Gathering information about processes

Processes are the running instance of a program. Several processes run on a computer, and each process is assigned a unique identification number called a **process ID** (**PID**). Multiple instances of the same program with the same name can be executed at the same time, but they all will have different PIDs. A process consists of several attributes, such as which user owns the process, the amount of memory used by the program, CPU time used by the program, and so on. This recipe shows how to gather information about processes.

Getting ready

Important commands related to process management are `top`, `ps`, and `pgrep`. Let's see how we can gather information about processes.

How to do it...

`ps` is an important tool for gathering information about the processes. It provides information on which user owns the process, the time when a process started, the command path used for executing the process, PID, the terminal it is attached with (TTY), the memory used by the process, CPU time used by the process, and so on. For example:

```
$ ps
  PID TTY          TIME CMD
 1220 pts/0    00:00:00 bash
 1242 pts/0    00:00:00 ps
```

The `ps` command is usually used with a set of parameters. When it is run without any parameter, `ps` will display processes that are running on the current terminal (TTY). The first column shows the process ID (PID), the second column is the TTY (terminal), the third column is how much time has elapsed since the process started, and finally CMD (the command).

In order to show more columns consisting of more information, use `-f` (stands for **full**) as follows:

```
$ ps -f
UID        PID  PPID  C STIME TTY          TIME CMD
slynux    1220  1219  0 18:18 pts/0    00:00:00 -bash
slynux    1587  1220  0 18:59 pts/0    00:00:00 ps -f
```

The preceding `ps` commands are not useful, as it does not provide any information about processes other than the ones attached to the current terminal. In order to get information about every process running on the system, add the `-e` (**every**) option. The `-ax` (**all**) option will also produce an identical output.

The `-x` argument along with `-a` specifies to remove the default TTY restriction imparted by `ps`. Usually, using `ps` without arguments prints processes that are attached to the terminal only.

Run one of these commands: `ps -e`, `ps -ef`, `ps -ax`, or `ps -axf`.

```
$ ps -e | head
  PID TTY          TIME CMD
1 ?        00:00:00 init
2 ?        00:00:00 kthreadd
3 ?        00:00:00 migration/0
4 ?        00:00:00 ksoftirqd/0
5 ?        00:00:00 watchdog/0
6 ?        00:00:00 events/0
7 ?        00:00:00 cpuset
8 ?        00:00:00 khelper
9 ?        00:00:00 netns
```

It will be a long list. The example filters the output using `head`, so we only get the first 10 entries.

The `ps` command supports several details to be displayed along with the process name and PID. By default, `ps` shows the information as different columns, and some of them may not be useful for us. We can specify the columns to be displayed using the `-o` flag and hence, thereby print only the required columns. Different parameters associated with a process are specified with options for that parameter. The list of parameters and usage of `-o` are discussed next.

In order to display the required columns of output using `ps`, use:

```
$ ps [OTHER OPTIONS] -o parameter1,parameter2,parameter3 ..
```

Parameters for `-o` are delimited by using the comma (`,`) operator. It should be noted that there is no space in between the comma operator and the next parameter. Usually, the `-o` option is combined with the `-e` (every) option (`-oe`), as it should list every process running in the system. However, when certain filters are used along with `-o`, such as those used for listing the processes owned by specified users, `-e` is not used along with `-o`. Usage of `-e` with a filter will nullify the filter and it will show all process entries.