IEEE P1003.2 Draft 11.2 – September 1991

Copyright © 1991 by the
Institute of Electrical and Electronics Engineers, Inc.
345 East 47th Street
New York, NY 10017, USA
All rights reserved as an unpublished work.

This is an unapproved and unpublished IEEE Standards Draft, subject to change. The publication, distribution, or copying of this draft, as well as all derivative works based on this draft, is expressly prohibited except as set forth below.

Permission is hereby granted for IEEE Standards Committee participants to reproduce this document for purposes of IEEE standardization activities only, and subject to the restrictions contained herein.

Permission is hereby also granted for member bodies and technical committees of ISO and IEC to reproduce this document for purposes of developing a national position, subject to the restrictions contained herein.

Permission is hereby also granted to the preceding entities to make limited copies of this document in an electronic form only for the stated activities.

The following restrictions apply to reproducing or transmitting the document in any form: 1) all copies or portions thereof must identify the document's IEEE project number and draft number, and must be accompanied by this entire notice in a prominent location; 2) no portion of this document may be redistributed in any modified or abridged form without the prior approval of the IEEE Standards Department.

Other entities seeking permission to reproduce this document, or any portion thereof, for standardization or other activities, must contact the IEEE Standards Department for the appropriate license.

Use of information contained in this unapproved draft is at your own risk.

IEEE Standards Department
Copyright and Permissions
445 Hoes Lane, P.O. Box 1331
Piscataway, NJ 08855-1331, USA
+1 (908) 562-3800
+1 (908) 562-1571 [FAX]

4.9 cksum — Write file checksums and sizes

4.9.1 Synopsis

```
cksum [file ...]
```

4.9.2 Description

The cksum utility shall calculate and write to standard output a cyclic redundancy check (CRC) for each input file, and also write to standard output the number of octets in each file. The CRC used is based on the polynomial used for 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 polynomial $M(x)$ of degree $n-1$. These n bits are the bits from the file, with the most significant bit being the most significant bit of the first octet of the file and the last bit being the least significant bit of the last octet, padded with zero bits (if necessary) to achieve an integral number of octets, followed by one or more octets representing the length of the file as a binary value, least significant octet first. The smallest number of octets capable of representing this integer shall be used.

(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.

(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.

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.

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

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.

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

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

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

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

```
static unsigned long crctab[] = {
0x0,
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,
```

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

```
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

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

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.

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

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.

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

4.10.5.2 Input Files

The input files can be any file type.

4.10.5.3 Environment Variables

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

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.

4.10.5.4 Asynchronous Events

Default.

4.10.6 External Effects

4.10.6.1 Standard Output

In the POSIX Locale, results of the comparison shall be written to standard output. When no options are used, the format shall be:

`"%s %s differ: char %d, line %d\n"`, *file1* , *file2* , *<byte number>* , *<line number>*

When the `-l` option is used, the format is:

`"%d %o %o\n"`, *<byte number>* , *<differing byte>*, *<differing byte>*

for each byte that differs. The first *<differing byte>* number is from *file1* while the second is from *file2* . In both cases, *<byte number>* shall be relative to the beginning of the file, beginning with 1.

The *<additional info>* field shall either be null or a string that starts with a `<blank>` and contains no `<newline>` characters.

No output shall be written to standard output when the `-s` option is used.

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

4.10.6.2 Standard Error

Used only for diagnostic messages. If *file1* and *file2* are identical for the entire length of the shorter file, in the POSIX Locale the following diagnostic message shall be written, unless the `-s` option is specified.

"`cmp: EOF on %s%s\n`", <*name of shorter file*>, <*additional info*>

4.10.6.3 Output Files

None.

4.10.7 Extended Description

None.

4.10.8 Exit Status

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

0 The files are identical.

1 The files are different; this includes the case where one file is identical to the first part of the other.

>1 An error occurred.

4.10.9 Consequences of Errors

Default.

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

Examples, Usage

The global language in Section 2 indicates that using two mutually-exclusive options together produces unspecified results. Some System V implementations consider the option usage:

