Note, that `some{string1,string2,string3}` expands as `somestring1 somestring2 somestring3`.

Exclude all `README` files in the search, as follows:

```
$ grep "main()" . -r --exclude "README"
```

To exclude directories, use the `--exclude-dir` option.

To read a list of files to exclude from a file, use `--exclude-from FILE`.

Using grep with xargs with zero-byte suffix

The `xargs` command is often used to provide a list of file names as a command-line argument to another command. When filenames are used as command-line arguments, it is recommended to use a zero-byte terminator for the filenames instead of a space terminator. Some of the filenames can contain a space character, and it will be misinterpreted as a terminator, and a single filename may be broken into two file names (for example, `New file.txt` can be interpreted as two filenames `New` and `file.txt`). This problem can be avoided by using a zero-byte suffix. We use `xargs` so as to accept a `stdin` text from commands such as `grep` and `find`. Such commands can output text to `stdout` with a zero-byte suffix. In order to specify that the input terminator for filenames is zero byte (`\0`), we should use `-0` with `xargs`.

Create some test files as follows:

```
$ echo "test" > file1
$ echo "cool" > file2
$ echo "test" > file3
```

In the following command sequence, `grep` outputs filenames with a zero-byte terminator (`\0`), because of the `-Z` option with `grep`. `xargs -0` reads the input and separates filenames with a zero-byte terminator:

```
$ grep "test" file* -lZ | xargs -0 rm
```

Usually, `-Z` is used along with `-l`.

Silent output for grep

Sometimes, instead of actually looking at the matched strings, we are only interested in whether there was a match or not. For this, we can use the quiet option (`-q`), where the `grep` command does not write any output to the standard output. Instead, it runs the command and returns an exit status based on success or failure.

We know that a command returns `0` on success, and non-zero on failure.

Let's go through a script that makes use of `grep` in a quiet mode, for testing whether a match text appears in a file or not.

```
#!/bin/bash
#Filename: silent_grep.sh
#Desc: Testing whether a file contain a text or not

if [ $# -ne 2 ]; then
  echo "Usage: $0 match_text filename"
  exit 1
fi

match_text=$1
filename=$2
grep -q "$match_text" $filename

if [ $? -eq 0 ]; then
  echo "The text exists in the file"
else
  echo "Text does not exist in the file"
fi
```

The `silent_grep.sh` script can be run as follows, by providing a match word (`Student`) and a file name (`student_data.txt`) as the command argument:

```
$ ./silent_grep.sh Student student_data.txt
The text exists in the file
```

Printing lines before and after text matches

Context-based printing is one of the nice features of `grep`. Suppose a matching line for a given match text is found, `grep` usually prints only the matching lines. But, we may need "n" lines after the matching line, or "n" lines before the matching line, or both. This can be performed by using context-line control in `grep`. Let's see how to do it.

In order to print three lines after a match, use the `-A` option:

```
$ seq 10 | grep 5 -A 3
5
6
7
8
```