

Synchronized People

Best Friend

Linux thrives on many tools and applications. Each designed to do a single task, but in turn to do that task perfectly, rather than large bloated software attempting to be a Jack-of-all-trades. Handling your email should be no exception. **BY MARCO FIORETTI**

Under Linux, email can be managed either with fully graphical clients, or with text ones, like Pine or Mutt. The second category still appears less featured and harder to many newbies, and/or justified only for gurus with special needs, like usability over SSH, or fast switching among twenty different GPG signatures.

The truth is that even newbies with normal needs can and should use Mutt, because email is an area where the Unix toolbox approach really shines. One can do everything possible with the heavy-weight clients by combining several small programs, just *faster*, even on an old computer. The real reason why this doesn't happen more often is that usually all the integration is left as an exercise for the user.

To fill this gap, this article will focus on a general method to make one's email environment more user friendly, and apply it to some areas of advanced email management. We will assume that the reader can already do the basics, i.e. send and receive email in different mailboxes, and discuss some of the nifty things that most Linux newbies (especially Outlook refugees) probably miss more.

(If what you miss more is your old email in Microsoft formats, just try `mbx2mbox` [1]: its home page also lists other similar tools, including one specific to Eudora mailboxes).

The real power of Mutt

Mutt is a console MUA able to thread email, color it according to user defined rules, and much more. You can configure it by copying one of the many examples

available [2], or with the online `Muttrc` builder [3].

Mutt has an *index* view, where all messages are listed, as well as a *pager* and a *composer* view, to read and, respectively, write messages. Mutt can execute *macros*, i.e. user defined sequences of both internal and external (scripts) commands. This capability, together with the *mailcap* files described later, can automate the interaction of Mutt with other programs, increasing user friendliness enormously. Of course, many of the tricks described here will work in every mailer able to accept user defined commands.

Centralize the control

What do the full blown graphical email clients really offer, apart from eye candy? They route incoming messages to different folders, manage an internal or external address database and display HTML email properly. Other useful functions include the capability to open a browser on any URL contained in the message body, neat printing, spell checking, and some mailbox search support.

Console based tools for all the above functions already exist. Knowing this, we can better understand the two real differences (at the functional and user friendliness level) between Mutt "out of the box" and the bigger, less flexible programs.

The first obvious observation is that big email clients look easy because you don't need anything else and everything is centralized: (almost) everything is configured in one file. The second difference, more subtle, but more important,



is that all the setup work is done from one place, the email client itself, without interrupting normal use! For example, if you want to create a new input filter while reading email, there is no need to start a new terminal or text editor, edit some other file, and maybe restart the mail client.

As a matter of fact, Mutt can achieve almost the same level of integration: apart from some "one time" tasks, you just need the right third party utilities and a little configuration magic to do a lot of cool things with just a few keystrokes or mouse clicks.

The necessary configuration files, reproduced here only partially, are available from [4]; many other examples are available in the Info box [5, 6, 7].

Opening attachments

Very often, opening attachments directly within the mail client is much more convenient than having to open another window after saving them somewhere, as I'm sure you'll agree.

Mutt can launch external programs to open attachments according to the instructions it finds in the `/etc/mailcap` or `$HOME/.mailcap` files. Each *mailcap* command explains what to do with a certain file type, depending on the situation (X running or not, interactive or non-interactive view...).

Listing 1 is an excerpt from a mailcap file, showing how it can deal with just about any document. The first three lines are for HTML files, and are discussed in the next section. The second group of options deals with MS Office attachments: `mutt_bgrun` is a handy script available from [5], which launches the viewer passed to it as the first argument in the background, on a temporary copy of the file. Without that script, Mutt would freeze until the viewer was running.

The last three settings handle faxes, images, and movie files. Of course, all the viewers in Listing 1 can be replaced with any program that supports the corresponding file format. If you just want to *print* something, the mailcap format supports that too. For HTML attachments, add something like:

```
text/html;; print= w3m -T
text/html -dump %s | a2ps
```

With a proper mailcap setup, all attachments can be handed off to external viewers or, if they can be converted to text, displayed directly inside Mutt's window. For example, chances are that if you are using Mutt, your most frequent attachment problem will be the Microsoft Office stuff. If OpenOffice.org or StarOffice are available, the mailcap lines shown in Listing 1 will be enough, but what should one do when that is not an option? The answer consists of lines like this:

```
auto_view text/html
application/msword
```

