



# Scheduling Tasks

In this book, you've learned about various tasks you can perform to keep Ubuntu running smoothly. You may decide that you want some of these tasks to occur on a regular basis. For example, perhaps you want your /home folder to be backed up every day, or perhaps you want to clean the /tmp folder to ensure that you always have enough free disk space. You could carry out each task individually, but human nature would no doubt step in, and you would forget, or you might perform the action twice, because you've forgotten that you've already done it.

As you might expect, Linux is able to automate the running of particular tasks. They can be run either periodically or as one-time tasks. Using Linux's scheduling features is explained in this chapter.

## Scheduling with cron

Under Linux, the traditional way of scheduling tasks is via the cron daemon. This works on behalf of the user to automate individual jobs and is also used by the system to run its own maintenance tasks. The cron command is useful for scheduling heavy loads at a time when you know the system will be underused.

For cron to run system tasks, it reads a file called /etc/crontab. Traditional cron starts soon after bootup and sits in the background while you work, checking every minute to see if a task is due. As soon as one comes up, it commences the task, and then returns to a waiting status.

## Creating a Scheduled Task

Users have their own crontab file, which is stored in the /var/spool/cron/crontabs/ directory, by username. This directory is owned by root, and normal users can't view each other's crontab. The user's crontab file is updated in a text editor, but a special command is used to do so. (It saves you from having to remember to run the crontab command after editing the file.)

Adding a scheduled task is done via a terminal window (Applications ► Accessories ► Terminal). Entering the following command will cause your personal crontab file to be loaded into the GNU nano text editor, ready for editing:

```
crontab -e
```

If this is the first time you've edited your crontab file, you'll see a comment line at the top. Comments in configuration files are nearly always preceded by hash symbols (#); the hash tells Ubuntu to ignore that line. This particular comment outlines the syntax for adding an entry to the file. You can delete the comment line if you want.

Adding a new entry is relatively easy, and normally takes this form:

```
01 12 15 * * tar -cjf /home/keir/mybackup.tar.bz2 /home/keir
```

Let's examine the line piece by piece. The first part—the numbers and asterisks—refers to when the task should be run. From left to right, the fields refer to the following:

- Minutes, from 0 to 59
- Hours, in 24-hour time, so from 0 to 23
- Day of the month, from 1 to 31 (assuming the month has that many days)
- Months, from 1 to 12
- Day of the week, either from 0 to 6 (0 is Sunday), or specified as a three-letter abbreviation (mon, tue, wed, and so on)

In the example, the task is set to run at the first minute at the twelfth hour (midday) on the fifteenth day of the month. But what do the asterisks stand for? They're wildcards, and tell cron that every possible value applies. Because an asterisk appears in the month field, this task will be run every month. As an asterisk appears in the day field, too, the task could run on any day of the week.

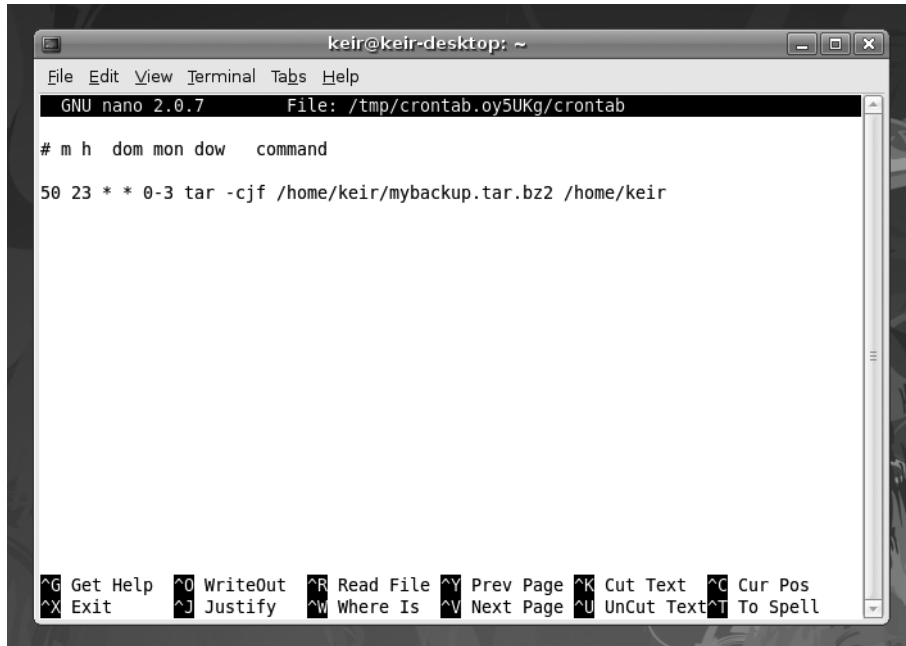
You might have noticed a logical contradiction here. How can you specify a day if you also specify a date in the month? Wouldn't this seriously limit the chances of the task ever running? Yes, it would. If you were to specify sat, for example, and put 15 in the date field, the task would run on only the fifteenth of the month if that happened to be a Saturday. This is why the two fields are rarely used in the same crontab entry, and an asterisk appears in one if the other is being used.

