The *xargs* utility constructs an argument list for a Unix command using standard input. Learn to use *xargs* with this example-laden tutorial.

Description

```
xargs [options] [command]
```

The *xargs* command creates an argument list for *command* from standard input. It is typically used with a pipe. For example,

```
$ find ~ -name 'proj1*' print | xargs cat
```

The *find* command searches the entire home directory structure for filenames that begin with *proj1*. The *xargs* command bundles the filenames output by *find* into an argument list for the *cat* command which prints them to the screen.

In many Unix shells there is a limit to the number of arguments allowed on a single command line. If the argument list read by *xargs* is larger than the maximum allowed by the shell, *xargs* will bundle the arguments into smaller groups and execute *command* separately for each argument bundle. Depending on the options used with *xargs*, arguments may be processed in smaller bundles (for example, one at a time).

If no *command* is specified, *xargs* works similar to the *echo* command and prints the argument list to standard output.

Options

Option	Description
-n#	Execute *command* once for every # argument. For example, -n2 bundles arguments into groups of two or less and executes *command* on each argument bundle.
-l#	Execute command once for every # lines of input. For example, -l1 creates a bundle of arguments for every one line of input and executes *command* on each argument bundle.
-i	Normally *xargs* places input arguments at the end of *command*. Used with the -i option, *xargs* will replace all instances of {} with input arguments. On most systems you must place a backslash (\) before each bracket to keep the shell from interpreting the special characters.
-t	Echo each *command* before executing.

-p	Prompts the user before executing each *command*.

Note: Not all Unix flavors support the above *xargs* options. Some flavors support more options than those shown above. Check your local man pages.

Examples

Xargs Basics

1. *Xargs* can be used to read the argument list for a command from standard input. Often arguments are lists of filenames passed to *xargs* via a pipe. For example,

   ```
   $ ls f*
   f1 f2 f3
   ```

   There are three files in the current directory that begin with the letter *f*. The following example

   ```
   $ ls f* | xargs cat
   contents of f1 ...
   contents of f2 ...
   contents of f3 ...
   ```

   prints the contents of each file to the screen. The *xargs* command takes output from *ls*, "*f1 f2 f3*", and uses it as arguments to the *cat* command, creating the command "*cat f1 f2 f3*".

   Notice that using the *xargs* command is different than piping the output of *ls* directly to *cat*. For example,

   ```
   $ ls f* | cat
   f1
   f2
   f3
   ```

   Used without arguments, *cat* reads standard input (in this case the filenames from *ls*) and prints the results to the screen.

2. Arguments read from standard input can follow options or other arguments. For example,

   ```
   $ ls f* | xargs grep -i 'to do' gfile
   gfile: To do by Wed: fix printer
   f2: I need to do my homework.
   ```

   The *xargs* command combines "*grep -i 'to do' gfile*" with the output from *ls*, creating the command "*grep -i 'to do' gfile f1 f2 f3*". The *grep* option *-i* and

argument *gfile* are typed at the command line but the arguments *f1*, *f2* and *f3* are read from standard input. (See Power Commands: Grep for more information on the *grep* command.)

3. `$ find ~ -name 'proj1*' print | xargs cat > proj1.all`

   The *find* command searches the entire home directory structure for filenames that begin with *proj1*. The *xargs* command groups all the filenames into an argument list for the *cat* command. The output from the *cat* command is saved in the file *proj1.all* using output redirection. (See Power Commands: Find for more information on the *find* command.)

Xargs vs. Command Substitution - Using _Xargs_ to Process Long Command Lines

1. Command substitution allows you to use the output from one Unix command as an argument to another Unix command. When part of a command is surrounded by backquotes, the shell will evaluate this text as a separate command then insert the output into the syntax of the original command. For example, the following command

   ```
   $ grep 'africa' `find ~ -type f -print`
   ```

   uses command substitution to search all regular files in the users home directory for the string *africa*. (See Power Commands: Grep and Power Commands: Find for more information on the *grep* and *find* commands.)

2. *Xargs* performs a similar function as command substitution. The command

   ```
   $ find ~ -type f -print | xargs grep 'africa'
   ```

   does the same search as the command in example one.

3. In some cases command substitution will create a command line too long for the Unix system. For example, try to search every document on the root directory for the string *OhNo*.

   ```
   $ grep 'OhNo' `find / -type f -print`
   grep: too many arguments
   ```

   Grep returns an error and does not finish the search. Note that the total number of arguments allowed on a command line varies between Unix shells.

4. *Xargs* will pass arguments in batches which are small enough not to exceed the maximum number of allowed by the system. For example, unlike the above example, the following command

```
$ find / -type f -print | xargs grep 'OhNo'
```

will not return an error. The *xargs* command allows *grep* to process more arguments than it could normally handle.

