Full credit is given to the above companies including the OS that this PDF file was generated!

Red Hat Enterprise Linux Release 9.2 Manual Pages on 'xargs.1p' command

$ man xargs.1p

XARGS(1P) POSIX Programmer's Manual XARGS(1P)

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

xargs ? construct argument lists and invoke utility

SYNOPSIS

xargs [-ptx] [-E eofstr] [-I replstr|-L number|-n number] [-s size] [utility [argument...]]

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 fit in length and number con? straints 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 one of the following occurs:

* An end-of-file condition is detected on standard input.
* An argument consisting of just the logical end-of-file string (see the -E eofstr option) is found on standard input after double-quote processing, <apostrophe> processing, and <backslash>-escape pro? cessing (see next paragraph). All arguments up to but not including

the argument consisting of just the logical end-of-file string shall be used as arguments in constructed command lines.

* An invocation of a constructed command line returns an exit status of 255.

The application shall ensure that arguments in the standard input are separated by unquoted <blank> characters, unescaped <blank> characters, or <newline> characters. A string of zero or more non-double-quote ("") characters and non-<newline> characters can be quoted by enclos? ing them in double-quotes. A string of zero or more non-<apostrophe> ('\'') characters and non-<newline> characters can be quoted by enclos? ing them in <apostrophe> characters. Any unquoted character can be es? caped by preceding it with a <backslash>. The utility named by utility shall be executed one or more times until the end-of-file is reached or the logical end-of file string is found. 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 in? voked, the combined argument and environment lists (see the exec family of functions in the System Interfaces volume of POSIX.1?2017) 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}.

OPTIONS

The xargs utility shall conform to the Base Definitions volume of POSIX.1?2017, Section 12.2, Utility Syntax Guidelines.

The following options shall be supported:

-E eofstr Use eofstr as the logical end-of-file string. If -E is not specified, it is unspecified whether the logical end-of-file string is the <underscore> character ('_') or the end-of-file string capability is disabled. When eofstr is the null string, the logical end-of-file string capability shall be

disabled and <underscore> characters shall be taken liter?
ally.

-I replstr

Insert mode: utility is executed for each logical line from standard input. Arguments in the standard input shall be sep? arated only by unescaped <newline> characters, not by <blank> characters. Any unquoted unescaped <blank> characters at the beginning of each line shall be ignored. The resulting argu? ment shall be inserted in arguments in place of each occur? rence of replstr. At least five arguments in arguments can each contain one or more instances of replstr. Each of these constructed arguments cannot grow larger than an implementa? tion-defined limit greater than or equal to 255 bytes. Option -x shall be forced on.

-L number The utility shall be executed for each non-empty number lines of arguments from standard input. The last invocation of utility shall be with fewer lines of arguments if fewer than number remain. A line is considered to end with the first <newline> unless the last character of the line is an un? escaped <blank>; a trailing unescaped <blank> signals contin? uation to the next non-empty line, inclusive.

-n number Invoke utility using as many standard input arguments as pos? sible, 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).
* The last iteration has fewer than number, but not zero, operands remaining.

-p Prompt mode: the user is asked whether to execute utility at each invocation. Trace mode (-t) is turned on to write the command instance to be executed, followed by a prompt to standard error. An affirmative response read from /dev/tty

shall execute the command; otherwise, that particular invoca?
tion of utility shall be skipped.

-s size Invoke utility using as many standard input arguments as pos?
sible yielding a command line length less than size (a posi?
tive decimal integer) bytes. Fewer arguments shall be used
if:

* The total number of arguments exceeds that specified by the -n option.
* The total number of lines exceeds that specified by the -L option.
* End-of-file is encountered on standard input before size bytes are accumulated.

Values of size up to at least {LINE_MAX} bytes shall be sup?
ported, provided that the constraints specified in the DE?
SCRIPTION are met. It shall not be considered an error if a
value larger than that supported by the implementation or ex?
ceeding the constraints specified in the DESCRIPTION is
given; xargs shall use the largest value it supports within
the constraints.

