百万人のMutt

～Mutt活用講座～

滝澤隆史
taki@cyber.email.ne.jp

第5回 日本語版での拡張機能

先日、「あるLinuxのディストリビューションのメーラにMuttの1.3系列が採用されている」という記事を読んだので、SRPMパッケージをダウンロードして展開してみたのですが、日本語パッチが当たったものではありませんでした。

基本的には、日本語パッチを当てなくても日本語が扱えるため、絶対に当てなければならないというものではないのですが、機種依存文字や生JISヘッダなどの日本語特有の問題に関しては、本家にフィードバックしていない（というか理解してもらえなくて拒否された）ため、実際問題、読めないメッセージが出てきます。しかし、そのディストリビューションでインストールされた日本語パッチの当たっていないMuttを使って「Muttって使えないじゃん」と言われるのは心外です。

そこで今回は、日本語パッチが何をしているのかを紹介したいと思います。いきなり問題点を解説しても読んでいて楽しいものではないので、前半は便利な拡張機能および関連する機能を紹介し、後半に日本語特有の問題を修正するために行っていることを紹介します。

本題に入る前に、Muttの最新状況について紹介します。

執筆時点での最新バージョンは1.3.22.1です。オリジナルの大きな変更点は`auto_view`の機能でメッセージを外部プログラムに渡す際に文字符号化方式の変換が行われなくなり、変換は外部プログラムに依存するようになったことです。そのため、表示がうまくいかない場合は`nkf`や`lv`などの文字符号化方式の変換を行なうフィルタプログラムを経由して外部プログラムを呼び出すように、`mailcap`ファイルを変更する必要があります。

日本語パッチでの大きな変更点は`./configure`のオプションとして、デフォルトで日本語の設定が行われる「`--enable-default-japanese`」を追加したことです。なお、この1.3.22.1は、次の正式版1.4のベータ版です。本誌が発売するころには1.4が出ているかもしれません。次号にて1.4の詳細をお伝えする予定です。

拡張機能

というわけで、まず日本語パッチの便利な拡張機能から紹介しましょう。

●プレフィックスなどの削除

日本のメーリングリストでは、

```
Subject: [prefix:0123] hogohoge
```

のように、Subjectフィールドにプレフィックスを付ける場合がほとんどです。しかし、プレフィックスが長いと、インデックス画面では肝心のSubjectの内容が行からはみ出て読めなくなってしまいます。そのような場合、次の設定を行うと、インデックス画面にプレフィックスが表示されなくなります。

```
set delete_prefix=yes
```

この設定を行うと、返信や転送する際にもプレフィックスは除去されるようになります。プレフィックスのパターンは、リスト1のように`$delete_regexp`を使って正規表現で設定することもできます（この例はデフォルト値）。

これをある特定のフォルダだけに適応するようにするには、`folder-hook`を使って設定します。例えば、あるメーリングリストで

```
Subject: foo_ml:0123: hogohoge
```

のようなプレフィックスを取り除くにはリスト2のように設定します。「`reset`」というコマンドは設定変数をデフォルト値に戻すコマンドです。

設定次第では、オリジナルのMuttにある設定変数`$reply_regexp`だけでも似たようなことができるので、チャレンジ精神のある方は試してみてはいかがでしょうか？

【リスト1】プレフィックスのパターンを$delete_regexpで設定する

```
set delete_regexp="^(\[[A-Za-z0-9_.: \-]*\][ ]*)"
```

【リスト2】特定のフォルダに対してプレフィックスを設定する

```
folder-hook . 'reset delete_regexp'
folder-hook foo_ml 'set delete_regexp="^[a-z_]*:[0-9]*:[ ]*"'
```

●ツリーに任意の文字を使う

スレッドやマルチパート構造の表示に使われるツリーには、ACS文字（罫線文字）が使われます（画面1）。しかし、一部の端末ではこのACS文字が使えないものもあります。そこでACS文字の代わりに、普通のASCII文字を使う変数`$ascii_chars`が用意されています（画面2）。しかし、このASCII文字のツリーはあまり美しくありません。そこで、日本語版ではツリーに使う文字を自由に設定できる`$tree_chars`を用意しました。JIS X 0208の罫線文字を使えばきれいなツリー表示ができます（画面3）。設定例を以下に示します。

```
set tree_chars=yes
set tree_llcorner="└"
set tree_ulcorner="┌"
set tree_ltee="├"
set tree_hline="─"
set tree_vline="│"
```

