NAME

xargs − build and execute command lines from standard input

SYNOPSIS

xargs [*options*] [*command* [*initial-arguments*]]

DESCRIPTION

This manual page documents the GNU version of **xargs**. **xargs** reads items from the standard input, delimited by blanks (which can be protected with double or single quotes or a backslash) or newlines, and executes the *command* (default is */bin/echo*) one or more times with any *initial-arguments* followed by items read from standard input. Blank lines on the standard input are ignored.

The command line for *command* is built up until it reaches a system-defined limit (unless the **−n** and **−L** options are used). The specified *command* will be invoked as many times as necessary to use up the list of input items. In general, there will be many fewer invocations of *command* than there were items in the input. This will normally have significant performance benefits. Some commands can usefully be executed in parallel too; see the **−P** option.

Because Unix filenames can contain blanks and newlines, this default behaviour is often problematic; filenames containing blanks and/or newlines are incorrectly processed by **xargs**. In these situations it is better to use the **−0** option, which prevents such problems. When using this option you will need to ensure that the program which produces the input for **xargs** also uses a null character as a separator. If that program is GNU **find** for example, the **−print0** option does this for you.

If any invocation of the command exits with a status of 255, **xargs** will stop immediately without reading any further input. An error message is issued on stderr when this happens.

OPTIONS

−0, −−null

Input items are terminated by a null character instead of by whitespace, and the quotes and backslash are not special (every character is taken literally). Disables the end of file string, which is treated like any other argument. Useful when input items might contain white space, quote marks, or backslashes. The GNU find −print0 option produces input suitable for this mode.

−a *file*, **−−arg−file=***file*

Read items from *file* instead of standard input. If you use this option, stdin remains unchanged when commands are run. Otherwise, stdin is redirected from */dev/null*.

−−delimiter=*delim*, **−d** *delim*

Input items are terminated by the specified character. The specified delimiter may be a single character, a C-style character escape such as **\n**, or an octal or hexadecimal escape code. Octal and hexadecimal escape codes are understood as for the **printf** command. Multibyte characters are not supported. When processing the input, quotes and backslash are not special; every character in the input is taken literally. The **−d** option disables any end-of-file string, which is treated like any other argument. You can use this option when the input consists of simply newline-separated items, although it is almost always better to design your program to use **−−null** where this is possible.

−E *eof-str*

Set the end of file string to *eof-str*. If the end of file string occurs as a line of input, the rest of the input is ignored. If neither **−E** nor **−e** is used, no end of file string is used.

−e[*eof-str*], **−−eof**[=*eof-str*]

This option is a synonym for the **−E** option. Use **−E** instead, because it is POSIX compliant while this option is not. If *eof-str* is omitted, there is no end of file string. If neither **−E** nor **−e** is used, no end of file string is used.

−I *replace-str*
Replace occurrences of *replace-str* in the initial-arguments with names read from standard input. Also, unquoted blanks do not terminate input items; instead the separator is the newline character. Implies **−x** and **−L** 1.

−i[*replace-str*], **−−replace**[=*replace-str*]
This option is a synonym for **−I***replace-str* if *replace-str* is specified. If the *replace-str* argument is missing, the effect is the same as **−I**{}. This option is deprecated; use **−I** instead.

−L *max-lines*
Use at most *max-lines* nonblank input lines per command line. Trailing blanks cause an input line to be logically continued on the next input line. Implies **−x**.

−l[*max-lines*], **−−max-lines**[=*max-lines*]
Synonym for the **−L** option. Unlike **−L**, the *max-lines* argument is optional. If *max-lines* is not specified, it defaults to one. The **−l** option is deprecated since the POSIX standard specifies **−L** instead.

−n *max-args*, **−−max−args**=*max-args*
Use at most *max-args* arguments per command line. Fewer than *max-args* arguments will be used if the size (see the **−s** option) is exceeded, unless the **−x** option is given, in which case **xargs will exit.**

−P *max-procs*, **−−max−procs**=*max-procs*
Run up to *max-procs* processes at a time; the default is 1. If *max-procs* is 0, **xargs** will run as many processes as possible at a time. Use the **−n** option or the **−L** option with **−P**; otherwise chances are that only one exec will be done. While **xargs** is running, you can send its process a SIGUSR1 signal to increase the number of commands to run simultaneously, or a SIGUSR2 to decrease the number. You cannot increase it above an implementation-defined limit (which is shown with −−show-limits). You cannot decrease it below 1. **xargs** never terminates its commands; when asked to decrease, it merely waits for more than one existing command to terminate before starting another.

Please note that it is up to the called processes to properly manage parallel access to shared resources. For example, if more than one of them tries to print to stdout, the output will be produced in an indeterminate order (and very likely mixed up) unless the processes collaborate in some way to prevent this. Using some kind of locking scheme is one way to prevent such problems. In general, using a locking scheme will help ensure correct output but reduce performance. If you don't want to tolerate the performance difference, simply arrange for each process to produce a separate output file (or otherwise use separate resources).

−o, −−open−tty
Reopen stdin as */dev/tty* in the child process before executing the command. This is useful if you want **xargs** to run an interactive application.

−p, −−interactive
Prompt the user about whether to run each command line and read a line from the terminal. Only run the command line if the response starts with 'y' or 'Y'. Implies **-t**.

−−process−slot−var=*name*
Set the environment variable *name* to a unique value in each running child process. Values are reused once child processes exit. This can be used in a rudimentary load distribution scheme, for example.

