VM User's Manual

First Edition, VM Version 4

June 1989

Kyle E. Jones
kyle@cs.odu.edu

Introduction

VM (View Mail) is an Emacs subsystem that allows UNIX mail to be read and disposed of within Emacs. Commands exist to do the normal things expected of a mail user agent, such as generating replies, saving messages to folders, deleting messages and so on. There are other more advanced commands that do tasks like bursting and creating digests, message forwarding, and organizing message presentation according to various criteria.

To invoke VM simply type `M-x vm`. VM gathers any mail that has arrived in your system mailbox and appends it to a file known as your *primary inbox*, and visits that file for reading. See Chapter 1 [Starting Up], page 3. A file visited for reading by VM is called the *current folder*.

If there are any messages in the primary inbox, VM selects the first new or unread message, and previews it. *Previewing* is VM's way of showing you part of message and allowing you to decide whether you want to read it. See Section 2.1 [Previewing], page 5. By default VM shows you the message's sender, recipient, subject and date headers. Typing `SPC` (`vm-scroll-forward`) exposes the body of the message and marks the message as read. Subsequent `SPC`'s scroll forward through the message, `b` scrolls backward. When you reach the end of a message, typing a `SPC` or `n` moves you forward to preview the next message.

If you do not want to read a message that's being previewed, just type `n` and VM will move on to the next message (if there is one). See Chapter 2 [Selecting Messages], page 4.

To save a message to a mail folder use `s` (`vm-save-message`). VM will prompt you for the folder name in the minibuffer. See Chapter 4 [Saving Messages], page 8.

Messages are deleted by typing `d` (`vm-delete-message`) while previewing or reading them. The message is not deleted right away; it is simply marked for deletion. If you change your mind about deleting a message just select it and type `u` (`vm-undelete-message`), and the message will be undeleted. See Chapter 5 [Deleting Messages], page 10. The actual removal of deleted messages from the current folder is called *expunging* and it is accomplished by typing `#` (`vm-expunge-folder`). The message is still present in the on-disk version of the folder until the folder is saved.

Typing `h` (`vm-summarize`) causes VM to pop up a window containing a summary of contents of the current folder. The summary is presented one line per message, by message number, listing each message's author, date sent, line and byte count, and subject. Also various letters appear beside the message number to indicate that a message is new, unread, marked for deletion, etc. An arrow '`->`' appears to the left of the line summarizing the current message. The summary format is user configurable, see Chapter 9 [Summaries], page 14.

When you are finished reading mail the current folder must be saved, so that the next time the folder is visited VM will know which messages have been already read, replied to and so on. Typing `S` (`vm-save-folder`) expunges all deleted messages and saves the folder. `C-x C-s` saves the folder without expunging deleted messages but the messages are still marked deleted. The next time the folder is visited these messages will still be marked for deletion.

To quit VM you can type `q` (`vm-quit`) or `x` (`vm-quit-no-change`). Typing `q` expunges and saves the current folder before quitting. Also, any messages marked new are changed to be marked unread, before saving. The `x` command quits VM without expunging, saving

or otherwise modifying the current folder. Quitting is not required; you can simply switch to another Emacs buffer when you've finished reading mail.

At any time while reading mail in the primary inbox you can type *g* (`vm-get-new-mail`) to check to see if new mail has arrived. If new mail has arrived it will be merged into the primary inbox. If you are not in the middle of another message, VM will also jump to the first new message.

1 Starting Up

There are two ways to start VM: *M-x vm* and *M-x vm-visit-folder*.

M-x vm causes VM to gather any mail present in your system mailbox and append it to a file known as your *primary inbox*, creating this file if necessary. The default name of this file is `~/INBOX`, but VM will use whatever file is named by the variable `vm-primary-inbox`.

VM transfers the mail from the system mailbox to the primary inbox via a temporary file known as the *crash box*. The variable `vm-crash-box` names the crash box file. VM first copies the mail to the crash box, deletes the system mailbox, merges the crash box contents into the primary inbox, and then deletes the crash box. If the system or Emacs should crash in the midst of this transfer, any message not present in the primary inbox will be either in the system mailbox or the crash box. Some messages may be duplicated but no mail will be lost.

If the file named by `vm-crash-box` already exists when VM is started up, VM will merge that with the primary inbox before getting any new messages from the system mailbox.

By default, the location of the system mailbox is determined heuristically by VM based on what type of system you're using. VM can be told explicitly where the system mailbox is through the variable `vm-spool-files`. The value of this variable should be a list of strings naming files VM should try when searching for newly arrived mail. Multiple mailboxes can be specified if you receive mail in more than one place.

