Remarks about specific systems ~
|os_390.txt| OS/390 Unix
|os_amiga.txt| Amiga
|os_beos.txt| BeOS and BeBox
|os_dos.txt| MS-DOS and MS-Windows NT/95 common items
|os_mac.txt| Macintosh
|os_mint.txt| Atari MiNT
|os_msdos.txt| MS-DOS (plain DOS and DOS box under Windows)
|os_os2.txt| OS/2
|os_qnx.txt| QNX
|os_risc.txt| RISC-OS
|os_unix.txt| Unix
|os_vms.txt| VMS
|os_win32.txt| MS-Windows 95/98/NT

==
os_390.txt   For Vim version 8.0. Last change: 2016 Feb 27

                  VIM REFERENCE MANUAL   by Ralf Schandl

                                          zOS *z/OS* *OS390* *os390* *MVS*
This file contains the particulars for the z/OS UNIX version of Vim.

Contributors: ~
The port to z/OS UNIX was done by Ralf Schandl for the Redbook mentioned
below.

Changes, bug-reports, or both by:

        David Moore
        Anthony Giorgio
        and others

==
1. ASCII/EBCDIC dependent scripts   *OS390-has-ebcdic* *zOS-has-ebcdic*

For the internal script language the feature "ebcdic" was added. With this
you can fix ASCII dependent scripts like this:
>
    if has("ebcdic")
        let space = 64
    else
        let space = 32
    endif
<

==
2. PuTTY and Colors   *OS390-PuTTY* *zOS-PuTTY*

If you see problems with syntax highlighting or screen corruptions when you
connect to z/OS using Putty, try the following:

- Configure Putty as "vt220" terminal (Connection->Data)
- Add the following 3 lines to your vimrc:

```
>
    set t_AB=^[[4%p1%dm
    set t_AF=^[[3%p1%dm
    set t_CO=8
<
```

Note: ^[is one character use <C-V><Esc> to enter it.

==
3. Motif Problems *OS390-Motif* *zOS-Motif*

Note: Seen with Vim 6.*, never tested since.

It seems that in porting the Motif library to z/OS, a translation from EBCDIC
to ASCII for the accelerator characters of the pull-down menus was forgotten.
Even after I tried to hand convert the menus, the accelerator keys continued
to only work for the opening of menus (like <Alt-F> to open the file menu).
They still do not work for the menu items themselves (like <Alt-F>O to open
the file browser).

There is no solution for this yet.

==
4. Bugs *OS390-bugs* *zOS-Bugs*

- Vim will consistently hang when a large amount of text is selected in
  visual block mode. This may be due to a memory corruption issue. Note that
  this occurs in both the terminal and gui versions.

==
5. Limitations *OS390-limitations* *zOS-limitations*

- No binary search in tag files.
  The program /bin/sort sorts by ASCII value by default. This program is
  normally used by ctags to sort the tags. There might be a version of
  ctags out there, that does it right, but we can't be sure. So this seems to
  be a permanent restriction.

- The cscope interface (|cscope|) doesn't work for the version of cscope
  that we use on our mainframe. We have a copy of version 15.0b12, and it
  causes Vim to hang when using the "cscope add" command. I'm guessing that
  the binary format of the cscope database isn't quite what Vim is expecting.
  I've tried to port the current version of cscope (15.3) to z/OS, without
  much success. If anyone is interested in trying, drop me a line if you
  make any progress.

