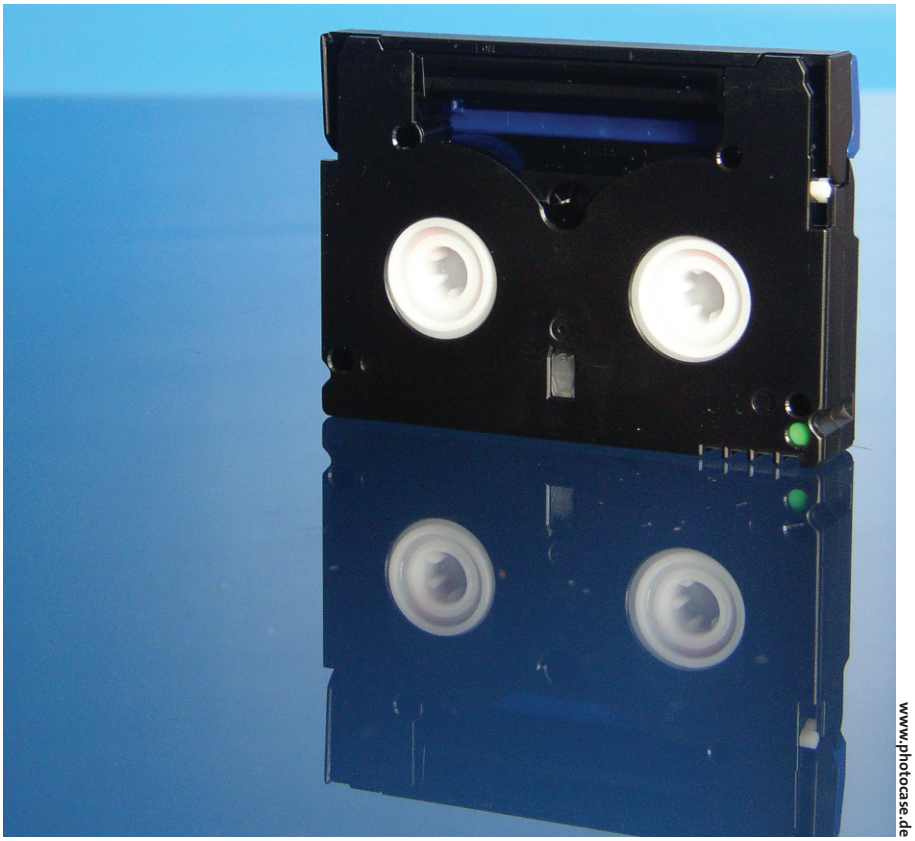
How to Email Your Video

Video & Radio Control

We all know that we can have a lot of fun with Linux. We also know that playing around with Multimedia can be fun. The question is, can we really put two diverse topics such as email and TV recording together, and still hope to produce a fun result? Steven Goodwin thinks we can do just that and sets off to prove it is possible with some programming knowledge.

BY STEVEN GOODWIN



www.photocase.de

When it comes to a futuristic vision of the future, hover cars and robot vacuum cleaners usually come top of the list! However, being able to email your video recorder must come fairly close. Now, thanks to Linux, it is not only possible, but free! Using nothing but common tools, some glue logic, and a little time, we can create such a system in the comfort of our own operating system!

Start of the Breakdown

The basic requirements consist of three areas. We'll consider these as a simplified top-down design:

Input Email arrives (for video@mydomain.com)

- Who receives the email?
- Can this user control the video?

Processing Let the email trigger a recording script at the appropriate time

- How can we validate the sender of the email?
- In what format is the email?
- Determining time on, time off, and channel to record.

Output The TV signal is recorded

- Accessing the TV card.
- Send a confirmation email?

Within each category there are several additional steps, each with its own problems to solve. Within the allotted space I shall be unable to cover every variation, or implement a complete professional solution. However, I will be able to demonstrate the steps involved in solving the problem, and the solutions produced. This is not the only solution, naturally. Just one of them, so consider this as an explanatory method.