```
cmp -l -s ...
```

to be an error. They also treat:

```
cmp -s -l ...
```

as if no options were specified. Both of these behaviors are considered bugs, but are allowed.

Although input files to `cmp` can be any type, the results might 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,

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

comparisons of character special files need not compare all of the data in those files.

The word `char` in the standard output format comes from historical usage, even though it is actually a byte number. When `cmp` is supported in other locales, implementations are encouraged to use the word `byte` or its equivalent in another language. Users should not interpret this difference to indicate that the functionality of the utility changed between locales.

History of Decisions Made

Some systems report on the number of lines in the identical-but-shorter file case. This is allowed by the inclusion of the *<additional info>* fields in the output format. The restriction on having a leading `<blank>` and no `<newline>`s is to make parsing for the file name easier. It is recognized that some file names containing white-space characters will make parsing difficult anyway, but the restriction does aid programs used on systems where the names are predominantly well behaved.

4.11 `comm` — Select or reject lines common to two files

4.11.1 Synopsis

`comm [-123]` *file1 file2*

4.11.2 Description

The `comm` utility shall read *file1* and *file2*, which should be ordered in the current collating sequence, and produce three text columns as output: lines only in *file1*; lines only in *file2*; and lines in both files.

If the lines in both files are not ordered according to the collating sequence of the current locale, the results are unspecified.

4.11.3 Options

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

The following options shall be supported by the implementation:

`-1` Suppress the output column of lines unique to *file1*.

`-2` Suppress the output column of lines unique to *file2*.

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

3261 −3 Suppress the output column of lines duplicated in *file1* and *file2*. 1

3262 ### 4.11.4 Operands

3263 The following operands shall be supported by the implementation:

3264 *file1* A pathname of the first file to be compared. If *file1* is −, the stan-
3265 dard input is used.

3266 *file2* A pathname of the second file to be compared. If *file2* is −, the
3267 standard input is used.

3268 If both *file1* and *file2* refer to standard input or to the same FIFO special, block
3269 special, or character special file, the results are undefined.

3270 ### 4.11.5 External Influences

3271 #### 4.11.5.1 Standard Input

3272 The standard input shall be used only if one of the *file1* or *file2* operands refers to
3273 standard input. See Input Files.

3274 #### 4.11.5.2 Input Files

3275 The input files shall be text files.

3276 #### 4.11.5.3 Environment Variables

3277 The following environment variables shall affect the execution of `comm`:

3278 **LANG** This variable shall determine the locale to use for the
3279 locale categories when both **LC_ALL** and the correspond-
3280 ing environment variable (beginning with **LC_**) do not
3281 specify a locale. See 2.6.

3282 **LC_ALL** This variable shall determine the locale to be used to over-
3283 ride any values for locale categories specified by the set-
3284 tings of **LANG** or any environment variables beginning
3285 with **LC_**.

3286 **LC_CTYPE** This variable shall determine the locale for the interpreta-
3287 tion of sequences of bytes of text data as characters (e.g.,
3288 single- versus multibyte characters in arguments and
3289 input files).

3290 **LC_COLLATE** This variable shall determine the locale for the collating
3291 sequence `comm` expects to have been used when the input
3292 files were sorted.

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

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

4.11.5.4 Asynchronous Events

Default.

4.11.6 External Effects

4.11.6.1 Standard Output

The `comm` utility shall produce output depending on the options selected. If the `-1`, `-2`, and `-3` options are all selected, `comm` shall write nothing to standard output.

If the `-1` option is not selected, lines contained only in *file1* shall be written using the format:

`"%s\n"`, *<line in file1>*

If the `-2` option is not selected, lines contained only in *file2* shall be written using the format:

`"%s%s\n"`, *<lead>*, *<line in file2>*

where the string *<lead>* is:

`<tab>` if the `-1` option is not selected, or

null string if the `-1` option is selected.

If the `-3` option is not selected, lines contained in both files shall be written using the format:

`"%s%s\n"`, *<lead>*, *<line in both>*

where the string *<lead>* is:

`<tab><tab>` if neither the `-1` nor the `-2` option is selected, or

`<tab>` if exactly one of the `-1` and `-2` options is selected, or

null string if both the `-1` and `-2` options are selected.

If the input files were ordered according to the collating sequence of the current locale, the lines written shall be in the collating sequence of the original lines.

4.11.6.2 Standard Error

Used only for diagnostic messages.

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

3322 #### 4.11.6.3 Output Files

3323 None.

3324 ### 4.11.7 Extended Description

3325 None.

3326 ### 4.11.8 Exit Status

3327 The `comm` utility shall exit with one of the following values:

3328 0 All input files were successfully output as specified.

3329 >0 An error occurred.

3330 ### 4.11.9 Consequences of Errors

3331 Default.

3332 ### 4.11.10 Rationale. *(This subclause is not a part of P1003.2)*

3333 **Examples, Usage**

3334 If the input files are not properly presorted, the output of `comm` might not be use-
3335 ful.

3336 If a file named `posix.2` contains a sorted list of the utilities in this standard, a
3337 file named `xpg3` contains a sorted list of the utilities specified in X/Open Portabil-
3338 ity Guide Issue 3, and a file named `svid89` contains a sorted list of the utilities in
3339 the System V Interface Definition Third Edition:

3340 ```
comm -23 posix.2 xpg3 | comm -23 - svid89
```

