IEEE P1003.2 Draft 11.2 – September 1991

This is an unapproved and unpublished IEEE Standards Draft, subject to change.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

The command:

```
umask -- -w
```

sets the mode mask so that subsequently created files have all their write bits cleared. Note that *mask* operands `−r`, `−w`, `−x`, or anything beginning with a hyphen, must be preceded by `--` to keep it from being interpreted as an option.

History of Decisions Made

The description of the historical utility was modified to allow it to use the symbolic modes of `chmod`. The `−s` option used in earlier drafts was changed to `−S` because `−s` could be confused with a *symbolic_mode* form of mask referring to the S_ISUID and S_ISGID bits.

The default output style is implementation defined to permit implementors to provide migration to the new symbolic style at the time most appropriate to their users. Earlier drafts of this standard specified an `−o` flag to force octal mode output. This was dropped because the octal mode may not be sufficient to specify all of the information that may be present in the file mode creation mask when more secure file access permission checks are implemented.

It has been suggested that trusted systems developers might appreciate softening the requirement that the mode mask ‘‘affects’’ the file access permissions, since it seems access control lists might replace the mode mask to some degree. The wording has been changed to say that it affects the file permission bits, and leaves the details of the behavior of how they affect the file access permissions to the description in POSIX.1 {8}.

4.68 `uname` — Return system name

4.68.1 Synopsis

```
uname [−amnrsv]
```

4.68.2 Description

By default, the `uname` utility shall write the operating system name to standard output. When options are specified, symbols representing one or more system characteristics shall be written to the standard output. The format and contents of the symbols are implementation defined. On systems conforming to POSIX.1 {8}, the symbols written shall be those supported by the POSIX.1 {8} *uname*() function.

4.68.3 Options

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

The following options shall be supported by the implementation:

`−a`	Behave as though all of the options `−mnrsv` were specified.
`−m`	Write the name of the hardware type on which the system is running to standard output.
`−n`	Write the name of this node within an implementation-specified communications network.
`−r`	Write the current release level of the operating system implementation.
`−s`	Write the name of the implementation of the operating system.
`−v`	Write the current version level of this release of the operating system implementation.

If no options are specified, the `uname` utility shall write the operating system name, as if the `−s` option had been specified.

4.68.4 Operands

None.

4.68.5 External Influences

4.68.5.1 Standard Input

None.

4.68.5.2 Input Files

None.

4.68.5.3 Environment Variables

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

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.

4.68.5.4 Asynchronous Events

Default.

4.68.6 External Effects

4.68.6.1 Standard Output

By default, the output shall be a single line of the following form:

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

If the `-a` option is specified, the output shall be a single line of the following form:

`"%s %s %s %s %s\n"`, *<sysname>* , *<nodename>* , *<release>*, *<version>* , *<machine>*

Additional implementation-defined symbols may be written; all such symbols shall be written at the end of the line of output before the `<newline>`.

If options are specified to select different combinations of the symbols, only those symbols shall be written, in the order shown above for the `−a` option. If a symbol is not selected for writing, its corresponding trailing `<blank>`s also shall not be written.

4.68.6.2 Standard Error

Used only for diagnostic messages.

4.68.6.3 Output Files

None.

4.68.7 Extended Description

None.

4.68.8 Exit Status

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

0	The requested information was successfully written.
>0	An error occurred.

4.68.9 Consequences of Errors

Default.

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

Examples, Usage

The following command:

```
uname −sr
```

writes the operating system name and release level, separated by one or more `<blank>`s.

Note that any of the symbols could include embedded `<space>`s, which may affect parsing algorithms if multiple options are selected for output.

The node name is typically a name that the system uses to identify itself for intersystem communication addressing.

History of Decisions Made

It was suggested that this utility cannot be used portably, since the format of the symbols is implementation defined. The POSIX.1 {8} working group could not achieve consensus on defining these formats in the underlying *uname*() function and there is no expectation that POSIX.2 would be any more successful. In any event, some applications may still find this historical utility of value. For example, the symbols could be used for system log entries or for comparison with operator or user input.

4.69 `uniq` — Report or filter out repeated lines in a file

4.69.1 Synopsis

`uniq` [−c| −d| −u] [−f *fields*] [−s *chars*] [*input_file* [*output_file*]]

Obsolescent Version:

`uniq` [−c| −d| −u] [−*n*] [+*m*] [*input_file* [*output_file*]]

4.69.2 Description

The `uniq` utility shall read an input file comparing adjacent lines, and write one copy of each input line on the output. The second and succeeding copies of repeated adjacent input lines shall not be written.

Repeated lines in the input shall not be detected if they are not adjacent.