Listing 1: example mailcap file

```
01 text/html; netscape -remote 'openURL(%s)' || netscape %s; test=sh -c
   'test $DISPLAY'
02 text/html; w3m %s; needsterminal; description=HTML
   Text;nametemplate=%s.html
03 text/html; w3m -dump %s; nametemplate=%s.html;copiousoutput
04
05 application/msword;          mutt_bgrun soffice %s
06 application/msword;          catdoc %s;copiousoutput
07 application/vnd.ms-excel;     mutt_bgrun soffice %s
08 application/vnd.ms-powerpoint; mutt_bgrun soffice %s
09
10 image/x-tiff; mutt_bgrun display %s; test=test -n "$DISPLAY
11 image/*;      mutt_bgrun display %s;test=sh -c 'test $DISPLAY'
12 video/*;      xanim %s > /dev/null;test=sh -c 'test $DISPLAY'
```

in your `.muttrc`. They will tell Mutt to convert such files to text with the non-interactive program associated to that content type. Programs to be used in this way (non-interactive) are marked with the *copiousoutput* option (if you don't need `auto_view`, the viewer can still be invoked by selecting the attachment in the corresponding Mutt window and hitting 'v'). The MS Word converter called in Listing 1 is the `catdoc` program [8]. Info [5] also mentions a method for converting MS Excel files to HTML and then to ASCII.

Remember that attachments can contain bad surprises: Mutt itself is quite robust from this point of view, but mailcap is used by other programs too, so it must be written carefully. For further explanations, and detailed tips, see the section "Secure use of mailcap" in the Mutt manual.

HTML email

HTML email is nasty, no doubt, but it's not *always* spam: luckily, it is possible to deal with it even without a graphical client.

In general, it is possible to read HTML email or attachments in an external browser or as formatted text within Mutt, with or without clickable links. There is no unique or best practises solution to this problem, however. We suggest that you try out (and customize) all the methods discussed here and in the rc files mentioned above, until they match your specific needs and taste.

The first method is the easiest. The mailcap netscape setting shown in Listing 1 will check (note the "test"

instruction) if X is running, and launch the browser.

When the requirement is simply plain readable text in the Mutt window, the message body must be converted *before* it is read. Again, Mutt can be told to pre-process the message using a mailcap setting like the third one in Listing 1. Another approach, which works for a whole site, regardless of the particular client used, requires a procmail recipe [9, 10] like this one:

```
:Obf:
* ^Content-Type:[ ]*text/html
| w3m -dump -T text/html
```

The first line says that what follows must be applied to the body of the message. If its format is HTML (as checked by the second line) the whole body is piped through an HTML to text converter. The example uses `w3m`, but any other HTML to ASCII converter would work: here, the first switch tells it to just dump the formatted text and exit, the second that the input is in HTML format.

One obvious disadvantage of plain text conversion is that all or some hyperlinks are lost. Consider this snippet of HTML code:

```
...To know more, visit <A href=
="http://www.me.com>http://www
.me.com</A>, to subscribe now
click <A href="http://www.me.
com/subscribe">HERE</A>
```

In its plain text version, only the first URL will still be useable (manual cut and paste or the trick explained on the next page). If the message is still in HTML format (i.e. if you didn't use procmail to convert it), its URLs are not lost, however. `urlview` gives you a list of all the URLs contained in a message, and starts your browser on the one you select. To run `urlview` on the current

Listing 2: Show headers

```
#!/bin/sh
# htmlviewer (method suggested
by Gary Johnson)
# add to .muttrc the line:
pager=_name_of_this_script
script_copying_headers_inside_htm
l | w3m -T text/html
```

message, add these two macros to your `.muttrc`:

```
macro index \cb |urlview\n ⌘
'list URLs contained in message'
macro pager \cb |urlview\n ⌘
'list URLs contained in message'
```

and then just press CTRL-b when required.

Clickable email

Urlview is fast, but shows the URLs totally out of context. The real solution would be to convert the message to plain formatted text, but keeping the strings related to URLs “clickable”. This is possible using a text browser from Mutt in *interactive* mode: the second mailcap entry of Listing 1, with its “needsterminal” option, does just that. Figure 1 shows the result: proper formatting, colors, selected link underlined... all in the Mutt window (the black rectangles were manually added to hide the sender’s name). To start an external browser in another window, just select the link (“Worldwide”) and hit ESC-M.

Is this perfect? Almost. Email headers are not displayed because they are external to the HTML <BODY> tags, but we can use a simple script to find them and add them to the body of the message. Such a script could be run by procmail instead of w3m in the recipe above, or be used to build an HTML-aware Mutt pager as in Listing 2.

Remember, however, that the Mutt pager has other uses too (such as showing online help for instance), and an HTML browser might not handle them correctly.