3341 would print a list of utilities in this standard not specified by either of the other
3342 documents,

3343 ```
comm -12 posix.2 xpg3 | comm -12 - svid89
```

3344 would print a list of utilities specified by all three documents, and

3345 ```
comm -12 xpg3 svid89 | comm -23 - posix.2
```

3346 would print a list of utilities specified by both XPG3 and *SVID*, but not specified in
3347 this standard.

3348 **History of Decisions Made**

3349 None.

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.
```

3350 ## 4.12 `command` — Execute a simple command

3351 ### 4.12.1 Synopsis

3352 `command [-p]` *command_name* `[`*argument* `...]`

3353 ### 4.12.2 Description

3354 The `command` utility shall cause the shell to treat the arguments as a simple com-
3355 mand, suppressing the shell function lookup that is described in 3.9.1.1 item 1
3356 (1)(b). 1

3357 If the *command_name* is the same as the name of one of the special built-in utili-
3358 ties, the special properties in the enumerated list at the beginning of 3.14 shall
3359 not occur. In every other respect, if *command_name* is not the name of a function,
3360 the effect of `command` shall be the same as omitting `command`.

3361 ### 4.12.3 Options

3362 The `command` utility shall conform to the utility argument syntax guidelines
3363 described in 2.10.2.

3364 The following option shall be supported by the implementation:

3365 `-p` Perform the command search using a default value for **PATH** that
3366 is guaranteed to find all of the standard utilities.

3367 ### 4.12.4 Operands

3368 The following operands shall be supported by the implementation:

3369 *argument* One of the strings treated as an argument to *command_name* .

3370 *command_name*
3371 The name of a utility or a special built-in utility.

3372 ### 4.12.5 External Influences

3373 #### 4.12.5.1 Standard Input

3374 None.

3375 #### 4.12.5.2 Input Files

3376 None.

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

4.12.5.3 Environment Variables

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

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).

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

PATH This variable shall determine the search path used during the command search described in 3.9.1.1, except as described under the `-p` option.

4.12.5.4 Asynchronous Events

Default.

4.12.6 External Effects

4.12.6.1 Standard Output

None.

4.12.6.2 Standard Error

Used only for diagnostic messages.

4.12.6.3 Output Files

None.

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

4.12.7 Extended Description

None.

4.12.8 Exit Status

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

126 The utility specified by *command_name* was found but could not be invoked.

127 An error occurred in the `command` utility or the utility specified by *command_name* could not be found.

Otherwise, the exit status of `command` shall be that of the simple command specified by the arguments to `command`.

4.12.9 Consequences of Errors

Default.

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

Examples, Usage

The order for command search in POSIX. 2 allows functions to override regular built-ins and path searches. This utility is necessary to allow functions that have the same name as a utility to call the utility (instead of a recursive call to the function).

The system default path is available using `getconf`; however, since `getconf` may need to have the **PATH** set up before it can be called itself, the following can be used:

