VM User's Manual

Second Edition, VM Version 5

June 1991

Kyle E. Jones
kyle@uunet.uu.net

License

GNU GENERAL PUBLIC LICENSE

Version 1, February 1989

Preamble

The license agreements of most software companies try to keep users at the mercy of those companies. By contrast, our General Public License is intended to guarantee your freedom to share and change free software—to make sure the software is free for all its users. The General Public License applies to the Free Software Foundation's software and to any other program whose authors commit to using it. You can use it for your programs, too.

When we speak of free software, we are referring to freedom, not price. Specifically, the General Public License is designed to make sure that you have the freedom to give away or sell copies of free software, that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of a such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must tell them their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS

1. This License Agreement applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any work containing the Program or a portion of it, either verbatim or with modifications. Each licensee is addressed as "you".
2. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish

on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this General Public License and to the absence of any warranty; and give any other recipients of the Program a copy of this General Public License along with the Program. You may charge a fee for the physical act of transferring a copy.

3. You may modify your copy or copies of the Program or any portion of it, and copy and distribute such modifications under the terms of Paragraph 1 above, provided that you also do the following:
   - cause the modified files to carry prominent notices stating that you changed the files and the date of any change; and
   - cause the whole of any work that you distribute or publish, that in whole or in part contains the Program or any part thereof, either with or without modifications, to be licensed at no charge to all third parties under the terms of this General Public License (except that you may choose to grant warranty protection to some or all third parties, at your option).
   - If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the simplest and most usual way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this General Public License.
   - You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

   Mere aggregation of another independent work with the Program (or its derivative) on a volume of a storage or distribution medium does not bring the other work under the scope of these terms.
4. You may copy and distribute the Program (or a portion or derivative of it, under Paragraph 2) in object code or executable form under the terms of Paragraphs 1 and 2 above provided that you also do one of the following:
   - accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Paragraphs 1 and 2 above; or,
   - accompany it with a written offer, valid for at least three years, to give any third party free (except for a nominal charge for the cost of distribution) a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Paragraphs 1 and 2 above; or,
   - accompany it with the information you received as to where the corresponding source code may be obtained. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form alone.)

   Source code for a work means the preferred form of the work for making modifications to it. For an executable file, complete source code means all the source code for all modules it contains; but, as a special exception, it need not include source code for modules which are standard libraries that accompany the operating system on which the executable file runs, or for standard header files or definitions files that accompany that operating system.

5. You may not copy, modify, sublicense, distribute or transfer the Program except as expressly provided under this General Public License. Any attempt otherwise to copy, modify, sublicense, distribute or transfer the Program is void, and will automatically terminate your rights to use the Program under this License. However, parties who have received copies, or rights to use copies, from you under this General Public License will not have their licenses terminated so long as such parties remain in full compliance.
6. By copying, distributing or modifying the Program (or any work based on the Program) you indicate your acceptance of this license to do so, and all its terms and conditions.
7. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein.
8. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

   Each version is given a distinguishing version number. If the Program specifies a version number of the license which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of the license, you may choose any version ever published by the Free Software Foundation.
9. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

10. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.
11. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT

LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

Appendix: How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to humanity, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

```
one line to give the program's name and a brief idea of what it does.
Copyright (C) 19yy  name of author

This program is free software; you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation; either version 1, or (at your option)
any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program; if not, write to the Free Software
Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.
```

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

```
Gnomovision version 69, Copyright (C) 19yy name of author
Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type `show w'.
This is free software, and you are welcome to redistribute it
under certain conditions; type `show c' for details.
```

The hypothetical commands 'show w' and 'show c' should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than 'show w' and 'show c'; they could even be mouse-clicks or menu items—whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the program, if necessary. Here a sample; alter the names:

```
Yoyodyne, Inc., hereby disclaims all copyright interest in the
program `Gnomovision' (a program to direct compilers to make passes
at assemblers) written by James Hacker.

signature of Ty Coon, 1 April 1989
Ty Coon, President of Vice
```

That's all there is to it!

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 9. 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 3.1 [Previewing], page 13. 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 flags the message as read. Subsequent `SPC`'s scroll forward through the message, `b` or `DEL` scrolls backward. When you reach the end of a message, typing `SPC` or `n` moves you forward to preview the next message. See Section 3.2 [Paging], page 14.

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 11.

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 5 [Saving Messages], page 18.

