Date: 29 Dec 2009 15:35
Topic: Flags, confronto ecc

The following reference cards provide a useful *summary* of certain scripting concepts. The foregoing text treats these matters in more depth, as well as giving usage examples.

Table B-1. Special Shell Variables

Variable	Meaning
`$0`	Filename of script
`$1`	Positional parameter #1
`$2 - $9`	Positional parameters #2 - #9
`${10}`	Positional parameter #10
`$#`	Number of positional parameters
`"$*"`	All the positional parameters (as a single word) *
`"$@"`	All the positional parameters (as separate strings)
`${#*}`	Number of command-line parameters passed to script
`${#@}`	Number of command-line parameters passed to script
`$?`	Return value
`$$`	Process ID (PID) of script
`$-`	Flags passed to script (using *set*)
`$_`	Last argument of previous command
`$!`	Process ID (PID) of last job run in background

* *Must be quoted*, otherwise it defaults to `"$@"`.

Table B-2. TEST Operators: Binary Comparison

Operator	Meaning	----	Operator	Meaning
		-		
Arithmetic Comparison			String Comparison	
-eq	Equal to		=	Equal to
			==	Equal to
-ne	Not equal to		!=	Not equal to
-lt	Less than		\<	Less than (ASCII) *
-le	Less than or equal to			
-gt	Greater than		\>	Greater than (ASCII) *
-ge	Greater than or equal to			
			-z	String is empty
			-n	String is not empty
Arithmetic Comparison	within double parentheses ((...))			
>	Greater than			

>=	Greater than or equal to			
<	Less than			
<=	Less than or equal to			

* *If within a double-bracket* [[...]] *test construct, then no escape* \ *is needed.*

Table B-3. TEST Operators: Files

Operator	Tests Whether	-----	Operator	Tests Whether
-e	File exists		-s	File is not zero size
-f	File is a *regular* file			
-d	File is a *directory*		-r	File has *read* permission
-h	File is a symbolic link		-w	File has *write* permission
-L	File is a *symbolic link*		-x	File has *execute* permission
-b	File is a block device			
-c	File is a character device		-g	*sgid* flag set
-p	File is a pipe		-u	*suid* flag set
-S	File is a socket		-k	"sticky bit" set
-t	File is associated with a *terminal*			

`-N`	File modified since it was last read		`F1 -nt F2`	File F1 is *newer* than F2 *
`-O`	You own the file		`F1 -ot F2`	File F1 is *older* than F2 *
`-G`	*Group id* of file same as yours		`F1 -ef F2`	Files F1 and F2 are *hard links* to the same file *
`!`	NOT (inverts sense of above tests)			

* *Binary* operator (requires two operands).

Table B-4. Parameter Substitution and Expansion

Expression	Meaning
`${var}`	Value of *var*, same as *$var*
`${var-DEFAULT}`	If *var* not set, evaluate expression as *$DEFAULT* *
`${var:-DEFAULT}`	If *var* not set or is empty, evaluate expression as *$DEFAULT* *
`${var=DEFAULT}`	If *var* not set, evaluate expression as *$DEFAULT* *
`${var:=DEFAULT}`	If *var* not set, evaluate expression as *$DEFAULT* *
`${var+OTHER}`	If *var* set, evaluate expression as *$OTHER*, otherwise as null string

`${var:+OTHER}`	If *`var`* set, evaluate expression as *`$OTHER`*, otherwise as null string
`${var?ERR_MSG}`	If *`var`* not set, print *`$ERR_MSG`* *
`${var:?ERR_MSG}`	If *`var`* not set, print *`$ERR_MSG`* *
`${!varprefix*}`	Matches all previously declared variables beginning with *`varprefix`*
`${!varprefix@}`	Matches all previously declared variables beginning with *`varprefix`*

* Of course if *`var`* *is* set, evaluate the expression as *`$var`*.

Table B-5. String Operations

Expression	Meaning
`${#string}`	Length of *`$string`*
`${string:position}`	Extract substring from *`$string`* at *`$position`*
`${string:position:length}`	Extract *`$length`* characters substring from *`$string`* at *`$position`*
`${string#substring}`	Strip shortest match of *`$substring`* from front of *`$string`*
`${string##substring}`	Strip longest match of *`$substring`* from front of *`$string`*

`${string%substring}`	Strip shortest match of *`$substring`* from back of *`$string`*
`${string%%substring}`	Strip longest match of *`$substring`* from back of *`$string`*
`${string/substring/replacement}`	Replace first match of *`$substring`* with *`$replacement`*
`${string//substring/replacement}`	Replace *all* matches of *`$substring`* with *`$replacement`*
`${string/#substring/replacement}`	If *`$substring`* matches *front* end of *`$string`*, substitute *`$replacement`* for *`$substring`*
`${string/%substring/replacement}`	If *`$substring`* matches *back* end of *`$string`*, substitute *`$replacement`* for *`$substring`*
`expr match "$string" '$substring'`	Length of matching *`$substring`** at beginning of *`$string`*
`expr "$string" : '$substring'`	Length of matching *`$substring`** at beginning of *`$string`*
`expr index "$string" $substring`	Numerical position in *`$string`* of first character in *`$substring`* that matches
`expr substr $string $position $length`	Extract *`$length`* characters from *`$string`* starting at *`$position`*
`expr match "$string" '\($substring\)'`	Extract *`$substring`** at beginning of *`$string`*
`expr "$string" : '\($substring\)'`	Extract *`$substring`** at beginning of *`$string`*

`expr match "$string" '.*\($substring\)'`	Extract *`$substring`** at end of *`$string`*
`expr "$string" : '.*\($substring\)'`	Extract *`$substring`** at end of *`$string`*

* Where *`$substring`* is a Regular Expression.

Table B-6. Miscellaneous Constructs

Expression	Interpretation
Brackets	
`if [ CONDITION ]`	Test construct
`if [[ CONDITION ]]`	Extended test construct
`Array[1]=element1`	Array initialization
`[a-z]`	Range of characters within a Regular Expression
Curly Brackets	
`${variable}`	Parameter substitution
`${!variable}`	Indirect variable reference
`{ command1; command2; . . . commandN; }`	Block of code
`{string1,string2,string3,...}`	Brace expansion
`{a..z}`	Extended brace expansion
`{}`	Text replacement, after find and xargs

Parentheses	
`( command1; command2 )`	Command group executed within a subshell
`Array=(element1 element2 element3)`	Array initialization
`result=$(COMMAND)`	Command substitution, new style
`>(COMMAND)`	Process substitution
`<(COMMAND)`	Process substitution
Double Parentheses	
`(( var = 78 ))`	Integer arithmetic
`var=$(( 20 + 5 ))`	Integer arithmetic, with variable assignment
`(( var++ ))`	*C-style* variable increment
`(( var-- ))`	*C-style* variable decrement
`(( var0 = var1<98?9:21 ))`	*C-style* trinary operation
Quoting	
`"$variable"`	"Weak" quoting
`'string'`	'Strong' quoting
Back Quotes	
`result=`COMMAND``	Command substitution, classic style