Crontables at the click of a mouse

TASK CREATOR AT YOUR SERVICE

BY PATRICIA JUNG

The Cron daemon may be very useful for making the computer execute one task or another at specific times, but the format in which it accepts requests takes a bit of getting used to. Which is where the graphic Crontab creation program can help.

Whether you want to cleanse your hard disk at regular intervals of the remains of dying programs (the *core* files), remember the birthday of your beloved in time, or want to be woken tomorrow by your MP3 collection – this is no problem when the computer is running almost round the clock. Simply record a task for the Cron daemon, and it will all be taken care of.

The catch: There is certainly a program named *crontab*. This retrieves your favourite editor and uses a rough syntax check to prevent too much mischief finding its way into the system, but the bottom line is that you have to write the *Cron table*, with its very demanding structure, yourself.

The acid test

Creating Crontab entries is not exactly one of those things where one automatically keeps up to date because they are done at least twice a day. It's all too easy for a little error to slip into the time details or the syntax. Some newcomers might be so frightened by the syntactical hurdles that they will have nothing more to do with this useful tool.

Graphical fond-ends programs for *crontab* suggest themselves as a way out of this dilemma. We are now going to take a closer look at a few of these little helpers.

Our test course is not completely undemanding. With a really complex invocation, we will check if the program can cope with long command lines. (The Cron daemons found under Linux usually follow up to 1024 characters per Cron table entry.) To do this we will build an alarm clock from the two command line tools *mix* (*ftp://sunsite.unc.edu/pub/Linux/apps/sound/mixers/mix-1.0.tar.gz*) and *mpg123*. This will search the directory */music* for MP3 files and play these from Tuesdays to Fridays at 07.25 at half volume (*vb=50*) in a random sequence (*-z*). It will also send all outputs from **Standard output and Standard error output** into data nirvana */dev/null*.

Four minutes later, hopefully, we are out of bed and so want the annoying cacophony to switch off automatically at 07.29 with *killall*. We should now check whether the program is actually making use of the whole time spectrum of the Cron daemon (every minute). For those who always find it hard to

***Standard output, standard error output**: command line tools send their results to the pre-defined standard output channel stdout, their error messages to the standard error output stderr. Both are normally 'linked' with the screen, while the standard input channel stdin is used by the keyboard. As user, you have the option of telling a command that instead of the screen, it should use a file as stdout: Command > file. Similarly, error messages can also be output in a file: Command 2 > file.*

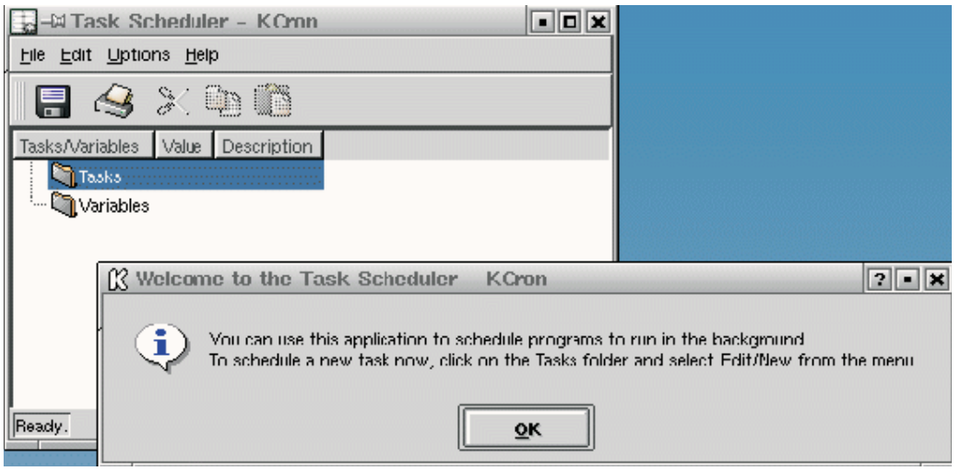


Fig. 1: kcron needs the initial user instructions

get out from under the duvet in the morning, the alarm clock goes off from 07.35 to 07.39 for a second time.

Next, we want to be presented with the morning paper every day at 8am (*netscape http://www.thetimes.co.uk*. Ideally, the program will let us know that we are going to have a problem with this, if we have not logged on and there is no X server running under our name.

