```
comm duplicate_files duplicate_sample  -2 -3 | tee /dev/stderr |
xargs rm
echo Removed duplicates files successfully.
```

3. Run it as:

```
$ ./remove_duplicates.sh
```

How it works...

The preceding commands will find the copies of the same file in a directory and remove all except one copy of the file. Let us go through the code and see how it works.

`ls -lS` will list the details of the files sorted by file size in the current directory. `--time-style=long-iso` tells `ls` to print dates in the ISO format. `awk` will read the output of `ls -lS` and perform comparisons on columns and rows of the input text to find out the duplicate files.

The logic behind the code is as follows:

- We list the files sorted by size so that the similarly sized files will be grouped together. The files having the same file size are identified as a first step to finding files that are the same. Next, we calculate the checksum of the files. If the checksums match, the files are duplicates and one set of the duplicates are removed.
- The `BEGIN{}` block of `awk` is executed first before the lines are read from the file. Reading lines takes place in the `{}` block and after the end of reading and processing all lines, the `END{}` block statements are executed. The output of `ls -lS` is:

```
total 16
-rw-r--r-- 1 slynux slynux 5 2010-06-29 11:50 other
-rw-r--r-- 1 slynux slynux 6 2010-06-29 11:50 test
-rw-r--r-- 1 slynux slynux 6 2010-06-29 11:50 test_copy1
-rw-r--r-- 1 slynux slynux 6 2010-06-29 11:50 test_copy2
```

- The output of the first line tells us the total number of files, which in this case is not useful. We use `getline` to read the first line and then dump it. We need to compare each of the lines and the next line for sizes. For that, we read the first line explicitly using `getline` and store the name and size (which are the eighth and fifth columns). Hence, a line is read ahead using `getline`. Now, when `awk` enters the `{}` block (in which the rest of the lines are read), that block is executed for every read of a line. It compares the size obtained from the current line and the previously stored size kept in the `size` variable. If they are equal, it means two files are duplicates by size. Hence, they are to be further checked by `md5sum`.

We have played some tricks on the way to the solution.

The external command output can be read inside `awk` as:

```
"cmd"| getline
```

Then, we receive the output in line `$0` and each column output can be received in `$1`, `$2`, ... ,`$n`, and so on. Here, we read the md5sum checksum of files in the `csum1` and `csum2` variables. Variables `name1` and `name2` are used to store consecutive filenames. If the checksums of two files are the same, they are confirmed to be duplicates and are printed.

We need to find a file from each group of duplicates so that we can remove all other duplicates. We calculate the `md5sum` value of the duplicates and print one file from each group of duplicates by finding unique lines, comparing `md5sum` only from each line using `-w 32` (the first 32 characters in the `md5sum` output; usually, the `md5sum` output consists of a 32-character hash followed by the filename). Therefore, one sample from each group of duplicates is written in `duplicate_sample`.

Now, we need to remove all the files listed in `duplicate_files`, excluding the files listed in `duplicate_sample`. The `comm` command prints files in `duplicate_files` but not in `duplicate_sample`.

For that, we use a set difference operation (refer to the recipes on intersection, difference, and set difference).

`comm` always accepts files that are sorted. Therefore, `sort -u` is used as a filter before redirecting to `duplicate_files` and `duplicate_sample`.

Here the `tee` command is used to perform a trick so that it can pass filenames to the `rm` command as well as `print`. The `tee` command writes lines that appear as `stdin` to a file and sends them to `stdout`. We can also print text to the terminal by redirecting to `stderr`. `/dev/stderr` is the device corresponding to `stderr` (standard error). By redirecting to a `stderr` device file, text that appears through `stdin` will be printed in the terminal as standard error.

Working with file permissions, ownership, and the sticky bit

File permissions and ownership are one of the distinguishing features of the Unix/Linux filesystems such as **extfs** (**extended FS**). In many circumstances while working on Unix/Linux platforms, we come across issues related to permissions and ownership. This recipe is a walk through the different use cases of these.

In Linux systems, each file is associated with many types of permissions. Out of these permissions, three sets of permissions (user, group, and others) are commonly manipulated.