

**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](mailto:dfrench@mtxia.com)**  
**615.556.0456**

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

[[ ... ]]	Double Square Bracket Test
-u <i>file</i>	true if <i>file</i> has SETUID bit set
-w <i>file</i>	true if <i>file</i> is writable by current user
-x <i>file</i>	true if <i>file</i> is executable by current user
-z <i>string</i>	true if <i>string</i> 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 <i>n1</i> is equal to <i>n2</i>
n1 -ne n2	true if <i>n1</i> is not equal to <i>n2</i>
n1 -lt n2	true if <i>n1</i> is less than <i>n2</i>
n1 -le n2	true if <i>n1</i> is less than or equal to <i>n2</i>
n1 -gt n2	true if <i>n1</i> is greater than <i>n2</i>
n1 -ge n2	true if <i>n1</i> is greater than or equal to <i>n2</i>

	Numeric Evaluation Commands
(( expr ))	true if <i>expression</i> evaluates to non-zero
\$( ( expr ) )	true if <i>expression</i> evaluates to non-zero, and substitutes <i>expression</i> for evaluated value
let 'expr'	true if <i>expression</i> evaluates to non-zero, can redirect STDOUT and STDERR

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

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

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

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

name[index]=value	Array Substitutions
<code>\${name[n]}</code>	substitute array element <i>n</i> of array <i>name</i>
<code>\${name[word]}</code>	substitute array element <i>word</i> of associative array <i>name</i>
<code>`\${name[*]}`</code>	all array elements, all values within a single pair of double quotes
<code>`\${name[@]}`</code>	all array elements, each value double quoted.
<code>`\${!name[*]}`</code>	all indexes of array <i>name</i> , all values within single pair of double quotes

name[index]=value	Array Substitutions
<code>`\${!name[@]}`</code>	all indexes of array <i>name</i> , each value double quoted.
<code>`\${#name[*]}`</code>	number of array elements
<code>`\${#name[@]}`</code>	number of array elements

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

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

name[index]=value	Array Assignments
	called <i>name</i> using an index of <i>word</i>
<code>name=( [word]="value" ... )</code>	assign one or more values to an associative array

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

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