【画面1】ACS文字によるツリー表示

【画面2】$ascii_charsによるツリー表示

【画面3】$tree_charsによるツリー表示

【画面4】$pagers_hdrs_only（メッセージ選択時）

【画面5】$pagers_hdrs_only（ページ送り操作後）

【画面6】$pager_spoil（メッセージ選択時）

●表示の軽量化

Muttはtty端末で動作しますから、リモートで使っている人もいるでしょう。LANでつながっているような環境では、ローカルで動かしているのと同じくらいの感覚で使えますが、マシンが遠隔地にある場合は、どうしても表示が重く感じるでしょう。この場合は、画面の書き換えをいかに少なくするかが鍵になります。まず、オリジナルの設定変数で画面の書き換えに関係あるものを紹介します。

インデックス画面でカーソルを動かすと、通常は1ページごとにスクロールします。$menu_scrollを設定すると、インデックスが1行ずつスクロールするため表示が重くなります。そのため、この変数は設定しない方がよいです（デフォルト無効）。とはいっても、インデックスとページャを同時に表示している場合はインデックスは1行ずつスクロールします。

インデックスのカーソルは通常はカーソル行全体がハイライトされた状態で表示されます。そのため、カーソルを動かすたびにカーソル行全体の書き換えが発生します。$arrow_cursorを設定すると、カーソルの形状が「->」になり、通し番号の左側に表示されるようになります。そのため、カーソルの移動による書き換えが少なくなります。

```
set arrow_cursor=yes
```

次に日本語版で拡張された画面の書き換えに関係する設定変数を紹介します。

$pager_hdrs_onlyを設定すると、インデックスでメッセージを選択したときにページャにはヘッダのみ表示され、本文は表示されません（画面4）。このときにページ送りなどの操作をすると、本文が表示されるようになります（画面5）。本文を表示させない限りページャの書き換えは最小限になります。

```
set pager_hdrs_only=yes
```

また、さらに$pager_spoilを設定すると、メッセージを選択したときにページャには本文の文字を$pager_spoilerで設定した文字（デフォルトは「"*"」）に置き換えて表示されます（画面6）。このときにページ送りなどの操作をすると、本文が表示されるようになります。この機能は書き換えという意味では効果がないですが。

```
set pager_spoil=yes
set pager_spoiler="*"
```

以上の設定を、実際にどの程度の効果があるか筆者の環境で試してみました。ADSL接続している自宅から、ADSL接続している遠隔地のマシンにsshで接続し、そのマシン上でMuttを動かしてみたというものです。0.5Mbpsくらいの通信速度だと、表示が少しもたつく程度で操作にストレスを感じないためか、設定の違いはほとんど実感できませんでした[*1]。遠隔地にあるマシンの演算速度が遅いとか、通信速度がかなり遅いとかの環境でないとはっきりとした効果は出ないのかもしれません。

●MHフォルダの判定

日本語版で拡張された変数$mh_pathでMHフォルダのパスを指定すると、そのディレクトリ以下のMHフォルダがMHフォルダではないという誤判定を減らすことができます。設定例は次の通りです。

```
set mh_path=~/Mail
```

詳しくは、本誌10月号のコラム「Muttを歩けば棒に当たる」をお読みください。実装者の川口銀河さんによる詳しい解説があります。

●添付ファイルのファイル名

添付ファイルのファイル名について詳しくない方は、まずコラム「添付ファイルの日本語ファイル名」をお読みください。

US-ASCII以外の文字符号化方式のファイル名はRFC2231形式で符号化しなければなりませんが、多くのメーラはRFC2047形式（B encodingあるいはQ encoding）で符号化します。Muttはこの RFC2047形式のファイル名の付いた添付ファイル付きのメッセージを受け取っても、ファイル名の符号化の形式がRFC違反であるため標準では復号化しようとしません。しかし、これでは困ることが多いので、RFC2047形式で符号化されたファイル名を認識する設定変数$rfc2047_parameterが用意されています。

```
set rfc2047_parameters=yes
```

逆に、Muttが生成するファイル名はRFC2231形式によるものだけです。RFC2047形式のファイル名は生成しません。しかし、相手が認識できなければどうしようもないので、日本語パッチでは、コラムで紹介したRFC2231形式とRFC2047形式の併用型の添付ファイル名を生成する設定変数$create_rfc2047_parameterを用意しました（mutt-1.3.19i-ja0から）。

```
set create_rfc2047_parameters=yes
```

日本語パッチとは関係ない話ですが、添付ファイルを保存するときにファイル名を任意の文字符号化方式に変えたいと思うかも知れません。しかし、これはMuttではできません。いったん保存した後で、Samba日本語版の関連ツールのSMBCHARTOOL（記事末のResource [1]を参照）を使ってファイル名の変換をしてください。