- No glib/gtk support. I have not been able to successfully compile glib on
  z/OS UNIX. This means you'll have to live without the pretty gtk toolbar.

```
Disabled at compile time:
    - Multibyte support         (|multibyte|)
    - Right-to-left mode        (|rileft|)
    - Farsi key map             (|Farsi|)
    - Arabic language support   (|Arabic|)
    - Spell checking            (|spell|)

Never tested:
    - Perl interface            (|perl|)
    - Hangul input              (|hangul|)
    - Encryption support        (|encryption|)
    - Langmap                   (|'langmap'|)
```

```
    - Python support            (|Python|)
    - Right-to-left mode        (|'rightleft'|)
    - TCL interface             (|tcl|)
    ...

==============================================================================
6. Open source on z/OS UNIX             *OS390-open-source* *zOS-open-source*

If you are interested in other Open Source Software on z/OS UNIX, have a
look at the following Redbook:

    Mike MacIsaac et al
    "Open Source Software for z/OS and OS/390 UNIX"
    IBM Form Number: SG24-5944-01
    ISBN: 0738424633
    http://www-03.ibm.com/systems/resources/
servers_eserver_zseries_zos_unix_redbook_sg245944.pdf

Also look at:
    http://www.redbooks.ibm.com
    http://www-03.ibm.com/systems/z/os/zos/features/unix/
    http://www-03.ibm.com/systems/z/os/zos/features/unix/library/IBM+Redbooks/
index.html



------------------------------------------------------------------------------
 vim:tw=78:fo=tcq2:ts=8:ft=help:norl:
*os_amiga.txt*  For Vim version 8.0.  Last change: 2010 Aug 14


                  VIM REFERENCE MANUAL    by Bram Moolenaar


                                                        *Amiga*
This file contains the particularities for the Amiga version of Vim.
There is also a section specifically for |MorphOS| below.

NOTE: The Amiga code is still included, but has not been maintained or tested.

Installation on the Amiga:
- Assign "VIM:" to the directory where the Vim "doc" directory is.  Vim will
  look for the file "VIM:doc/help.txt" (for the help command).
  Setting the environment variable $VIM also works.  And the other way around:
  when $VIM used and it is not defined, "VIM:" is used.
- With DOS 1.3 or earlier: Put "arp.library" in "libs:".  Vim must have been
  compiled with the |+ARP| feature enabled.  Make sure that newcli and run are
  in "C:" (for executing external commands).
- Put a shell that accepts a command with "-c" (e.g. "Csh" from Fish disk
  624) in "c:" or in any other directory that is in your search path (for
  executing external commands).

If you have sufficient memory you can avoid startup delays by making Vim and
csh resident with the command "rez csh vim".  You will have to put
"rezlib.library" in your "libs:" directory.  Under 2.0 you will need rez
version 0.5.

If you do not use digraphs, you can save some memory by recompiling without
the |+digraphs| feature.  If you want to use Vim with other terminals you can
recompile with the TERMCAP option.  Vim compiles with Manx 5.x and SAS 6.x.
See the makefiles and feature.h.
```

If you notice Vim crashes on some files when syntax highlighting is on, or
when using a search pattern with nested wildcards, it might be that the stack
is too small. Try increasing the stack size. In a shell use the Stack
command before launching Vim. On the Workbench, select the Vim icon, use the
workbench "Info" menu and change the Stack field in the form.

If you want to use different colors set the termcap codes:
        t_mr (for inverted text)
        t_md (for bold text)
        t_me (for normal text after t_mr and t_md)
        t_so (for standout mode)
        t_se (for normal text after t_so)
        t_us (for underlined text)
        t_ue (for normal text after t_us)
        t_ZH (for italic text)
        t_ZR (for normal text after t_ZH)

Standard ANSI escape sequences are used. The codes are:
30 grey char   40 grey cell   >0 grey background   0 all attributes off
31 black char 41 black cell >1 black background   1 boldface
32 white char 42 white cell >2 white background   2 faint
33 blue char   43 blue cell   >3 blue background   3 italic
34 grey char   44 grey cell   >4 grey background   4 underscore
35 black char 45 black cell >5 black background   7 reverse video
36 white char 46 white cell >6 white background   8 invisible
37 blue char   47 blue cell   >7 blue background

The codes with '>' must be the last. The cell and background color should be
the same. The codes can be combined by separating them with a semicolon. For
example to get white text on a blue background: >
  :set t_me=^V<Esc>[0;32;43;>3m
  :set t_se=^V<Esc>[0;32;43;>3m
  :set t_ue=^V<Esc>[0;32;43;>3m
  :set t_ZR=^V<Esc>[0;32;43;>3m
  :set t_md=^V<Esc>[1;32;43;>3m
  :set t_mr=^V<Esc>[7;32;43;>3m
  :set t_so=^V<Esc>[0;31;43;>3m
  :set t_us=^V<Esc>[4;32;43;>3m
  :set t_ZH=^V<Esc>[3;32;43;>3m

When using multiple commands with a filter command, e.g. >
  :r! echo this; echo that
Only the output of the last command is used. To fix this you have to group the
commands. This depends on the shell you use (that is why it is not done
automatically in Vim). Examples: >
  :r! (echo this; echo that)
  :r! {echo this; echo that}

Commands that accept a single file name allow for embedded spaces in the file
name. However, when using commands that accept several file names, embedded
spaces need to be escaped with a backslash.

