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All rights reserved as an unpublished work.

This is an unapproved and unpublished IEEE Standards Draft, subject to change.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]

14256 #### 4.72.6.3 Output Files

14257 None.

14258 ### 4.72.7 Extended Description

14259 None.

14260 ### 4.72.8 Exit Status

14261 The `xargs` utility shall exit with one of the following values:

14262 0 All invocations of *utility* returned exit status zero.

14263 1–125 A command line meeting the specified requirements could not be assembled, one or more of the invocations of *utility* returned a nonzero exit status, or some other error occurred.

14266 126 The utility specified by *utility* was found but could not be invoked.

14267 127 The utility specified by *utility* could not be found.

14268 ### 4.72.9 Consequences of Errors

14269 If a command line meeting the specified requirements cannot be assembled, the
14270 utility cannot be invoked, an invocation of the utility is terminated by a signal, or
14271 an invocation of the utility exits with exit status 255, the `xargs` utility shall write
14272 a diagnostic message and exit without processing any remaining input.

14273 ### 4.72.10 Rationale. *(This subclause is not a part of P1003.2)*

14274 **Examples, Usage**

14275 The `xargs` utility is usually found only in System V-based systems; BSD systems
14276 provide an `apply` utility that provides functionality similar to `xargs -n` *number* .
14277 The *SVID* lists `xargs` as a software development extension; POSIX. 2 does not
14278 share the view that it is used only for development, and therefore it is not
14279 optional.

14280 Note that input is parsed as lines and `<blank>`s separate arguments. If `xargs` is
14281 used to bundle output of commands like `find dir -print` or `ls` into commands
14282 to be executed, unexpected results are likely if any file names contain any
14283 `<blank>`s or `<newline>`s. This can be fixed by using `find` to call a script that
14284 converts each file found into a quoted string that is then piped to `xargs`. Note
14285 that the quoting rules used by `xargs` are not the same as in the shell. They were
14286 not made consistent here because existing applications depend on the current
14287 rules and the shell syntax is not fully compatible with it. An easy rule that can be
14288 used to transform any string into a quoted form that `xargs` will interpret
14289 correctly is to precede each character in the string with a backslash.

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

The following command will combine the output of the parenthesized commands onto one line, which is then written to the end of file `log`:

```
(logname; date; printf "%s\n" "$0 $*") | xargs >>log
```

The following command will invoke `diff` with successive pairs of arguments originally typed as command line arguments (assuming there are no embedded <blank>s in the elements of the original argument list):

```
printf "%s\n" "$*" | xargs -n 2 -x diff
```

On implementations with a large value for {ARG_MAX}, `xargs` may produce command lines longer than {LINE _MAX}. For invocation of utilities, this is not a problem. If `xargs` is being used to create a text file, users should explicitly set the maximum command line length with the `-s` option.

History of Decisions Made

The list of options has been scaled down extensively. As it had stood, the `xargs` utility did not exhibit an economy of powerful, modular, or extensible functionality.

The classic application of the `xargs` utility is in conjunction with the `find` utility to reduce the number of processes launched by a simplistic use of the `find -exec` combination. The `xargs` utility is also used to enforce an upper limit on memory required to launch a process. With this basis in mind, POSIX. 2 selected only the minimal features required.

The `-n` *number* option was classically used to evoke a utility using pairs of operands, yet the general case has problems when *utility* spawns child processes of its own. The `xargs` utility can sap resources from these children, especially those sharing the parent's environment.

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.

Although the 255 exit status is mostly an accident of historical implementations, it allows a utility being used by `xargs` to tell `xargs` to terminate if it knows no further invocations using the current data stream will succeed. Any nonzero exit status from a utility will fall into the 1–125 range when `xargs` exits. There is no statement of how the various nonzero utility exit status codes are accumulated by `xargs`. The value could be the addition of all codes, their highest value, the last one received, or a single value such as 1. Since no algorithm is arguably better than the others, and since many of the POSIX. 2 standard utilities say little more

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

(portably) than ''pass/fail,'' no new algorithm was invented.