日本語特有の問題

日本語特有の問題はすべて日本語の文字符号化方式によるものであるので、この辺りの話に詳しくない方はコラム「日本語の符号化文字集合と文字符号化方式」をお読みください。

●機種依存文字対策

先のコラムの項目の「文字符号化方式の変換に伴う問題」で説明していますが、受け取った

＊1　7年前に買った28.8kbpsのモデムを引っぱり出せば、また違う結果が出たかもしれませんが、そこまでして試そうとは思わなかったので。

メッセージに機種依存文字が含まれていると、iconvライブラリは文字符号化方式の変換に失敗します。

この対策として設定変数$sanitize_ja_charsを設けました。この変数を設定すると、iconvライブラリに渡す前に機種依存文字を「〓」(げた記号)に置き換えるようになります。ただし、冗長なことをやっているため数%程度遅くなっていると思います。設定例は次の通りです。

```
set sanitize_ja_chars=yes
```

また、この変数を設定すると、コラムの最後に説明したASCIIとJIS X 0201ラテン文字集合の違いによって発生する問題をISO-2022-JPのエスケープシーケンスを置き換えることにより回避します。

●生JIS問題

一部のメーラは、文字符号化方式を指定しないメッセージを生成します。ヘッダ・フィールドにMIME B encodingを行っていない生のISO-2022-JPの文字を記述したり、ISO-2022-JPでメッセージが記述されているにもかかわらずContent-Typeフィールドにcharsetパラメータを指定しなかったりするものです。また、一部のメーラが添付ファイルのファイル名に符号化していない生のISO-2022-JPの文字を付けてくるものがあります。

Muttは、「文字符号化方式の指定なきものはUS-ASCII」とみなすため、文字符号化方式の変換が行われず、文字化け状態になります。

この対策として$assumed_charsetにコロン区切りのリスト形式で変数を設定し、文字符号化方式の指定がないものに対して、その文字符号化方式を推定するようにしました。MIMEに厳格な仕様を崩すという意味も込めて$strict_mimeを「no」にしないと有効になりません。設定例をリスト3に示します。

実際に内部でやっていることは$assumed_charsetの先頭から文字符号化方式を順に取り出し、文字列をiconvライブラリに放り込んでその文字符号化方式から$charsetで指定したものに変換できるかを試し、変換できたらその文字符号化方式が正しいとみなすということです。

ヘッダフィールドと添付ファイル名はリストのすべてを試みますが、ボディに関しては先頭のものしか試さないので注意してください。なお、Content-TypeフィールドのcharsetパラメータにUS-ASCIIの指定がしてあって、実際のメッセージはISO-2022-JPである場合はMuttはcharsetパラメータの値を信用しますので、$assumed_charsetは有効に働きません。このときは「^E(キー操作でCtrl-E)」(edit-type)でcharsetパラメータの値を修正してください。

●添付ファイルの文字符号化方式

Windowsのメーラは、添付ファイルがテキストファイルであろうがなかろうが問答無用にBase64に符号化するものが多いようです。Muttの場合は添付ファイルがテキストファイルである場合にはそのファイルの文字符号化方式を$charsetで指定したものとみなして、送信用の文字符号化方式に変換します。しかし、日本語の場合はテキストファイルの文字符号化方式にはさまざまなものが使われるため、$charsetで指定した文字符号化方式以外の場合は問題が生じます。

この問題はテキストファイルの添付だけではなく、メッセージのMIMEカプセル化した転送の場合にも生じます。

この対策として先に紹介した$assumed_charsetと同じような働きをする$file_charsetを用意しました。この変数を設定すると、添付するテキストファイルの文字コードを推定します。設定例をリスト4に示します。

●旧形式のPGPメッセージ

Muttは、「旧形式の(PGP/MIMEではない)PGPメッセージはUS-ASCIIしか扱わず、他の文字符号化方式を使う場合はPGP/MIMEを使え」という仕様なので、文字符号化方式の変換は一切しません。そのため、表示のために文字符号化方式の変換を必要とする日本語のメッセージは表示できません。

この対策として、文字符号化方式の変換を行うようにしました。

●不必要なfoldingの問題

一部のメーラは、日本語の文字とASCII文字を含んだSubjectの文字列を符号化するときに、符号化した日本語の文字列(encoded-word)と符号化しないASCIIの文字列(text)との間でfolding(折り返し)を行うものがあります*2。例えば「これはLinux Japanの原稿です」というSubjectを符号化した際にリスト5のようになります。