Vim for MorphOS *MorphOS*

[this section mostly by Ali Akcaagac]

For the latest info about the MorphOS version:
        http://www.akcaagac.com/index_vim.html

Problems ~

```
There are a couple of problems which are not MorphOS related but more Vim and
UN*X related.  When starting up Vim in ram: it complains with a nag requester
from MorphOS please simply ignore it.  Another problem is when running Vim as
is some plugins will cause a few problems which you can ignore as well.
Hopefully someone will be fixing it over the time.

To pass all these problems for now you can either run:

        vim <file to be edited>

or if you want to run Vim plain and enjoy the motion of Helpfiles etc. it then
would be better to enter:

        vim --noplugins <of course you can add a file>


Installation ~

1) Please copy the binary 'VIM' file to c:
2) Get the Vim runtime package from:

        ftp://ftp.vim.org/pub/vim/amiga/vim62rt.tgz

   and unpack it in your 'Apps' directory of the MorphOS installation.  For me
   this would create following directory hierarchy:

        MorphOS:Apps/Vim/Vim62/...

3) Add the following lines to your s:shell-startup (Important!).

        ;Begin VIM
        Set VIM=MorphOS:Apps/Vim/Vim62
        Assign HOME: ""
        ;End VIM

4) Copy the '.vimrc' file to s:

5) There is also a file named 'color-sequence' included in this archive.  This
   will set the MorphOS Shell to show ANSI colors.  Please copy the file to s:
   and change the s:shell-startup to:

        ;Begin VIM
        Set VIM=MorphOS:Apps/Vim/Vim62
        Assign HOME: ""
        Execute S:Color-Sequence
        Cls
        ;End VIM


 vim:tw=78:ts=8:ft=help:norl:
*os_beos.txt*   For Vim version 8.0.  Last change: 2016 Mar 28


                  VIM REFERENCE MANUAL    by Bram Moolenaar


                                                        *BeOS* *BeBox*
This is a port of Vim 5.1 to the BeOS Preview Release 2 (also known as PR2)
or later.

This file contains the particularities for the BeBox/BeOS version of Vim.  For
```

matters not discussed in this file, Vim behaves very much like the Unix
|os_unix.txt| version.

1. General *beos-general*

The default syntax highlighting mostly works with different foreground colors
to highlight items. This works best if you set your Terminal window to a
darkish background and light letters. Some middle-grey background (for
instance (r,g,b)=(168,168,168)) with black letters also works nicely. If you
use the default light background and dark letters, it may look better to
simply reverse the notion of foreground and background color settings. To do
this, add this to your .vimrc file (where <Esc> may need to be replaced with
the escape character): >

```
  :if &term == "beos-ansi"
  :    set t_AB=<Esc>[3%dm
  :    set t_AF=<Esc>[4%dm
  :endif
```

2. Compiling Vim *beos-compiling*

From the Advanced Access Preview Release (AAPR) on, Vim can be configured with
the standard configure script. To get the compiler and its flags right, use
the following command-line in the shell (you can cut and paste it in one go):

```
CC=$BE_C_COMPILER CFLAGS="$BE_DEFAULT_C_FLAGS -O7" \
    ./configure --prefix=/boot/home/config
```

$BE_C_COMPILER is usually "mwcc", $BE_DEFAULT_C_FLAGS is usually "-I- -I."

When configure has run, and you wish to enable GUI support, you must edit the
config.mk file so that the lines with GUI_xxx refer to $(BEOSGUI_xxx) instead
of $(NONE_xxx).
Alternatively you can make this change in the Makefile; it will have a
more permanent effect. Search for "NONE_".

After compilation you need to add the resources to the binary. Add the
following few lines near the end (before the line with "exit $exit_value") of
the link.sh script to do this automatically.

```
    rmattr BEOS:TYPE vim
    copyres os_beos.rsrc vim
    mimeset vim
```

Also, create a dummy file "strip":

```
    #!/bin/sh
    mimeset $1
    exit 0
```

You will need it when using "make install" to install Vim.

Now type "make" to compile Vim, then "make install" to install it.

If you want to install Vim by hand, you must copy Vim to $HOME/config/bin, and create a bunch of symlinks to it ({g,r,rg}{vim,ex,view}). Furthermore you must copy Vim's configuration files to $HOME/config/share/vim: vim-5.0s/{*.vim,doc,syntax}. For completeness, you should also copy the nroff manual pages to $HOME/config/man/man1. Don't forget ctags/ctags and xxd/xxd!

Obviously, you need the unlimited linker to actually link Vim. See http://www.metrowerks.com for purchasing the CodeWarrior compiler for BeOS. There are currently no other linkers that can do the job.

