KORN SHELL 93 – TESTING & SUBSTITUTIONS

Mt Xia publishes information on a variety of topics such as Business Continuity, Disaster Recovery, High Availability, AIX, and Shell Programming.

Mt Xia Inc.
113 East Rich
Norman, OK 73069

Dana French, President
dfrench@mtxia.com
615.556.0456

[[...]]	Double Square Bracket Test
-a *file*	true if *string* is not null (obsolete)
-b *file*	true if *file* is a block device
-c *file*	true if *file* is a character device
-C *file*	true if *file* is a contiguous file
-d *file*	true if *file* is a directory
-e *file*	true if *file* exists
-f *file*	true if *file* is a regular file
-g *file*	true if *file* has SETGID bit set
-G *file*	true if *file's* group is effective GID
-h *file*	true if *file* is a symbolic link
-k *file*	true if *file* has sticky bit set
-L *file*	true if *file* is a symbolic link
-n *string*	true is *string* has non-zero length
-o *option*	true if *option* is on
-O *file*	true if *file's* owner is effective UID
-p *file*	true if *file* is a pipe (FIFO)
-r *file*	true if *file* is readable by current user
-s *file*	true if *file* has non-zero size
-S *file*	true if *file* is a socket
-t *filedes*	true if *filedes* is a terminal

[[...]]	Double Square Bracket Test
-u *file*	true if *file* has SETUID bit set
-w *file*	true if *file* is writable by current user
-x *file*	true if *file* is executable by current user
-z *string*	true if *string* has zero length

[[...]]	File and String Comparison
file1 -nt file2	true if file1 is newer than file2 or file 2 does not exist
file1 -ot file2	true if file1 is older than file2 or file 2 does not exist
file1 -ef file2	true if file1 and file2 are the same file
string == pattern	true if string matches pattern
string != pattern	true if string doesn't match pattern
string1 < string2	true if string1 is lexically less than string2
string1 > string2	true if string1 is lexically greater than string2

[[...]]	Numeric Comparison Tests (obsolete)
n1 -eq n2	true if *n1* is equal to *n2*
n1 -ne n2	true if *n1* is not equal to *n2*
n1 -lt n2	true if *n1* is less than *n2*
n1 -le n2	true if *n1* is less than or equal to *n2*
n1 -gt n2	true if *n1* is greater than *n2*
n1 -ge n2	true if *n1* is greater than or equal to *n2*

	Numeric Evaluation Commands
((expr))	true if *expr*ession evaluates to non-zero
$((expr))	true if *expr*ession evaluates to non-zero, and substitutes *expr*ession for evaluated value
let 'expr'	true if *expr*ession evaluates to non-zero, can redirect STDOUT and STDERR

((...))	Numeric Comparision tests
var = *expr*	evaluate *expr*ession and assign to *var*. true if *expr* evaluates to non-zero
n1 == n2	true if *n1* is equal to *n2*
n1 != n2	true if *n1* is not equal to *n2*
n1 < n2	true if *n1* is less than *n2*
n1 <= n2	true if *n1* is less than or equal to *n2*
n1 > n2	true if *n1* is greater than *n2*
n1 >= n2	true if *n1* is greater than or equal to *n2*

((...))	Numeric Evaluation Operators
var = expr	evaluate *expr*ession and assign result to var
+ -	addition, subtraction
* / %	multiplication, division, modulo
**	exponentiation
++ --	auto-increment, auto-decrement
&& ||	boolean 'and' boolean 'or'

name=value	Variable Substitution / Testing
${*name*}	substituted for value of *name*
${#name}	number of characters in value
${*name*:-*word*}	if *name* is unset or null, use *word*
${*name*:=*word*}	if *name* is unset or null, assign *word* to name and substitute *word*
${*name*:?*word*}	if *name* is unset or null, print *word* on STDERR and exit.
${*name*:+*word*}	if *name* is unset or null, use null, otherwise use *word*
${!name}	name of variable index
${!*prefix**]	all variables beginning with prefix.
${!*prefix*@}	all variables beginning with *prefix*.