Several other `xargs` options were withdrawn because simple alternatives already exist within the standard. For example, the `-e`*eofstr* option has a `sed` work around. The `-i`*replstr* option can be just as efficiently performed using a shell `for` loop. Since `xargs` will *exec*() with each input line, the `-i` option will usually not exploit `xarg`'s grouping capabilities.

The `-s` option was reinstated since many of the balloters on Draft 8 felt that it was preferable to the `-r` option invented for that draft that required the implementation to use {ARG_MAX} – *size* bytes for command lines.

The requirement that `xargs` never produce command lines such that invocation of *utility* is within 2048 bytes of hitting the POSIX.1 {8} *exec* {ARG_MAX} limitations is intended to guarantee that the invoked utility has a little bit of room to modify its environment variables and command line arguments and still be able to invoke another utility. Note that the minimum {ARG_MAX} allowed by POSIX.1 {8} is 4096 and the minimum value allowed by POSIX.2 is 2048; therefore, the 2048-byte difference seems reasonable. Note, however, that `xargs` may never be able to invoke a utility if the environment passed in to `xargs` comes close to using {ARG_MAX} bytes.

The version of `xargs` required by POSIX.2 is required to wait for the completion of the invoked command before invoking another command. This was done because existing scripts using `xargs` assumed sequential execution. Implementations wanting to provide parallel operation of the invoked utilities are encouraged to add an option enabling parallel invocation, but should still wait for termination of all of the children before `xargs` terminates normally.

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

Section 5: User Portability Utilities Option

Editor's Note: This empty section is placeholder for a future revision (the User Portability Extension, P1003.2a) to contain descriptions of utilities that are suitable for user portability on asynchronous character terminals. P1003.2a is currently balloting within the IEEE. Contact the IEEE Standards Office to obtain a copy of the latest draft.

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

Section 6: Software Developm ent Utilities Option

This section describes utilities used for the development of applications, including compilation or translation of source code, the creation and maintenance of library archives, and the maintenance of groups of interdependent programs.

The utilities described in this section may be provided by the conforming system; however, any system claiming conformance to the **Software Development Utilities Option** shall provide all of the utilities described here.

6.1 `ar` — Create and maintain library archives

6.1.1 Synopsis

`ar –d [–v]` *archive file* ...

`ar –p [–v]` *archive* [*file* ...]

`ar –r [–cuv]` *archive file* ...

`ar –t [–v]` *archive* [*file* ...]

`ar –x [–v]` *archive* [*file* ...]

6.1.2 Description

The `ar` utility can be used to create and maintain groups of files combined into an archive. Once an archive has been created, new files can be added, and existing files can be extracted, deleted, or replaced. When an archive consists entirely of valid object files, the implementation shall format the archive so that it is usable as a library for link editing (see A.1 and C.2). When some of the archived files are not valid object files, the suitability of the archive for library use is undefined.

All *file* operands can be pathnames. However, files within archives shall be named by a filename, which is the last component of the pathname used when the file was entered into the archive. The comparison of *file* operands to the names of files in archives shall be performed by comparing the last component of the operand to the name of the archive file.

It is unspecified whether multiple files in the archive may be identically named. In the case of such files, however, each *file* operand shall match only the first

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

archive file having a name that is the same as the last component of the *file* operand.

6.1.3 Options

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

The following options shall be supported by the implementation:

`-c` Suppress the diagnostic message that is written to standard error by default when the archive file *archive* is created.

`-d` Delete *file*(s) from *archive*.

`-p` Write the contents of the *file*(s) from *archive* to the standard output. If no *file*(s) are specified, the contents of all files in the archive shall be written in the order of the archive.

`-r` Replace or add *file*(s) to *archive* . If the archive named by *archive* does not exist, a new archive file shall be created and a diagnostic message shall be written to standard error (unless the `-c` option is specified). If no *file*(s) are specified and the *archive* exists, the results are undefined. Files that replace existing files shall not change the order of the archive. Files that do not replace existing files shall be appended to the archive.

`-t` Write a table of contents of *archive* to the standard output. The files specified by the *file* operands shall be included in the written list. If no *file* operands are specified, all files in *archive* shall be included in the order of the archive.