−r, −−no−run−if−empty
If the standard input does not contain any nonblanks, do not run the command. Normally, the command is run once even if there is no input. This option is a GNU extension.

-s *max-chars*, **−−max−chars**=*max-chars*
Use at most *max-chars* characters per command line, including the command and initial-arguments and the terminating nulls at the ends of the argument strings. The largest allowed value is system-dependent, and is calculated as the argument length limit for exec, less the size of your environment, less 2048 bytes of headroom. If this value is more than 128KiB, 128Kib is used as the default value; otherwise, the default value is the maximum. 1KiB is 1024 bytes. **xargs** automatically adapts to tighter constraints.

−−show−limits
Display the limits on the command-line length which are imposed by the operating system, **xargs**' choice of buffer size and the **−s** option. Pipe the input from */dev/null* (and perhaps specify **--no-run-if-empty**) if you don't want **xargs** to do anything.

−t, −−verbose
Print the command line on the standard error output before executing it.

−x, −−exit
Exit if the size (see the **−s** option) is exceeded.

−−help Print a summary of the options to **xargs** and exit.

−−version
Print the version number of **xargs** and exit.

The options **−−max-lines** (**−L**, **−l**), **−−replace** (**−I**, **−i**) and **−−max-args** (**−n**) are mutually exclusive. If some of them are specified at the same time, then **xargs** will generally use the option specified last on the command line, i.e., it will reset the value of the offending option (given before) to its default value. Additionally, **xargs** will issue a warning diagnostic on *stderr*. The exception to this rule is that the special *max-args* value *1* ('**−n***1*') is ignored after the **−−replace** option and its aliases **−I** and **−i**, because it would not actually conflict.

EXAMPLES

find /tmp −name core −type f −print | xargs /bin/rm −f

Find files named **core** in or below the directory **/tmp** and delete them. Note that this will work incorrectly if there are any filenames containing newlines or spaces.

find /tmp −name core −type f −print0 | xargs −0 /bin/rm −f

Find files named **core** in or below the directory **/tmp** and delete them, processing filenames in such a way that file or directory names containing spaces or newlines are correctly handled.

find /tmp −depth −name core −type f −delete

Find files named **core** in or below the directory **/tmp** and delete them, but more efficiently than in the previous example (because we avoid the need to use **fork**(2) and **exec**(2) to launch **rm** and we don't need the extra **xargs** process).

cut −d: −f1 < /etc/passwd | sort | xargs echo

Generates a compact listing of all the users on the system.

EXIT STATUS

xargs exits with the following status:

0 if it succeeds

123 if any invocation of the command exited with status 1-125

124	if the command exited with status 255
125	if the command is killed by a signal
126	if the command cannot be run
127	if the command is not found
1	if some other error occurred.

Exit codes greater than 128 are used by the shell to indicate that a program died due to a fatal signal.

STANDARDS CONFORMANCE

As of GNU xargs version 4.2.9, the default behaviour of **xargs** is not to have a logical end-of-file marker. POSIX (IEEE Std 1003.1, 2004 Edition) allows this.

The −l and −i options appear in the 1997 version of the POSIX standard, but do not appear in the 2004 version of the standard. Therefore you should use −L and −I instead, respectively.

The −o option is an extension to the POSIX standard for better compatibility with BSD.

The POSIX standard allows implementations to have a limit on the size of arguments to the **exec** functions. This limit could be as low as 4096 bytes including the size of the environment. For scripts to be portable, they must not rely on a larger value. However, I know of no implementation whose actual limit is that small. The **−−show−limits** option can be used to discover the actual limits in force on the current system.

BUGS

It is not possible for **xargs** to be used securely, since there will always be a time gap between the production of the list of input files and their use in the commands that **xargs** issues. If other users have access to the system, they can manipulate the filesystem during this time window to force the action of the commands **xargs** runs to apply to files that you didn't intend. For a more detailed discussion of this and related problems, please refer to the ''Security Considerations'' chapter in the findutils Texinfo documentation. The **−execdir** option of **find** can often be used as a more secure alternative.

When you use the **−I** option, each line read from the input is buffered internally. This means that there is an upper limit on the length of input line that **xargs** will accept when used with the **−I** option. To work around this limitation, you can use the **−s** option to increase the amount of buffer space that **xargs** uses, and you can also use an extra invocation of **xargs** to ensure that very long lines do not occur. For example:

somecommand | xargs −s 50000 echo | xargs −I '{}' −s 100000 rm '{}'

Here, the first invocation of **xargs** has no input line length limit because it doesn't use the **−i** option. The second invocation of **xargs** does have such a limit, but we have ensured that it never encounters a line which is longer than it can handle. This is not an ideal solution. Instead, the **−i** option should not impose a line length limit, which is why this discussion appears in the BUGS section. The problem doesn't occur with the output of **find**(1) because it emits just one filename per line.

REPORTING BUGS

GNU findutils online help: <https://www.gnu.org/software/findutils/#get-help>
Report any translation bugs to <https://translationproject.org/team/>

Report any other issue via the form at the GNU Savannah bug tracker:
<https://savannah.gnu.org/bugs/?group=findutils>
General topics about the GNU findutils package are discussed at the *bug−findutils* mailing list:
<https://lists.gnu.org/mailman/listinfo/bug-findutils>

COPYRIGHT

SEE ALSO

find(1), **kill**(1), **locate**(1), **updatedb**(1), **fork**(2), **execvp**(3), **locatedb**(5), **signal**(7)

Full documentation <https://www.gnu.org/software/findutils/xargs>
or available locally via: **info xargs**