Xargs Echoing Feature

1. Used without a command, *xargs* functions similar to *echo*. It bundles the input lines and prints them to standard output. For example,

   ```
   $ cat f1
   line 1 of f1
   line 2 of f1
   line 3 of f1
   ```

   Now try

   ```
   $ cat f1 | xargs
   line 1 of f1 line 2 of f1 line 3 of f1
   ```

   Notice that *xargs* has grouped separate lines together. If *f1* were a particularly long file then *xargs* would create more than one bundle of output. For example,

   ```
   $ wc -l bigfl
   4012
   ```

   the file, *bigfl*, has 4012 lines. (See the *wc* command glossary entry for more information on the word count command.) Now try

   ```
   $ cat bigfl | xargs > xbigfl
   $ wc -l xbigfl
   8
   ```

   The output from *xargs* is saved in the file *xbigfl* which has eight lines. *Xargs* bundled the output into groups small enough for the shell to process without error. In this case eight bundles.

2. The echoing feature of *xargs* is particularly useful when combining the output from multiple commands. For example,

   ```
   $ date +%D
   08/15/01
   ```

   prints today's date (See the *Power Commands: Date* article for more information.) and

   ```
   $ du -s ~
   1934  /home/lizr
   ```

prints the total amount of disk space used by the home directory. (See Determining Disk Usage for more information.) Now

```
$ date +%D ; du -s ~
08/15/01
1934  /home/lizr
```

executes both command on one line. (See Entering Multiple Commands on a Single Line for more information.)

The following command uses a pipe and *xargs* to append the output from both commands on one line in the file *log*.

```
$ ( date +%D ; du -s ~ ) | xargs >> log
$ cat log
...
08/15/01  1934  /home/lizr
```

Run a Command Every *N* Words or Lines of Input

1. The *-n#* option with *xargs* executes a command with up to # arguments. For example,

   ```
   $ ls | xargs -n1
   f1
   f2
   f3
   f4
   ```

   using the *-n1* option, *xargs* processes only one argument at a time, while

   ```
   $ ls | xargs -n3
   f1 f2 f3
   f4
   ```

   using the *-n3* option, *xargs* bundles up to three arguments at a time.

2. Displays the contents of a file one word per line.

   ```
   $ cat filename | xargs -n1
   ```

3. The *-l#* option with *xargs* executes a command every # lines of input. For example,

   ```
   $ cat f1
   line 1
   line 2
   line 3
   line 4
   ```

The file *f1* has four lines.

```
$ cat f1 | xargs
line 1 line 2 line 3 line 4
```

With no options *xargs* bundles lines of input into the longest arguments list that the shell can process. In this case all lines are bundled into one argument list. Use *-l2* to bundle every two lines of input together.

```
$ cat f1 | xargs -l2
line 1 line 2
line 3 line 4
```

Position Standard Input Arguments Among Other Arguments

1. Typically *xargs* places input arguments at the end of a command. Used with the *-i* option, *xargs* will replace all instances of {} with input arguments. On most systems you must place a backslash before each bracket to keep the shell from interpreting the special characters. For example, the following command moves all files in *dir1* into the directory *dir2*.

   ```
   $ ls dir1 | xargs -i mv dir1/\{\} dir2/\{\}
   ```

2. In the current directory there are three files whose filenames end with *.ascii*.

   ```
   $ ls *.ascii
   f1.ascii f2.ascii f3.ascii
   ```

   The following example renames all files whose filenames end with *.ascii* so that their filenames end with *.txt*.

   ```
   $ ls *.ascii | xargs -i basename \{\} .ascii \
   | xargs -i mv \{\}.ascii \{\}.txt
   ```

   (Note that the backslash (\) after *ascii* is used to spread a single command across two lines.)

   How does this command work? The *basename* command prints a filename minus the extension. For example,

   ```
   $ basename f1.ascii .ascii
   f1
   ```

   So

   ```
   $ ls *.ascii | xargs -i basename \{\} .ascii
   f1
   ```

```
f2
f3
```

prints each filename without the *.ascii* extension. This output is then sent to the second *xargs* command which creates the commands

```
mv f1.ascii f1.txt
mv f2.ascii f2.txt
mv f3.ascii f3.txt
```

Print or Query Before Executing Commands

1. Used with the *-t* option, *xargs* echoes each command before executing. For example, the following command moves all files in *dir1* into the directory *dir2*.

```
$ ls dir1 | xargs -i -t mv dir1/\{\} dir2/\{\}
mv dir1/f1 dir2/f1
mv dir1/f2 dir2/f2
mv dir1/f3 dir2/f3
```

2. Used with the *-p* option, *xargs* prompts the user before executing each command. For example,

```
$ ls dir1 | xargs -i -p mv dir1/\{\} dir2/\{\}
mv dir1/f1 dir2/f1 ?...y
mv dir1/f2 dir2/f2 ?...n
mv dir1/f3 dir2/f3 ?...y
```

   Files *f1* and *f3* were moved but file *f2* was not.

3. Use the query (-p) option, to choose which files in the current directory should be *compressed*.

```
$ ls | xargs -n1 -p compress
compress largef1 ?...y
compress largef2 ?...y
compress smallf1 ?...n
compress smallf2 ?...n
```