After the time and date fields comes the command itself: tar. As you learned in the previous chapter, this tar command can be used to back up your personal data (in this example, as long as your username is keir).

The cron daemon isn't clever enough to interpret symbols such as the tilde (~) as a way of referring to your /home directory. For this reason, it's best to be very thorough when defining a cron job, and always use absolute paths.

Let's take a look at another example (shown in Figure 32-1):

```
50 23 * * 0-3 tar -cjf /home/keir/mybackup.tar.bz2 /home/keir
```



**Figure 32-1.** *Editing crontab lets you schedule tasks using the nano text editor.*

The first field says that this task will run at the fiftieth minute of the twenty-third hour (that is, ten minutes before midnight). The day and month fields have asterisks, so this implies that the task could run on any day and every month. However, the day field contains 0-3. This says that the task should run only on days 0 through to 3, or Sunday through Wednesday.

You can have as many cron entries as you like; simply give each a separate line. You don't need to put them in date or time order. You can just add them as and when you see fit.

When you're finished, save the file with Ctrl+O and quit GNU nano by pressing Ctrl+X. You'll be prompted to save the file to the /tmp directory, which is fine. The cron file will be copied into the correct location as soon as nano quits, and you should see the following output in the terminal:

```
crontab: installing new crontab
```



---

**Note** The `/etc/anacrontab` file runs scripts contained in the directories `/etc/cron.daily`, `/etc/cron.weekly`, and so on, depending on when the tasks are meant to run (every day, week, or month). The average user never needs to bother with system-wide `anacron` jobs. Programs create their own entries as and when necessary.

---

Each line in `anacrontab` takes the following form:

*days*      *delay*      *name of task*      *command*

The `days` field holds the number of days in between the running of the task. To set the task to run every day, you would enter 1. To make the task run every nine days, you would add 9. To set it to run monthly, you would type `@monthly`.

The `delay` field tells `anacron` how long to wait before running the task, specified in minutes. It is necessary because `anacron` is run at boot time by default. If it were to run all the scheduled tasks simultaneously, the machine could grind to a halt under the load. A delay of five minutes is usually adequate, although if some tasks are already scheduled to run on the same day before that task, you should allow enough time for them to finish.

The `name of task` field is for your personal reference and shouldn't contain either slashes or spaces (hint: separate words using underscores or periods).

The `command` field is, as with `crontab`, the shell command that should be run.

---

**Note** `anacron` is run as the root user, so if you do add your own entry to `anacrontab`, any files it creates will be owned by root, too. If you use `anacron` to create a backup of your `/home` directory, for example, the resultant backup file will be owned by root, and you'll need to use the `chown` command to change its ownership so you can access it. See Chapter 14 for more information about the `chown` command.

---

Let's look at an example of an `anacrontab` entry:

```
1    15    backup_job    tar -cjf /home/keir/mybackup.tar.bz2 /home/keir
```

This will run the specified `tar` command every day (because 1 is in the `days` field), and with a delay of 15 minutes after `anacron` is first run.

`anacron` is run automatically every time you boot, but you can also run it manually by simply typing it at the command prompt (with superuser powers):

```
sudo anacron
```

## Using `at` to Schedule One-Off Tasks

What if you quickly want to schedule a one-time-only task? For this, you can use the `at` command.

Adding a job with `at` is very easy, largely because the `at` command accepts a wide variety of time formats. For example, typing the following at the command prompt will run a job at lunchtime tomorrow:

```
at noon tomorrow
```

It really is as simple as that!

Alternatively, you can specify a time, date, and even a year:

```
at 13:00 jun 25 2009
```

This will run the job at 1 p.m. on June 25, 2009. The various time and date formats are explained in the `at` command's man page.

Once the `at` command containing the date has been entered, you'll be presented with a mock shell prompt. Here, you can type the commands you want to run. Many shell commands can be entered, one after the other; just press Enter between them. Then press Ctrl+D to signal that you're finished editing. At this point, `at` will confirm the time and write the task into its list.

You can view the list at any time by typing `atq`. This will show a list of numbered jobs. You can remove any job by typing `atrm`, followed by its `atq` job number. For example, the following will remove the job numbered 9 in the `atq` list:

```
atrm 9
```

## Summary

In this brief chapter, we looked at how you can schedule tasks under Ubuntu, which essentially means making programs run at certain times. We examined the `cron` and `anacron` daemons, which can schedule tasks to run at specific times or periodically, and we also examined the `at` command, which can schedule one-off tasks.

In the final chapter of this book, we will look at how you can access your Ubuntu computer remotely—theoretically, from any Internet-equipped location in the world.