-- --

IEEE P1003.2 Draft 11.2 – September 1991

4.9 `cksum` — Write file checksums and sizes 2

4.9.1 Synopsis

`cksum` [*file* ...]

4.9.2 Description

The `cksum` utility shall calculate and write to standard output a cyclic redun- 2
dancy check (CRC) for each input file, and also write to standard output the 2
number of octets in each file. The CRC used is based on the polynomial used for 2
CRC error checking in the networking standard ISO 8802-3 {B7}.

The CRC checksum shall be obtained in the following way:

The encoding is defined by the generating polynomial:

$$G(x) = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^{8} + x^{7} + x^{5} + x^{4} + x^{2} + x + 1$$

Mathematically, the CRC value corresponding to a given file shall be defined by the following procedure:

(1) The n bits to be evaluated are considered to be the coefficients of a mod 2 2
polynomial $M(x)$ of degree $n-1$. These n bits are the bits from the file, 2
with the most significant bit being the most significant bit of the first 2
octet of the file and the last bit being the least significant bit of the last 2
octet, padded with zero bits (if necessary) to achieve an integral number 2
of octets, followed by one or more octets representing the length of the file 2
as a binary value, least significant octet first. The smallest number of 2
octets capable of representing this integer shall be used. 2

(2) $M(x)$ is multiplied by x^{32} (i.e., shifted left 32 bits) and divided by $G(x)$
using mod 2 division, producing a remainder $R(x)$ of degree ≤ 31. 2

(3) The coefficients of $R(x)$ are considered to be a 32-bit sequence.

(4) The bit sequence is complemented and the result is the CRC. 2

4.9.3 Options

None.

4.9.4 Operands

The following operand shall be supported by the implementation:

file A pathname of a file to be checked. If no *file* operands are specified, the standard input is used.

4.9.5 External Influences

4.9.5.1 Standard Input

The standard input is used only if no *file* operands are specified. See Input Files.

4.9.5.2 Input Files

The input files can be any file type.

4.9.5.3 Environment Variables

The following environment variables shall affect the execution of `cksum`:

LANG This variable shall determine the locale to use for the locale categories when both **LC_ALL** and the corresponding environment variable (beginning with **LC_**) do not specify a locale. See 2.6.

LC_ALL This variable shall determine the locale to be used to override any values for locale categories specified by the settings of **LANG** or any environment variables beginning with **LC_**.

LC_CTYPE This variable shall determine the locale for the interpretation of sequences of bytes of text data as characters (e.g., single- versus multibyte characters in arguments).

LC_MESSAGES This variable shall determine the language in which messages should be written.

4.9.5.4 Asynchronous Events

Default.

4.9.6 External Effects

4.9.6.1 Standard Output

For each file processed successfully, the `cksum` utility shall write in the following format: 2

`"%u %d %s\n"`, <*checksum*> , <*# of octets*>, <*pathname*> 2

If no *file* operand was specified, the pathname and its leading space shall be omitted.

4.9.6.2 Standard Error

Used only for diagnostic messages.

4.9.6.3 Output Files

None.

4.9.7 Extended Description

None.

4.9.8 Exit Status

The `cksum` utility shall exit with one of the following values:

- 0 All files were processed successfully.
- >0 An error occurred.

4.9.9 Consequences of Errors

Default.

4.9.10 Rationale. *(This subclause is not a part of P1003.2)*

Examples, Usage

The `cksum` utility is typically used to quickly compare a suspect file against a trusted version of the same. However, no claims are made by POSIX.2 that this comparison is cryptographically secure; the historical `sum` utility from which `cksum` was inspired has traditionally been used mainly to ensure that files transmitted over noisy media arrive intact. The chances of a damaged file producing the same CRC as the original are astronomically small; deliberate deception is difficult, but probably not impossible.

Although input files to `cksum` can be any type, the results need not be what would be expected on character special device files or on file types not described by POSIX.1 {8}. Since POSIX.2 does not specify the block size used when doing input, checksums of character special files need not process all of the data in those files.

The algorithm is expressed in terms of a bitstream divided into octets. If a file is transmitted between two systems and undergoes any data transformation (such as moving 8-bit characters into 9-bit bytes or changing ‘‘little Endian’’ byte ordering to ‘‘big Endian’’), identical CRC values cannot be expected. Implementations performing such transformations may extend `cksum` to handle such situations.

The following C-language program can be used as a model to describe the algorithm. It assumes that a `char` is one octet. It also assumes that the entire file is