As the fourth task, a self-written script from the subdirectory *bin* of your own home directory will perform a few clearing-up tasks daily between 0.00 and 23.00 every six hours. We want to save ourselves the bother of specifying the precise path to *clear up* when entering the command and therefore set the Crontab *PATH* variable in advance to *$HOME/bin*. With this we can test whether the front-end can set variables and not just globally once at the start of the Cron table, out also before a new entry.

As a rule, Linux Cron daemons only evaluate a few special variables. The result of this is that a really good GUI-Crontab program does not allow just any variable to be set.

Listing 1 shows the appropriate hand-written Crontab. This should also be able to be read in by all the programs. Particular attention is paid here to periods specially produced with - and / (*0-23/6*). These are recognised by the Cron daemons commonly used under Linux, but not by Cron implementations of other Unix operating systems. Another test criterion for reading in the hand-written Crontab lies in the correct reproduction of the comments.

Lastly, we will now check how the program acts in this connection with respect to impossible dates such as 30 February or 31 November. This, and a plausibility check of the program to be executed, may be requirements that not even the command line*crontab* program meets. Since click programs are specially intended to make access easier for the less adept user, though, they will have to be measured against stricter criteria.

Table 1 shows the most important criteria and information on each individual program. For comparison, the command line tool installed as standard on almost every Linux system, *crontab*, is executed at the same time.

Kcron

The fact that an all-round carefree desktop environment like KDE comes with a Crontab front-end, is more or less to be expected. In fact this has been the case only since KDE 2.0: *kcron* from the *kdeadmin* packet or else installed as an individual *rpm* packet, can be found in the *K* menu under *System/task scheduler Kcron* or can be retrieved using the command *kcron &* (Figure 1).

Apart from the actual application window, a short set of instructions also appears – and these really are needed: The fact that a new task is being created by selecting the *Tasks* folder with a double click, is something most would presumably expect. But then nothing happens, which means we must fiddle about instead by selecting, in the *Edit* menu, the entry *New...* or quickly learning the keyboard shortcut *Ctrl-N*.

The task creation window (Figure 2) is clear, but this only allows tasks at a rate of one every five minutes. When importing an existing Crontab there

Listing 1: Example of a personal Cron table

```
# DO NOT EDIT THIS FILE - edit the master and reinstall.
# (/tmp/crontab.1514 installed on Wed Jan 6 21:44:50 1999)
# (Cron version — $Id: cron2 red1.html,v 1.4 1999/03/06 22:44:46 lm Exp $)

# Alarm
25,35 7 * * Tue-Fri (/usr/local/bin/mix vb=50; /usr/bin/mpg123 -b 2048 -z `find
/music -name \*.mp3`) > /dev/null 2>&1
# after 12 minutes turn off the annoying alarm clock
29,39 7 * * tue-fri killall mpg123

# The morning paper
0 8 * * * netscape http://www.thetimes.co.uk/

# From now on only programs in $HOME/bin without path specification
# will be found
PATH=$HOME/bin
# Clear up script
0 0-23/6 * * * clear up
```