RFC2047によると、encoded-word間のfoldingや空白文字は復号化する際に取り除くように記述されているので、Muttのページャはそのように動作しますが、それ以外のfoldingはそのまま表示されます。そのため、先の例のように不必要なfoldingがある場合は次のように表示されます。

```
Subject: これは
 Linux Japan
 の原稿です
```

これでは非常に読みにくいので、設定変数$strict_mimeを「no」にすることで、次のようにencoded-wordとtextとの間のfoldingと空白文字を取り除き、スペースに置き換えるようにしました。

```
Subject: これは Linux Japan の原稿です。
```

ただし、通常のtext間のfoldingはそのまま表示されます。

●文末のISO-2022-JP

作成したメッセージが改行コードで終わらずに日本語の文字で終わっている場合はメッセージの最後のISO-2022-JPのシフト状態がASCIIに戻りません。これはiconvライブラリがASCIIの文字に出会わない限りシフト状態をASCIIの戻さないためです。

この対策として作成したメッセージが改行コードで終わっていない場合は改行コードを文末に追加するようにしました。

最後に

ここで紹介した問題点は海外製の国際化されたメーラで日本語を扱う際に生じやすい問題だと思います。まあ、他にも問題点はあったのですが、すでにオリジナルにフィードバックされているので特に紹介はしませんでした*3。海外製のメーラの開発に参加している方の参考になれば幸いです。

【リスト3】$assumed_charsetを使った文字化け対策

```
set strict_mime=no
set assumed_charset="iso-2022-jp:euc-jp:shift_jis:utf-8"
```

【リスト4】$file_charsetを使った添付ファイルの文字符号化方式

```
set file_charset="iso-2022-jp:euc-jp:shift_jis:utf-8"
```

【リスト5】「これはLinux Japanの原稿です」を符号化した結果

```
Subject: =?iso-2022-jp?B?GyRCJDMkbCRPGyhC?=
 Linux Japan
 =?iso-2022-jp?B?GyRCJE44NjlGJEckORsoQg==?=
```

＊2　1行の文字数が76文字を越えない限り特にfoldingをする必要はありません。　＊3　RFC1468「Japanese Character Encoding for Internet Messages」に記述されていることに関して特別ルールとしてオリジナルのコードに追加しています。また、Mutt付属のワイド文字関数がワイド文字である日本語を扱えなかったという笑い話のような問題もあったのですが……。

Column 日本語の符号化文字集合と文字符号化方式

■符号化文字集合と文字符号化方式

厳密な定義を行おうとすると、いろいろ議論のあるものだとは思いますが、ここでは次のように定義しておきます。

符号化文字集合（CCS：Coded Character Set）とは文字集合（文字の集まりを定めたもの）に含まれる文字に対して1対1の数値を割り当てたものです。「ASCII」、「JIS X 0201」、「JIS X 0208」、「ISO 10646」、「Unicode」、「ISO 8859シリーズ」などがこれに相当します。

例えば、JIS X 0208の文字「漢」、「字」は、それぞれ「20区33点」、「27区90点」に割り当てられ、ASCIIの文字「K」、「A」、「N」、「J」、「I」は、それぞれ16進数で「4B」、「41」、「4E」、「4A」、「49」に割り当てられています。

文字符号化方式（CES：Character Encoding Scheme）とは1つ以上の符号化文字集合に含まれる文字に対して実際に使用する数値を対応させる方法のことです。「ISO-2022-JP」、「EUC-JP」、「UTF-8」などはこれに相当します。

例えば、先の例で示したJIS X 0208とASCIIの文字を組合わせた文字列「漢字KANJI」をISO-2022-JPで表記（16進数）すると、

```
1B 24 42 34 41 3B 7A 1B 28 42 4B 41 4E 4A 49
         漢    字          K  A  N  J  I
```

となります。ここで、「1B 24 42」、「1B 28 42」は、それぞれ「符号化文字集合をJIS X 0208、ASCIIに切り替える」ことを指示するエスケープシーケンスです。また、EUC-JPでは

```
B4 C1 BB FA 4B 41 4E 4A 49
 漢    字   K  A  N  J  I
```

となります。共にASCIIの文字の数値はそのままですが、JIS X 0208の文字は、その区点番号の数値にある演算（ISO-2022-JPの場合は16進数で20を加え、EUC-JPの場合はA0を加える）を施して符号化されていることが分かります。

このように2つの用語があるものの、欧米では「ASCII」、「ISO 8859シリーズ」のように、どちらにも使えるようなものがあるためか、使い分けが混乱しているようにも思えます。その混乱の1つに「charset」パラメータがあります。文字符号化方式の指定という意味では「encoding」にすべきだとは思うのですが、なぜか「charset」で定着しています。Muttの各種設定変数や文書でも同様の扱いとなっています。