M-x vm-visit-folder (*v* from within VM) allows you to visit some other mail folder than the primary inbox. The folder name will be prompted for in the minibuffer.

Once VM has read the folder, the first new or unread message will be selected. If there is no such message, the first message in the folder is selected.

The variable `vm-startup-with-summary` controls whether VM automatically displays a summary of the folder's contents at startup. A value of `nil` gives no summary; a value of `t` gives a full screen summary. A value that is neither `t` nor `nil` splits the screen between the summary and the folder display. The latter only works if the variable `pop-up-windows`'s value is non-`nil`, and the value of `vm-mutable-windows` is non-`nil`. The default value of `vm-startup-with-summary` is `nil`.

The variable `vm-mail-window-percentage` tells VM what percentage of the screen should be given to the folder display when both it and the folder summary are being displayed. Note that Emacs enforces a minimum window size limit, so a very high or very low value for this variable may squeeze out one of the displays entirely. This variable's default value is 75, which works with Emacs' default minimum window size limit, on a 24 line terminal. Note that the value of `vm-mutable-windows` must be `t` or VM will not do window resizing regardless of the value of `vm-mail-window-percentage`.

A non-`nil` value for the variable `vm-inhibit-startup-message` disables the display of the VM's copyright, copying and warranty disclaimer. If you must, set this variable in your own `.emacs` file; don't set it globally for everyone. Users should be told their rights. The startup messages abort at the first keystroke after startup, so they do not impede mail reading.

2 Selecting Messages

The primary commands for selecting messages in VM are *n* (`vm-next-message`) and *p* (`vm-previous-message`). These commands move forward and backward through the current folder. When they go beyond the end or beginning of the folder they wrap to the beginning and end respectively. By default these commands skip messages marked for deletion. This behavior can be disabled by setting the value of the variable `vm-skip-deleted-messages` to `nil`. These commands can also be made to skip messages that have been read; set `vm-skip-read-messages` to `t` to do this. If all the messages in the current folder would be skipped (i.e all are read and/or deleted), *n* and *p* simply move to the next message.

The commands *n* and *p* also take prefix arguments that specify the number of messages to move forward or backward. If the magnitude of the prefix argument is greater than 1, no message skipping will be done regardless of the settings of the previously mentioned skip control variables.

The variable `vm-circular-folders` determines whether VM folders will be considered circular by various commands. *Circular* means VM will wrap from the end of the folder to the start and vice versa when moving the message pointer, deleting, undeleting or saving messages before or after the current message.

A value of `t` causes all VM commands to consider folders circular. A value of `nil` causes all of VM commands to signal an error if the start or end of the folder would have to be passed to complete the command. For movement commands, this occurs after the message pointer has been moved as far it can go. For other commands the error occurs before any part of the command has been executed, i.e. no moves, saves, etc. will be done unless they can be done in their entirety. A value other than `nil` or `t` causes only VM's movement commands to consider folders circular. Saves, deletes and undeletes will behave as if the value is `nil`. The default value of `vm-circular-folders` is `0`.

Other commands to select messages:

RET (`vm-goto-message`)
: Go to message number *n*. *n* is the prefix argument, if provided, otherwise it is prompted for in the minibuffer.

TAB (`vm-goto-message-last-seen`)
: Go to message last previewed or read.

N (`vm-Next-message`)
P (`vm-Previous-message`)
: Go to the next (previous) message, ignoring the settings of the skip control variables.

M-n (`vm-next-unread-message`)
M-p (`vm-previous-unread-message`)
: Move forward (backward) to the nearest new or unread message. If no such message exists then these commands work like *n* and *p*.

M-s (`vm-isearch-forward`)
: This works just like Emacs' normal incremental search except that when the search ends, VM selects the message containing point. If the value of the variable `vm-search-using-regexps` is non-`nil`, a regular expression may be

used instead of a fixed string for the search pattern. VM defaults to the fixed string search. See Section "Incremental Search" in *the GNU Emacs Manual*.

Selecting a message within VM normally causes VM to preview it. See Section 2.1 [Previewing], page 5.

2.1 Previewing

Previewing is VM's way of showing you a small portion of a message and allowing you to decide whether you want to read it. Typing `SPC` exposes the body of the message.

By default the sender, recipient, subject and date headers are shown when previewing; the rest of the message is hidden. This behavior may be augmented by the settings of two variables: `vm-visible-headers` and `vm-preview-lines`.