The basic requirements for this project are a Linux-based computer with a TV card and Internet connectivity.

The In Crowd

Our first task is to create a specific email user to handle our requests. This is a simple job for root:

```
adduser video
```

The *video* user will need to access the TV card (*/dev/video*) and the audio mixer (*/dev/mixer*) to be able to record from it, so those devices must have the appropriate permissions to permit this. Often, these permissions will have already been decided according to the requirements of other users. In this case we should place the *video* user into the appropriate groups (perhaps *video* of the /dev/video device, and *audio* for /dev/mixer).

```
# usermod -G audio,video video
```

In order to test this, give the video user a password, log on, and run your TV application of choice. For example, *xawtv*. When the system is ready for deployment, we should remove the password, preventing anyone from being able to log

Listing 1: /home/video/.procmailrc

```
01 PATH=$HOME/bin:/usr/bin:/usr/local/bin:
02 MAILDIR=$HOME/Mail
03 DEFAULT=$MAILDIR/inbox
04 LOGFILE=$MAILDIR/log
05 LOCKFILE=$MAILDIR/.lockmail
06
07 :0
08         * ^Subject: help
09         | formail -s emailvideo.pl help;
10 :0
11         * ^Subject: record
12         | formail -s emailvideo.pl record;
```

in as *video*, and exploiting any security holes that might exist.

We also intend to email responses back from the video. Perhaps saying that the recording has started, or that the request could not be fulfilled. To this end, we should remember to include a specific comment in the user field that will appear on all emails. Otherwise, they will appear to come from ",,,"!

```
usermod -c "My Video" video
```

Processing email automatically is easy, thanks to a program called *procmail* (installed by default on most systems). *Procmail* is a mail filter that sits between your mail transfer agent (Exim, Sendmail, or Postfix, perhaps) and your mailbox, to process each message before it arrives. It can be programmed to move mail into specific folders (like spam, for instance), or forward on to other email accounts.

As with many tools, Procmail works by having one global configuration file (*/etc/procmailrc*) and several local ones, which live in each users' home directory and are called *.procmailrc*. These files consist of a number of *recipes*. Each recipe describes what to do with that mail when a particular set of conditions has been met. For example: move any email from *Dean* into the *Curry* folder! Procmail works through each recipe in turn as it is listed in *.procmailrc*. When it finds one that matches the criteria, it processes the email accordingly and exits, checking no further recipes.

For this project, we need to call an external script whenever an email is received by the *video* user. If the subject of this email is 'help', we shall reply with the instructions for using our email-based video. If the subject happens to be 'record' we shall respond by recording the television program. We shall therefore create two recipes, using the subject line to distinguish between them, and vary the parameter passed to our script. See Listing 1.

By default, procmail is case *in*sensitive so *help* is sufficient to match 'help', 'HELP' or 'hElP'. The structure of the *.procmailrc* file itself can be found in man pages, or in one of the many tutorials available on Internet. Including the one at [1].

Before calling our script, the entire email is piped through an extra program called *formail*. This converts the message into mailbox format. Although simple, this is an immensely useful tool that understands the features of RFC 822 (which describes the format in which an email should be) and outputs the email with a unified layout. Therefore, we only have to concern ourselves with the one format, and not several. *Formail* takes the email as standard input, and produces the mailbox (some would say, correct!) version to standard out. We must therefore write our *emailvideo.pl* script to deal with stdin.

