

Appendix B. Reference Cards

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

Table B-1. Special Shell Variables

Variable	Meaning
\$0	Name 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 <i>set</i>)
_	Last argument of previous command
\$_!	Process ID (PID) of last job <u>run</u> 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 <i>regular</i> file			
-d	File is a <i>directory</i>		-r	File has <i>read</i> permission
-h	File is a <i>symbolic link</i>		-w	File has <i>write</i> permission
-L	File is a <i>symbolic link</i>		-x	File has <i>execute</i> permission
-b	File is a <i>block device</i>			
-c	File is a <i>character device</i>		-g	<i>sgid</i> flag set
-p	File is a <i>pipe</i>		-u	<i>suid</i> flag set
-S	File is a <i>socket</i>		-k	"sticky bit" set
-t	File is associated with a <i>terminal</i>			
-N	File modified since it was last read		F1 -nt F2	File F1 is <i>newer</i> than F2 *
-O	You own the file		F1 -ot F2	File F1 is <i>older</i> than F2 *
-G	<i>Group id</i> of file same as yours		F1 -ef F2	Files F1 and F2 are <i>hard links</i> to the same file *
!	"NOT" (reverses sense of above tests)			

* *Binary* operator (requires two operands).

Table B-4. Parameter Substitution and Expansion

Expression	Meaning
\${var}	Value of <i>var</i> , same as <i>\$var</i>
\${var-DEFAULT}	If <i>var</i> not set, evaluate expression as <i>\$DEFAULT</i> *
\${var:-DEFAULT}	If <i>var</i> not set or is empty, evaluate expression as <i>\$DEFAULT</i> *
\${var=DEFAULT}	If <i>var</i> not set, evaluate expression as <i>\$DEFAULT</i> *
\${var:=DEFAULT}	If <i>var</i> not set, evaluate expression as <i>\$DEFAULT</i> *
\${var+OTHER}	If <i>var</i> set, evaluate expression as <i>\$OTHER</i> , otherwise as <u>null</u> string
\${var:+OTHER}	If <i>var</i> set, evaluate expression as <i>\$OTHER</i> , otherwise as null string
\${var?ERR_MSG}	If <i>var</i> not set, print <i>\$ERR_MSG</i> *
\${var:?ERR_MSG}	If <i>var</i> not set, print <i>\$ERR_MSG</i> *
\${!varprefix*}	Matches all previously declared variables beginning with <i>varprefix</i>
\${!varprefix@}	Matches all previously declared variables beginning with <i>varprefix</i>

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

Table B-5. String Operations

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

* Where *\$substring* is a *regular expression*.

Table B-6. Miscellaneous Constructs

Expression	Interpretation
<i>Brackets</i>	
<code>if [CONDITION]</code>	Test construct
<code>if [[CONDITION]]</code>	Extended test construct
<code>Array[1]=element1</code>	Array initialization
<code>[a-z]</code>	Range of characters within a Regular Expression
<i>Curly Brackets</i>	

<code>\${variable}</code>	Parameter substitution
<code>\${!variable}</code>	Indirect variable reference
<code>{ command1; command2 }</code>	Block of code
<code>{string1,string2,string3,...}</code>	Brace expansion
<i>Parentheses</i>	
<code>(command1; command2)</code>	Command group executed within a subshell
<code>Array=(element1 element2 element3)</code>	Array initialization
<code>result=\$(COMMAND)</code>	Execute command in subshell and assign result to variable
<code>>(COMMAND)</code>	Process substitution
<code><(COMMAND)</code>	Process substitution
<i>Double Parentheses</i>	
<code>((var = 78))</code>	Integer arithmetic
<code>var=\$((20 + 5))</code>	Integer arithmetic, with variable assignment
<i>Quoting</i>	
<code>"\$variable"</code>	"Weak" quoting
<code>'string'</code>	"Strong" quoting
<i>Back Quotes</i>	
<code>result=`COMMAND`</code>	Execute command in subshell and assign result to variable