Clickable URLs

There is another, completely different method to open URLs without the need for manual cutting and pasting, which is available for plain ASCII email. It can be extended to interact with the system in many other ways. There are command line programs that can echo whatever text is

Listing 3: Using xclip

```
01 #!/bin/bash
02 # contributed by Joel Hammer
03 #
04
05 a=`ps x | grep netscape | grep
-v grep`
06 [ -n "$a" ] && {
07 /usr/bin/netscape -remote
"openURL(`xclip -o`) " &>
/dev/null &
08 }
09 [ -z "$a" ] && {
10 /usr/bin/netscape "`xclip -
o`" &> /dev/null &
11 }
```

highlighted with the mouse to the standard output. One of these is `xclip` [11]. Once you have installed it along with the simple shell script in Listing 3, place the following macro in your `.muttrc`:

```
macro pager \cn "!start_browser⌘
.sh\n" 'open URL'
```

The `start_browser.sh` script launches your favorite browser (`/usr/bin/netscape` in this case) in another window, if needed. The URL it has to load is grabbed from the X clipboard and echoed by `xclip`.

Important: if the URL in the message doesn’t have the “`http://`” prefix the browser will see it as a local file and fail. The solution is to pipe the output of `xclip`

to a perl/awk one liner which adds the prefix if missing. Another thing to keep in mind is that on some distributions “netscape” is a shell wrapper around the browser executable: this may create quoting problems, and weird message errors.

In general, however, the main interest of this shell script is that `xclip` is in no way restricted to this, or to email, for that matter: Making mouse selections available to scripts opens up endless possibilities...

For example, all the remote browser control techniques described in “Client Side Web Scripting” could be applied here to just add the URL to your bookmark file, or send it straight to the printer.

Address management

Mutt can handle email aliases, as explained in its manual. We could define them directly inside `.muttrc`, but since there could be a lot of them, many people put them in a separate file, which is then referenced by the main file.

The alias file can be managed directly from within Mutt if you install the `mail2muttalias` Python script [12] and bind it to a pager macro like this:

```
macro pager A |'/CORRECT/PATH⌘
/HERE/mail2muttalias.py'\n
```

At this point, just hitting “A” when you are reading a message in the pager will

list all the addresses it contains, allow you to alias them, and write everything directly to the standard alias file. The script can be generalized to write to other addresses files (abook for example), or to prompt the user for further information, comments, etc.

If addresses only occurred in the body and headers of an email, that would be it, but often they are coded in the vcard attachment format. This can look like a real business card, but (by default) is unusable in text based

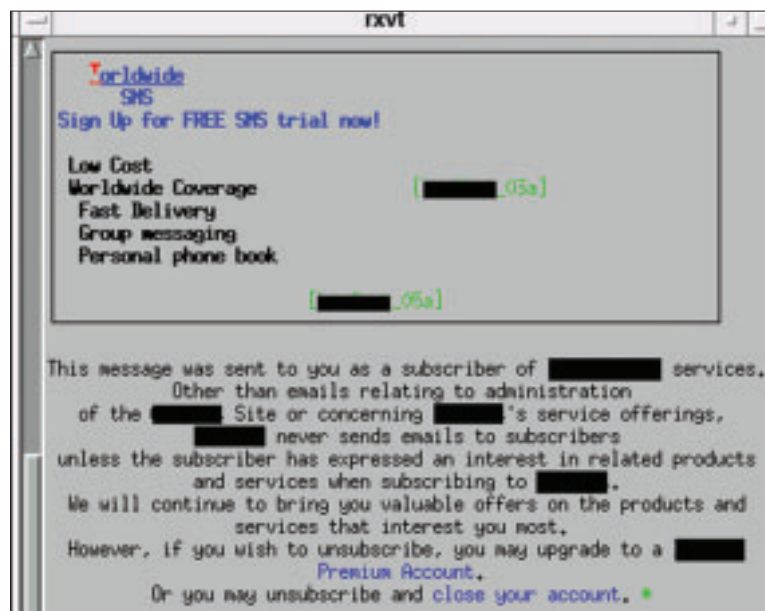


Figure 1: With the correct formatting

mailers. Luckily, a specific filter is available to read the content of these attachments [7]. Once it is placed in your path and made executable, just create a proper mailcap entry for the vcard file format:

```
text/x-vcard; mutt.vcard
.filter; copiousoutput
```

and add it to the `auto_view` list of attachments if you want mutt to display it automatically. Even this filter, being a simple Perl script, can be easily modified to write contact data to any address book format you might use.