[top]
Fig. 2: Cleared up, but not finely adjustable

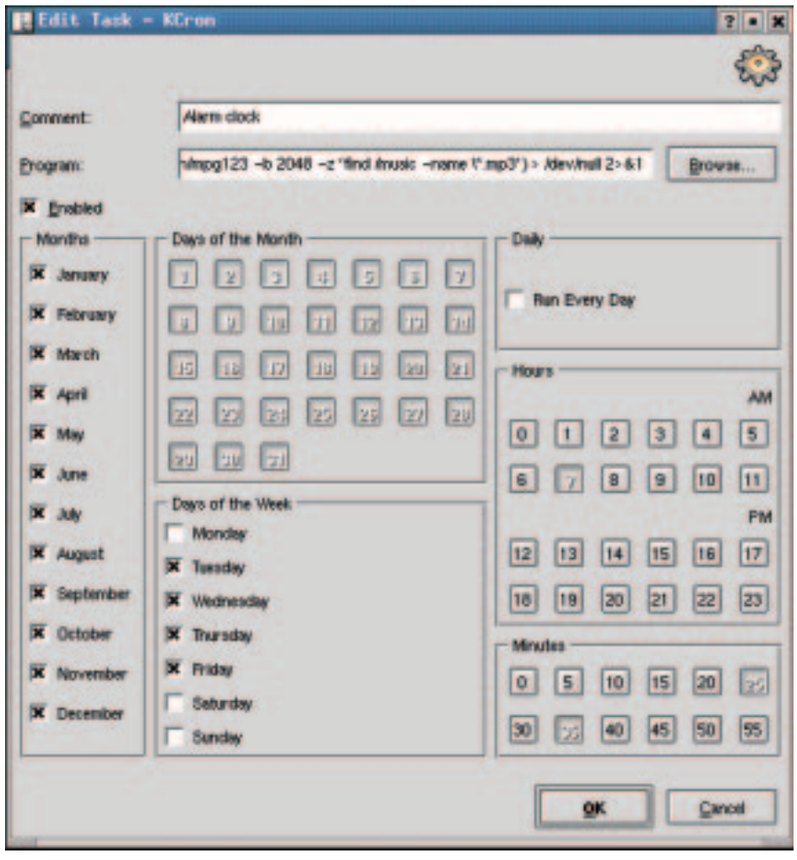
[above]
Fig. 3: Shame kcrontab did not get there after KDE 2

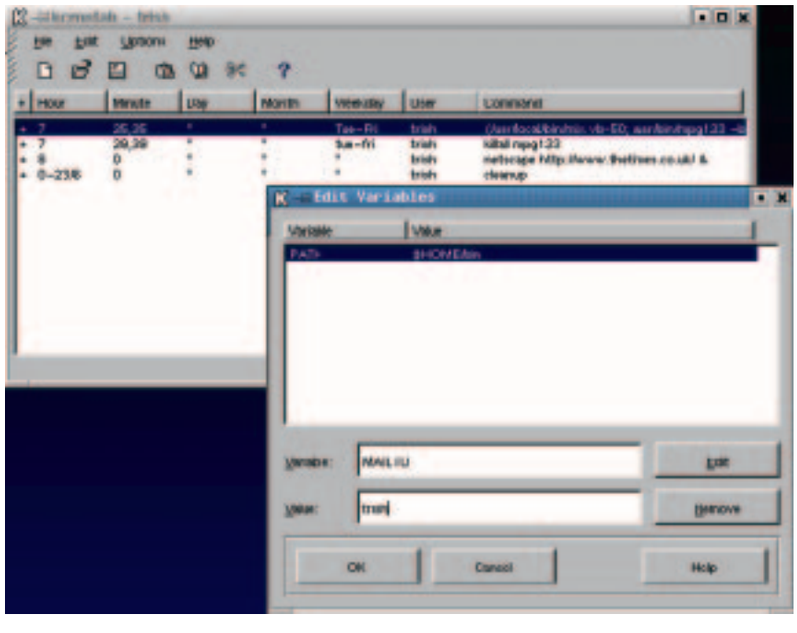
are in fact no problems even with a number of minutes, which is not divisible by 5 – unless you want to alter these entries. If no month is entered, the program complains, and you are forced to make a cross every 12 months individually, if you want to imitate the little star in the month column with *kcron*.

The details of the variables are explained: With Cron's permission these can be selected direct from the menu and are immediately provided with an explanatory comment. However, *kcron* does not refuse fantasy variables, and changing the environment for individual commands, by placing the appropriate variables before each Crontab entry, is also impossible.

If you make use of the option of searching via the *Select* menu for the program to be executed, one way or another this will be specified with the complete path, so that this takes the sting out of criticism for the *PATH* variable. If you click the command to be executed in this way, you will also get a free test of executability rights.

Both variables and Cronjobs may be commented. The only shame with this is that *kcron* refuses to import multi-line comments. Plus, the program has a surprise in the form of a very useful feature, even if it's not visible at first glance: Crontab entries can be deactivated in *kcron*. Behind the scenes such an entry is specially decommented, and is thus retained, can be modified and if required, reactivated.





KCrontab

Anyone who has not yet changed over to KDE 2 can still make use of *kcrontab*, even if it is not that easy to start up this first KDE-Crontab manager. Minutes and hours are entered manually (meaning you have complete format control, so can also use *), and thanks to the *every month* selection item, you don't have to click yourself to death.

