbler" –cnotlike "w*"	**("dove","Warbler")**
–inotlike	"dog" –cnotlike "D?g"	**True**	"dove","wren","Warbler" –notlike "*"	**( )**
Regular expression [3]	`<target> <op> <regex>`	Boolean [4]	`<array> <op> <regex>`	Sub-list
–match	"archaeopteryx" –match "arch.*"	**True**	"nutria","beaver","muskrat" –match "[mn]u.*"	**("nutria","muskrat")**
–cmatch	"archaeopteryx" –match ".*(ae|ea).*"	**True**	"a4.001","b3.902","c3.4he" –match "\.[0-9]{2,}"	**("a4.001","b3.902")**
–imatch	"archaeopteryx" –match "ae|ea"	**True**	"notebook","book","bookend" –match "book$"	**("notebook","book")**
			"notebook","book","bookend" –match "^book$"	**("book")**
Regex/negated [3]	`<target> <op> <regex>`	Boolean [4]	`<array> <op> <regex>`	Sub-list
–notmatch	"bird" -notmatch "Bird.*"	**False**	"dove","wren","Warbler" -notmatch "w.*"	**("dove")**
–cnotmatch	"bird" -cnotmatch "Bird.*"	**True**	"dove","wren","Warbler" -cnotmatch "w.*"	**("dove","Warbler")**
–inotmatch				
Membership	`<target>.contains(<value>)`	Boolean	*Not Available*	
contains()	"archaeopteryx".contains("aeo")	**True**		
	"archaeopteryx".contains("aeiou")	**False**		
Membership	`<target> <op> <value>`	Boolean [5]	`<array> <op> <value>`	Boolean
–contains	"dog" –contains "Dog"	**True**	"dog","dogwood" –contains "Dog"	**True**
–ccontains	"dog" –ccontains "Dog"	**False**	"dog","dogwood" –ccontains "Dog"	**False**
–icontains	"dog" –contains "d"	**False**	"dog","dogwood","catfish" –ccontains "cat"	**False**
Membership/negated	`<target> <op> <value>`	Boolean [5]	`<array> <op> <value>`	Boolean
–notcontains	"dog" –notcontains "Dog"	**False**	"dog","dogwood" –notcontains "Dog"	**False**
–cnotcontains	"dog" –cnotcontains "Dog"	**True**	"dog","dogwood" –cnotcontains "Dog"	**True**
–inotcontains				

Switch command [6]
This syntax applies to all variants below.
	`switch ( <value> )`
	`{`
	`  <choice> {<statements>}`
	`  <choice> {<statements>}`
	`  . . .`
	`}`
		Arbitrary (or ***no*** return value)	`switch ( <array> )`
			`{ # iterates through the list`
			`  <choice> {<statements>}`
			`  <choice> {<statements>}`
			`  . . .`
			`}`
				Arbitrary (or ***no*** return value)

Branch/equality
Switch [–Exact]
[–CaseSensitive]
	Switch ("maybe") {
	"yes" {10}
	"no" {20}
	}
		Null	Switch ("dog","bird","lizard") {
			{ "dog","cat" –contains $_ } { "$_ : housepet" }
			Default { "$_ : not sure" }
			}
				dog : housepet
				bird : not sure
				lizard : not sure

Branch/wildcard [2]
Switch –Wildcard
[–CaseSensitive]
	Switch –wildcard ("a13") {
	"a??" {10}
	"b??" {20}
	default {$null}
	}
		10	Switch –wildcard –case ("dog","bird","Dog") {
			"D*" { "$_ : housepet" }
			"b??d" { "$_ : housepet" }
			Default { "$_ : not sure" }
			}
				dog : not sure
				bird : housepet
				Dog : housepet