This won't be able to include the Perl or Python interfaces even if you have the appropriate files installed. |beos-perl|

3. Timeout in the Terminal *beos-timeout*

Because some POSIX/UNIX features are still missing[1], there is no direct OS support for read-with-timeout in the Terminal. This would mean that you cannot use :mappings of more than one character, unless you also :set notimeout. |'timeout'|

To circumvent this problem, I added a workaround to provide the necessary input with timeout by using an extra thread which reads ahead one character. As a side effect, it also makes Vim recognize when the Terminal window resizes.

Function keys are not supported in the Terminal since they produce very indistinctive character sequences.

These problems do not exist in the GUI.

[1]: there is no select() on file descriptors; also the termios VMIN and VTIME settings do not seem to work properly. This has been the case since DR7 at least and still has not been fixed as of PR2.

4. Unicode vs. Latin1 *beos-unicode* *beos-utf8*

BeOS uses Unicode and UTF-8 for text strings (16-bit characters encoded to 8-bit characters). Vim assumes ISO-Latin1 or other 8-bit character codes. This does not produce the desired results for non-ASCII characters. Try the command :digraphs to see. If they look messed up, use :set isprint=@ to (slightly) improve the display of ISO-Latin1 characters 128-255. This works better in the GUI, depending on which font you use (below).

You may also use the /boot/bin/xtou command to convert UTF-8 files from (xtou -f iso1 filename) or to (xtou -t iso1 filename) ISO-Latin1 characters.

5. The BeOS GUI *beos-gui*

The BeOS GUI is no longer included. It was not maintained for a while and most likely didn't work. If you want to work on this: get the Vim 6.x version

and merge it back in.

6. The $VIM directory *beos-vimdir*

$VIM is the symbolic name for the place where Vims support files are stored.
The default value for $VIM is set at compile time and can be determined with >

```
:version
```

The normal value is /boot/home/config/share/vim. If you don't like it you can
set the Vim environment variable to override this, or set 'helpfile' in your
.vimrc: >

```
:if version >= 500
:    set helpfile=~/vim/vim54/doc/help.txt
:    syntax on
:endif
```

7. Drag & Drop *beos-dragndrop*

You can drop files and directories on either the Vim icon (starts a new Vim
session, unless you use the File Types application to set Vim to be "Single
Launch") or on the Vim window (starts editing the files). Dropping a folder
sets Vim's current working directory. |:cd| |:pwd| If you drop files or
folders with either SHIFT key pressed, Vim changes directory to the folder
that contains the first item dropped. When starting Vim, there is no need to
press shift: Vim behaves as if you do.

Files dropped set the current argument list. |argument-list|

8. Single Launch vs. Multiple Launch *beos-launch*

As distributed Vim's Application Flags (as seen in the FileTypes preference)
are set to Multiple Launch. If you prefer, you can set them to Single Launch
instead. Attempts to start a second copy of Vim will cause the first Vim to
open the files instead. This works from the Tracker but also from the command
line. In the latter case, non-file (option) arguments are not supported.

NB: Only the GUI version has a BApplication (and hence Application Flags).
This section does not apply to the GUI-less version, should you compile one.

9. Fonts *beos-fonts*

Set fonts with >

```
:set guifont=Courier10_BT/Roman/10
```

where the first part is the font family, the second part the style, and the
third part the size. You can use underscores instead of spaces in family and
style.

Best results are obtained with monospaced fonts (such as Courier). Vim
attempts to use all fonts in B_FIXED_SPACING mode but apparently this does not
work for proportional fonts (despite what the BeBook says).