```
command -p getconf _CS_PATH
```

Since `command` appears in Table 2-2, it will always be found prior to the **PATH** search.

There is nothing in the description of `command` that implies the command line is parsed any differently than for any other simple command. For example,

```
command a | b ; c
```

is not parsed in any special way that causes `|` or `;` to be treated other than a pipe operator or semicolon or that prevents function lookup on `b` or `c`.

Examples : Make a version of `cd` that always prints out the new working directory exactly once:

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

```
cd() {
        command cd "$@" >/dev/null
        pwd
}
```

Start off a ‘‘secure shell script’’ in which the script avoids being spoofed by its parent:

```
IFS='
'
#       The preceding value should be <space><tab><newline>.
#       Set IFS to its default value.
\unset -f command
#       Ensure command is not a user function.
#       Note that unset is escaped to prevent an alias being used
#       for unset on implementations that support aliases.
PATH="$(\command -p getconf _CS_PATH):$PATH"
#       Put on a reliable PATH prefix.
#       Now, unset all utility names that will be used (or
#       invoke them with \command each time).
#       ...
```

At this point, given correct permissions on the directories called by **PATH**, the script has the ability to ensure that any utility it calls is the intended one. It is being very cautious because it assumes that implementation extensions may be present that would allow user aliases and/or functions to exist when it is invoked; neither capability is specified by POSIX.2, but neither is prohibited as an extension. For example, the proposed UPE supplement to POSIX.2 introduces a **ENV** variable that precedes the invocation of the script with a user startup script. Such a script could have used the aliasing facility from the UPE or the functions in POSIX.2 to spoof the application.

The `command`, `env`, `nohup`, and `xargs` utilities have been specified to use exit code 127 if an error occurs so that applications can distinguish ‘‘failure to find a utility’’ from ‘‘invoked utility exited with an error indication.’’ The value 127 was chosen because it is not commonly used for other meanings; most utilities use small values for ‘‘normal error conditions’’ and the values above 128 can be confused with termination due to receipt of a signal. The value 126 was chosen in a similar manner to indicate that the utility could be found, but not invoked. Some scripts produce meaningful error messages differentiating the 126 and 127 cases. The distinction between exit codes 126 and 127 is based on KornShell practice that uses 127 when all attempts to *exec* the utility fail with [ENOENT], and uses 126 when any attempt to *exec* the utility fails for any other reason.

History of Decisions Made

The `command` utility is somewhat similar to the Eighth Edition `builtin` command, but since `command` also goes to the file system to search for utilities, the name `builtin` would not be intuitive.

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

The `command` utility will most likely be provided as a regular built-in. In an earlier draft, it was a special built-in. This was changed for the following reasons:

— The removal of exportable functions made the special precedence of a special built-in unnecessary.

— A special built-in has special properties (see the enumerated list at the beginning of 3.14) that were inappropriate for invoking other utilities. For example, two commands such as

```
date > unwritable-file
command date > unwritable-file
```

would have entirely different results; in a noninteractive script, the former would continue to execute the next command, the latter would abort. Introducing this semantic difference along with suppressing functions was seen to be nonintuitive.

— There are some advantages of suppressing the special characteristics of special built-ins on occasion. For example:

```
command exec > unwritable-file
```

will not cause a noninteractive script to abort, so that the output status can be checked by the script.

An earlier draft presented a larger number of options. Most were removed because they were not useful to real portable applications, given the new command search order.

The `-p` option is present because it is useful to be able to ensure a safe path search that will find all the POSIX.2 standard utilities. This search might not be identical to the one that occurs through one of the POSIX.1 {8} *exec* functions when **PATH** is unset, as explained in 2.6.1. At the very least, this feature is required to allow the script to access the correct version of `getconf` so that the value of the default path can be accurately retrieved.

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

4.13 cp — Copy files

4.13.1 Synopsis