Security

Security is always a concern. Every network port, program, or service we open to the outside world is another potential security flaw for people to attack. Some services might appear harmless, but giving our machine the capability of recording video may give a hacker opportunity to utilise all our disk space. Some prevention ideas might include,

- Record everything to a separate partition. Even if it fills up, the system still has space for log files
- Use PGP to validate the sender
- Reply to sender, asking for confirmation before adding the request to the queue
- Don't reply to any email that it doesn't have to. Spammers might use it as if it were an open relay
- Use passwords that expire, limiting the window of opportunity to days, or even hours

Email

```
01 ch:2
02 at:18:00
03 length:25
04 pass:my_secret_password
```

Finally, we could extend *.procmailrc* by including a final 'catch-all' recipe. This is placed at the end of the file, since procmail stops processing as soon as a match is found, and this case will match every email not yet caught. Any email not fitting either of the two patterns above could be sent to the bit-bucket. By default, unmatched messages would arrive in the user's mailbox, something that is redundant, knowing that *video* is not a real user!

A user-friendly system would reply to the sender with an error code. Without adequate safeguards, however, this could mean that *video* tries to reply to non-existent (i.e. faked) email addresses, or gets stuck in a loop by replying to itself ad infinitum. This is why the 'help and 'record' subjects must be specified precisely, so as to reject a subject line of 're: help' (as present in some spam emails).

```
:0
     *
     /dev/null
```

Testing this code is fairly straightforward since all our tools can be run on the command line, using standard input and output streams. Begin by sending a message to the new *video* user with 'help' as the subject. It should arrive in */var/mail/video* before too long.

If you pipe this file through *formail* you can see the output that will arrive in your Perl script. It will have all the email headers intact, ready for parsing. Redirect this output into a file and keep it in a safe place. We'll be using it later to test our Perl script, without having to commit a denial of service on our own mail server!

```
formail -s </var/mail/video ➥
>testmail
```

Friends will be Friends

We now have a vacancy for a Perl script! It must take an email from stdin, and prepare our TV card for recording. Now, although it might appear that the Perl script should be responsible for recording the TV program, this is not actually the case. There will very likely be a time lag between sending the email, and the programme starting.

So instead, we will create another script that does the actual recording, and use Perl to parse the email and determine when *that* script should be run. Perl can also validate the email, and reply immediately with any problem or ill-configured options. The second script will be written in bash, to make an obvious distinction between the two 'scripts'.

Parsing an email in Perl is best done with an existing module. CPAN [2] provides so many modules, for so many tasks, that it is impossible for many of us to know which is better for any particular job.

I am using Mail::Internet [3] for instance. Not because I have any mystical insider knowledge of it, but I've used it before, and it does the job I need doing. Other people have their own favorites! You can install it from the command line with:

```
# perl -MCPAN -e 'install Mail::Internet'
```

Remembering, of course, that you may need to be root. If this is the first time you've used the CPAN module to install software, you will be asked a series of questions to configure it. The defaults are perfectly acceptable. Once installed, Mail::Internet provides a wealth of email-related functionality. We, however, will only be using a small part of it. We need to determine the sender from the header, and read the email body to determine the recording parameters. First off, we must create a mail object,

```
#!/usr/bin/perl
use Mail::Internet;
use strict;

my $mail = Mail::Internet->new( [ <STDIN> ] );
```

This object is our gateway into the message. Everything about the email can be referenced through $mail, or a member function of it. We can then use the module code to provide the headers,

```
my $headers = $mail->head->header_hashref;
my $from = $headers->{From}[0];
```

Checking the *from* line gives us very little in the way of security. These can, and are, faked by spammers. We shall validate the requests by use of a simple plain text password included in the email. This password will be transient, and once it has been used, will not work again. It's not perfect, but it's a start. Other security ideas are presented in the BOXOUT: Security.

Using the information in *$headers,* it is possible to create a reply for this email by building a new header with all the appropriate information. However, since this is a common feature, the Mail::Internet module includes a method for doing this. Sensibly enough, it's called *reply*!

```
my $REPLY = $mail->reply();
$REPLY->body("A new body");
$REPLY->send();
```

We can make our first test here! Feed the test mail we created earlier into the current script, and an email should arrive in your inbox.

```
./emailvideo.pl <testmail
```

LIRC

The Linux Infra-red Remote Control project [6] has provided the community with a driver and a good set of tools for IR communication. It allows a Linux-oriented PC to receive IR signals that can be used to control applications (like Mozilla or mutt), or transmit messages to other devices!