The value of `vm-preview-lines` should be a number that tells VM how many lines of the text of the message should be visible. The default value of this variable is 0. If `vm-preview-lines` is `nil`, then previewing is not done at all; when a message is first presented it is immediately exposed in its entirety and is marked as read.

The value of `vm-visible-headers` should be a list of regular expressions matching the beginnings of headers that should be made visible when a message is presented. The regexps should be listed in the preferred order of presentation for the headers they match.

Another variable of interest is `vm-highlighted-header-regexp`. The value of this variable should be a single regular expression that matches the beginnings of any header that should be presented in inverse video when previewing. For example, a value of '`"^From\\|^Subject"`' causes the From and Subject headers to be highlighted.

By default VM previews all messages, even if they have already been read. To have VM preview only those messages that have not been read, set the value of `vm-preview-read-messages` to `nil`.

3 Sending Messages

When sending messages from within VM, you will be using the standard Mail major mode provided with GNU Emacs. See Section "Mail Mode" in *the GNU Emacs Manual*. However, '`*mail*`' buffers created by VM have extra command keys:

C-c C-y
: Copies a message from the current folder into the '`*mail*`' buffer. The message number is read from the minibuffer. By default each line of the copy is prepended with the value of the variable `vm-included-text-prefix`. If a prefix argument is given, this prepending is not done.

C-c C-v <Any VM command key>
: All VM commands may be accessed in the '`*mail*`' buffer by prefixing them with C-c C-v.

The simplest command is *`m`* (`vm-mail`) which sends a mail message much as *`M-x mail`* does but allows the added commands described above.

3.1 Replying

VM has special commands that make it easy to reply to a message. When a reply command is invoked VM fills in the subject and recipient headers for you, since it is apparent to whom the message should be sent. You can change these headers manually if you wish.

VM also helps you quote material from a message to which you are replying by providing *included text* as a feature of some of the commands. *Included text* is a copy of the message being replied to with some fixed string prepended to each line so that included text can be distinguished from the text of the reply. The variable `vm-included-text-prefix` specifies what the prepended string will be.

The variable `vm-included-text-attribution-format` specifies the format for the attribution of included text. This attribution is a line of text that tells who wrote the text that is to be included; it will be inserted before the included text. If non-`nil`, the value of `vm-included-text-attribution-format` should be a string format specifiation similar to `vm-summary-format`. See Chapter 9 [Summaries], page 14. A `nil` value causes the attribution to be omitted.

The variable `vm-in-reply-to-format` specifies the format of the In-Reply-To header that is inserted into header section of the reply buffer. Like `vm-included-text-attribution-format`, `vm-in-reply-to-format` should be a string similar to that of `vm-summary-format`. A `nil` value causes the In-Reply-To header to be omitted.

The recipient headers generated for reply messages are normally made by simply copying the appropriate headers for the message to which you are replying. This includes any full name information, comments, etc. in these headers. If the variable `vm-strip-reply-headers` is non-`nil`, the reply headers will stripped of all information but the actual addresses.

The reply commands are:

r (`vm-reply`)
: Replies to the author of the current message.

R (`vm-reply-include-text`)
: Replies to the author of the current message and provides included text.

f (`vm-followup`)
: Replies to the all recipients of the current message.

F (`vm-followup-include-text`)
: Replies to the all recipients of the current message and provides included text.

All the reply commands mark the message to which you are responding as "replied" when the reply is actually sent.

3.2 Forwarding Messages

VM has two commands to forward messages: *z* (`vm-forward-message`) and `@` (`vm-send-digest`).

Typing *z* puts you into a '`*mail*`' buffer just like *m*, except the current message appears as the body of the message in the '`*mail*`' buffer. The forwarded message is surrounded by RFC 934 complaint message delimiters. If the variable `vm-rfc934-forwarding` is non-`nil` `"^-"` to `"- -"` character stuffing is done to the forwarded message (this is the default). This behavior is required if the recipient of the forwarded message wants to use a RFC 934 standard bursting agent to access the message. If the variable `vm-forwarding-subject-format` is non-`nil` it should specify the format of the Subject header of the forwarded message. This subject will be used as the contents of the Subject header automatically inserted into the '`*mail*`' buffer. A `nil` value causes the Subject header to be left blank.The command `@` (`vm-send-digest`) works like *z* except that a digest of all the messages in the current folder is made and inserted into the '`*mail*`' buffer.

4 Saving Messages

Mail messages are normally saved to files that contain only mail messages. Such files are called *folders*.