Vim also tries to use the B_ISO8859_1 encoding, also known as ISO Latin 1.
This also does not work for all fonts. It does work for Courier, but not for
ProFontISOLatin1/Regular (strangely enough). You can verify this by giving the >

```
  :digraphs

command, which lists a bunch of characters with their ISO Latin 1 encoding.
If, for instance, there are "box" characters among them, or the last character
isn't a dotted-y, then for this font the encoding does not work.

If the font you specify is unavailable, you get the system fixed font.

Standard fixed-width system fonts are:

              ProFontISOLatin1/Regular
                  Courier10_BT/Roman
                  Courier10_BT/Italic
                  Courier10_BT/Bold
                  Courier10_BT/Bold_Italic

Standard proportional system fonts are:

                    Swis721_BT/Roman
                    Swis721_BT/Italic
                    Swis721_BT/Bold
                    Swis721_BT/Bold_Italic
                Dutch801_Rm_BT/Roman
                Dutch801_Rm_BT/Italic
                Dutch801_Rm_BT/Bold
                Dutch801_Rm_BT/Bold_Italic
                   Baskerville/Roman
                   Baskerville/Italic
                   Baskerville/Bold
                   Baskerville/Bold_Italic
                 SymbolProp_BT/Regular

Try some of them, just for fun.


10. The meta key modifier                                *beos-meta*

The META key modifier is obtained by the left or right OPTION keys.  This is
because the ALT (aka COMMAND) keys are not passed to applications.


11. Mouse key mappings                                   *beos-mouse*

Vim calls the various mouse buttons LeftMouse, MiddleMouse and RightMouse.  If
you use the default Mouse preference settings these names indeed correspond to
reality.  Vim uses this mapping:

    Button 1 -> LeftMouse,
    Button 2 -> RightMouse,
    Button 3 -> MiddleMouse.

If your mouse has fewer than 3 buttons you can provide your own mapping from
mouse clicks with modifier(s) to other mouse buttons.  See the swapmouse
package for an example:                                |gui-mouse-mapping|
$VIMRUNTIME/pack/dist/opt/swapmouse/plugin/swapmouse.vim


12. Color names                                          *beos-colors*

Vim has a number of color names built-in.  Additional names are read from the
file $VIMRUNTIME/rgb.txt, if present.  This file is basically the color
```

database from X. Names used from this file are cached for efficiency.

13. Compiling with Perl *beos-perl*

Compiling with Perl support enabled is slightly tricky. The Metrowerks compiler has some strange ideas where to search for include files. Since several include files with Perl have the same names as some Vim header files, the wrong ones get included. To fix this, run the following Perl script while in the vim-5.0/src directory: >

```
   preproc.pl > perl.h

    #!/bin/env perl
    # Simple #include expander, just good enough for the Perl header files.

    use strict;
    use IO::File;
    use Config;

    sub doinclude
    {
        my $filename = $_[0];
        my $fh = new IO::File($filename, "r");
        if (defined $fh) {
            print "/* Start of $filename */\n";

            while (<$fh>) {
                if (/^#include "(.*)"/) {
                    doinclude($1);
                    print "/* Back in $filename */\n";
                } else {
                    print $_;
                }
            }
            print "/* End of $filename */\n";

            undef $fh;
        } else {
            print "/* Cannot open $filename */\n";
            print "#include \"$filename\"\n";
        }
    }

    chdir     $Config{installarchlib}."/CORE";
    doinclude "perl.h";
```

It expands the "perl.h" header file, using only other Perl header files.

Now you can configure & make Vim with the --enable-perlinterp option.
Be warned though that this adds about 616 kilobytes to the size of Vim!
Without Perl, Vim with default features and GUI is about 575K, with Perl
it is about 1191K.

-Olaf Seibert

[Note: these addresses no longer work:]
<rhialto@polder.ubc.kun.nl>
http://polder.ubc.kun.nl/~rhialto/be

 vim:tw=78:ts=8:ft=help:norl:
os_mac.txt   For Vim version 8.0. Last change: 2017 Apr 28

VIM REFERENCE MANUAL by Bram Moolenaar et al.

mac *Mac* *macintosh* *Macintosh*

This file documents the particularities of the Macintosh version of Vim.

NOTE: This file is a bit outdated. You might find more useful info here:
http://macvim.org/

There was a Mac port for version 3.0 of Vim. Here are the first few lines
from the old file:

VIM Release Notes
Initial Macintosh release, VIM version 3.0
19 October 1994

Eric Fischer
<enf1@midway.uchicago.edu>, <eric@jcp.uchicago.edu>, <etaoin@uchicago.edu>
5759 N. Guilford Ave
Indianapolis IN 46220 USA

==
1. Filename Convention *mac-filename*

Starting with Vim version 7 you can just use the unix path separators with
Vim. In order to determine if the specified filename is relative to the
current folder or absolute (i.e. relative to the "Desktop"), the following
algorithm is used:

If the path start by a "/", the path is absolute
If the path start by a ":", the path is relative
If the path doesn't start by neither a "/" nor ":",
and a ":" is found before a "/" then the path is absolute
>
:e /HD/text
:e HD:text
< Edit the file "text" of the disk "HD" >
:e :src:main.c
:e src/main.c
< Edit the file "main.c" in the folder "src" in the current folder >
:e os_mac.c
< Edit the file "os_mac.c" in the current folder.