- 日本語の符号化文字集合
  日本語の符号化文字集合には表1のものが使われています（制定年やバージョンは省略）。
- 日本語の文字符号化方式
  日本語が扱える文字符号化方式としては表2に示すものがあります。このうち、「ISO-2022-JP-1」と「ISO-2022-JP-2」は実質的に使われていません。「ISO-2022-JP-3」はIANAに登録されていないのでインターネットで使用してはいけません。「CP932」はWindowsで独自に拡張した文字を使っているためインターネットでは使用してはいけません。

■文字符号化方式の変換に伴う問題

日本語のメールで使われる文字符号化方式は通常はISO-2022-JPです。しかし、メッセージの作成時に用いられる文字符号化方式は大抵はShift_JISかEUC-JPであるため、メッセージ作成後にISO-2022-JPに変換する必要があります。

まず、ここで問題が生じます。Shift_JISのメッセージにはいわゆる半角カナ（JIS X 0201片仮名）が含まれている可能性があります。また、Windowsで使われている文字符号化方式は厳密にいえば、Shift_JISを独自に拡張したCP932であるため、作成したメッセージに機種依存文字（JIS X 0208の空き領域やShift_JISの保留域を勝手に使ったもの）が含まれている可能性があります。Macintoshでメッセージを作成した場合も同様に機種依存文字が含まれている可能性があります。これらの文字はISO-2022-JPで使用している符号化文字集合では扱えないため本来は変換はできないのですが、コード変換の演算によって無理やりISO-2022-JPに割り当ててしまうメーラがたくさんあります。これに起因するトラブルとして、演算によって無理やり変換するため、IBM選定IBM拡張漢字は8ビットコードになり、配送経路上でbase64やquoted-printableに変換されることがあります。なお、半角カナをそのまま送るメーラは最近はほとんどなくなったので、それによるトラブルはあまりないでしょう。

このようにして送られた機種依存文字を含んだメッセージを受け取った際に実害が生じます。ISO-2022-JPの文字列を表示するためにShift_JISやEUC-JPに変換する必要があるのですが、glibc 2.2のiconvライブラリやGNU libiconvなどのようにISO/IEC 10646ベースで変換を行なっている場合にはISO-2022-JPに定義されていない文字（機種依存文字）が含まれていると、ISO-2022-JPの文字列として認識されず、変換に失敗します。そうなると、メーラには単に文字化けした（というよりは変換していない）文字列が表示するということになります。

もう1つの問題はASCIIとJIS X 0201ラテン文字集合との違いです。この2つの符号化文字集合はほとんど同じですが、2つだけ異なり、ASCIIにおける「\」（バックスラッシュ）と「~」（チルダ）の位置にJIS X 0201では「¥」（円マーク）と「‾」（オーバーライン）が入っています。ISO-2022-JPはこの両方とも扱うことができるのですが、EUC-JPやShift_JISはそうではありません。EUC-JPはASCIIを、Shift_JISはJIS X 0201ラテン文字集合を扱えますが、その逆は扱えません*4。例えば、ISO-2022-JPの文字列中にJIS X 0201の「¥」が含まれている場合、その文字列をEUC-JPに変換する際に「¥」を変換することができません。実際には、文字コードの位置だけを取り出して変換しているケースが多いわけですが、ISO/IEC 10646ベースのiconvライブラリを使っている場合、内部処理上全く異なる文字として認識されます。そのため変換が失敗します。

（滝澤隆史）

【表1】日本語の符号化文字集合

規格名称	文字集合	説明
JIS X 0201	ラテン文字用図形文字集合	ASCIIとほとんど同じ。2文字だけ異なる
JIS X 0201	片仮名用図形文字集合	いわゆる半角カナと呼ばれる存在
JIS X 0208	漢字集合（第1水準、第2水準）	通常用いられている漢字
JIS X 0212	補助漢字集合	ほとんど使われていない
JIS X 0213	拡張漢字集合（第3水準、第4水準）	JIS X 0208で足りない文字を補うために拡張したもの
Unicode	Unicode	世界各地の文字集合を収容することを目的とした民間規格
ISO/IEC 10646	国際化符号化文字集合（UCS）	世界各地の文字集合を収容することを目的とした国際規格