The most efficient solution for general address management is `abook` [13], which stores email as well as snail mail addresses, phone numbers, etc. It runs on the command line but can be queried directly from within Mutt. In fact, the mailer, supports a general mechanism for external address queries through the `query_command` variable. A line like this in `.muttrc`:

```
set query_command="abook
--mutt-query '%s'"
```

will run `abook` inside Mutt (the default binding is “Q”) and let you select the people to which the message should be sent.

Advanced email filtering

Procmail is, without doubt, the default solution for sorting, filtering, and generally preprocessing incoming email: its home page, and its mailing list archives, contain plenty of useful examples.

There is also a nice tool, called TK-Red [14], which is perfect both for generating simple recipes, and, even more importantly, to help beginners learn the syntax.

At any rate, there are three limits to the standard procmail approach. To begin with, it must be set up separately; also it

occurs too late to save bandwidth (at least for dialup users), and, finally, it's static.

The first problem is easily solved with `mail2procmailrc`, another Perl script. If launched via a macro while you are reading a message, it will ask some questions directly in the Mutt window, and then write and save the corresponding procmail recipe. Another way to change the configuration file of procmail (or any other utility!) on the fly in Mutt is explained in the “No more Spam!” article at [15].

The second problem is even easier to define: if we are sure that a message must be deleted, why bother downloading it (especially if you have a slow, pay-per-minute dial-up connection)?

Freshmeat offers several tools capable of retrieving just the mail headers from a POP3 server, and then deleting those messages that match some predefined rules. `Animail` [16] is one of them, with a GUI for initial setup, but the less sophisticated `popfilter` script [4] can also be useful.

All spam is “unwanted” mail, but does that make the opposite true? No, and that's why static filtering doesn't work. Consider a generic mailing list to which you regularly subscribe.

A vanilla procmail recipe will save all the list email in its own folder, including threads in which you have no interest: imagine one hundred messages about multi-processor optimization when you only have one CPU. In Mutt you could tag the whole thread (ESC-t) and then delete it (“;d”), but we can do better.

This “thread-level” filtering should also be performed only for a few days after the thread appears, or it would do more harm than good in the long run. How long would it take to scan the POP3 server for all the irrelevant threads that appeared over the last two years?

The `popfilter` scripts show how to take care of this problem, and their approach can be used with other tools. The headers of an uninteresting message can still be extracted using `formail`:

```
macro pager X "|formail -Xfrom:
-XSubject: -Xto:|/usr/local
/bin/popfilter_format.pl\n"
```

but this time `popfilter_format.pl` prepends a time stamp to each entry before saving to a file as shown in Listing 4.

The `popfilter` script will then read this blacklist, and ask the POP3 server to delete only the messages with matching `From:` and `Subject:` headers, and a timestamp less than (for example) ten days old.

Summary and credits

I am grateful to Gary Johnson and to all the members of the `mutt`, `w3m` and `procmail` lists (specific credits are given in the complete configuration scripts) for being patient and sharing many secrets. I also hope to have given at least an idea of what Mutt macros, mailcap and the right utilities can do together. If you have similar tips and tricks that you would like share, feel free to drop me a line. ■

INFO

- [1] Converting Microsoft mailboxes: <http://mbx2mbox.sourceforge.net/>
- [2] Examples of `.muttrc` files: http://www.dotfiles.com/index.php3?app_id=27
- [3] Muttrc Builder: <http://mutt.netliberte.org/>
- [4] `popfilter` script: <http://www.rule-project.org/en/sw/>
- [5] Gary Johnson's Mutt Page: <http://www.spocom.com/users/gjohnson/mutt/>
- [6] 3rd party applications for Mutt: <http://www.symonds.net/~prahladv/mutt.php>
- [7] Dave Pearson's Mutt Page (and `mutt.vcard.filter`): <http://www.davep.org/mutt/>
- [8] `catdoc` converter: <http://www.ice.ru/~vitus/catdoc/catdoc.man.html>
- [9] Create procmail recipes on the fly: <http://www.linuxgazette.com/issue62/okopnik.html>
- [10] `mail2procmailrc`: <http://www.ghettohack.net/~timball/>
- [11] `xclip` source code: <http://packages.qa.debian.org/x/xclip.html>
- [12] `Mail2muttaliases`: <http://webrium.uni-mannheim.de/jura/moritz/mail2muttaliases.shtml>
- [13] `abook`: <http://abook.sourceforge.net>
- [14] `Tk-RED`: <http://naskita.com/linux/tk-red/tk-red.shtml>
- [15] No More Spam!: www.linuxgazette.com/issue62/okopnik.html
- [16] Delete email before download: <http://animail.sourceforge.net/>

Listing 4: time-limited blacklist

```
20010211 =>psyche-
list@redhat.com<= => Floppy
20010211 =>psyche-
list@redhat.com<= => Networks
```