Messages are deleted by typing `d` (`vm-delete-message`) while previewing or reading them. The message is not deleted right away; it is simply flagged 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 6 [Deleting Messages], page 20. 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, flagged for deletion, etc. An arrow '`->`' appears to the left of the line summarizing the current message. The summary format is user configurable, see Chapter 12 [Summaries], page 26.

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 flagged deleted. The next time the folder is visited these messages will still be flagged 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 flagged new are changed

to be flagged 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 moved from the system spool area and merged into the primary inbox. If you are not in the middle of another message, VM will also jump to the first new message, if you are not in the midst of reading another message.

If `vm-get-new-mail` is given a prefix argument, it will prompt for another file from which to gather messages instead of the usual spool files. In this case the source folder is copied but not deleted.

1 Starting Up

There are three ways to start VM: *M-x vm*, *M-x vm-visit-folder* and `vm-mode`. The first time VM is started in an Emacs session (by any of these methods), it attempts to load the file `~/.vm`. If present this file should contain Lisp code, much like the `.emacs` file. Since VM has in excess of forty configuration variables, use of the `~/.vm` can considerably reduce clutter in the `.emacs` file. You can force the reloading of this file on demand by typing *L* from within VM.

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 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. The value of `vm-spool-files` will be inherited from the shell environmental variables MAILPATH or MAIL if either of these variables are defined.

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.

M-x vm-mode can be used on a buffer already loaded into Emacs to put it into the VM major mode so that VM commands can be executed from within it. This command is suitable for use in Lisp programs, and for inclusion in `auto-mode-alist` to automatically start VM on a file based on a particular filename suffix. `vm-mode` foregoes some of VM's startup procedures (e.g. starting up a summary) to faciliate noninteractive use.

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 flagged 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.

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 deletions, 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*.

3 Reading Messages

Once a message has been selected, VM will present it to you. By default presentation is done in two stages: *previewing* and *paging*.

3.1 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, and from there you can repeatedly type `SPC` to page through 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 altered by changing the settings of two variables: `vm-visible-headers`, `vm-invisible-header-regexp` 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 flagged 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 presentation order of the headers they match.

If non-`nil`, the variable `vm-invisible-header-regexp` specifies what headers should *not* be displayed. Its value should be a string containing a regular expression that matches all headers you do not want to see. Setting this variable non-`nil` implies that you want to see all headers not matched by it; therefore the value of `vm-visible-headers` is only used to determine the order of the visible headers in this case. Headers not matched by `vm-invisible-header-regexp` or `vm-visible-headers` are displayed last.

If you change the value of either `vm-visible-headers` or `vm-invisible-header-regexp` is the middle of a VM sesssion the effects will not be immediate. You will need to use the command `vm-discard-cached-data` on each message (bound to `j` by default) to force VM rearrange the message headers. A good way to do this is to mark all the messages in the folder and apply `vm-discard-cached-data` to the marked messages. See Chapter 8 [Message Marks], page 22.

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`.

Typing `t` (`vm-expose-hidden-headers`) causes VM toggle between exposing and hiding headers that would ordinarily be hidden.

3.2 Paging

Typing SPC during a message preview exposes the body of the message. If the message was new or previously unread, it will be flagged "read". At this point you can use SPC to scroll forward, and *b* or DEL to scroll backward a windowful of text at a time. Typing space at the end of a message moves you to the next message. If the value of `vm-auto-next-message` is `nil`, SPC will not move to the next message; you must type *n* explicitly.

If the value of `vm-honor-page-delimiters` is non-`nil`, VM will recognize and honor page delimiters. This means that when you scroll through a document, VM will display text only up to the next page delimiter. Text ater the delimiter will be hidden until you type another SPC, at which point the text preceding the delimiter will becom hidden. The Emacs variable `page-delimiter` determines what VM will consider to be a page delimiter.

You can "unread" a message (so to speak) by typing *U* (`vm-unread-message`). The current message will be flagged unread.

4 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 (`vm-yank-message`)
: 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`. All message headers are yanked along with the text. Point is left before the inserted text, the mark after. Any hook functions bound to mail-yank-hooks are run, aftert inserting the text and setting point and mark. If a prefix argument is given, this tells VM to ignore mail-yank-hooks, don't set the mark, don't prepend the value of vm-included-text-prefix to every yanked line, and don't yank any headers other than those specified in vm-visible-headers/vm-invisible-headers.

C-c y (`vm-yank-message-other-folder`)
: Work like `vm-yank-message`, but it first prompts for the name of a folder from which to yank the message.

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.