name=value	Variable Substitution / Testing
${*name*#*pat*}	delete smallest matching *pat*tern from the beginning of value of name.
${*name*##*pat*}	delete the largest matching *pat*tern from the beginning of value of name.
${*name*%*pat*}	delete the smallest matching *pat*tern from the end of value of name.
${*name*%%*pat*}	delete the largest matching *pat*tern from the end of value of name.
${*name*:*start*}	substitute substring of value from position *start* beginning at zero.
${*name*:*start*:*length*}	substitute substring value from position *start* beginning at zero for *length* number of characters.
${*name*/*pat*/*string*}	substitute first occurrence of *pat*tern with *string*
${*name*//*pat*/*string*}	substitute all occurrences of *pat*tern with *string*
${*name*/#*pat*/*string*}	substitute occurrence of *pat*tern at beginning of value with *string*
${*name*/%*pat*/*string*}	substitute occurrence of *pat*tern at end of value with *string*

name[index]=value	Array Substitutions
${*name*[n]}	substitute array element n of array *name*
${*name*[*word*]}	substitute array element *word* of associative array *name*
“${name[*]}”	all array elements, all values within a single pair of double quotes
“${name[@]}”	all array elements, each value double quoted.
“${!*name*[*]}”	all indexes of array *name*, all values within single pair of double quotes

name[index]=value	Array Substitutions
“${!*name*[@]}”	all indexes of array *name*, each value double quoted.
${#name[*]}	number of array elements
${#name[@]}	number of array elements

[[:class:]]	Character Class		
[:alnum:]	alphanumeric	[:print:]	printable
[:alpha:]	alphabetic	[:punct:]	punctuation
[:blank:]	space or tab	[:space:]	whitespace
[:cntrl:]	control	[:upper:]	uppercase
[:digit:]	decimal	[:lower:]	lowercase
[:graph:]	non-spaces	[:xdigit:]	hexadecimal
[:word:] =	[[:alnum:]_]		
+(\d) =	[[:digit:]]	+(\D) =	[![:digit:]]
+(\s) =	[[:space:]]	+(\S) =	[![:space:]]
+(\w) =	[[:word:]]	+(\W) =	[![:word:]]

name[index]=value	Array Assignments
name[n]=”value”	assign a single array element *n* to a value
name=(...)	assign one or more values to an array called *name*
set -A name val1 ...	assign one or more values to an array called *name*
read -A name	read values into an array called *name*
typeset -A name	declare an associative array, must be defined before any values can be assigned.
name[word]=”value”	assign a single value to an associative array

name[index]=value	Array Assignments
	called *name* using an index of *word*
name=([word]=”value” ...)	assign one or more values to an associative array

	Pattern – filenames and strings
?	match one single character
*	match 0 or more characters
[...]	match any single character from the set of characters between the brackets
[!...]	match any single character not matching the set of characters between the brackets

	Pattern Operators
pat|pat|...	pattern list can be one or more patterns. separated by pipe symbol '|' means 'or'.
pat&pat&...	pattern list can be one or more patterns. separated by ampersand '&' means 'and'
?(pat-list)	match 0 or 1 occurrences of patterns
*(pat-list)	match 0 or more occurrences of patterns
+(pat-list)	match 1 or more occurrences of patterns
@(pat-list)	match exactly one occurrence of pattern
!(pat-list)	match anything but any of the patterns
\n	text matched by *n*th sub-pattern in (...)
{n}(pat-list)	match exactly n of any of the patterns
{n,m}(pat-list)	match *n* to *m* of any of the patterns
~(-i:pattern)	enable case sensitive option
~(+i:pattern)	enable ignore case option
~(-g:pattern)	enable shortest matching pattern option
~(+g:pattern)	enable longest matching pattern option