4.69.3 Options

The `uniq` utility shall conform to the utility argument syntax guidelines described in 2.10.2; the obsolescent version does not, as one of the options begins with + and the −*m* and +*n* options do not have option letters.

The following options shall be supported by the implementation:

`−c`	Precede each output line with a count of the number of times the line occurred in the input.
`−d`	Suppress the writing of lines that are not repeated in the input.
`−f` *fields*	Ignore the first *fields* fields on each input line when doing comparisons, where *fields* shall be a positive decimal integer. A field is the maximal string matched by the basic regular expresssion: `[[:blank:]]*[^[:blank:]]*` If the *fields* option-argument specifies more fields than appear on an input line, a null string shall be used for comparison.

`−s` *chars*	Ignore the first *chars* characters when doing comparisons, where *chars* shall be a positive decimal integer. If specified in conjunction with the `−f` option, the first *chars* characters after the first *fields* fields shall be ignored. If the *chars* option-argument specifies more characters than remain on an input line, a null string shall be used for comparison.
`−u`	Suppress the writing of lines that are repeated in the input.
−*n*	(Obsolescent.) Equivalent to `−f` *fields* with *fields* set to *n*.
+*m*	(Obsolescent.) Equivalent to `−s` *chars* with *chars* set to *m*.

4.69.4 Operands

The following operands shall be supported by the implementation:

input_file	A pathname of the input file. If the *input_file* operand is not specified, or if the *input_file* is −, the standard input shall be used.
output_file	A pathname of the output file. If the *output_file* operand is not specified, the standard output shall be used. The results are unspecified if the file named by *output_file* is the file named by *input_file* .

4.69.5 External Influences

4.69.5.1 Standard Input

The standard input shall be used only if no *input_file* operand is specified or if *input_file* is −. See Input Files.

4.69.5.2 Input Files

The input file shall be a text file.

4.69.5.3 Environment Variables

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

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) and which characters constitute a `<blank>` in the current locale.
LC_MESSAGES	This variable shall determine the language in which messages should be written.

4.69.5.4 Asynchronous Events

Default.

4.69.6 External Effects

4.69.6.1 Standard Output

The standard output shall be used only if no *output_file* operand is specified. See Output Files.

4.69.6.2 Standard Error

Used only for diagnostic messages.

4.69.6.3 Output Files

If the `-c` option is specified, the output file shall be empty or each line will be of the form:

`"%d %s"`, <*number of duplicates*> , <*line*>

otherwise, the output file will be empty or each line will be of the form:

`"%s"`, <*line*>

4.69.7 Extended Description

None.

4.69.8 Exit Status

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

0	The utility executed successfully.
>0	An error occurred.

4.69.9 Consequences of Errors

Default.

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

Examples, Usage

Some historical implementations have limited lines to be 1080 bytes in length, which will not meet the implied {LINE_MAX} limit.

The `sort` utility (see 4.58) can be used to cause repeated lines to be adjacent in the input file.

The following input file data (but flushed left) was used for a test series on `uniq`:

```
#01 foo0 bar0 foo1 bar1
#02 bar0 foo1 bar1 foo1
#03 foo0 bar0 foo1 bar1
#04
#05 foo0 bar0 foo1 bar1
#06 foo0 bar0 foo1 bar1
#07 bar0 foo1 bar1 foo0
```

What follows is a series of test invocations of the `uniq` utility that use a mixture of `uniq`'s options against the input file data. These tests verify the meaning of *adjacent*. The `uniq` utility views the input data as a sequence of strings delimited by `\n`. Accordingly, for the *fields* th member of the sequence, `uniq` interprets unique or repreated adjacent lines strictly relative to the *fields* +1th member.

This first example tests the line counting option, comparing each line of the input file data starting from the second field:

```
uniq -c -f 1 uniq_0I.t
  1 #01 foo0 bar0 foo1 bar1
  1 #02 bar0 foo1 bar1 foo0
  1 #03 foo0 bar0 foo1 bar1
  1 #04
  2 #05 foo0 bar0 foo1 bar1
  1 #07 bar0 foo1 bar1 foo0
```

The number `2`, prefixing the fifth line of output, signifies that the `uniq` utility detected a pair of repeated lines. Given the input data, this can only be true when `uniq` is run using the `-f 1` option (which causes `uniq` to ignore the first field on each input line).