Apart from forgetting an essential time detail there are no more plausibility tests in the pipeline, not even when setting variables. When entering a variable for the first time it takes some getting used to, having to click on *Edit*, to enter it into the list after setting. Nevertheless *kcrontab* is much better to use, (especially when the Crontab format is not a complete unknown), than the KDE-2 program.

Cromagnon

Calculated for what is, for Cronjobs, such a common time at midnight, the program once conceived as standard Crontab manager for the GNOME desktop refuses: One you realise that the witching hour (midnight) in Cromagnon's user dialog starts at 12am (Figure 4), then you will no longer be disappointed. Erroneously, this detail has been stored as 24 instead of as 0. If you have just got over the fact that the GNOME-Crontab manager personally so far only *creates* Crontables, but does not file them in the spool directory, you will now be saving the generated Crontab in a separate file and try to install it with *crontab \filename*. *crontab* may, however, correctly fail to be convinced of the erroneous 24-hour specification and need editing by hand.

Variables are not an issue for Cromagnon; comments on the other hand can not only be made,

but even have to be entered in the column *Description*. On the basis of the comment lines generated by it, commencing with *#CroMagnon:* Cromagnon namely recognises whether he it is confident enough to modify the entry found in the following Crontab line. All Cronjobs not marked in this way are quite simply taboo for Cromagnon – which is annoying, since the Crontable painstakingly made by hand cannot be modified. On the other hand this has the advantage that non-standard extensions of the Crontab syntax, as used by special Cron daemons such as *ucrond* or *hc-cron* remain enclosed. But anyone interested in such special features has already got the knack of using Crontable by hand.

The option of duplicating an existing entry using the *Duplicate* button in order to adapt the clone later, saves time. There is no help in creating the task command, and since nothing has happened to do with the code for a long time, it is to be presumed that the project will not become a mature aid in the foreseeable future.