You can use the |$VIM| and |$VIMRUNTIME| variable. >

:so $VIMRUNTIME:syntax:syntax.vim

==
2. .vimrc and .vim files *mac-vimfile*

It is recommended to use Unix style line separators for Vim scripts, thus a
single newline character.

When starting up Vim will load the $VIMRUNTIME/macmap.vim script to define
default command-key mappings.

On older systems files starting with a dot "." are discouraged, thus the rc
files are named "vimrc" or "_vimrc" and "gvimrc" or "_gvimrc". These files
can be in any format (mac, dos or unix). Vim can handle any file format when
the |'nocompatible'| option is set, otherwise it will only handle mac format
files.

==
3. Standard mappings *mac-standard-mappings*

The following mappings are available for cut/copy/paste from/to clipboard.

key	Normal	Visual	Insert	Description ~	
Command-v	"*P	"-d"*P	<C-R>*	paste text	*<D-v>*
Command-c		"*y		copy Visual text	*<D-c>*
Command-x		"*d		cut Visual text	*<D-x>*
Backspace		"*d		cut Visual text	

==
4. Mac FAQ *mac-faq*

On the internet: http://macvim.org/OSX/index.php#FAQ

Q: I can't enter non-ASCII character in Apple Terminal.
A: Under Window Settings, Emulation, make sure that "Escape non-ASCII
characters" is not checked.

Q: How do I start the GUI from the command line?
A: Assuming that Vim.app is located in /Applications:
```
open /Applications/Vim.app
```
Or:
```
/Applications/Vim.app/Contents/MacOS/Vim -g {arguments}
```

Q: How can I set $PATH to something reasonable when I start Vim.app from the
GUI or with open?
A: The following trick works with most shells. Put it in your vimrc file.
This is included in the system vimrc file included with the binaries
distributed at macvim.org . >
```
let s:path = system("echo echo VIMPATH'${PATH}' | $SHELL -l")
let $PATH = matchstr(s:path, 'VIMPATH\zs.\{-}\ze\n')
```

==
5. Mac Lack *mac-lack*

In a terminal CTRL-^ needs to be entered as Shift-Control-6. CTRL-@ as
Shift-Control-2.

==
6. Mac Bug Report *mac-bug*

When reporting any Mac specific bug or feature change, please use the vim-mac
maillist |vim-mac|. However, you need to be subscribed. An alternative is to
send a message to the current MacVim maintainers:

```
mac@vim.org
```

==

7. Compiling Vim *mac-compile*

See the file "src/INSTALLmac.txt" that comes with the source files.

==
8. The Darwin Feature *mac-darwin-feature*

If you have a Mac that isn't very old, you will be running OS X, also called
Darwin. The last pre-Darwin OS was Mac OS 9. The darwin feature makes Vim
use Darwin-specific properties.

What is accomplished with this feature is two-fold:

- Make Vim interoperable with the system clipboard.
- Incorporate into Vim a converter module that bridges the gap between some
  character encodings specific to the platform and those known to Vim.

Needless to say, both are not to be missed for any decent text editor to work
nicely with other applications running on the same desktop environment.

As Vim is not an application dedicated only to macOS, we need an extra feature
to add in order for it to offer the same user experience that our users on
other platforms enjoy to people on macOS.

For brevity, the feature is referred to as "darwin" to signify it one of the
Vim features that are specific to that particular platform.

The feature is a configuration option. Accordingly, whether it is enabled or
not is determined at build time; once it is selected to be enabled, it is
compiled in and hence cannot be disabled at runtime.

The feature is enabled by default. For most macOS users, that should be
sufficient unless they have specific needs mentioned briefly below.

If you want to disable it, pass `--disable-darwin` to the configure script: >

    ./configure --disable-darwin <other options>

and then run `make` to build Vim. The order of the options doesn't matter.

To make sure at runtime whether or not the darwin feature is compiled in, you
can use `has('macunix')` which returns 1 if the feature is compiled in; 0
otherwise.

Notable use cases where `--disable-darwin` is turned out to be useful are:

- When you want to use |x11-selection| instead of the system clipboard.
- When you want to use |x11-clientserver|.

Since both have to make use of X11 inter-client communication for them to work
properly, and since the communication mechanism can come into conflict with
the system clipboard, the darwin feature should be disabled to prevent Vim
from hanging at runtime.

 vim:tw=78:ts=8:ft=help:norl:
os_mint.txt For Vim version 8.0. Last change: 2005 Mar 29

MiNT *Atari*
This file contains the particularities for the Atari MiNT version of Vim.