`vm-mail` can be invoked outside of VM by typing *`M-x vm-mail`*. However, of the above commands, only `C-c y` (`vm-yank-message-other-folder`) will work; all the other commands require a parent folder.

If you send a message and it is returned by the mail system because it was undeliverable, you an easily resend the message by typing *`C-r`* (`vm-resend-bounced-message`). VM will extract the old message and its pertinent headers from the returned message, and place you in a '`*mail*`' buffer. You can then change the recipient addresses or do whatever is necessary to correct the original problem and resend the message.

4.1 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 and what the subject should be. There is an old convention of prepending the string '`"Re: "`' to the subject of replies if the string isn't present already. VM supports this indirectly by providing the variable `vm-reply-subject-prefix`. Its value should be a string to prepend to the subject of replies, if the said string isn't present already. A `nil` value means don't prepend anything to the subject (this is the default). In any case you can edit any of the message 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 12 [Summaries], page 26. 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 created 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.

These commands all accept a numeric prefix argument *n*, which if present, causes VM to reply to the next (or previous if the argument is negative) *n-1* message as well as the current message. Also all the reply commands set the "replied" attribute of the messages to which you are responding, but only when the reply is actually sent. The reply commands can also be applied to marked messages, see Chapter 8 [Message Marks], page 22.

If you are one of multiple recipients of a message and you use *f* and *F*, your address will be included in the recipients of the reply. You can avoid this by judicious use of the variable `vm-reply-ignored-addresses`. Its value should be a list of regular expressions that match addresses that VM should automatically remove from the recipient headers of replies.

4.2 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 forwarded message is flagged "forwarded".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. Also, `vm-send-digest` can be applied to marked messages. See Chapter 8 [Message Marks], page 22. When applied to marked messages, `vm-send-digest` will only bundle marked messages, as opposed to the usual bundling of all messages in the current folder. If you give `vm-send-digest` a prefix argument, VM will insert a list of preamble lines at the beginning of the digest, one line per digestified message. The variable `vm-digest-preamble-format` determines the format of the preamble lines. If the value of `vm-digest-center-preamble` is non-`nil`, the preamble lines will be centered.

5 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 flagged "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. If the result of evaluating *folder-name* is a list, then the list will be treated as another auto-folder-alist and will be descended recursively.

Whether matching is case sensitive depends on the value of the variable `vm-auto-folder-case-fold-search`. A non-`nil` value makes matching case insensitive. The default value is `t`, which means matching is case sensitive. Note that the matching of header names is always case insensitive becasue RFC 822 specifies that header names are case indistinct.

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 value of `t` cuases VM to always visit a folder before saving message to it. A `nil` value causes VM to always append directly to the folder file.

In this case VM will not save messages to the disk copy of a folder that is being visited. This restriction is necessary to insure that the buffer and on-disk copies of the folder are consistent. If the value of *vm-visit-when-saving* is not `nil` and not `t` (e.g. 0, the default), VM will append to the folder's buffer if the buffer is currently being visited, otherwise VM will append to the file itself.

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 flag 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 flagged "written".

A (`vm-auto-archive-messages`)
: Save all unfiled messages that auto-match a folder via `vm-auto-folder-alist` to their appropriate folders. Messages that are flagged for deletion are not saved by this command. If invoked with a prefix argument, confirmation will be requested for each save.

`|` (`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 itself is not altered.

6 Deleting Messages

In VM, messages are flagged 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`)
: Flags the current message for deletion. A prefix argument *n* causes the current message and the next *n-1* messages to be flagged. A negative *n* causes the current message and the previous *n-1* messages to be flagged.

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

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

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

Setting the variable `vm-move-after-deleting` non-`nil` causes VM to move past the messages after flagging them for deletion. Setting `vm-move-after-undeleting` non-`nil` causes similar movement after undeletes.

7 Editing Messages

To edit a message, type `e` (`vm-edit-message`). The current message is copied into a temporary buffer, and this buffer is selected for editing. The major mode of this buffer is controlled by the variable `vm-edit-message-mode`. The default is Text mode.

Use *C-c ESC* (`vm-edit-message-end`) when you have finished editing the message. The message will be inserted into its folder, replacing the old version of the message. If you want to quit the edit without your edited version replacing the original, use *C-c C-]* (`vm-edit-message-abort`), or you can just kill the edit buffer with *C-x k* (`kill-buffer`).

If you give a prefix argument to `vm-edit-message`, then the current message will be flagged unedited.

