BEGINNERS

Program scheduling with Cron
COUNT DOWN

Linux users can save themselves time and effort by scheduling jobs with Cron. Juergen Jentsch checks out exactly what Cron's capable of

Did you realise that Linux encourages laziness? If your computer is running, it can take care of a whole bunch of tasks without any help from you. Just tell your computer what to do and let it get on with the job.

Once only

Linux provides the *at* command for tasks that you need to run once at a given time. Suppose you need to convert the files in your MP3 folder to wav files overnight so you can create an audio CD the next morning. You can use the following script to perform the conversion job:

```
#!/bin/sh
for file in *.mp3
do
  mpg123 -w "$file.wav" "$file"
done
```

Save the script as "mp32wav" in the directory with the MP3 files you need to convert and type *chmod a+x mp32wav* to make the script executable.

```
pingi@server:~/mp3$ at 23:00
warning: commands will be executed using
/bin/sh
at> mp32wav
at> [Ctrl][D]
```

Assuming that you did not run out of hard disk capacity, the wav files should appear right next to your original MP3 files just in time for breakfast next morning. However, if you want to run the script at 11 p.m. in three days time (because you will be spending the next few nights hacking), you might like to try the following:

```
pingi@server:~/mp3$ at 23:00 + 3 days
at> mp32wav
at> [Ctrl][D]
```

Of course, your computer must be up and running on the day and at the time in question. If you have installed a local mail server, you will be sent a message confirming that your script has been run. The mail is always sent to the user who entered the *at* job.

You can use *atq* to list the jobs that *at* has scheduled. Normal users will only be able to view their own jobs, but root will be able to view a list containing all the scheduled jobs:

```
root@server:~# atq
2       2002-04-13 23:00 a pingi
```

If you change your mind in the meantime, you can use *atrm job_number* to delete a job:

```
pingi@server:~$ atrm 2
```

The *batch* command is a subcategory of the *at* command that is seldom seen on today's workstations. The command has a similar effect to *at*, the major difference being that the command is only launched when the system load drops below a given threshold (**load average** < 0.8). If you are working on a server with a heavy daytime load, the batch command will wait until your colleagues have left the office:

```
pingi@server:~/mp3$ batch -m
at> mp32wav
at> [Ctrl][D]
job 8 at 2002-04-13 14:32
```

The *-m* ("*m*ail") switch, which you can also use with *at*, allows you to send mail on completion even though the command itself does not produce output on your standard output device.

Again, you can use *atq* to view and *atrm* to delete any jobs scheduled using *batch*.

Nice and regular

If you need to schedule recurring tasks, don't worry about the syntax for all those *at* commands. Instead, use the cron daemon, a program that specialises in taking care of those repetitive tasks.

A GUI tool is probably the easiest way to create a job-list, or *crontab*. KDE provides *kcron* for this purpose (see Figure 1). Just click on *Edit/New* to create a new entry.

Type a name in the comment text box to remind you what this entry does, and then type the command you want to launch in the Program text box. You can then use the various time buttons to

for loop The *for* loop is a simple counter based loop that repeats while the runtime variable is within a given range. In bash the loop has the following generic format:

```
for runtime_variable
List
do
  [various
instructions]
done
```

Our example calls the mpg123 command once only for each .mp3 file in the current directory.

$ To query the content of a variable in bash you simply prefix the variable with the $ operator. Our example uses this technique to generate a name for each wave file it creates using the content of the file variable and the string ".wav".

specify the dates and times on which you want to run the current job. Click on OK to return to the main window. Unfortunately *File/Save* does not perform as expected in some *kcron* versions (although there are no surprises with KDE 3.0). However, when you close the program it should not be too difficult to manually transfer the jobs shown to the cron daemon.

If clicking all those buttons seems like a waste of time to you, you can resort to the traditional method instead of using a GUI tool. Use *crontab –l* to output the current cron job refer to Listing 1 for an example.