Branch/regex [3]
Switch –Regex
[–CaseSensitive]
	Switch –regex ("sR9X2T") { [4]
	"^[a-l]" {10}
	"^[m-y]" {20}
	"^[z]" {99}
	default {$null}
	}
		20	switch –regex ("dog", "cat", "catfish", "catbird") {
			"cat(?!fish)" { "$_ : land" }
			"seal|whale|dolphin|catfish" { "$_ : sea" }
			"owl|eagle|osprey|catbird" { "$_ : air" }
			default { ("$_ : " + $null) }
			}
				dog : Null
				cat : land
				catfish : sea
				catbird : land
				catbird : air [7]

Select–String *This syntax applies to all variants below.*	`<target> <op> <value>`	string	`<target> <op> <value>`	Sub-list
Select–String/equality	"dog" | ss –simple "dog"	**"dog"**	"dog","Dog" | ss –simple "dog"	**("dog","Dog")**
ss [8] –SimpleMatch [–CaseSensitive]	"dog" | ss –simple "do"	**"dog"**	"dog","Dog","dogbone" | ss –case –simple "dog"	**("dog","dogbone")**
Select–String/wildcard	*Not Available*		*Not Available*	
Select–String/regex	"coelacanth" | ss "c..l.*th"	**"coelacanth"**	"a1","a2","ab3","AB3" | ss "ab.*"	**("ab3","AB3")**
ss [8] [–CaseSensitive]	"coelacanth" | ss "c.*"	**"coelacanth"**	"a1","a2","ab3","AB3" | ss –case "ab.*"	**("ab3")**
			"ab3","abcd","ado" | ss "ab*" [9]	**("ab3","abcd","ado")**
Select–String/negated	"dog" | ss –simple -NotMatch "dog"	**Null**	"dog","Cat","catfish" | ss –not "Cat.*h"	**("dog","Cat")**
ss [8] –NotMatch	"dog" | ss –simple -NotMatch "cat"	**"dog"**	"dog","Cat","catfish" | ss –simple -not -case "Cat"	**("dog","catfish")**
[–SimpleMatch] [–CaseSensitive]	"dog" | ss –not ""	***<illegal>***	"dog","dogbone" | ss –not "dog"	**Null**

LEGEND

🟩	Equality
🟪	Wilcard
🟥	Regex

1 Each operator has three variations:
> **default** (e.g. –eq),
> **case-sensitive** (e.g. –ceq), and
> **case-insensitive** e.g. –ieq).
Note that the default in each case is case–insensitive so **–eq** is exactly equivalent to **–ieq**; the latter is provided if you have a preference for being explicit.
See about_Comparison_Operators.

2 Wildcards include:
> asterisk (*) for any number of chars;
> question mark (?) for any single char;
> brackets ([ ]) for single, enumerated char or char range.
Must match input in its entirety.
See about_Wildcards.

3 Regular expressions provide a powerful but complex matching construct; the PowerShell reference (about_Regular_Expressions) documents only a portion of it; PowerShell actually supports the full .NET implementation—see Regular Expression Language Elements .

4 Populates **$Matches** where:
> **$Matches**[0] contains entire match
> **$Matches** [*n*] contains ***n***th match

5 **–contains** technically only operates on a list; with a scalar it is equivalent to **–eq**.

6 The **switch** statement implicitly uses **–eq** in selecting a match; specifying **–CaseSensitive** modifies this to **–ceq**. The **–Wildcard** and **–Regex** parameters may be used to effect **–like** or **–match**, respectively. Similarly adding **–CaseSensitive** modifies these to **–clike** or **–cmatch**. Switch syntax even allows specifying your own arbitrary operator or more complex Boolean expression: instead of specifying a choice as a simple value (string, number, or variable) use a code block to specify an expression, where the standard **$_** automatic variable references the input value.
See about_Switch.

7 This deliberate error shows that **switch** evaluates every expression unless you use **break** statements!

8 **Select–String** examples use a custom **ss** alias for brevity.

9 This might look like a wildcard, but it is a regex! As a wildcard, it would have returned ("ab3","abcd") only.

Other References:
about_Operators
Conditional Operators
Operator enumeration
Mastering PowerShell, chapter 7