As with VM '`*mail*`' buffers, all VM commands can be accessed from the edit buffer through the command prefix *C-c C-v*.

8 Message Marks

VM provides general purpose *marks* that may be applied to any and all messages within a given folder. Certain VM commands can be subsequently invoked only on those message that are marked.

To mark the current message, type `C-c C-@` (`vm-mark-message`). If you give a numeric prefix argument *n*, the next *n-1* messages will be marked as well. A negative prefix argument means mark the previous *n-1*. An asterisk ('`*`') will appear to the right of the message numbers of all marked messages in the summary window.

To remove a mark from the current message, use `C-c SPC` (`vm-unmark-message`. Prefix arguments work as with `vm-mark-message`.

Use `C-c C-a` to mark all message in the current folder; `C-c a` removes marks from all messages.

To apply a VM command to all marked message you must prefix it with the key sequence `C-c RET` (`vm-next-command-uses-marks`). The next VM command will apply to all marked messages, provided the command can be applied to such messages in a meaningful and useful way. The current commands that can be applied to marked message are: `vm-delete-message`, `vm-discard-cached-data`, `vm-followup`, `vm-followup-include-text`, `vm-reply`, `vm-reply-include-text`, `vm-save-message`, `vm-save-message-sans-headers`, `vm-send-digest`, `vm-undelete-message`, and `vm-unread-message`.

9 Undoing

VM provides a special form of undo which allows changes to message attributes 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, saves and message edits are *not* undoable.

10 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.

'`recipient`'
Mesage with the same recipients are grouped together.

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

'`physical-order`'
Message presentation reverts to physical message order of the foler (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.

11 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 10 [Grouping Messages], page 24. 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.

If you give a prefix argumet to `vm-burst-digest`, it will attempt to cope with non-RFC 934 compliant digests. If `vm-burst-digest` seems to be breaking digests at inappropriate places, most likely the digest is not compliant with the standard. In this case try using the prefix arg.

12 Summaries

Typing *h* (`vm-summarize`) causes VM to display a summary of contents of the current folder. The information in the summary is automatically updated as changes are made to the current folder. An arrow '`->`' appears to the left of the line summarizing the current message. The variable `vm-auto-center-summary` controls whether VM will keep the summary arrow vertically centered within the summary window. A value of `t` causes VM to always keep arrow cenered. A value of `nil` means VM will never bother centering the arrow. A value that is not `nil` and not `t` causes VM to center the arrow only if the summary window is not the only existing window.

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 four characters wide)
The first char is 'D', 'N', 'U' or ' ' for deleted, new, unread and read messages respectively.
The second char is 'F', 'W' or ' ' for filed (saved) or written messages.
The third char is 'R', 'Z' or ' ' for messages replied to, and forwarded messages.
The fourth char is 'E' if the message has been edited, ' ' otherwise.

A - longer version of attributes indicators (six characters wide)
The first char is 'D', 'N', 'U' or ' ' for deleted, new, unread and read messages respectively.
The second is 'r' or ' ', for message replied to.
The third is 'z' or ' ', for messages forwarded.
The fourth is 'f' or ' ', for messages filed.
The fifth is 'w' or ' ', for messages written.
The sixth is 'e' or ' ', for messages that have been edited.

c - number of characters in message (ignoring headers)
d - numeric day 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
M - numeric month message sent (January = 1)
n - message number
s - message subject
t - addresses of the recipients of the message, in a comma-separated list
T - full names of the recipients of the message, in a comma-separated list
If a full name cannot be found, the corresponding address is used instead.
w - day of the week message sent

y - year message sent
z - timezone of date when the message was sent
* - '*' if the current message is marked, ' ' otherwise

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 9.

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.

13 Miscellaneous

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

vm-confirm-quit
: A value of t causes VM to always ask for confirmation before ending a VM visit of a folder. A nil value means VM will ask only when messages will be lost unwittingly by quitting, i.e. not removed by intentional delete and expunge. A value that is not nil and not t causes VM to ask only when there are unsaved changes to message attributes or message will be lost.

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 and %t 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-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 its 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.

mail-yank-hooks
: Value should be a list of functions to be called after a message is yanked into a '`*mail*`' buffer via `vm-yank-message`. When each hook function is called, point will be at the beginning of the yanked text and mark at the end.

  This is not a VM specific variable, but rather an external variable that VM honors so that citation packages such as *SUPERCITE* can be used with VM.

Key Index

X

Z

Command Index

Variable Index

M

V

Short Contents

Table of Contents