【表2】日本語が扱える文字符号化方式

文字符号化方式	説明
ISO-2022-JP	ISO 2022を参考にしてASCII、JIS X 0201ラテン文字、JIS X 0208を扱えるようにしたもの。7ビットコード
ISO-2022-JP-1	ISO-2022-JPをJIS X 0212を扱えるように拡張したもの
ISO-2022-JP-2	ISO-2022-JPをJIS X 0212、GB2312、KSC5601、ISO-8859-1、ISO-8859-7を扱えるように拡張したもの
ISO-2022-JP-3	ISO-2022-JPをJIS X 0213を扱えるように拡張したもの
EUC-JP	ISO 2022に準拠してASCII、JIS X 0208、JIS X 0201片仮名、JIS X 0212を扱えるようにしたもの。UNIX系OSでよく使われている。
Shift_JIS	JIS X 0208をJIS X 0201と併用できる空間に押し込んだもの。パソコン用OSでのデファクトスタンダード
CP932	Shift_JISにNEC特殊文字とIBM拡張漢字を拡張したもの。Windowsで使われている（IANAには「Windows-31J」として登録）
UTF-8	Unicode/ISO 10646をASCII文字をそのまま扱えるように符号化したもの

＊4　EUC-JPでJIS X 0201ラテン文字集合を扱えるという話（[2]）もありますが、ここでの話はIANA character-setsに登録された情報およびGNU libiconvの動作によっています。

Column

Muttも歩けば棒に当たる

FreeBSD使いの私がMuttと出会ったのはもう3年以上前、PocketBSDと出会い、この貧弱なハードウェア環境で動くいいMUAはないかと悩んでいたときでした。mewはmuleの起動だけで14秒もかかるし、pineはいまいちしっくりこないし……と、ちょうどそのとき使っていたニュースリーダslrnのslangつながりでMuttに出会い、ほどなくメインの環境でもMuttを使うようになりました。

私の使い方は、受信は滝澤さんと同じくfetchmailとmaildropで、すべてMaildirで処理しています。`~/Maildir/`の下に約40個のMaildir形式のmailboxがあります。送信はpostfixにお任せしています。

■ Muttの売り

今までMuttを使ってきて、「ここら辺が売りかな？」と思う所を書いてみます。

●見た目がシンプルなのに機能てんこ盛り

未だにすべての機能を使い切れてはいません。使ったことがないというより把握できてないというか。たまに「こんなことできないかな〜」と思ってマニュアルを読んだりすると、そのたびに新たな発見があったりします。

●UNIXの勉強になる

Mutt自身はSMTPを話さないので、SMTPサーバをきちんと設定するか、SMTPサーバの代わりになってくれるプログラムを使う必要があります。私の場合、sendmailで苦労したり、私の無知からMLで迷惑をかけることもありました。もちろん勉強にはなりましたが。

今はよい時代で、SMTPサーバを選べばそれほど苦労なく使えると思いますが、やはり他のMUAと比べると、UNIXに対する理解が求められるところは多いと思います。PC-UNIXがはやり、ユーザー＝システム管理者ともいえる時代では、これは逆に長所だと思います。

●パイプ機能

キーボードマクロを使って簡単にパイプが使えます。私は、オムロンソフトの翻訳ソフト「翻訳魂」を呼び出すフィルタプログラムをpythonで書き（[3]）、`macro`でパイプコマンドとしてキーに割り当て、キー一発で翻訳できるようにしてます。`~/.muttrc`の記述は**リストA**のような感じです。設定後はCtrl-Yで起動します。

Muttには必須ともいえる「urlview」がもしなかったとしても、こんな感じで結構簡単に作れそうですね。パイプは、もっといろんな使い方ができるんじゃないかな〜と思います。

●LDAP対応

LDAPの機能はまだ全然使ってないのですが、ZaurusもLANにつながるようになったので、Zaurusとメールアドレスを共有できるんじゃないかと思ってます。というわけで現在LDAP勉強中で、その次はZaurusでLDAPクライアント作成。と、道は長いです。私の場合、アイデアに技術と時間が追い付かないことが多いので、完成するまで誰にも言わない予定だったのですが(^^;。

■ノートPCでMuttを賢く使うには

今までノートPCを使うことが多かったので、当然MuttもノートPC上で使ってきました。それなりに経験を積んだので、ノートPC上（というよりダイヤルアップ環境？）で使う際に有用と思われる事項を紹介しましょう。

●SMTPサーバを選ぶ

取りあえず必要そうな機能は、

・メールの配送を、指定するまで遅らせることができる
・勝手にDNSを引きにいかない

sendmailにもメールの配送を遅らせる機能はあるのですが、他の設定でかなり苦労しました。PocketBSDではマシンパワーの都合で、gn付属のgnspoolを使ったりしていました。私はまだ使った事ないんですが、今モバイルで使うならやっぱりNomailでしょうか。postfixでも問題無さそうですが、Nomailはドキュメントが日本語なので:-)。