-t Enable trace mode. Each generated command line shall be writ?
ten to standard error just prior to invocation.

-x Terminate if a constructed command line will not fit in the
implied or specified size (see the -s option above).

OPERANDS

The following operands shall be supported:

utility The name of the utility to be invoked, found by search path
using the PATH environment variable, described in the Base
Definitions volume of POSIX.1?2017, Chapter 8, Environment
Variables. If utility is omitted, the default shall be the
echo utility. If the utility operand names any of the special
built-in utilities in Section 2.14, Special Built-In Utili?
ties, the results are undefined.

argument An initial option or operand for the invocation of utility.

STDIN

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>.

INPUT FILES

The file /dev/tty shall be used to read responses required by the -p option.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of xargs:

LANG Provide a default value for the internationalization vari? ables that are unset or null. (See the Base Definitions vol? ume of POSIX.1?2017, Section 8.2, Internationalization Vari? ables for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_COLLATE

Determine the locale for the behavior of ranges, equivalence classes, and multi-character collating elements used in the extended regular expression defined for the yesexpr locale keyword in the LC_MESSAGES category.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files) and the behavior of character classes used in the ex? tended regular expression defined for the yesexpr locale key? word in the LC_MESSAGES category.

LC_MESSAGES

Determine the locale used to process affirmative responses, and the locale used to affect the format and contents of di? agnostic messages and prompts written to standard error.

NLSPATH Determine the location of message catalogs for the processing

of LC_MESSAGES.

PATH Determine the location of utility, as described in the Base Definitions volume of POSIX.1?2017, Chapter 8, Environment Variables.

ASYNCHRONOUS EVENTS

Default.

STDOUT

Not used.

STDERR

The standard error shall be used for diagnostic messages and the -t and -p options. If the -t option is specified, the utility and its con? structed argument list shall be written to standard error, as it will be invoked, prior to invocation. If -p is specified, a prompt of the following format shall be written (in the POSIX locale):

"?..."

at the end of the line of the output from -t.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values shall be returned:

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 non-zero exit status, or some other error occurred.

126 The utility specified by utility was found but could not be in? voked.

127 The utility specified by utility could not be found.

CONSEQUENCES OF ERRORS

If a command line meeting the specified requirements cannot be assem? bled, 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.

The following sections are informative.

APPLICATION USAGE

The 255 exit status 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. Thus, utility should explicitly exit with an ap? propriate value to avoid accidentally returning with 255.

Note that since input is parsed as lines, <blank> characters separate arguments, and <backslash>, <apostrophe>, and double-quote characters are used for quoting, if xargs is used to bundle the output of commands like find dir -print or ls into commands to be executed, unexpected re? sults are likely if any filenames contain <blank>, <newline>, or quot? ing characters. This can be solved by using find to call a script that converts each file found into a quoted string that is then piped to xargs, but in most cases it is preferable just to have find do the ar? gument aggregation itself by using -exec with a '+' terminator instead of ';'. Note that the quoting rules used by xargs are not the same as in the shell. They were not made consistent here because existing ap? plications depend on the current rules. An easy (but inefficient) method that can be used to transform input consisting of one argument per line into a quoted form that xargs interprets correctly is to pre? cede each non-<newline> character with a <backslash>. More efficient alternatives are shown in Example 2 and Example 5 below.

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.

The command, env, nice, nohup, time, 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 val? ues 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 differ? entiating 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.

EXAMPLES

1. The following command combines the output of the parenthesized com? mands (minus the <apostrophe> characters) onto one line, which is then appended to the file log. It assumes that the expansion of "0*" does not include any <apostrophe> or <newline> characters.

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

2. The following command invokes diff with successive pairs of argu? ments originally typed as command line arguments. It assumes there are no embedded <newline> characters in the elements of the origi? nal argument list.

   ```
   printf "%s\n$@" | sed 's/[^[:alnum:]]/\\&/g' |
       xargs -E "" -n 2 -x diff
   ```

3. In the following commands, the user is asked which files in the current directory (excluding dotfiles) are to be archived. The files are archived into arch; a, one at a time or b, many at a time. The commands assume that no filenames contain <blank>, <new? line>, <backslash>, <apostrophe>, or double-quote characters.

