-- --

IEEE P1003.2 Draft 11.2 – September 1991

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.

4.72.6.3 Output Files

None.

4.72.7 Extended Description

None.

4.72.8 Exit Status

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

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

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. 1

126 The utility specified by *utility* was found but could not be invoked. 1

127 The utility specified by *utility* could not be found. 1

4.72.9 Consequences of Errors

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

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

Examples, Usage

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

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

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 1
utility’’ from ‘‘invoked utility exited with an error indication.’’ The value 127 was 1
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 con-
fused with termination due to receipt of a signal. The value 126 was chosen in a 1
similar manner to indicate that the utility could be found, but not invoked. Some 1
scripts produce meaningful error messages differentiating the 126 and 127 cases. 1
The distinction between exit codes 126 and 127 is based on KornShell practice 2
that uses 127 when all attempts to *exec* the utility fail with [ENOENT], and uses 2
126 when any attempt to *exec* the utility fails for any other reason. 2

Although the 255 exit status is mostly an accident of historical implementations, 1
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 1
status from a utility will fall into the 1−125 range when `xargs` exits. There is no 1
statement of how the various nonzero utility exit status codes are accumulated by 1
`xargs`. The value could be the addition of all codes, their highest value, the last 1
one received, or a single value such as 1. Since no algorithm is arguably better 1
than the others, and since many of the POSIX.2 standard utilities say little more 1

(portably) than ‘‘pass/fail,’’ no new algorithm was invented. 1

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.

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.