`-u` Update older files. When used with the `-r` option, files within the archive will be replaced only if the corresponding *file* has a modification time that is at least as new as the modification time of the file within the archive.

`-v` Give verbose output. When used with the option characters `-d`, `-r`, or `-x`, write a detailed file-by-file description of the archive creation and maintenance activity, as described in 6.1.6.1.

When used with `-p`, write the name of the file to the standard output before writing the file itself to the standard output, as described in 6.1.6.1.

When used with `-t`, include a long listing of information about the files within the archive, as described in 6.1.6.1.

`-x` Extract the files named by the *file* operands from *archive* . The contents of the archive file shall not be changed. If no *file* operands are given, all files in the archive shall be extracted. If the filename of a file extracted from the archive is longer than that supported in the directory to which it is being extracted, the

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

results are undefined. The modification time of each file extracted shall be set to the time the file is extracted from the archive.

6.1.4 Operands

The following operands shall be supported by the implementation:

archive A pathname of the archive file.

file A pathname. Only the last component shall be used when comparing against the names of files in the archive. If two or more *file* operands have the same last pathname component (basename), the results are unspecified. The implementation's archive format shall not truncate valid filenames of files added to, or replaced in, the archive.

6.1.5 External Influences

6.1.5.1 Standard Input

None.

6.1.5.2 Input Files

The input file named by *archive* shall be a file in the format created by `ar -r`.

6.1.5.3 Environment Variables

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

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.

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

LC_TIME This variable shall determine the format and content for date and time strings written by `ar`.

6.1.5.4 Asynchronous Events

Default.

6.1.6 External Effects

6.1.6.1 Standard Output

If the `-d` option is used with the `-v` option, the standard output format is:

`"d - %s\n"`, *<file>*

where *file* is the operand specified on the command line.

If the `-p` option is used with the `-v` option, `ar` shall precede the contents of each file with:

`"\n<%s>\n\n"`, *<file>*

where *file* is the operand specified on the command line, if *file* operands were specified, and the name of the file in the archive if they were not.

If the `-r` option is used with the `-v` option, and *file* is already in the archive, the standard output format is:

`"r - %s\n"`, *<file>*

where *file* is the operand specified on the command line.

If *file* is being added to the archive with the `-r` option, the standard output format is:

`"a - %s\n"`, *<file>*

where *file* is the operand specified on the command line.

If the `-t` option is used, `ar` writes the names of the files to the standard output in the format:

`"%s\n"`, *<file>*

where *file* is the operand specified on the command line, if *file* operands were specified, or the name of the file in the archive if they were not.

If the `-t` option is used with the `-v` option, the standard output format is:

`"%s %u/%u %u %s %d %d:%d %d %s\n"`, *<member mode>*, *<user ID>*, *<group ID>*, *<number of bytes in member>*, *<abbreviated month>*, *<day-of-month>*, *<hour>*, *<minute>*, *<year>*, *<file>*

Where:

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

file shall be the operand specified on the command line, if *file* operands were specified, or the name of the file in the archive if they were not.

<member mode> shall be formatted the same as the *<file mode>* string defined in 4.39.6.1 (Standard Output of `ls`), except that the first character, the *<entry type>*, is not used; the string represents the file mode of the archive member at the time it was added to, or replaced in, the archive.

The following represent the last-modification time of a file when it was most recently added to or replaced in the archive:

<abbreviated month> shall be equivalent to the `%b` format in `date` (see 4.15).

<day-of-month> shall be equivalent to the `%e` format in `date`.

<hour> shall be equivalent to the `%H` format in `date`.

<minute> shall be equivalent to the `%M` format in `date`.

<year> shall be equivalent to the `%Y` format in `date`.

When **LC_TIME** does not specify the POSIX Locale, a different format and order of presentation of these fields relative to each other may be used in a format appropriate in the specified locale.

If the `-x` option is used with the `-v` option, the standard output format is:

```
"x - %s\n", <file>
```

where *file* is the operand specified on the command line, if *file* operands were specified, or the name of the file in the archive if they were not.

6.1.6.2 Standard Error

Used only for diagnostic messages. The diagnostic message about creating a new archive when `-c` is not specified shall not modify the exit status.