   ```
   a. ls | xargs -E "" -p -L 1 ar -r arch
   b. ls | xargs -E "" -p -L 1 | xargs -E "" ar -r arch
   ```

4. The following command invokes command1 one or more times with mul? tiple arguments, stopping if an invocation of command1 has a non- zero exit status.

   ```
   xargs -E "" sh -c 'command1 "$@" || exit 255' sh < xargs_input
   ```

5. On XSI-conformant systems, the following command moves all files

from directory $1 to directory $2, and echoes each move command just before doing it. It assumes no filenames contain <newline> characters and that neither $1 nor $2 contains the sequence "{}".

```
ls -A "$1" | sed -e 's/"/"\\""/g' -e 's/.*/"&"/' |
    xargs -E "" -I {} -t mv "$1"/{} "$2"/{}
```

RATIONALE

The xargs utility was usually found only in System V-based systems; BSD systems included an apply utility that provided functionality similar to xargs -n number. The SVID lists xargs as a software development ex? tension. This volume of POSIX.1?2017 does not share the view that it is used only for development, and therefore it is not optional.

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, this volume of POSIX.1?2017 selected only the mini? mal features required.

Although the 255 exit status is mostly an accident of historical imple? mentations, it allows a utility being used by xargs to tell xargs to terminate if it knows no further invocations using the current data stream shall succeed. Any non-zero exit status from a utility falls into the 1?125 range when xargs exits. There is no statement of how the various non-zero 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 standard utili? ties say little more (portably) than ``pass/fail'', no new algorithm was invented.

Several other xargs options were removed because simple alternatives already exist within this volume of POSIX.1?2017. For example, the -i replstr option can be just as efficiently performed using a shell for loop. Since xargs calls an exec function with each input line, the -i option does not usually exploit the grouping capabilities of xargs.

The requirement that xargs never produces command lines such that invo? cation of utility is within 2048 bytes of hitting the POSIX exec {ARG_MAX} limitations is intended to guarantee that the invoked utility has 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 the System Interfaces volume of POSIX.1?2017 is 4096 bytes and the minimum value allowed by this volume of POSIX.1?2017 is 2048 bytes; therefore, the 2048 bytes 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 this volume of POSIX.1?2017 is re? quired to wait for the completion of the invoked command before invok? ing another command. This was done because historical 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 termina? tion of all of the children before xargs terminates normally.

The -e option was omitted from the ISO POSIX?2:1993 standard in the be? lief that the eofstr option-argument was recognized only when it was on a line by itself and before quote and escape processing were performed, and that the logical end-of-file processing was only enabled if a -e option was specified. In that case, a simple sed script could be used to duplicate the -e functionality. Further investigation revealed that:

* The logical end-of-file string was checked for after quote and es? cape processing, making a sed script that provided equivalent func? tionality much more difficult to write.
* The default was to perform logical end-of-file processing with an <underscore> as the logical end-of-file string.

To correct this misunderstanding, the -E eofstr option was adopted from the X/Open Portability Guide. Users should note that the description of the -E option matches historical documentation of the -e option (which was not adopted because it did not support the Utility Syntax Guide? lines), by saying that if eofstr is the null string, logical end-of-

file processing is disabled. Historical implementations of xargs actu? ally did not disable logical end-of-file processing; they treated a null argument found in the input as a logical end-of-file string. (A null string argument could be generated using single or double-quotes (" or ""). Since this behavior was not documented historically, it is considered to be a bug.

The -I, -L, and -n options are mutually-exclusive. Some implementations use the last one specified if more than one is given on a command line; other implementations treat combinations of the options in different ways.

FUTURE DIRECTIONS

None.

SEE ALSO

Chapter 2, Shell Command Language, diff, echo, find

The Base Definitions volume of POSIX.1?2017, Chapter 8, Environment Variables, Section 12.2, Utility Syntax Guidelines

The System Interfaces volume of POSIX.1?2017, exec

COPYRIGHT