なお、sendmailで死ぬほど苦労するのは時間の無駄なのでやめましょう。もちろんsendmail好きな人はsendmailで構いませんが。

●envelope fromの設定

ノートPCに限ったことではありませんが、「`envelope from`」はきちんと設定しましょう。サーバによっては、`envelope from`をチェックして、存在しないサーバからのmailを拒否することもあります。Muttで一番お手軽な対応は、`~/.muttrc`で

```
set envelope_from=yes
```

とすることだったりしますが、Mutt以外のMUAも使うという場合は、SMTPサーバの設定で対応した方がいい場合もあるでしょう。

私がPC-UNIXを使い始めた当時は`envelope from`に触れたドキュメントが少なくて、かなりはまりました（今もそんなに多くないかな……）。FreeBSDでは、問題を報告するときに「`send-pr`」というコマンドを使うのですが、送り先のfreebsd.orgが`envelope from`をチェックするサーバだったりして、思えばこれがはまりの第一歩でした。

●Maildirにしてrsyncを使う

ノートPCとデスクトップPCを併用する人は、mailboxをMaildirにして、rsyncなどを併用すると便利です。2台のマシンで同じPOPサーバにアクセスする場合、一般的には片方のマシンで「POPサーバにメールを残す」設定にしますよね。でも、mailboxがMaildirだと、rsyncでメールの同期ができるため、そのような設定にする必要はありません。

1.（もし希望するなら）外出前にrsyncで「デスクトップPC→ノートPC」の方向で同期
2.出先でノートPCで受信（POPサーバから削除）
3.外出先から戻って来たら「ノートPC→デスクトップPC」の方向で同期

こうしておくと、ノートPC側でもPOPサーバからメールを削除できて、POPサーバのメールスプールが溢れるのを心配しなくて済みます。これはPOPサーバのmailboxの容量が少ない場合、精神的にもよろしく快適です。特に数日にわたる出張や帰省時などにお勧めです。

難点は、Maildirの場合、`reply`などでファイルネームが変わるため、1.でノートPCに持ち込んだメールで`reply`などした後に3.の同期をとると、`reply`元のメールが重複して増えてしまう点と、rsyncのオプション指定方法がややこしい（一度決まったらシェルスクリプトにしましょう）、初心者はネットワークまわりの設定でつまずきやすい、ってことでしょうか。

というわけで多少問題はあるものの、この方法はそれなりに使い道があると思うんですが、どうでしょうか？　この方法を思い付いたときは、「これがUNIXというものか！」とちょっと感動したものです（今考えると、かなりずれてる気がしないでもない:-)。

●rsyncの設定

rsyncは、rshかsshを使います。というわけでまずrsh／sshを使える状態にしましょう。rshとsshのどちらを使うかですが、マシンパワーに問題がない限りsshの方がいいでしょう。rsyncも、現在ではセキュリティに気を遣うディストリビューションが増えたため、設定自体もsshの方がしやすいのではないかと思います。

ssh そのものの設定はここでは省略しますが、デスクトップPCでsshdが動いて、ノートPC側からsshでデスクトップPCにログインできればOKです。この場合、rsyncはノートPCにインストールしておきます。

両方とも`~/Maildir`以下にメールを格納していると仮定すると、**【実行例A】**と**【実行例B】**に示すようなコマンドをノートPC側で実行することになります。「`user@DesktopPC`」の部分は、デスクトップPC側での「ユーザー名@ホスト名」になります。これは自分の環境に合わせて下さい。動作が確認できたら、シェルスクリプトにしましょう。　（岩下洋治）

【リストA】パイプ機能を利用するための~/.muttrcの記述

```
macro pager \cy |~/bin/honyaku.py\ -m|jless\n
macro index \cy |~/bin/honyaku.py\ -m|jless\n
```

【実行例A】デスクトップPC→ノートPC方向の同期

```
# rsync -atuv -e ssh user@DesktopPC:Maildir/ ~/Maildir/
```

【実行例B】ノートPC→デスクトップPC方向の同期

```
# rsync -atuv -e ssh ~/Maildir/ user@DesktopPC:Maildir/
```

Column 添付ファイルの日本語ファイル名

■ファイル名の指定方法

添付ファイルのファイル名の付け方はRFC2183で定義されていて、「Content-Dispositionフィールドのfilenameパラメータに指定する」ことになっています。**リストB**に「example.jpeg」というJPEGファイルを添付したときのヘッダの例を示します。なお、実際にはヘッダに「Content-Transfer-Encoding」フィールドも付きますが、このコラムのリストでは省略しました。