The hash sign (#) at the beginning of a line indicates a comment – just like in a shell script. Any other lines must contain the following entries (separated by spaces): minutes, hours, days, months, day of week and the command you want to run. Use a numeric value for the weekday: *0* represents Sunday and *6* is Saturday. Use and asterisk (*) as a wildcard, if you do not need to specify an entry. To use several values, type a comma as a separator and do not use space characters before or after the entries. The counter in Listing 1 is thus launched on the full and half hour, every hour and every day.

If you want to start from scratch, you can type *crontab –r* to delete the whole crontab. *crontab –e* calls the editor defined in your *EDITOR* environment variable (normally *vi*). Should you prefer to avoid using the standard editor, just launch your preferred editor and save the entries you need in an ASCII file. You can then use *crontab filename* to save your entries in the crontab.

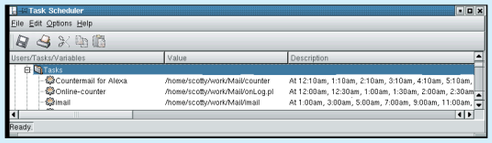

Figure 1: Main window in kcron 3.0

Almost on schedule

The cron daemon has one disadvantage: your computer must be up and running, at the scheduled launch times. But *anacron* provides a useful solution for this issue. If this package is not part of your current distribution, refer to the Info boxout for a download source. The installation is typically performed as follows:

```
tar -xzf anacron-2.3.tar.gz
cd anacron-2.3
make
su
make install
```

anacron's control centre is called */etc/anacrontab*; each line in this configuration file (Listing 2), which is administered by root, corresponds to a single task.

Listing 2: Anacron Control File

```
# /etc/anacrontab-Example
SHELL=/bin/sh
PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:
/usr/sbin:/usr/bin
#Interval delay Job-Identifier     command
1       5       whatis   makewhatis
7       15      Week statistics
/home/scotty/work/Mail/stats
```

In contrast to cron, *anacron* does not work with exact times, instead the first column is used to define the number of days that elapse before performing a given task. If the command resides in a directory defined in the *PATH* variable, you can simply type the name of the command, if not, you must supply the path.

If you call *anacron* on booting your system, it first reads the file referenced by the string in the third column to check whether a sufficient number of days have passed. This file is re-written after *anacron* has performed a task. To avoid launching thousands of jobs simultaneously when you start *anacron* (for example, after restarting a computer that has been down for a considerable period), you can enter a delay in minutes in the second column.

It makes sense to start the program from your boot script using *anacron –s*. If you normally leave your computer running overnight, you might consider entering *anacron –s* in your *root* cron table. Of course, this assumes that you really want to repeat a job at a given interval.

Info

anacron *http://software.linux.com/projects/anacron/*

load average You can use the *uptime* command to view the average load on your system:

```
pingi@server:~$ uptime
  2:05pm up       8:52,
4 user,  load average:
0.62, 0.35, 0.22
```

The command outputs the system time, the time elapsed since the last start, the number of users logged on and the average load for the last minute, the last five minutes and the last quarter of an hour.

Listing 1: A personalised cron table

```
# DO NOT EDIT THIS FILE - edit the master and reinstall.
# (/tmp/crontab.999 installed on Wed Mar 27 17:29:15 2002)
# (Cron version -- $Id: crontab.c,v 2.13 1994/01/17 03:20:37 vixie Exp
$)
# Countermail for Alexa
10      0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23 * *
* /home/scotty/work/Mail/counter
# Online-Counter
0,30 0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23 * * *
/home/scotty/work/Mail/onLog.pl >/tmp/file
# imail
0       1,3,5,7,9,11,13,15,17,19,21,23                        * * *
/home/scotty/work/Mail/imail
# This file was written by KCron. Copyright (c) 1999, Gary Meyer
# Although KCron supports most crontab formats, use care when editing.
# Note: Lines beginning with "#\" indicates a disabled task.
```