6.1.6.3 Output Files

Archives are files with unspecified formats.

6.1.7 Extended Description

None.

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

6.1.8 Exit Status

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

0 Successful completion.

>0 An error occurred.

6.1.9 Consequences of Errors

Default.

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

Examples, Usage

The archive format is not described. It is recognized that there are several known `ar` formats, which are not compatible. The `ar` utility is being included, however, to allow creation of archives that are intended for use only on the same machine. The archive file is specified as a file and it can be moved as a file. This does allow an archive to be moved from one machine to another machine that uses the same implementation of `ar`.

Utilities such as `pax` (and its forebears `tar` and `cpio`) also provide portable "archives." This is a not a duplication; the `ar` interface is included in the standard to provide an interface primarily for `make` and the compilers, based on a historical model.

In historical implementations, the `-q` option is known to execute quickly because `ar` does not check whether the added members are already in the archive. This is useful to bypass the searching otherwise done when creating a large archive piece-by-piece. The remarks may or may not hold true for a brand-new POSIX.2 implementation; and hence, these remarks have been moved out of the specification and into the Rationale.

Likewise, historical implementations maintain a symbol table to speed searches, particularly when the archive contains object files. However, future implementors may or may not use a symbol table, and the `-s` option was removed from this clause to permit implementors freedom of choice. Instead, the requirement that archive libraries be suitable for link editing was added to ensure the intended functionality. Systems such as System V maintain the symbol table without requiring the use of `-s`, so adding `-s` (even if it were worded as allowing a no-op) would essentially require all portable applications to use it in all invocations involving libraries.

The Operands subclause requires what might seem to be true without specifying it: the archive cannot truncate the filenames below {NAME_MAX}. Some historical implementations do so, however, causing unexpected results for the application. Therefore, POSIX.2 makes the requirement explicit to avoid misunderstandings.

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

According to the System V documentation, the options `–dmpqrtx` are not required to begin with a hyphen (–). POSIX.2 requires that a conforming application use the leading hyphen.

When extracting files with long filenames into a file system that supports only shorter filenames, an undefined condition occurs. Typical implementation actions might be one of the following:

— Extract and truncate the filename only when an existing file would not be overlaid.

— Extract and truncate the filename and overlay an existing file only if some extension such as another command-line option were used to override this safety feature.

— Refuse to extract any files unless an extension overrode the default.

The archive format used by the 4.4BSD implementation is documented in the rationale as an example:

> A file created by `ar` begins with the ''magic'' string ''`!<arch>\n`''. The rest of the archive is made up of objects, each of which is composed of a header for a file, a possible filename, and the file contents. The header is portable between machine architectures, and, if the file contents are printable, the archive is itself printable.
>
> The header is made up of six ASCII fields, followed by a two-character trailer. The fields are the object name (16 characters), the file last modification time (12 characters), the user and group IDs (each 6 characters), the file mode (8 characters) and the file size (10 characters). All numeric fields are in decimal, except for the file mode, which is in octal.
>
> The modification time is the file *st_mtime* field. The user and group IDs are the file *st_uid* and *st_gid* fields. The file mode is the file *st_mode* field. The file size is the file *st_size* field. The two-byte trailer is the string ''`<newline>`''.
>
> Only the name field has any provision for overflow. If any filename is more than 16 characters in length or contains an embedded space, the string ''`#1/`''followed by the ASCII length of the name is written in the name field. The file size (stored in the archive header) is incremented by the length of the name. The name is then written immediately following the archive header.
>
> Any unused characters in any of these fields are written as `<space>` characters. If any fields are their particular maximum number of characters in length, there will be no separation between the fields.
>
> Objects in the archive are always an even number of bytes long; files that are an odd number of bytes long are padded with a `<newline>` character, although the size in the header does not reflect this.

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

History of Decisions Made

The `ar` utility description requires that (when all its members are valid object files) `ar` produce an object code library, which the linkage editor can use to extract object modules. If the linkage editor needs a symbol table to permit random access to the archive, `ar` must provide it; however, `ar` does not require a symbol table. The historical `-m` and `-q` positioning options were omitted, as were the positioning modifiers formerly associated with the `-m` and `-r` options, because the two functions of positioning are handled by the `ranlib`-style (a utility found on some historical systems to create symbol tables within the archive) symbol tables and/or the ability of portable applications to create multiple archives instead of loading from a single archive.

