Appendix E. Exit Codes With Special Meanings

Table E-1. *Reserved* Exit Codes

Exit Code Number	Meaning	Example	Comments
1	Catchall for general errors	let "var1 = 1/0"	Miscellaneous errors, such as "divide by zero" and other impermissible operations
2	Misuse of shell builtins (according to Bash documentation)	empty_function() {}	Missing keyword or command, or permission problem (and *diff* return code on a failed binary file comparison).
126	Command invoked cannot execute	/dev/null	Permission problem or command is not an executable
127	"command not found"	illegal_command	Possible problem with $PATH or a typo
128	Invalid argument to exit	exit 3.14159	**exit** takes only integer args in the range 0 - 255 (see first footnote)
128+n	Fatal error signal "n"	*kill -9* $PPID of script	**$?** returns 137 (128 + 9)
130	Script terminated by Control-C	*Ctl-C*	Control-C is fatal error signal 2, (130 = 128 + 2, see above)
255*	Exit status out of range	exit -1	**exit** takes only integer args in the range 0 - 255

According to the above table, exit codes 1 - 2, 126 - 165, and 255 [1] have special meanings, and should therefore be avoided for user-specified exit parameters. Ending a script with *exit 127* would certainly cause confusion when troubleshooting (is the error code a "command not found" or a user-defined one?). However, many scripts use an *exit 1* as a general bailout-upon-error. Since exit code 1 signifies so many possible errors, it is not particularly useful in debugging.

There has been an attempt to systematize exit status numbers (see /usr/include/sysexits.h), but this is intended for C and C++ programmers. A similar standard for scripting might be appropriate. The author of this document proposes restricting user-defined exit codes to the range 64 - 113 (in addition to 0, for success), to conform with the C/C++ standard. This would allot 50 valid codes, and make troubleshooting scripts more straightforward. [2] All user-defined exit codes in the accompanying examples to this document conform to this standard, except where overriding circumstances exist, as in Example 9-2.

Issuing a $? from the command-line after a shell script exits gives results consistent with the table above only from the Bash or *sh* prompt. Running the *C-shell* or *tcsh* may give different values in some cases.

Notes

[1] Out of range exit values can result in unexpected exit codes. An exit value greater than 255 returns an exit code modulo 256. For example, *exit 3809* gives an exit code of 225 (3809 % 256 = 225).

[2] An update of /usr/include/sysexits.h allocates previously unused exit codes from 64 - 78. It may be anticipated that the range of unallotted exit codes will be further restricted in the future. The author of this document will *not* do fixups on the scripting examples to conform to the changing standard. This should not cause any problems, since there is no overlap or conflict in usage of exit codes between compiled C/C++ binaries and shell scripts.

Prev
Parsing and Managing Pathnames

Home

Next
A Detailed Introduction to I/O and I/O Redirection