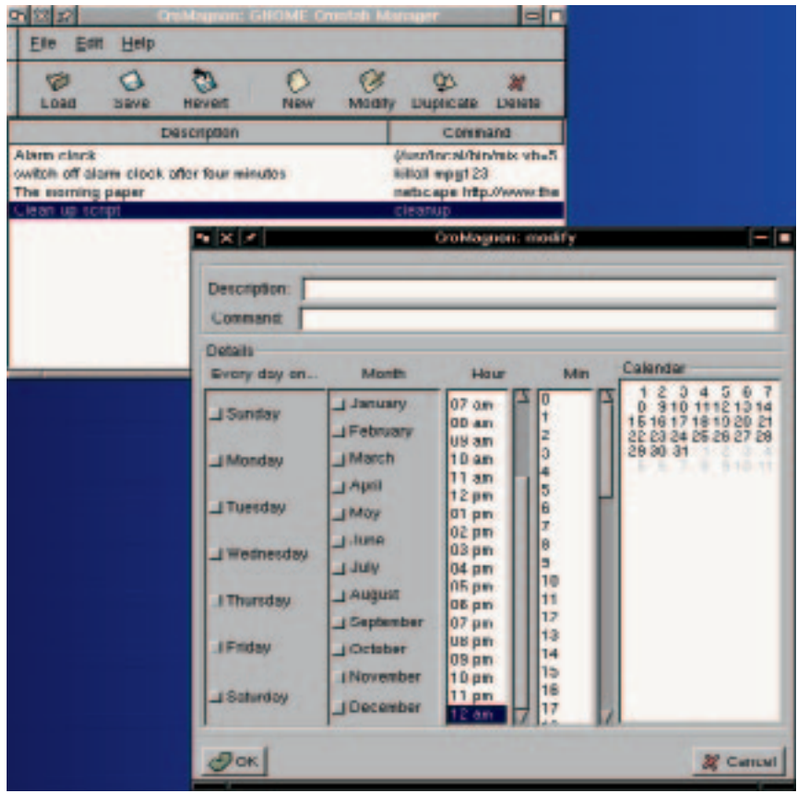


Fig. 4: Blackbox: Cromagnon recognises only its own Crontable entries

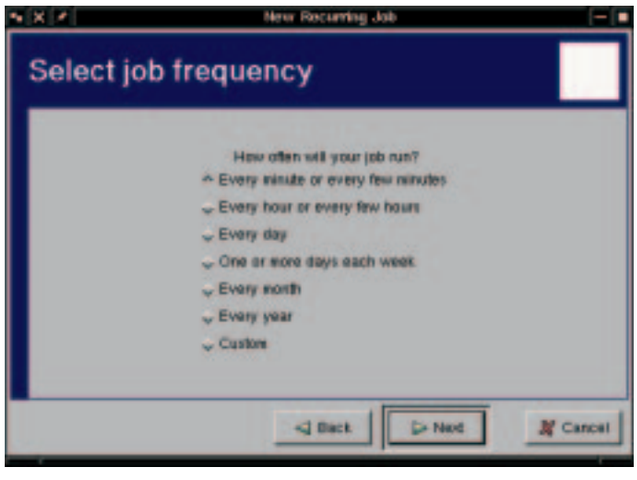
gat

The gap which Cromagnon leaves in the GNOME project, is one which *gat* is trying to occupy – and in doing so it is taking refreshing new paths. Confusing to Crontab veterans at first glance, the *0 8 * * ** entry from Listing 1 is displayed as *Every day at 8:00am*, but this translation into natural language means the program is to some extent intuitively used by users without knowledge of Crontab syntax. Unfortunately this concept is not always kept to consistently.

The GTK Task Scheduler tries as far as possible to hide the cryptic Crontab syntax from the user. This is especially evident from the fact that a wizard leads the way through the job creation. But this basically good idea is not always clearly thought out and/or completely realised.

So for example, it is only possible via the point *Custom* (Figure 5a) to create a task which is to be executed twice an hour but not exactly every half hour, and here you need to know your way around a bit in terms of the Crontab syntax. Too much knowledge, however, is also dangerous, because for example if a (legal) abbreviation for a day of the week such as *mon* (for Monday) is entered, nothing whatsoever happens, not even an error message.

When entering a Cronjob command, the wizard only provides one *Browse...* button as an aid (Figure 6) and at the same time also accepts non-executable files without any problem. But this should not be held against the program, because as it does have a *Test job* button in the main window, it is the only one in the test field to have a test option for the input command. All that was really missing here was a warning that the Cron daemon can execute graphics programs badly if one has not



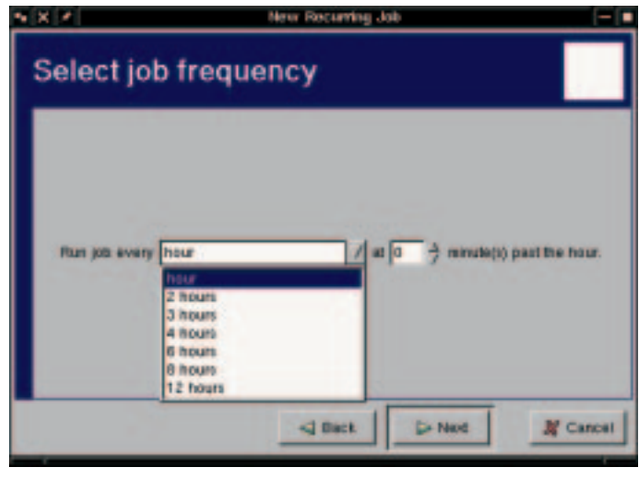


Fig. 5 : Good ideas, not always ideally realised – gat

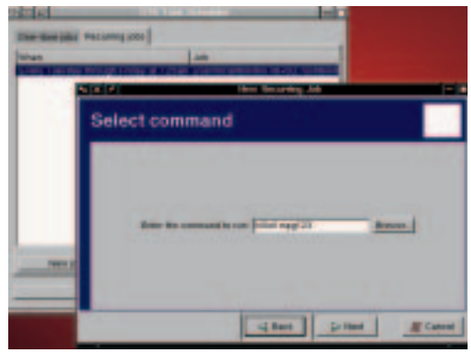


Fig. 6 : Plus points for the only Crontab manager with test options for Cronjobs

At-Job: *With this program at jobs can be defined which must be executed at a certain time (but unlike Cron, once only instead of repeatedly).*
GUI-Toolkit: *A programmer library, made available to programmers as a toolbox of components for constructing graphical user Interfaces graphical User Interfaces windows, menus, buttons, selection buttons etc.*

logged on under X, then this point would be realised almost perfectly.