For compiling Vim on the Atari running MiNT see "INSTALL" and "Makefile"
in the src directory.

Vim for MiNT behaves almost exactly like the Unix version.
The Unix behavior described in the documentation also refers to the
MiNT version of Vim unless explicitly stated otherwise.

For wildcard expansion of <~> (home directory) you need a shell that
expands the tilde. The vanilla Bourne shell doesn't recognize it.
With csh and ksh it should work OK.

The MiNT version of vim needs the termcap file /etc/termcap with the
terminal capabilities of your terminal. Builtin termcaps are
supported for the vt52 terminal. Termcap entries for the TOSWIN window
manager and the virtual console terminals have been appended to the
termcap file that comes with the Vim distribution.

If you should encounter problems with swapped <BS> and <Del> keys, see
|:fixdel|.

Because terminal updating under MiNT is often slow (e.g. serial line
terminal), the 'showcmd' and 'ruler' options are default off.
If you have a fast terminal, try setting them on. You might
also want to set 'ttyfast'.

Send bug reports to

        Jens M. Felderhoff, e-mail: <jmf@infko.uni-koblenz.de>

 vim:tw=78:ts=8:ft=help:norl:
os_msdos.txt For Vim version 8.0. Last change: 2016 Feb 26

                  VIM REFERENCE MANUAL   by Bram Moolenaar

msdos *ms-dos* *MSDOS* *MS-DOS*
This file used to contain the particularities for the MS-DOS version of Vim.
MS-DOS support was removed in patch 7.4.1399. If you want to use it you will
need to get a version older than that. Note that the MS-DOS version doesn't
work, there is not enough memory. The DOS32 version (using DJGPP) might still
work on older systems.

 vim:tw=78:ts=8:ft=help:norl:
os_os2.txt   For Vim version 8.0. Last change: 2015 Dec 31

                  VIM REFERENCE MANUAL   by Paul Slootman

os2 *OS2* *OS/2*
This file used to contain the particularities for the OS/2 version of Vim.

The OS/2 support was removed in patch 7.4.1008.

 vim:tw=78:ts=8:ft=help:norl:
os_qnx.txt   For Vim version 8.0. Last change: 2005 Mar 29

VIM REFERENCE MANUAL by Julian Kinraid

QNX *qnx*

==

1. General *qnx-general*

Vim on QNX behaves much like other unix versions. |os_unix.txt|

2. Compiling Vim *qnx-compiling*

Vim can be compiled using the standard configure/make approach. If you want to compile for X11, pass the --with-x option to configure. Otherwise, running ./configure without any arguments or passing --enable-gui=photon, will compile vim with the Photon gui support. Run ./configure --help , to find out other features you can enable/disable.

3. Terminal support *qnx-terminal*

Vim has support for the mouse and clipboard in a pterm, if those options are compiled in, which they are normally.

The options that affect mouse support are |'mouse'| and |'ttymouse'|. When using the mouse, only simple left and right mouse clicking/dragging is supported. If you hold down shift, ctrl, or alt while using the mouse, pterm will handle the mouse itself. It will make a selection, separate from what vim's doing.

When the mouse is in use, you can press Alt-RightMouse to open the pterm menu. To turn the mouse off in vim, set the mouse option to nothing, set mouse=

4. Photon GUI *photon-gui*

To start the gui for vim, you need to run either gvim or vim -g, otherwise the terminal version will run. For more info - |gui-x11-start|

Supported features:

