Replacing a pattern with text in all the files in a directory

There will be numerous occasions when we will need to replace a particular text with a new text in every file in a directory. An example would be changing a common URI everywhere in a website's source directory. Using the shell for this is one of the quickest methods out there.

How to do it...

From what we have learnt up to now, we can first use `find` to locate the files we want to perform the text replacement on. Then, we can use `sed` to do the actual replacement.

Let's say we want to replace the text `Copyright` with the word `Copyleft` in all `.cpp` files:

```
$ find . -name *.cpp -print0 |  xargs -I{} -0 sed -i 's/Copyright/
Copyleft/g' {}
```

How it works...

We use `find` on the current directory to find all the files of `.cpp`, and use `print0` to print a null-separated list of files (recall that this helps, if the filenames have spaces in them). We then pipe this list to `xargs`, which will pass these files to `sed`, which in turn will make the modifications.

There's more...

If you recall, `find` has an option `-exec`, which can be used to run a command on each of the files that `find` will match. We can use this option to achieve the same effect or replace the text with a new one, as follows:

```
$ find . -name *.cpp -exec sed -i 's/Copyright/Copyleft/g' \{\} \;
```

Or:

```
$ find . -name *.cpp -exec sed -i 's/Copyright/Copyleft/g' \{\} \+
```

While they perform the same function, the first form will call `sed` once for every file that is found, while in the second form, `find` will combine multiple filenames and pass them together to `sed`.

Text slicing and parameter operations

This recipe walks through some of the simple text-replacement techniques and parameter-expansion shorthands available in Bash. A few simple techniques can often help us avoid having to write multiple lines of code.

How to do it...

Let's get into the tasks.

Replacing some text from a variable can be done as follows:

```
$ var="This is a line of text"
$ echo ${var/line/REPLACED}
This is a REPLACED of text"
```

`line` is replaced with `REPLACED`.

We can produce a substring by specifying the start position and string length, by using the following syntax:

```
${variable_name:start_position:length}
```

To print from the fifth character onwards, use the following command:

```
$ string=abcdefghijklmnopqrstuvwxyz
$ echo ${string:4}
efghijklmnopqrstuvwxyz
```

To print eight characters starting from the fifth character, use the following command:

```
$ echo ${string:4:8}
efghijkl
```

The index is specified by counting the start letter as `0`. We can also specify counting from the last letter as `-1`. It is used inside a parenthesis. `(-1)` is the index for the last letter:

```
echo ${string:(-1)}
z
$ echo ${string:(-2):2}
yz
```

See also

- The *Iterating through lines, words, and characters in a file* recipe in this chapter explains slicing of a character from a word