The 'Recommended' rating is however foolishly thrown away by *gat*, because it provides no editing option of any kind for jobs once created – so at present all you can do is delete and rewrite. Also, some of you may take a bit longer than expected when first starting, before finding out that to create Crontabs the rider *Recurring jobs* has to be selected: by default, you land in the index card *One-time jobs* to create **At-Job**s.

Visual Cron

Apparently just as popular as **GUI toolkits** Qt and GTK with Crontab substitute authors is the combination of the script language Tcl and the toolkit Tk. The protagonist with the longest history in this illustrious tour is called *vcron*, and helps, not only with the syntactical tricks of a Cron table, but (like *gat*) also in elegantly handling that of an At-job.

But there's no fool like an old fool, and so *vcron* disappoints by supporting neither comments nor variables. Our long-winded alarm clock entry flummoxes the tool as soon as you try to change it and the *0-23/6* entry is not correctly interpreted.

Table 1: Comparison of Crontab managers

Name/Version	kcron 2.0pre	kcrontab 0.2.2	cromagnon 0.1	gat 0.9	vcron 1.5	tkt 1.1	crontab (Vixie-Cron)
URL	KDE-Mirrors	RPMFind, z. B. http://rpmfind.net /linux/RPM/ powertools/6.0/ i386/kcrontab- 0.2.2-1.i386.html	http://www. andrews.edu/~ aldy/cromagnon.html	http://www.cs. duke.edu/~ reynolds/gat/	http://www.linux- kheops.com/pub/ vcron/vcronGB.html	http://www.spin. net.au/~mich/ tkcron/	(usually part of basic installation)
Requirements	KDE 2.0/Qt 2.x	KDE 1.0/Qt 1.x	GNOME/GTK	GNOME/GTK	Tcl/Tk 8.0	Tcl/Tk 8.0	command line
Accuracy when creating the Crontab	5-minute cycle	1-minute cycle	1- minute cycle	1- minute cycle	1- minute cycle	1- minute cycle	1- minute cycle
Reject impossible dates like 30.02. or 31.11.	no	no	no	no	no	no	no
Time details with - and /	import	import and when creating	no	Import	no "/" when import	import and when creating	yes
Command length	100 symbols	100 symbols	2064 symbols	1024 symbols (does write longer commands, but then crashes)	> 4000 symbols	> 4000 symbols used	depends on editor
Testing commands	Check for executability when using Selection dialog	no	no	"Test job" button	no	no	no
Comments	for variables- and task details	for task details	for task details	no	no	yes	yes
Variables	any with menu selection for those actually understood by Cron, each settable only once per Crontab	any, each settable only once per Crontab settable	no	no	no	MAILTO, SHELL, PATH	yes
Read in with crontab-installed Crontables	yes	yes	yes	yes (removes comments as soon as a new job is created with gat)	with cuts	no	yes
Import hand-written Crontabs from other files	no	no	no	no	no	only from ~/.tct	yes
File the created Crontable in the Cron spool directory	yes	yes	no	yes	yes	yes	yes
Support for at	no	no	no	yes	yes	no	no

Another disappointment was the user interface: Why tabulator symbols are displayed as \t, when it makes no difference to Cron, whether one uses as column separator a blank space or tabulators, is something you will have to ask the author himself. The fact that the *at* window, even when empty, still claims the same volume as the Cron part, also no longer corresponds to modern GUI-standards. And the helpful feature of having the current time always in view does not really make up for this (Figure 7).

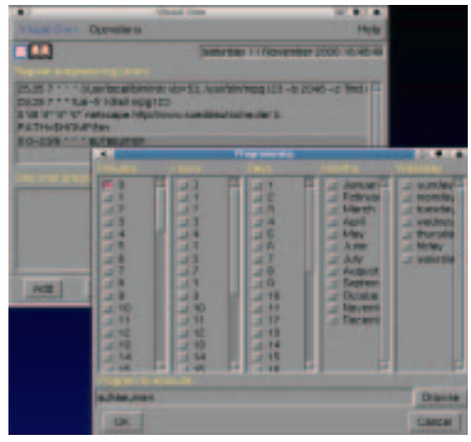


Fig. 7: vcron even falsifies standard Cron entries

tct

The second candidate from the Tcl/Tk faction is hesitant at first even to read in an existing Cron table: Only when the installed Cron table e.g. with *crontab -l > ~/.tct* has been read into the personal *tct*-configuration file, can this be loaded using the *Get Crontab* button into the editor window and edit it there, too. *Install* then actually does what it is meant to and installs the modified Cron table.