The second example tests the option to suppress unique lines, comparing each line of the input file data starting from the second field:

```
uniq -d -f 1 uniq_0I.t
#05 foo0 bar0 foo1 bar1
```

This test suppresses repeated lines, comparing each line of the input file data starting from the second field:

```
uniq -u -f 1 uniq_0I.t
     #01 foo0 bar0 foo1 bar1
     #02 bar0 foo1 bar1 foo1
     #03 foo0 bar0 foo1 bar1
     #04
     #07 bar0 foo1 bar1 foo0
```

This suppresses unique lines, comparing each line of the input file data starting from the third character:

```
     uniq -d -s 2 uniq_0I.t
```

In the last example, the `uniq` utility found no input matching the above criteria.

History of Decisions Made

The `-f` and `-s` options were added to replace the obsolescent −*n* and +*m* options so that `uniq` could meet the syntax guidelines in an upward-compatible way.

The output specifications in Output Files do not show a terminating `<newline>` because they both specify <*line*>, which includes its own `<newline>` (because of the definition of *line*).

4.70 `wait` — Await process completion

4.70.1 Synopsis

`wait` [*pid* . . .]

4.70.2 Description

When an asynchronous list (see 3.9.3.1) is started by the shell, the process ID of the last command in each element of the asynchronous list shall become known in the current shell execution environment; see 3.12. 1

If the `wait` utility is invoked with no operands, it shall wait until all process IDs known to the invoking shell have terminated and exit with a zero exit status.

If one or more *pid* operands are specified that represent known process IDs, the `wait` utility shall wait until all of them have terminated. If one or more *pid* operands are specified that represent unknown process IDs, `wait` shall treat them as if they were known process IDs that exited with exit status 127. The exit

specified by *string1* the set of all characters in the current character set, as defined

If the `-c` option is specified without `-d`, the complement of the characters

4.70.3 Options

None.

4.70.4 Operands

The following operand shall be supported by the implementation:

pid	The unsigned decimal integer process ID of a command, for which the utility is to wait for the termination.

4.70.5 External Influences

4.70.5.1 Standard Input

None.

4.70.5.2 Input Files

None.

4.70.5.3 Environment Variables

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

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.

4.70.5.4 Asynchronous Events

Default.

4.70.6 External Effects

4.70.6.1 Standard Output

None.

4.70.6.2 Standard Error

Used only for diagnostic messages.

4.70.6.3 Output Files

None.

4.70.7 Extended Description

None.

4.70.8 Exit Status

If one or more operands were specified, all of them have terminated or were not known by the invoking shell, and the status of the last operand specified is known, then the exit status of `wait` shall be the exit status information of the command indicated by the last operand specified. If the process terminated abnormally due to the receipt of a signal, the exit status shall be greater than 128 and shall be distinct from the exit status generated by other signals, but the exact value is unspecified. (See the `kill −l` option in 4.32.) Otherwise, the `wait` utility shall exit with one of the following values:

0	The `wait` utility was invoked with no operands and all process IDs known by the invoking shell have terminated.
1−126	The `wait` utility detected an error.
127	The command identified by the last *pid* operand specified is unknown.

4.70.9 Consequences of Errors

Default.

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

Examples, Usage

On most implementations, `wait` is a shell built-in. If it is called in a subshell or separate utility execution environment, such as one of the following:

```
(wait)
nohup wait ...
find . -exec wait ... \;
```

it will return immediately because there will be no known process IDs to wait for in those environments.

Although the exact value used when a process is terminated by a signal is unspecified, if it is known that a signal terminated a process, a script can still reliably figure out which signal using `kill` as shown by the following script:

```
sleep 1000&
pid=$!
kill -kill $pid
wait $pid
echo $pid was terminated by a SIG$(kill -l $?) signal.
```

Historical implementations of interactive shells have discarded the exit status of terminated background processes before each shell prompt. Therefore, the status of background processes was usually lost unless it terminated while `wait` was waiting for it. This could be a serious problem when a job that was expected to run for a long time actually terminated quickly with a syntax or initialization error because the exit status returned was usually zero if the requested process ID was not found. POSIX.2 requires the implementation to keep the status of terminated jobs available until the status is requested, so that scripts like:

```
j1&
p1=$!
j2&
wait $p1
echo Job 1 exited with status $?
wait $!
echo Job 2 exited with status $?
```

will work without losing status on any of the jobs. The shell is allowed to discard the status of any process that it determines the application cannot get the process ID from the shell. It is also required to remember only {CHILD_MAX} number of processes in this way. Since the only way to get the process ID from the shell is by using the ! shell parameter, the shell is allowed to discard the status of an asynchronous list if `$!` was not referenced before another asynchronous list was started. (This means that the shell only has to keep the status of the last asynchronous list started if the application did not reference `$!`. If the implementation of the shell is smart enough to determine that a reference to `$!` was not ‘‘saved’’ anywhere that the application can retrieve it later, it can use this information to trim the list of saved information. Note also that a successful call to `wait` with no operands discards the exit status of all asynchronous lists.)