available for one pass through the function. This was done for simplicity in 2
demonstrating the algorithm, rather than as an implementation model. 2

```
static unsigned long crctab[] = {                                        /* 2 */
0x0,                                                                     /* 2 */
0x77073096, 0xee0e612c, 0x990951ba, 0x076dc419, 0x706af48f,
0xe963a535, 0x9e6495a3, 0x0edb8832, 0x79dcb8a4, 0xe0d5e91e,
0x97d2d988, 0x09b64c2b, 0x7eb17cbd, 0xe7b82d07, 0x90bf1d91,
0x1db71064, 0x6ab020f2, 0xf3b97148, 0x84be41de, 0x1adad47d,
0x6ddde4eb, 0xf4d4b551, 0x83d385c7, 0x136c9856, 0x646ba8c0,
0xfd62f97a, 0x8a65c9ec, 0x14015c4f, 0x63066cd9, 0xfa0f3d63,
0x8d080df5, 0x3b6e20c8, 0x4c69105e, 0xd56041e4, 0xa2677172,
0x3c03e4d1, 0x4b04d447, 0xd20d85fd, 0xa50ab56b, 0x35b5a8fa,
0x42b2986c, 0xdbbbc9d6, 0xacbcf940, 0x32d86ce3, 0x45df5c75,
0xdcd60dcf, 0xabd13d59, 0x26d930ac, 0x51de003a, 0xc8d75180,
0xbfd06116, 0x21b4f4b5, 0x56b3c423, 0xcfba9599, 0xb8bda50f,
0x2802b89e, 0x5f058808, 0xc60cd9b2, 0xb10be924, 0x2f6f7c87,
0x58684c11, 0xc1611dab, 0xb6662d3d, 0x76dc4190, 0x01db7106,
0x98d220bc, 0xefd5102a, 0x71b18589, 0x06b6b51f, 0x9fbfe4a5,
0xe8b8d433, 0x7807c9a2, 0x0f00f934, 0x9609a88e, 0xe10e9818,
0x7f6a0dbb, 0x086d3d2d, 0x91646c97, 0xe6635c01, 0x6b6b51f4,
0x1c6c6162, 0x856530d8, 0xf262004e, 0x6c0695ed, 0x1b01a57b,
0x8208f4c1, 0xf50fc457, 0x65b0d9c6, 0x12b7e950, 0x8bbeb8ea,
0xfcb9887c, 0x62dd1ddf, 0x15da2d49, 0x8cd37cf3, 0xfbd44c65,
0x4db26158, 0x3ab551ce, 0xa3bc0074, 0xd4bb30e2, 0x4adfa541,
0x3dd895d7, 0xa4d1c46d, 0xd3d6f4fb, 0x4369e96a, 0x346ed9fc,
0xad678846, 0xda60b8d0, 0x44042d73, 0x33031de5, 0xaa0a4c5f,
0xdd0d7cc9, 0x5005713c, 0x270241aa, 0xbe0b1010, 0xc90c2086,
0x5768b525, 0x206f85b3, 0xb966d409, 0xce61e49f, 0x5edef90e,
0x29d9c998, 0xb0d09822, 0xc7d7a8b4, 0x59b33d17, 0x2eb40d81,
0xb7bd5c3b, 0xc0ba6cad, 0xedb88320, 0x9abfb3b6, 0x03b6e20c,
0x74b1d29a, 0xead54739, 0x9dd277af, 0x04db2615, 0x73dc1683,
0xe3630b12, 0x94643b84, 0x0d6d6a3e, 0x7a6a5aa8, 0xe40ecf0b,
0x9309ff9d, 0x0a00ae27, 0x7d079eb1, 0xf00f9344, 0x8708a3d2,
0x1e01f268, 0x6906c2fe, 0xf762575d, 0x806567cb, 0x196c3671,
0x6e6b06e7, 0xfed41b76, 0x89d32be0, 0x10da7a5a, 0x67dd4acc,
0xf9b9df6f, 0x8ebeeff9, 0x17b7be43, 0x60b08ed5, 0xd6d6a3e8,
0xa1d1937e, 0x38d8c2c4, 0x4fdff252, 0xd1bb67f1, 0xa6bc5767,
0x3fb506dd, 0x48b2364b, 0xd80d2bda, 0xaf0a1b4c, 0x36034af6,
0x41047a60, 0xdf60efc3, 0xa867df55, 0x316e8eef, 0x4669be79,
0xcb61b38c, 0xbc66831a, 0x256fd2a0, 0x5268e236, 0xcc0c7795,
0xbb0b4703, 0x220216b9, 0x5505262f, 0xc5ba3bbe, 0xb2bd0b28,
0x2bb45a92, 0x5cb36a04, 0xc2d7ffa7, 0xb5d0cf31, 0x2cd99e8b,
0x5bdeae1d, 0x9b64c2b0, 0xec63f226, 0x756aa39c, 0x026d930a,
0x9c0906a9, 0xeb0e363f, 0x72076785, 0x05005713, 0x95bf4a82,
0xe2b87a14, 0x7bb12bae, 0x0cb61b38, 0x92d28e9b, 0xe5d5be0d,
0x7cdcefb7, 0x0bdbdf21, 0x86d3d2d4, 0xf1d4e242, 0x68ddb3f8,
0x1fda836e, 0x81be16cd, 0xf6b9265b, 0x6fb077e1, 0x18b74777,
0x88085ae6, 0xff0f6a70, 0x66063bca, 0x11010b5c, 0x8f659eff,
0xf862ae69, 0x616bffd3, 0x166ccf45, 0xa00ae278, 0xd70dd2ee,
0x4e048354, 0x3903b3c2, 0xa7672661, 0xd06016f7, 0x4969474d,
0x3e6e77db, 0xaed16a4a, 0xd9d65adc, 0x40df0b66, 0x37d83bf0,
```

```
0xa9bcae53, 0xdebb9ec5, 0x47b2cf7f, 0x30b5ffe9, 0xbdbdf21c,
0xcabac28a, 0x53b39330, 0x24b4a3a6, 0xbad03605, 0xcdd70693,
0x54de5729, 0x23d967bf, 0xb3667a2e, 0xc4614ab8, 0x5d681b02,
0x2a6f2b94, 0xb40bbe37, 0xc30c8ea1, 0x5a05df1b, 0x2d02ef8d
};

unsigned long memcrc(const unsigned char *b, size_t n)
{
/*      Input arguments:
 *      const char*     b == byte sequence to checksum
 *      size_t          n == length of sequence
 */

        register unsigned int   i, c, s = 0;

        for (i = n; i > 0; --i) {
                c = (unsigned int)(*b++);
                s = (s << 8) ^ crctab[(s >> 24) ^ c];
        }

        /* extend with the length of the string */
        while (n != 0) {
                c = n & 0377;
                n >>= 8;
                s = (s << 8) ^ crctab[(s >> 24) ^ c];
        }

        return ~s;
}
```