ただ、MIMEの初版であるRFC1341では、「Content-Type: application/octet-stream」でファイル名としてnameパラメータを定義していた影響からか、1995年にContent-Dispositionフィールドの登場（RFC1806）によってnameパラメータは廃止されたにもかかわらず、現在でも**リストC**のように、ファイル名をnameパラメータに指定しているものを多く見かけます。

■日本語ファイル名の生成方法

添付ファイルに日本語の（というよりはUS-ASCII以外の）ファイル名を使う方法は、RFC2231に記述されています。例えば「ほごほげ.jpeg」というファイルを添付したときは**リストD**のようになります。このRFCが公開されたのはせいぜい4年前であり、主要なメーラの半数近くがその復号化に対応しているに過ぎず、まだ普及しているとは言いがたい状況です。そのため、このRFCが公開される以前から勝手に使われている、次の2つの方法も依然として使われています。

- 生のISO-2022-JP
- MIME B encoding（RFC2047）による符号化

1つ目の方法は、少ないながらもまだ見かけます。ISO-2022-JPの文字列の中に文法上意味のある文字（「"」や「\」など）が含まれているため、「ISO-2022-JPのシフト状態を考慮しないとパラメータの文字列を抜き出せない」という問題があります。

2つ目の方法は、現状で最もよく使われている方法で、**リストE**や**リストF**のようにして使われています。しかし、パラメータの値としてMIME B encodingを用いることはRFC2047で明確に禁止されています。

RFC2231によるファイル名を認識できるメーラが半数で、MIME B encodingによるファイル名を認識できるメーラがほとんどである現状を考えて、**リストG**のように、その両方を生成するメーラもあります。RFC2231に対応していないようなメーラはContent-Typeフィールドのnameパラメータでファイル名を解析するだろうという考えです。

添付ファイルの日本語ファイル名の処理に関係するRFCを表Aにまとめましたので参考にしてください。

（滝澤隆史）

【リストB】JPEGファイルexample.jpegを添付したときのヘッダ

```
Content-Type: image/jpeg
Content-Disposition: attachment; filename=example.jpeg
```

【リストC】nameパラメータで添付ファイルを指定した例

```
Content-Type: image/jpeg; name=example.jpeg
```

【リストD】RFC2231による添付ファイル名の符号化例

```
Content-Type: image/jpeg
Content-Disposition: attachment;
        filename*=iso-2022-jp''%1B$B$%5B$4$%5B$2%1B%28B%2Ejpeg
```

【リストE】RFC2047による添付ファイル名の符号化例（1）

```
Content-Type: image/jpeg; name="=?iso-2022-jp?B?GyRCJFskNCRbJDIbKEIuanBl?=
        =?iso-2022-jp?B?Zw==?="
```

【リストF】RFC2047による添付ファイル名の符号化例（2）

```
Content-Type: image/jpeg;
Content-Disposition: attachment; filename="=?iso-2022-jp?B?GyRCJFskNCRbJDIbKEIuanBl?=
        =?iso-2022-jp?B?Zw==?="
```

【リストG】添付ファイル名の符号化例（RFC2231とRFC2047の両方）

```
Content-Type: image/jpeg; name="=?iso-2022-jp?B?GyRCJFskNCRbJDIbKEIuanBl?=
        =?iso-2022-jp?B?Zw==?="
Content-Disposition: attachment;
        filename*=iso-2022-jp''%1B$B$%5B$4$%5B$2%1B%28B%2Ejpeg
```

【表A】添付ファイルの日本語ファイル名に関係するRFC

RFC No.	名称
RFC1341	MIME (Multipurpose Internet Mail Extensions): Mechanisms for Specifying and Describing the Format of Internet Message Bodies
RFC2047	MIME (Multipurpose Internet Mail Extensions) Part Three: Message Header Extensions for Non-ASCII Text
RFC1806	Communicating Presentation Information in Internet Messages: The Content-Disposition Header
RFC2183	Communicating Presentation Information in Internet Messages: The Content-Disposition Header Field
RFC2231	MIME Parameter Value and Encoded Word Extensions: Character Sets, Languages, and Continuations

Resource

[1] SMBCHARTOOL
http://www.samba.gr.jp/project/contrib/smbchartool.html

[2] 日本語EUCの定義
http://euc.jp/i18n/euc-jp.txt

[3] 翻訳魂pythonモジュール
http://www5.xdsl.ne.jp/~shuna/python/honyaku.html

- Mutt Japanese Edition
http://www.emaillab.org/mutt/