Earlier drafts had elaborate descriptions in the Asynchronous Events subclause about how signals were caught and then resent to itself. These were removed in favor of the default case because they are essentially implementation details, unnecessary for the application. Similarly, information about where (and if) temporary files are created was removed from earlier drafts.

The BSD `-o` option was omitted. It is a rare portable application that will use `ar` to extract object code from a library with concern for its modification time, since this can only be of importance to `make`. Hence, since this functionality is not deemed important for applications portability, the modification time of the extracted files is set to the current time.

There is at least one known implementation (for a small computer) that can accommodate only object files for that system, disallowing mixed object and other files. The ability to handle any type of file is not only existing practice for most implementations, but is also a reasonable expectation.

Consideration was given to changing the output format of `ar -tv` to the same format as the output of `ls -l`. This would have made parsing the output of `ar` the same as that of `ls`. This was rejected in part because the current `ar` format is commonly used and changes would break existing usage. Second, `ar` gives the user ID and group ID in numeric format separated by a slash. Changing this to be the user name and group name would not be right if the archive were moved to a machine that contained a different user database. Since `ar` cannot know whether the archive file was generated on the same machine, it cannot tell what to report.

The text on the `-ur` option combination is historical practice—since one filename can easily represent two different files (e.g., `/a/foo` and `/b/foo`), it is reasonable to replace the member in the archive even when the modification time in the archive is identical to that in the file system.

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

6.2 make — Maintain, update, and regenerate groups of programs

6.2.1 Synopsis

```
make [-einpqrst][-f makefile ]... [ -k | -S ][macro=name ]...
     [target_name ... ]
```

6.2.2 Description

The make utility can be used as a part of software development to update files that are derived from other files. A typical case is one where object files are derived from the corresponding source files. The make utility examines time relationships and updates those derived files (called targets) that have modified times earlier than the modified times of the files (called prerequisites) from which they are derived. A description file (''makefile'') contains a description of the relationships between files, and the commands that must be executed to update the targets to reflect changes in their prerequisites. Each specification, or rule, shall consist of a target, optional prerequisites, and optional commands to be executed when a prerequisite is newer than the target. There are two types of rules:

— Inference rules, which have one target name with at least one period (.) and no slash (/)

— Target rules, which can have more than one target name

In addition, make shall have a collection of built-in macros and inference rules that infer prerequisite relationships to simplify maintenance of programs.

To receive exactly the behavior described in this clause, a portable makefile shall:

— Include the special target .POSIX (see 6.2.7.3)

— Omit any special target reserved for implementations (a leading period followed by uppercase letters) that has not been specified by this clause.

The behavior of make is unspecified if either or both of these conditions are not met.

6.2.3 Options

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

The following options shall be supported by the implementation:

-e Cause environment variables, including those with null values, to override macro assignments within makefiles.

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

309 `-f` *makefile* Specify a different makefile. The argument *makefile* is a path-
310 name of a description file, which is also referred to as the
311 *makefile* . A pathname of `"-"` shall denote the standard input.
312 There can be multiple instances of this option, and they shall be
313 processed in the order specified. The effect of specifying the same
314 option-argument more than once is unspecified. See 6.2.7.1.

315 `-i` Ignore error codes returned by invoked commands. This mode is
316 the same as if the special target `.IGNORE` were specified without 1
317 prerequisites. See 6.2.7.2. 1

318 `-k` Continue to update other targets that do not depend on the
319 current target if a nonignored error occurs while executing the
320 commands to bring a target up to date.

321 `-n` Write commands that would be executed on standard output, but
322 do not execute them. However, lines with a plus-sign (+) prefix
323 shall be executed. In this mode, lines with an at-sign (@) charac-
324 ter prefix shall be written to standard output.

325 `-p` Write to standard output the complete set of macro definitions
326 and target descriptions. The output format is unspecified.