History of Decisions Made

The historical practice of writing the number of ‘‘blocks’’ has been removed in favor of writing the number of octets since the latter is not only more useful, but historical implementations have not been consistent in defining what a ‘‘block’’ meant. Octets are used instead of bytes because bytes can differ in size between systems.

The algorithm used was selected to increase the robustness of the utility’s operation. Neither the System V nor BSD `sum` algorithm was selected. Since each of these was different and each was the default behavior on those systems, no realistic compromise was available if either were selected—some set of historical applications would break. Therefore, the name was changed to `cksum`. Although the historical `sum` commands will probably continue to be provided for many years to come, programs designed for portability across systems should use the new name.

The algorithm selected is based on that used by the Ethernet standard for the Frame Check Sequence Field. The algorithm used does not match the technical definition of a *checksum*; the term is used for historical reasons. The length of the file is included in the CRC calculation because this parallels Ethernet’s inclusion of a length field in its CRC, but also because it guards against inadvertent collisions between files that begin with different series of zero octets. The chance that two different files will produce identical CRCs is much greater when their lengths are not considered. Keeping the length and the checksum of the file itself separate would yield a slightly more robust algorithm, but historical usage has always been that a single number (the checksum as printed) represents the signature of the file. It was decided that historical usage was the more important consideration.

Earlier drafts contained modifications to the Ethernet algorithm that involved extracting table values whenever an intermediate result became zero. This was demonstrated to be less robust than the current method and mathematically difficult to describe or justify.

Editor's Note: The following bibliographic references will be cleaned up before the standard is completed.

The calculation used is identical to that given in pseudo-code on page 1011 of *Communications of the ACM*, August, 1988 in the article ‘‘Computation of Cyclic Redundancy Checks Via Table Lookup’’ by Dilip V. Sarwate. The pseudo-code rendition is:

```
X <- 0;  Y <- 0;
for i <- m -1 step -1 until 0 do
        begin
        T <- X(1) ^ A[i];
        X(1) <- X(0); X(0) <- Y(1); Y(1) <- Y(0); Y(0) <- 0;
        comment: f[T] and f'[T] denote the T-th words in the
                table f and f' ;
        X <- X ^ f[T]; Y <- Y ^ f'[T];
        end
```

The pseudo-code is reproduced exactly as given; however, note that in `cksum`'s case, `A[i]` represents a byte of the file, the words `X` and `Y` are a treated as a single 32-bit value, and the tables `f` and `f'` are a single table containing 32-bit values.

The article also discusses generating the table(s).

Other sources consulted about CRC's:

> ‘‘A Tutorial on CRC Computations,’’ Ramabadran and Gaitonde, *IEEE Micro,* p. 62, August 1988;

> *Computer Networks*, Andrew Tanenbaum, Prentice-Hall, Inc.

4.10 `cmp` — Compare two files

4.10.1 Synopsis

`cmp [ −l | −s ]` *file1 file2*

4.10.2 Description

The `cmp` utility shall compare two files. The `cmp` utility shall write no output if the files are the same. Under default options, if they differ, it shall write to standard output the byte and line number at which the first difference occurred. Bytes and lines shall be numbered beginning with 1.

4.10.3 Options

The `cmp` utility shall conform to the utility argument syntax guidelines described in 2.10.2.

The following options shall be supported by the implementation:

`−l` (Lowercase ell.) Write the byte number (decimal) and the differing bytes (octal) for each difference.

`−s` Write nothing for differing files; return exit status only.

4.10.4 Operands

The following operands shall be supported by the implementation:

file1 A pathname of the first file to be compared. If *file1* is −, the standard input shall be used.

file2 A pathname of the second file to be compared. If *file2* is −, the standard input shall be used.

If both *file1* and *file2* refer to standard input or refer to the same FIFO special, block special, or character special file, the results are undefined.

4.10.5 External Influences

4.10.5.1 Standard Input

The standard input shall be used only if the *file1* or *file2* operand refers to standard input. See Input Files.