This new functionality was added because it is needed to accurately determine the exit status of any asynchronous list. The only compatibility problem that this change creates is for a script like:

```
    while sleep 60
    do
            job&
            echo Job started $(date) as $!
    done
```

which will cause the shell to keep track of all of the jobs started until the script terminates or runs out of memory. This would not be a problem if the loop did not reference `$!` or if the script would occasionally `wait` for jobs it started.

If the exit status of `wait` is greater than 128, there is no way for the application to know if the waited for process exited with that value or was killed by a signal. Since most utilities exit with small values, there is seldom any ambiguity. Even in the ambiguous cases, most applications just need to know that the asynchronous job failed; it does not matter whether it detected an error and failed or was killed and did not complete its job normally.

History of Decisions Made

The description of `wait` does not refer to the *waitpid* () function from POSIX.1 {8}, because that would needlessly overspecify this interface. However, the wording requires that `wait` is required to wait for an explicit process when it is given an argument, so that the status information of other processes is not consumed. Historical implementations use POSIX.1 {8} *wait* () until *wait* () returns the requested process ID or finds that the requested process does not exist. Because this means that a shell script could not reliably get the status of all background children if a second background job was ever started before the first job finished, it is recommended that the `wait` utility use a method such as the functionality provided by the *waitpid* () function in POSIX.1 {8}.

The ability to wait for multiple *pid* operands was adopted from the KornShell at the request of ballot comments and objections.

Some implementations of `wait` support waiting for asynchronous lists identified by the use of job identifiers. For example, `wait %1` would wait for the first background job. This standard does not address job control issues, but allows these features to be added as extensions. Job control facilities will be provided by the UPE.

4.71 `wc` — Word, line, and byte count

4.71.1 Synopsis

`wc [−clw]` [*file* ...]

4.71.2 Description

The `wc` utility shall read one or more input files and, by default, write the number of `<newline>`s, words, and bytes contained in each input file to the standard output.

The utility also shall write a total count for all named files, if more than one input file is specified.

The `wc` utility shall consider a *word* to be a nonzero-length string of characters delimited by white space.

4.71.3 Options

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

The following options shall be supported by the implementation:

`−c`	Write to the standard output the number of bytes in each input file.
`−l`	Write to the standard output the number of `<newline>`s in each input file.
`−w`	Write to the standard output the number of words in each input file.

When any option is specified, `wc` shall report only the information requested by the specified option(s).

4.71.4 Operands

The following operand shall be supported by the implementation:

file	A pathname of an input file. If no *file* operands are specified, the standard input shall be used.

4.71.5 External Influences

4.71.5.1 Standard Input

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

4.71.5.2 Input Files

The input files may be of any type.

4.71.5.3 Environment Variables

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

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) and which characters are defined as ‘‘white space’’ characters.
LC_MESSAGES	This variable shall determine the language in which messages should be written.

4.71.5.4 Asynchronous Events

Default.

4.71.6 External Effects

4.71.6.1 Standard Output

By default, the standard output shall contain a line for each input file of the form:

`"%d %d %d %s\n"`, *<newlines>* , *<words>* , *<bytes>*, *<file>*

If any options are specified and the `-l` option is not specified, the number of `<newline>`s shall not be written.

If any options are specified and the `−w` option is not specified, the number of words shall not be written.

If any options are specified and the `−c` option is not specified, the number of bytes shall not be written.

If no input *file* operands are specified, no name shall be written and no `<blank>`s preceding the pathname shall be written.

If more than one input *file* operand is specified, an additional line shall be written, of the same format as the other lines, except that the word `total` (in the POSIX Locale) shall be written instead of a pathname and the total of each column shall be written as appropriate. Such an additional line, if any, shall be written at the end of the output.

4.71.6.2 Standard Error

Used only for diagnostic messages.

4.71.6.3 Output Files

None.

4.71.7 Extended Description

None.

4.71.8 Exit Status

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

0	Successful completion.
>0	An error occurred.

4.71.9 Consequences of Errors

Default.

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

Examples, Usage

None.

History of Decisions Made

The output file format pseudo-*printf*() string was derived from the HP-UX version of `wc`; the System V version:

```
"%7d%7d%7d %s\n"
```

produces possibly ambiguous and unparsable results for very large files, as it assumes no number will exceed six digits.

Some historical implementations use only `<space>`, `<tab>`, and `<newline>` as word separators. The equivalent of the C Standard {7} *isspace*() function is more appropriate.