327 `-q` Return a zero exit value if the target file is up-to-date; otherwise,
328 return an exit value of 1. Targets shall not be updated if this
329 option is specified. However, a command line (associated with the
330 targets) with a plus-sign (+) prefix shall be executed.

331 `-r` Clear the suffix list and do not use the built-in rules.

332 `-S` Terminate `make` if an error occurs while executing the commands
333 to bring a target up-to-date. This shall be the default and the
334 opposite of `-k`.

335 `-s` Do not write command lines or touch messages (see `-t`) to stan-
336 dard output before executing. This mode shall be the same as if
337 the special target `.SILENT` were specified without prerequisites. 1
338 See 6.2.7.2. 1

339 `-t` Update the modification time of each target as though a `touch`
340 *target* had been executed. See `touch` in 4.63. Targets that have 1
341 prerequisites but no commands (see 6.2.7.3), or that are already 1
342 up-to-date, shall not be touched in this manner. Write messages 1
343 to standard output for each target file indicating the name of the
344 file and that it was touched. Normally, the command lines associ-
345 ated with each target are not executed. However, a command line
346 with a plus-sign (+) prefix shall be executed.

347 If the `-k` and `-S` options are both specified on the command line, by the
348 **MAKEFLAGS** environment variable, or by the `MAKEFLAGS` macro, the last one
349 evaluated shall take precedence. The **MAKEFLAGS** environment variable shall
350 be evaluated first and the command line shall be evaluated second. Assignments
351 to the `MAKEFLAGS` macro shall be evaluated as described in 6.2.5.3.

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

6.2.4 Operands

The following operands shall be supported by the implementation:

target_name Target names, as defined in 6.2.7. If no target is specified, while `make` is processing the makefiles, the first target that `make` encounters that is not a special target or an inference rule shall be used.

macro=name Macro definitions, as defined in 6.2.7.4.

If the *target_name* and *macro=name* operands are intermixed on the command line, the results are unspecified.

6.2.5 External Influences

6.2.5.1 Standard Input

The standard input shall be used only if the *makefile* option-argument is −. See Input Files.

6.2.5.2 Input Files

The input file, otherwise known as the makefile, is a text file containing rules, macro definitions, and comments. (See 6.2.7.)

6.2.5.3 Environment Variables

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

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 and input files).

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

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

MAKEFLAGS This variable shall be interpreted as a character string representing a series of option characters to be used as the default options. The implementation shall accept both of the following formats (but need not accept them when intermixed):

(1) The characters are option letters without the leading hyphens or <blank> separation used on a command line.

(2) The characters are formatted in a manner similar to a portion of the make command line: options are preceded by hyphens and <blank>-separated as described in 2.10.2. The *macro* =*name* macro definition operands can also be included. The difference between the contents of **MAKEFLAGS** and the command line is that the contents of the variable shall not be subjected to the word expansions (see 3.6) associated with parsing the command line values.

When the command-line options −f or −p are used, they shall take effect regardless of whether they also appear in **MAKEFLAGS** . If they otherwise appear in **MAKEFLAGS** , the result is undefined.

The **MAKEFLAGS** variable shall be accessed from the environment before the makefile is read. At that time, all of the options (except −f and −p) and command-line macros not already included in MAKEFLAGS shall be added to the MAKEFLAGS macro. The MAKEFLAGS macro shall be passed into the environment as an environment variable for all child processes. If the MAKEFLAGS macro is subsequently set by the makefile, it shall replace the **MAKEFLAGS** variable currently found in the environment.

The value of the **SHELL** environment variable shall not be used as a macro and shall not be modified by defining the SHELL macro in a makefile or on the command line. All other environment variables, including those with null values, shall be used as macros, as defined in 6.2.7.4.

6.2.5.4 Asynchronous Events

If not already ignored, make shall trap SIGHUP , SIGTERM , SIGINT , and SIGQUIT and remove the current target unless the target is a directory or the target is a prerequisite of the special target .PRECIOUS or unless one of the −n, −p, or −q options was specified. Any targets removed in this manner shall be reported in diagnostic messages of unspecified format, written to standard error. After this cleanup process, if any, make shall take the standard action for all other signals; see 2.11.5.4.