Anyone wanting to click together a new job, should get ready for a tiresome procedure: Each of the five time-setting buttons has to be set to a specified value and a command selected, before the task can be included in the editor window via the *Add* button, where it can only be altered manually.

The respective selection dialogs are very powerful, though there is no display showing for which button a value has already been defined, and the whole thing is not very helpful, since the typical Tcl/Tk error messages usually give more cryptic than meaningful information. Astonishingly, on the other hand, our over-complicated *0-23/6* details when converted with *tct* turned into the identical, but more compact form **/6* in the Cron table.

Once all times have been set, they are stored as default values for the current session, until something else is changed on them. So for additional jobs only times which deviate from the previous time settings have to be set anew. The *Find* button for selecting a command produced only error messages, though, in the tested version, so that there is no alternative to entering the command by hand.

Using the *Environment* button the three most common Cron environmental variables can be entered at the start in the Cron table. Anyone wanting to set them specially for a job has to modify them in the editor.

This is presumably where the main benefit of *tct* lies hidden: One quickly tires of the fiddly user interface, but thanks to the editor being visible at all times this is a valuable learning tool. One very soon learns how to write Crontables by hand.

What's left

The conclusion of this test is sobering. Apart from *gat,* all the programs presented try to imitate the *crontab* original graphically, with varying degrees of success. But, not least because of this paucity of ideas, not one of them manages to be an adequate (never mind a better) substitute. Apart from the ability of clicking together the time details (which is often in need of improvement itself) additional benefits are few and far between.

Perhaps the most useful programs to use are those which make themselves superfluous as quickly as possible by turning out to be learning aids for Crontab syntax. One candidate or another may help newcomers in simple Cronjobs get over the syntax hurdles, but for more demanding contemporaries, all the tools tested get stuck down blind alleys or are too basic. The only comfort here is that in any case one can only rely on the aid of Cron table managers for personal Cron tables. When it comes to managing system-wide */etc/crontab* it would be better to use your favourite text editor. ■

Info

Cron-Daemonen & Co. are available for downloading: ftp://sunsite.unc.edu/pub/Linux/system/daemons/cron/

■

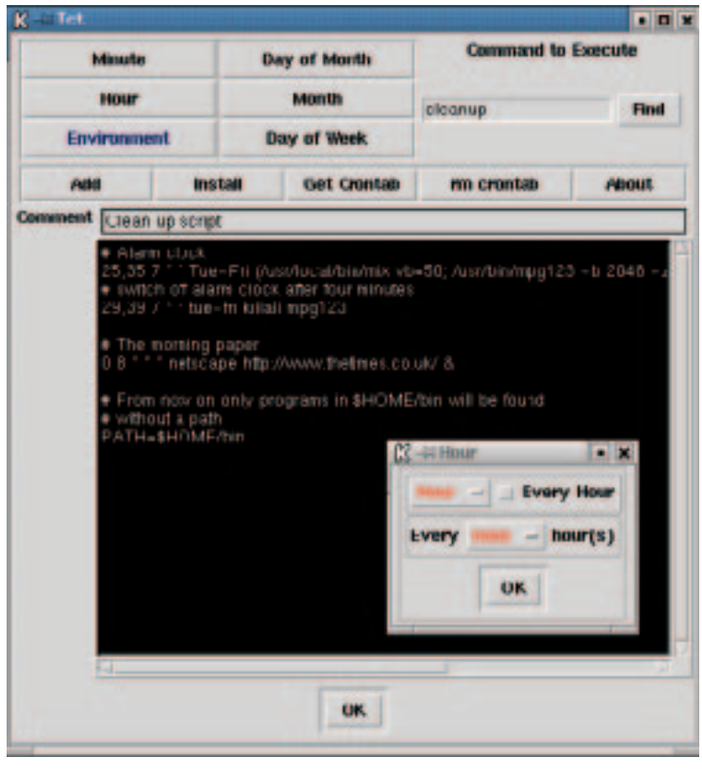


Fig. 8: Powerful, but extremely fiddly – tct