Most TV cards provide IR capabilities with a small remote control and receiver 'eye'. These are well supported by LIRC, but usually work on a different IR protocol to the universal remote controls, and TV handsets available for traditional domestic use, and so are not interchangeable.

However, LIRC has produced simple circuits that can be built and attached directly to the serial port to provide bi-directional control of standard IR devices. This means you can send IR signals to a traditional VCR with LIRC.

There is no feedback loop unfortunately (so you can not tell if the video actually received your signal), but it can provide identical functionality if you don't have a TV capture card, or for when the hard drive runs out of space. Since the above design is modular, we only need replace a small amount of code in the *getvid.sh* script.

Listing 2: Recording script

```
1  #!/bin/bash
2  # Parameters are, in order: Station. Duration(mins). Reply email
3
4  TVNAME[0]=SVideo; TVMAP[0]=0
5  TVNAME[1]=BBC1; TVMAP[1]=55
6  TVNAME[2]=BBC2; TVMAP[2]=62
7  TVNAME[3]=ITV1; TVMAP[3]=59
8  TVNAME[4]=Ch_4; TVMAP[4]=65
9  TVNAME[5]=Five; TVMAP[5]=37
10
11 TARGETDIR=/media/tv
12
13 DURATION=$2
14 CHANNEL=${TVMAP[$1]}
15 NAME="Vid_`/bin/date +%Y%m%d_%H%M`_`echo ${TVNAME[$1]} | tr [.] [_]`".avi
16
17 if [ $CHANNEL -eq 0 ]; then
18  INPUT="input=1";
19 else
20  FREQ=$[($CHANNEL*8)+303]
21  FREQ=$FREQ.25
22
23  INPUT="input=0:freq=$FREQ";
24 fi
25
26 mencoder -o $TARGETDIR/$NAME -tv on:driver=v4l:$INPUT:width=320:height=240 -oac copy -ovc lavc -endpos $DURATION >/dev/null 2>/dev/null
27
28 REPLY=$REPLY"$NAME is awaiting you when you get home :)"
29
30 echo "$REPLY" | mail -s "Video complete !" $3
```

The contents of the actual reply need to be determined by the original email. This might be a request for help (just because it's our system, doesn't mean we'll always remember the format), or a command to record video. Our procmail recipe already passes an extra argument to the Perl script that can be used to determine the purpose of our email. This argument can be retrieved with,

```
my $ARGUMENT = shift;

if ($ARGUMENT=~/help/i)
{ $reply = do_help(); }
elsif ($ARGUMENT=~/record/i)
{ $reply = do_record($FREQ,
$DURATION, $TIME); }
else  { $reply = do_error(); }
```

The parameters for the *do_record* sub-routine need to be determined from the email body. Now, there is no easy way to do this… unless you consider regular expressions easy! Fortunately, the expressions we need are not so difficult, since we can design a layout that we're comfortable parsing. We shall adopt the strategy that every parameter will consist of,

```
NAME:VALUE
```

Where the name shall be case insensitive, and the value is interpreted literally as either a number, of a piece of text. This text will be passed into our other tools directly. An example of such an email can be seen in the BOXOUT: Email. Our parsing loop will read the first few lines of the email (as to avoid lengthy signatures that might confuse us), and works like so:

```
01 for(my $i=0;$i<10;$i++)
02 {
03 my $line = $mail->body()->[$i];
04
05 if ($line=~/ch[^:\s]*[:\s]*
   (.+)/i) { $FREQ = $1; }
06 if ($line=~/length[^:\s]*
   [:\s]*([0-9]*)/i) {
   $DURATION = $1*60; }
07 if ($line=~/length[^:\s]*
   [:\s]*([0-9]*)\s*hours/i)
   { $DURATION = $1*60*60; }
08 if ($line=~/pass[^:\s]*
   [:\s]*(.+)/i) { $PASSWORD =
   $1; }
09 if ($line=~/at[^:\s]*
   [:\s]*(.+)/i) { $TIME = $1; }
10 }
```

Table 1: UK station list

Station	Input	Channel *	Frequency *
BBC 1	0	55	743.25
BBC 2	0	62	799.25
ITV	0	59	775.25
Channel 4	0	65	823.25
Five	0	37	599.25
Composite 1	1	-	-

* Applicable to my area, possibly not yours!

