The preceding command prints the name (path) of all the files that are not from the `.git` directories.

Here, `\( -name ".git" -prune \)` is the exclude portion, which specifies that the `.git` directory should be excluded and `\( -type f -print \)` specifies the action to be performed. The actions to be performed are placed in the second block `-type f -print` (the action specified here is to print the names and path of all the files).

Playing with xargs

We use pipes to redirect `stdout` (standard output) of a command to `stdin` (standard input) of another command. For example:

```
cat foo.txt | grep "test"
```

Some of the commands accept data as command-line arguments rather than a data stream through `stdin` (standard input). In that case, we cannot use pipes to supply data through command-line arguments.

We should try alternate methods. `xargs` is a command that is very helpful in handling standard input data to the command-line argument conversions. It can manipulate `stdin` and convert to command-line arguments for the specified command. Also, `xargs` can convert any one-line or multiple-line text inputs into other formats, such as multiple lines (specified number of columns) or a single line and vice versa.

All Bash users love one-liner commands, which are command sequences that are joined by using the pipe operator, but do not use the semicolon terminator (`;`) between the commands used. Crafting one-line commands makes tasks more efficient and simpler to solve. It requires proper understanding and practice to formulate one-liners for solving text processing problems. `xargs` is one of the important components for building one-liner commands.

Getting ready

When using the pipe operator, the `xargs` command should always be the first thing to appear after the operator. `xargs` uses standard input as the primary data stream source. It uses `stdin` and executes another command by providing command-line arguments for that executing command using the `stdin` data source. For example:

```
command | xargs
```

How to do it...

The `xargs` command can supply arguments to a command by reformatting the data received through `stdin`.

`xargs` can act as a substitute that can perform similar actions as the `-exec` argument in the case of the `find` command. Let's see a variety of hacks that can be performed using the `xargs` command.

- **Converting multiple lines of input to a single-line output**: Multiple-line input can be converted simply by removing the newline character and replacing with the " " (space) character. '`\n`' is interpreted as a newline character, which is the delimiter for the lines. By using `xargs`, we can ignore all the newlines with space so that multiple lines can be converted into a single-line text as follows:

```
$ cat example.txt # Example file
1 2 3 4 5 6
7 8 9 10
11 12

$ cat example.txt | xargs
1 2 3 4 5 6 7 8 9 10 11 12
```

- **Converting single-line into multiple-line output**: Given a maximum number of arguments in a line `= n`, we can split any `stdin` (standard input) text into lines of `n` arguments each. An argument is a piece of a string delimited by " " (space). Space is the default delimiter. A single line can be split into multiple lines as follows:

```
$ cat example.txt | xargs -n 3
1 2 3
4 5 6
7 8 9
10 11 12
```

How it works...

The `xargs` command is appropriate to be applied to many problem scenarios with its many options. Let's see how these options can be used wisely to solve problems.

We can also use our own delimiter towards separating arguments. To specify a custom delimiter for input, use the `-d` option as follows:

```
$ echo "splitXsplitXsplitXsplit" | xargs -d X
split split split split
```

In the preceding code, `stdin` contains a string consisting of multiple `X` characters. We can use `X` as the input delimiter by using it with `-d`. Here, we have explicitly specified `X` as the input delimiter, whereas in the default case `xargs` takes the **internal field separator** (space) as the input delimiter.