As an example usage case, we can consider the case of the Apache web server. The PHP files in the web server require proper permissions to execute. We can find out the PHP files that don't have proper execute permissions as follows:

```
$ find . -type f -name "*.php" ! -perm 644 -print
```

We can also search files based on ownership of the files. The files owned by a specific user can be found out using the `-user USER` option.

The `USER` argument can be a username or UID.

For example, to print the list of all files owned by the user `slynux`, you can use the following command:

```
$ find . -type f -user slynux -print
```

Executing commands or actions with find

The `find` command can be coupled with many of the other commands using the `-exec` option. It is one of the most powerful features that comes with `find`.

Consider the example in the previous section. We used `-perm` to find out the files that do not have proper permissions. Similarly, in the case where we need to change the ownership of all files owned by a certain user (for example, `root`) to another user (for example, `www-data`, the default Apache user in the web server), we can find all the files owned by root by using the `-user` option and using `-exec` to perform the ownership change operation.

> You must run the `find` command as root if you want to change ownership of files or directories.

Let's have a look at the following example:

```
# find . -type f -user root -exec chown slynux {} \;
```

In this command, `{}` is a special string used with the `-exec` option. For each file match, `{}` will be replaced with the filename for `-exec`. For example, if the `find` command finds two files `test1.txt` and `test2.txt` with owner `slynux`, the `find` command will perform:

```
chown slynux {}
```

This gets resolved to `chown slynux test1.txt` and `chown slynux test2.txt`.

> Sometimes we don't want to run the command for each file. Instead, we might want to run it a fewer times with a list of files as parameters. For this, we use `+` instead of `;` in the `exec` syntax.

Another usage example is to concatenate all the C program files in a given directory and write it to a single file, say, `all_c_files.txt`. We can use `find` to match all the C files recursively and use the `cat` command with the `-exec` flag as follows:

```
$ find . -type f -name "*.c" -exec cat {} \;>all_c_files.txt
```

`-exec` is followed by any command. `{}` is a match. For every matched filename, `{}` is replaced with the filename.

To redirect the data from `find` to the `all_c_files.txt` file, we have used the `>` operator instead of `>>` (append) because the entire output from the `find` command is a single data stream (`stdin`). `>>` is necessary only when multiple data streams are to be appended to a single file.

For example, to copy all the `.txt` files that are older than 10 days to a directory `OLD`, use the following command:

```
$ find . -type f -mtime +10 -name "*.txt" -exec cp {} OLD  \;
```

Similarly, the `find` command can be coupled with many other commands.

-exec with multiple commands

We cannot use multiple commands along with the `-exec` parameter. It accepts only a single command, but we can use a trick. Write multiple commands in a shell script (for example, `commands.sh`) and use it with `-exec` as follows:

```
-exec ./commands.sh {} \;
```

`-exec` can be coupled with `printf` to produce a very useful output. For example:

```
$ find . -type f -name "*.txt" -exec printf "Text file: %s\n" {} \;
```

Skipping specified directories when using the find command

Skipping certain subdirectories for performance improvement is sometimes required while doing a directory search and performing an action. For example, when programmers look for particular files on a development source tree, which is under the version control system such as Git, the source hierarchy will always contain the `.git` directory in each of the subdirectories (`.git` stores version-control-related information for every directory). Since version-control-related directories do not produce useful output, they should be excluded from the search. The technique of excluding files and directories from the search is known as **pruning**. It can be performed as follows:

```
$ find devel/source_path  \( -name ".git" -prune \) -o \( -type f -print
\)

# Instead of \( -type -print \), use required filter.
```