The password must then be checked and validated before passing control onto the *do_record* sub-routine that will set the timer. In this example we are looking for, and then deleting, a file with an identical name located in a special *.pass.d* directory. Again, more complex solutions would be required in a commercial environment. However, this establishes our proof of concept.

```
sub is_password_valid()
{
  if (-f "$HOME/.pass.d/$_[0]")
 { return 1; }
  return 0;
}
```

In the code provided on the Linux Magazine web site we have extended this idea, allowing each command the option of requiring a password, or not. This allows help to be provided free of charge, so to speak.

Sound and Vision

The most common tool for running software at a specific time is *at*. This simple utility runs in the background and wakes up every minute to check for outstanding tasks. Cron works in the same way, but is targeted at jobs that are repeated on a regular basis. *at* can only run a single program (with optional arguments) so we shall use a bash script to achieve the more complex functionality we need to record a TV programme.

The parameters supplied in the email fall into one of two categories: those that determine when the recording should begin, and how the recording should behave.

The first category of parameters are used in conjuncture with the *at* command to trigger a script, and consists of the start time (check the *at* man page for the format).

Our second category (the recording behavior) will consist of the channel, and duration, of the recording process. These will be the parameters for our script. We will also append the email address for a completion message. If we wanted to run the script immediately we would type,

```
getvid.sh 2 25
steev@mydomain.com
```

Listing 3: getradio.sh. An equivalent getvid.sh

```
1  #!/bin/bash
2
3  FREQ=$1
4  DURATION=$2
5  EMAIL=$3
6
7  FILEPATH="/media/radio/"
8
9  NAME="Radio_`/bin/date
  +%Y%m%d_%H%M`_`echo $FREQ | tr
  [.] [_]`"
10
11 FILENAME=$FILEPATH$NAME
12 echo "$NAME" | mail -s "Radio
   recording started..." $EMAIL
13
14 BODY=""
15 BODY=$BODY`fm $FREQ $VOL`
16 BODY=$BODY`sound-recorder -S
   $DURATION -f=wav $FILENAME.wav
   2>&1`
17
18 # Encode and send
19 lame $FILENAME.wav
   $FILENAME.mp3 >/dev/null 2>&1
20
21 uuencode $FILENAME.mp3
   $FILENAME.mp3 | mail -s "Radio
   program ($FILENAME)!" $EMAIL
   >/dev/null 2>&1
22
23 # Clean up
24 /bin/rm $FILENAME.wav
25 /bin/rm $FILENAME.mp3
```

This would record a 25-minute program on BBC 2, for instance. To begin recording at some point in the future we would issue a command like,

```
$ at 18:00
warning: commands will be executed using /bin/sh
at> /home/video/bin/getvid.sh 2 25 steev@mydomain.com
at> <EOT>  # this is produced by closing the input stream by pressing ctrl+d
```

Notice the full path used for the script. Also note that the parameters we read from the email are unchecked, and passed through directly to this script. *at* takes the commands from standard input which, by default, is the keyboard. A simple pipe will allow this to work non-interactively:

```
echo $cmd | at $when
```

Bad Young Brother

Having written this much code, we're now ready for the final stage. Recording the actual program. For that, we have a vacancy for a bash script!

This part, believe it or not, is actually very easy. In this version, we are recording information directly from the TV capture card in the local machine. If you wish to control a physical VCR this is also possible, but you'll need some other software, as shown in the boxout: LIRC.