The VM command to save a message to a folder is `s` (`vm-save-message`); invoking this command causes the current message to be saved to a folder whose name you specify in the minibuffer. If `vm-save-message` is given a prefix argument *n*, the current message plus the next *n-1* message are saved. If *n* is negative, the current message and the previous *n-1* messages are saved. Messages saved with `vm-save-message` are marked "filed".

If the value of the variable `vm-confirm-new-folders` is non-`nil`, VM will ask for confirmation before creating a new folder on interactive saves.

If you have a directory where you keep all your mail folders, you should set the variable `vm-folder-directory` to point to it. If this variable is set, `vm-save-message` will insert this directory name into the minibuffer before prompting you for a folder name; this will save you some typing.

Another aid to selecting folders in which to save mail is the variable `vm-auto-folder-alist`. The value of this variable should be a list of the form,

```
((header-name
  (regexp . folder-name) ...)
 ...)
```

where *header-name* and *regexp* are strings, and *folder-name* is a string or an s-expression that evaluates to a string.

If any part of the contents of the message header named by *header-name* is matched by the regular expression *regexp*, VM will evaluate the corresponding *folder-name* and use the result as the default when prompting for a folder to save the message in. If the resulting folder name is a relative pathname it resolves to the directory named by `vm-folder-directory`, or the `default-directory` of the currently visited folder if `vm-folder-directory` is nil.

When *folder-name* is evaluated, the current buffer will contain only the contents of the header named by *header-name*. It is safe to modify this buffer. You can use the match data from any '`\( ... \)`' grouping constructs in *regexp* along with the function buffer-substring to build a folder name based on the header information.

All `vm-auto-folder-alist` matching is case sensitive.

VM can save messages to a folder in two distinct ways. The message can be appended directly to the folder on disk, or the folder can be visited as Emacs would visit any other file and the message be appended to that buffer. In the latter method you must save the buffer yourself to change the on-disk copy of the folder. The variable `vm-visit-when-saving` controls which method is used. A `nil` value (the default) causes VM to append directly to the folder file, a non-`nil` value makes VM load the file into a buffer and append to that.

After a message is saved to a folder, the usual thing to do next is to delete it. If the variable `vm-delete-after-saving` is non-`nil` VM will mark messages for deletion automatically after saving them. This applies only to saves to folders, not for the `w` command (see below).

Other commands:

w (`vm-save-message-sans-headers`)
: Saves a message or messages to a file without their headers. This command responds to a prefix argument exactly as `vm-save-message` does. Messages saved this way are *not* marked as filed, as "filed" is meant to mean saved to a folder. You should *not* use this command to save to mail folders.

A (`vm-auto-archive-messages`)
: Save all unfiled messages that auto-match a folder via `vm-auto-folder-alist` to their appropriate folders.

`|` (`vm-pipe-message-to-command`)
: Runs a shell command with the some or all of the current message as input. By default the entire message is used.

  If invoked with one `C-u` the text portion of the message is used.
  If invoked with two `C-u`'s the header portion of the message is used.

  If the shell command generates any output, it is displayed in a '`*Shell Command Output*`' buffer. The message is not altered or marked as filed.

5 Deleting Messages

In VM messages are marked for deletion, and then are subsequently *expunged* or removed from the folder. The messages are not removed from the on-disk copy of the folder until the folder is saved.

d (`vm-delete-message`)
: Marks the current message for deletion. A prefix argument *n* causes the current message and the next *n-1* messages to be marked. A negative *n* causes the current message and the previous *n-1* messages to be marked.

u (`vm-undelete-message`)
: Removes the deletion mark from the current message. A prefix argument *n* causes the current message and the next *n-1* messages to be unmarked. A negative *n* causes the current message and the previous *n-1* messages to be unmarked.

k (`vm-kill-subject`)
: Marks all message with the same subject as the current message (ignoring "Re:") for deletion.

(`vm-expunge-folder`)
: Does the actual removal of messages marked for deletion in the current folder.

Setting the variable `vm-move-after-deleting` non-`nil` causes VM to move past the messages after marking them for deletion.

6 Undoing

VM provides a special form of undo which allows message attribute changes to be undone.

Typing `C-x u` or `C-_` (`vm-undo`) undoes the last attribute change. Consecutive `vm-undo`'s undo further and further back. Any intervening command breaks the undo chain, after which the undos themselves become undoable by subsequent invocations of `vm-undo`.

Note that expunges and saves are *not* undoable.

7 Grouping Messages

In order to make numerous related messages easier to cope with, VM provides the command `G` (`vm-group-messages`), which groups all message in a folder according to some criterion. *Grouping* causes messages that are related in some way to be presented consecutively. The actual order of the folder is not altered; the messages are simply numbered and presented differently. Grouping should not be confused with sorting; grouping only moves messages that occur later in the folder backward to "clump" with other related messages.

