If we want to match either of the multiple criteria, we can use OR conditions as shown in the following:

```
$ ls
new.txt  some.jpg  text.pdf
$ find . \( -name "*.txt" -o -name "*.pdf" \) -print
./text.pdf
./new.txt
```

The previous command will print all of the `.txt` and `.pdf` files, since the `find` command matches both `.txt` and `.pdf` files. `\(` and `\)` are used to treat `-name "*.txt" -o -name "*.pdf"` as a single unit.

The `-path` argument can be used to match the file path for files that match the wildcards. `-name` always matches using the given filename. However, `-path` matches the file path as a whole. For example:

```
$ find /home/users -path "*/slynux/*" -print
This will match files as following paths.
/home/users/list/slynux.txt
/home/users/slynux/eg.css
```

The `-regex` argument is similar to `-path`, but `-regex` matches the file paths based on regular expressions.

Regular expressions are an advanced form of wildcard matching, which enables us to specify text with patterns. By using patterns, we can make matches to the text and print them. A typical example of text matching using regular expressions is: parsing all e-mail addresses from a given pool of text. An e-mail address takes the form `name@host.root`. So, it can be generalized as `[a-z0-9]+@[a-z0-9]+.[a-z0-9]+`. The `+` sign signifies that the previous class of characters can occur one or more times, repeatedly, in the characters that follow.

The following command matches the `.py` or `.sh` files:

```
$ ls
new.PY  next.jpg  test.py
$ find . -regex ".*\(\.py\|\.sh\)$"
./test.py
```

Similarly, using `-iregex` ignores the case for the regular expressions that are available. For example:

```
$ find . -iregex ".*\(\.py\|\.sh\)$"
./test.py
./new.PY
```

We will learn more about regular expressions in *Chapter 4*, *Texting and Driving*.

Negating arguments

`find` can also exclude things that match a pattern using `!`:

```
$ find . ! -name "*.txt" -print
```

This will match all the files whose names do not end in `.txt`. The following example shows the result of the command:

```
$ ls
list.txt  new.PY  new.txt  next.jpg  test.py

$ find . ! -name "*.txt" -print
.
./next.jpg
./test.py
./new.PY
```

Search based on the directory depth

When the `find` command is used, it recursively walks through all the subdirectories as much as possible, until it reaches the leaf of the subdirectory tree. We can restrict the depth to which the `find` command traverses using some depth parameters given to `find`. `-maxdepth` and `-mindepth` are the parameters.

In most of the cases, we need to search only in the current directory. It should not further descend into the subdirectories from the current directory. In such cases, we can restrict the depth to which the `find` command should descend using depth parameters. To restrict `find` from descending into the subdirectories from the current directory, the depth can be set as `1`. When we need to descend to two levels, the depth is set as `2`, and so on for the rest of the levels.