We can prepare the mixer (to record audio from the line in socket) beforehand, and test it with stand-alone applications like *xawtv* to check that the access rights have been correctly set for the *video* user. This includes the target directory for the resultant file.

The tool I have chosen to record the TV signal is *mencoder*. This is part of the *MPlayer* package from [4]. It provides a large set of low-level options to tune the channel, set the compression format, and determine programme duration. A full explanation can be found in the man pages. The parameters given below should be applicable to most systems. Depending on the speed of your processor however, you may need to record at a different resolution.

```
mencoder -o /media/tv/filename.avi -tv on:driver=v4l: input=0:freq=799.25:width=320:height=240 -oac copy -ovc lavc -endpos 25 >/dev/null 2>/dev/null
```

You can probably spot the parameters that our bash script needs to change: the filename, the endpos (aka duration) and the input/freq pair. Only the latter is non-obvious. *input* indicates whether the recording is to come from the tuner (0), or the S-Video socket (1). Your TV card may have other inputs.

The *freq* parameter requires an actual frequency in MHz. Not a channel number, or a station ID. Determining the frequency is a two-stage process. Firstly, we need to find the channel number (which generally ranges between 21 and 68) for a particular station (BBC 1, BBC2, etc). Secondly we need to convert that into a frequency.

The channel numbers will vary depending on where you live in the country, and are based on the nearest television transmitter to your location. Anyone using a TV card will probably have these channel numbers already living in the *.xawtv* file. If not, they can be produced by the *scantv* program.

UK readers can find these numbers on the BBC's web site at [5]. Five is not shown here, but normally lives on channel 37 or 39. There is no logic to the channel numbers, you simply have to find them, either through research, or brute force.

Channels can then be converted into frequencies with a straightforward equation. This is the same across the country. *frequency = (channel * 8) + 303.25*. This reflects the PAL-i vision frequency of a particular channel, which are spaced 8Mhz apart from each other.

Performing the arithmetic in bash requires a little syntactical massaging, as shown in line 20, although the addition of the fractional component (0.25) is done with concatenation. This builds into the data set present in Table 1.

Otherwise, the bash script is fairly uneventful. We use the time and date to determine a filename, eliminating the need for a 'get_unique_filename' function, and write our AVI into the */media/tv directory* (which will usually be accessible to other users). There's also an option for recording from the composite video input of the TV card. This could be an external DVD player, or even a security camera.

Short Stories

All that's now needed is the testing and deployment. And we should still have our sample email ready for that purpose. The completed script consists of a tree-felling 143 lines, and so has be relegated to the Linux Magazine web site, but all the pertinent details can be found within this article.

This is just one idea (and one solution) for automated email systems. Most mid-range TV cards include an FM radio receiver too. Adapting our existing system to record programs from the radio requires very little work! Granted, we need a new user (radio) to pick up the emails, and we must replace the video-related tools, but there's no more than an hours work! See Listing 3.

I use one package (fmtools) to control the radio, and another (sound-recorder) to record the WAV file. Then the lame MP3 encoder is invoked to create an MP3 of the radio show requested. Because it is much smaller in size than a movie, the MP3 can be uuencoded and emailed back to the user that requested it. This also requires that the audio mixer is setup correctly beforehand.

As for other ideas, there are several out there just waiting to be picked up. How about grabbing teletext pages from the TV card? Or adding files to your peer2peer download queue? Or a song request service? The tools are available for free. The information is available above. The fun is available whenever you find it! ■

INFO

[1] Procmail tutorial: *http://www.perlcode.org/tutorials/procmail/proctut/*

[2] Comprehensive Perl Archive Network: *http://www.cpan.org*

[3] Mail::Internet module: *http://search.cpan.org/~markov/MailTools-1.60/Mail/Internet.pm*

[4] MPlayer: *http://www.mplayerhq.hu/*

[5] BBC frequencies: *http://www.bbc.co.uk/reception/tv_transmitters/*

[6] Linux Infra-red Remote Control: *http://www.lirc.org*