```
cp [-fip] source_file target_file
cp [-fip] source_file ... target
cp -R [-fip] source_file ... target
cp -r [-fip] source_file ... target
```

4.13.2 Description

The first synopsis form is denoted by two operands, neither of which are existing files of type directory. The `cp` utility shall copy the contents of *source_file* to the destination path named by *target_file.*

The second synopsis form is denoted by two or more operands where the `-R` or `-r` options are not specified and the first synopsis form is not applicable. It shall be an error if any *source_file* is a file of type directory, if *target* does not exist, or if *target* is a file of a type defined by POSIX.1 {8}, but is not a file of type directory. The `cp` utility shall copy the contents of each *source_file* to the destination path named by the concatenation of *target,* a slash character, and the last component of *source_file.*

The third and fourth synopsis forms are denoted by two or more operands where the `-R` or `-r` options are specified. The `cp` utility shall copy each file in the file hierarchy rooted in each *source_file* to a destination path named as follows.

If *target* exists and is a file of type directory, the name of the corresponding destination path for each file in the file hierarchy shall be the concatenation of *target,* a slash character, and the pathname of the file relative to the directory containing *source_file.*

If *target* does not exist, and two operands are specified, the name of the corresponding destination path for *source_file* shall be *target;* the name of the corresponding destination path for all other files in the file hierarchy shall be the concatenation of *target,* a slash character, and the pathname of the file relative to *source_file.*

It shall be an error if *target* does not exist and more than two operands are specified, or if *target* exists and is a file of a type defined by POSIX.1 {8}, but is not a file of type directory.

In the following description, *source_file* refers to the file that is being copied, whether specified as an operand or a file in a file hierarchy rooted in a *source_file* operand. The term *dest_file* refers to the file named by the destination path.

For each *source_file,* the following steps shall be taken:

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

(1) If *source_file* references the same file as *dest_file,* `cp` may write a diagnostic message to standard error; it shall do nothing more with *source_file* and shall go on to any remaining files.

(2) If *source_file* is of type directory, the following steps shall be taken:

(a) If neither the `-R` or `-r` options were specified, `cp` shall write a diagnostic message to standard error, do nothing more with *source_file,* and go on to any remaining files.

(b) If *source_file* was not specified as an operand and *source_file* is dot or dot-dot, `cp` shall do nothing more with *source_file* and go on to any remaining files.

(c) If *dest_file* exists and it is a file type not specified by POSIX.1 {8}, the behavior is implementation defined.

(d) If *dest_file* exists and it is not of type directory, `cp` shall write a diagnostic message to standard error, do nothing more with *source_file* or any files below *source_file* in the file hierarchy, and go on to any remaining files.

(e) If the directory *dest_file* does not exist, it shall be created with file permission bits set to the same value as those of *source_file* , modified by the file creation mask of the user if the `-p` option was not specified, and then bitwise inclusively ORed with S_IRWXU. If *dest_file* cannot be created, `cp` shall write a diagnostic message to standard error, do nothing more with *source_file,* and go on to any remaining files. It is unspecified if `cp` shall attempt to copy files in the file hierarchy rooted in *source_file.*

(f) The files in the directory *source_file* shall be copied to the directory *dest_file*, taking the four steps [(1)-(4)] listed here with the files as *source_file* s.

(g) If *dest_file* was created, its file permission bits shall be changed (if necessary) to be the same as those of *source_file,* modified by the file creation mask of the user if the `-p` option was not specified.

(h) The `cp` utility shall do nothing more with *source_file* and go on to any remaining files.

(3) If *source_file* is of type regular file, the following steps shall be taken:

(a) If *dest_file* exists, the following steps are taken:

[1] If the `-i` option is in effect, the `cp` utility shall write a prompt to the standard error and read a line from the standard input. If the response is not affirmative, `cp` shall do nothing more with *source_file* and go on to any remaining files.

[2] A file descriptor for *dest_file* shall be obtained by performing actions equivalent to the POSIX.1 {8} *open* () function call using *dest_file* as the *path* argument, and the bitwise inclusive OR of O_WRONLY and O_TRUNC as the *oflag* argument.

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.