The grouping criteria currently supported are:

'`subject`' Messages with the same subject (ignoring "Re:" prefixes) are grouped together.

'`author`' Messages with the same author are grouped together.

'`date-sent`'
Messages sent on the same day are grouped together.

'`arrival-time`'
Message presentation reverts to arrival time ordering (the default).

If the variable `vm-group-by` has a non-`nil` value it specifies the default grouping that will be used for all folders. So if you like having your mail presented to you grouped by subject, then put `(setq vm-group-by "subject")` in your `.emacs` file to get this behavior.

8 Reading Digests

A *digest* is one or more mail messages encapsulated in a single message.

VM supports digests by providing a command to "burst" them into their individual messages. These messages can then be handled like any other messages under VM.

The command `*` (`vm-burst-digest`) bursts a digest into its individual messages and appends them to current folder. These messages are then assimilated into the current folder using the default grouping. See Chapter 7 [Grouping Messages], page 12. The original digest message is not altered, and the messages extracted from it are not part of the on-disk copy of the folder until a save is done.

9 Summaries

Typing `h` (`vm-summarize`) causes VM to display a summary of contents of the current folder. An arrow '`->`' appears to the left of the line summarizing the current message. The information in the summary is automatically updated as changes are made to the current folder.

The variable `vm-summary-format` controls the format of each message's summary. Its value should be a string. This string should contain printf-like "%" conversion specifiers which substitute information about the message into the final summary.

Recognized specifiers are:

a - attribute indicators (always three characters wide)
The first char is 'D', 'N', 'U' or ' ' for deleted, new, unread and read message respectively.
The second char is 'F' for filed (saved) messages.
The third char is 'R' if the message has been replied to.
c - number of characters in message (ignoring headers)
d - date of month message sent
f - author's address
F - author's full name (same as f if full name not found)
h - hour message sent
i - message ID
l - number of lines in message (ignoring headers)
m - month message sent
n - message number
s - message subject
w - day of the week message sent
y - year message sent
z - timezone of date when the message was sent

Use "%%" to get a single "%".

A numeric field width may be specified between the "%" and the specifier; this causes right justification of the substituted string. A negative field width causes left justification. The field width may be followed by a "." and a number specifying the maximum allowed length of the substituted string. If the string is longer than this value, it is truncated.

The summary format need not be one line per message but it must end with a newline, otherwise the message pointer will not be displayed correctly in the summary window.

You can have a summary generated automatically at startup, see Chapter 1 [Starting Up], page 3.

All VM commands are available in the summary buffer just as they are in the folder buffer itself. If you set `vm-follow-summary-cursor` non-`nil`, VM will select the message under the cursor in the summary window before executing commands that operate on the current message. Note that this occurs *only* when executing a command from the summary buffer window.

10 Miscellaneous

Here are some VM customization variables that don't really fit into the other chapters.

vm-berkeley-mail-compatibility
: A non-`nil` value means to read and write BSD *Mail(1)* style Status: headers. This makes sense if you plan to use VM to read mail archives created by *Mail*.

vm-gargle-uucp
: A non-`nil` value means to use a crufty regular expression that does surprisingly well at beautifying UUCP addresses that are substituted for %f as part of summary and attribution formats.

vm-mode-hooks
: A non-`nil` value should be a list of hook functions to run when a buffer enters vm-mode. These hook functions should generally be used to set key bindings and local variables. Mucking about in the folder buffer is certainly possible, but it is not encouraged.

vm-folder-type
: This variable specifies the type of mail folder VM should expect to read and write. `Nil` means expect the UNIX style folders characterized by the "\n\nFrom " message separators. The only other supported value for this variable is the symbol `mmdf` which causes VM to use "^A^A^A^A\n" MMDF style leaders and trailers.

vm-delete-empty-folders
: A non-`nil` value for this variable causes VM to remove empty (zero length) folder files after saving them.

vm-mutable-windows
: This variable's value controls VM's window usage. A value of `t` gives VM free run of the Emacs display; it will commandeer the entire screen for its purposes. A value of `nil` restricts VM's window usage to the window from which it was invoked. VM will not create, delete, or use any other windows, nor will it resize it's own window. A value that is neither `t` nor `nil` allows VM to use other windows, but it will not create new ones, or resize or delete the current ones.

Key Index

Z

Command Index

Variable Index

Short Contents

Table of Contents