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

6.2.6 External Effects

6.2.6.1 Standard Output

The `make` utility shall write all commands to be executed to standard output unless the `-s` option was specified, the command is prefixed with an at-sign, or the special target `.SILENT` has either the current target as a prerequisite or has no prerequisites. If `make` is invoked without any work needing to be done, it shall write a message to standard output indicating that no action was taken.

6.2.6.2 Standard Error

Used only for diagnostic messages.

6.2.6.3 Output Files

None. However, utilities invoked by `make` may create additional files.

6.2.7 Extended Description

The `make` utility attempts to perform the actions required to ensure that the specified target(s) are up-to-date. A target is considered out-of-date if it is older than any of its prerequisites or if it does not exist. The `make` utility shall treat all prerequisites as targets themselves and recursively ensure that they are up-to-date, processing them in the order in which they appear in the rule. The `make` utility shall use the modification times of files to determine if the corresponding targets are out-of-date. (See 2.9.1.6.)

After `make` has ensured that all of the prerequisites of a target are up-to-date, and if the target is out-of-date, the commands associated with the target entry shall be executed. If there are no commands listed for the target, the target shall be treated as up-to-date.

6.2.7.1 Makefile Syntax

A makefile can contain rules, macro definitions (see 6.2.7.4), and comments. There are two kinds of rules: inference rules (6.2.7.5) and target rules (6.2.7.3). The `make` utility shall contain a set of built-in inference rules. If the `-r` option is present, the built-in rules shall not be used and the suffix list shall be cleared. Additional rules of both types can be specified in a makefile. If a rule or macro is defined more than once, the value of the rule or macro shall be that of the last one specified. Comments start with a number-sign (#) and continue until an unescaped `<newline>` is reached.

By default, the file `./makefile` shall be used. If `./makefile` is not found, the file `./Makefile` shall be tried. If neither `./makefile` nor `./Makefile` are found, other implementation-defined pathnames may also be tried.

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

The −f option shall direct make to ignore ./makefile and ./Makefile (and any implementation-defined variants) and use the specified argument as a makefile instead. If the − argument is specified, standard input shall be used.

The term *makefile* is used to refer to any rules provided by the user whether in ./makefile, ./Makefile, or specified by the −f option.

The rules in makefiles shall consist of the following types of lines: target rules, including special targets (see 6.2.7.3); inference rules (see 6.2.7.5); macro definitions (see 6.2.7.4); empty lines; and comments. Comments start with a number sign (#) and continue until an unescaped <newline> is reached.

When an escaped <newline> (one preceded by a backslash) is found anywhere in the makefile, it shall be replaced, along with any leading white space on the following line, with a single <space>.

6.2.7.2 Makefile Execution

Command lines shall be processed one at a time by writing the command line to the standard output (unless one of the conditions listed below under ''@'' suppresses the writing) and executing the command(s) in the line. A <tab> character may precede the command to standard output. Commands shall be executed by passing the command line to the command interpreter in the same manner as if the string were the argument to the function in 7.1.1 [such as the *system* () function in the C binding].

The environment for the command being executed shall contain all of the variables in the environment of make. The macros from the command line to make shall be added to make's environment. Other implementation-defined variables may also be added to make's environment. If any command-line macro has been defined elsewhere, the command-line value shall overwrite the existing value. If the **MAKEFLAGS** variable is not set in the environment in which make was invoked, in the makefile, or on the command line, it shall be created by make, and shall contain all options specified on the command line except for the −f and −p options. It may also contain implementation-defined options.

By default, when make receives a nonzero status from the execution of a command, it terminates with an error message to standard error.

Command lines can have one or more of the following prefixes: a hyphen (−), an at-sign (@), or a plus-sign (+). These modify the way in which make processes the command. When a command is written to standard output, the prefix shall not be included in the output.

- If the command prefix contains a hyphen, or the −i option is present, or the special target .IGNORE has either the current target as a prerequisite or has no prerequisites, any error found while executing the command shall be ignored.

@ If the command prefix contains an at-sign and the command-line −n option is not specified, or the −s option is present, or the special target .SILENT has either the current target as a prerequisite or has no prerequisites, the

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