The `−c` option stands for ‘‘character’’ count, even though it counts bytes. This stems from the sometimes erroneous historical view that bytes and characters are the same size.

Earlier drafts only specified the results when input files were text files. The current specification more closely matches existing practice. (Bytes, words, and `<newline>`s are counted separately and the results are written when an end-of-file is detected.)

Historical implementations of the `wc` utility only accepted one argument to specify the options `−c`, `−l`, and `−w`. Some of them also had multiple occurrences of an option cause the corresponding count to be output multiple times and having the order of specification of the options affect the order of the fields on output, but did not document either of these. Because common usage either specifies no options or only one option and because none of this was documented, the changes required by this standard should not break many existing applications (and does not break any historical portable applications.)

4.72 `xargs` — Construct argument list(s) and invoke utility

4.72.1 Synopsis

```
xargs [-t][-n number [-x]][-s size][utility [argument ...]]
```

4.72.2 Description

The `xargs` utility shall construct a command line consisting of the *utility* and *argument* operands specified followed by as many arguments read in sequence from standard input as will fit in length and number constraints specified by the options. The `xargs` utility shall then invoke the constructed command line and wait for its completion. This sequence shall be repeated until an end-of-file condition is detected on standard input or an invocation of a constructed command line returns an exit status of 255. 1

Arguments in the standard input shall be separated by unquoted `<blank>`s, or unescaped `<blank>`s or `<newline>`s. A string of zero or more nondouble-quote (`"`) and non-`<newline>` characters can be quoted by enclosing them in double-quotes. A string of zero or more nonapostrophe (`'`) and non-`<newline>` characters can be quoted by enclosing them in apostrophes. Any unquoted character can be escaped by preceding it with a backslash. The *utility* shall be executed one or more times until the end-of-file is reached. The results are unspecified if the utility named by *utility* attempts to read from its standard input.

The generated command line length shall be the sum of the size in bytes of the utility name and each argument treated as strings, including a null byte terminator for each of these strings. The `xargs` utility shall limit the command line length such that when the command line is invoked, the combined argument and environment lists (see the *exec* family of functions in POSIX.1 {8} 3.1.2) shall not exceed {ARG_MAX}−2048 bytes. Within this constraint, if neither the `-n` nor the `-s` option is specified, the default command line length shall be at least {LINE_MAX}.

4.72.3 Options

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

The following options shall be supported by the implementation:

`-n` *number* Invoke *utility* using as many standard input arguments as possible, up to *number* (a positive decimal integer) arguments maximum. Fewer arguments shall be used if:

- The command line length accumulated exceeds the size specified by the `-s` option (or {LINE_MAX} if there is no `-s` option), or

	— The last iteration has fewer than *number* , but not zero, operands remaining.
`−s` *size*	Invoke *utility* using as many standard input arguments as possible yielding a command line length less than *size* (a positive decimal integer) bytes. Fewer arguments shall be used if: — The total number of arguments exceeds that specified by the `−n` option, or — End of file is encountered on standard input before *size* bytes are accumulated. Implementations shall support values of *size* up to at least {LINE_MAX} bytes, provided that the constraints specified in 4.72.2 are met. It shall not be considered an error if a value larger than that supported by the implementation or exceeding the constraints specified in 4.72.2 is given; `xargs` shall use the largest value it supports within the constraints.
`−t`	Enable trace mode. Each generated command line shall be written to standard error just prior to invocation.
`−x`	Terminate if a command line containing *number* arguments (see the `−n` option above) will not fit in the implied or specified size (see the `−s` option above).

4.72.4 Operands

The following operands shall be supported by the implementation:

utility	The name of the utility to be invoked, found by search path using the **PATH** environment variable, described in 2.6. If *utility* is omitted, the default shall be the `echo` utility (see 4.19). If the *utility* operand names any of the special built-in utilities in 3.14, the results are undefined.
argument	An initial option or operand for the invocation of *utility* .

4.72.5 External Influences

4.72.5.1 Standard Input

The standard input shall be a text file. The results are unspecified if an end-of-file condition is detected immediately following an escaped `<newline>`.

4.72.5.2 Input Files

None.

4.72.5.3 Environment Variables

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

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.

4.72.5.4 Asynchronous Events

Default.

4.72.6 External Effects

Any external effects are a result of the invocation of the utility *utility* , in a manner specified by that utility.

4.72.6.1 Standard Output

None.

4.72.6.2 Standard Error

Used for diagnostic messages and the `-t` option. If the `-t` option is specified, the *utility* and its constructed argument list shall be written to standard error, as it will be invoked, prior to invocation.