:browse command	|:browse|
:confirm command	|:confirm|
Cursor blinking	|'guicursor'|
Menus, popup menus and menu priorities	|:menu| |popup-menu| |menu-priority|
Toolbar	|gui-toolbar| |'toolbar'|
Font selector (:set guifont=*)	|photon-fonts|
Mouse focus	|'mousefocus'|
Mouse hide	|'mousehide'|

```
        Mouse cursor shapes                             |'mouseshape'|
        Clipboard                                       |gui-clipboard|

Unfinished features:
        Various international support, such as Farsi & Hebrew support,
        different encodings, etc.

        This help file

Unsupported features:
        Find & Replace window                           |:promptfind|
        Tearoff menus

        Other things which I can't think of so I can't list them


5. Fonts                                                *photon-fonts*

You set fonts in the gui with the guifont option >
        :set guifont=Lucida\ Terminal
<
The font must be a monospace font, and any spaces in the font name must be
escaped with a '\'.  The default font used is PC Terminal, size 8.  Using
'*' as the font name will open a standard Photon font selector where you can
select a font.

Following the name, you can include optional settings to control the size and
style of the font, each setting separated by a ':'.  Not all fonts support the
various styles.

The options are,
    s{size}     Set the size of the font to {size}
    b           Bold style
    a           Use antialiasing
    i           Italic style

Examples:

Set the font to monospace size 10 with antialiasing >
        :set guifont=monospace:s10:a
<
Set the font to Courier size 12, with bold and italics >
        :set guifont=Courier:s12:b:i
<
Select a font with the requester >
        :set guifont=*
<


6. Bugs & things To Do

Known problems:
        - Vim hangs sometimes when running an external program.  Workaround:
          put this line in your |vimrc| file: >
                set noguipty

Bugs:
        - Still a slight problem with menu highlighting.
        - When using phditto/phinows/etc., if you are using a font that
          doesn't support the bold attribute, when vim attempts to draw
          bold text it will be all messed up.
        - The cursor can sometimes be hard to see.
```

```
        - A number of minor problems that can fixed. :)

Todo:
        - Improve multi-language support.
        - Options for setting the fonts used in the menu and toolbar.
        - Find & Replace dialog.
        - The clientserver features.
        - Maybe tearoff menus.

        - Replace usage of fork() with spawn() when launching external
          programs.

 vim:tw=78:sw=4:ts=8:ts=8:ft=help:norl:
*os_risc.txt*   For Vim version 8.0.  Last change: 2011 May 10


                  VIM REFERENCE MANUAL    by Thomas Leonard


                                                *riscos* *RISCOS* *RISC-OS*
The RISC OS support has been removed from Vim with patch 7.3.187.
If you would like to use Vim on RISC OS get the files from before that patch.


 vim:tw=78:ts=8:ft=help:norl:
*os_unix.txt*   For Vim version 8.0.  Last change: 2005 Mar 29


                  VIM REFERENCE MANUAL    by Bram Moolenaar


                                                        *unix* *Unix*
This file contains the particularities for the Unix version of Vim.

For compiling Vim on Unix see "INSTALL" and "Makefile" in the src directory.

The default help file name is "/usr/local/lib/vim/help.txt"
The files "$HOME/.vimrc" and "$HOME/.exrc" are used instead of "s:.vimrc" and
"s:.exrc".  Additionally "/usr/local/etc/vimrc" is used first.
If "/usr/local/share" exists it is used instead of "/usr/local/lib".

Temporary files (for filtering) are put in "/tmp".  If you want to place them
somewhere else, set the environment variable $TMPDIR to the directory you
prefer.

With wildcard expansion you can use '~' (home directory) and '$'
(environment variable).

                                                        *fork* *spoon*
For executing external commands fork()/exec() is used when possible, otherwise
system() is used, which is a bit slower.  The output of ":version" includes
|+fork| when fork()/exec() is used, |+system()| when system() is used.  This
can be changed at compile time.
(For forking of the GUI version see |gui-fork|.)

Because terminal updating under Unix is often slow (e.g. serial line
terminal, shell window in suntools), the 'showcmd' and 'ruler' options
are default off.  If you have a fast terminal, try setting them on.  You might
also want to set 'ttyfast'.

When using Vim in an xterm the mouse clicks can be used by Vim by setting
'mouse' to "a".  If there is access to an X-server gui style copy/paste will
```

```
be used and visual feedback will be provided while dragging with the mouse.
If you then still want the xterm copy/paste with the mouse, press the shift
key when using the mouse.  See |mouse-using|.  Visual feedback while dragging
can also be achieved via the 'ttymouse' option if your xterm is new enough.

                                                        *terminal-colors*
To use colors in Vim you can use the following example (if your terminal
supports colors, but "T_Co" is empty or zero): >
   :set t_me=^[[0;1;36m     " normal mode (undoes t_mr and t_md)
   :set t_mr=^[[0;1;33;44m  " reverse (invert) mode
   :set t_md=^[[1;33;41m    " bold mode
   :set t_se=^[[1;36;40m    " standout end
   :set t_so=^[[1;32;45m    " standout mode
   :set t_ue=^[[0;1;36m     " underline end
   :set t_us=^[[1;32m       " underline mode start
[the ^[ is an <Esc>, type CTRL-V <Esc> to enter it]

For real color terminals the ":highlight" command can be used.

The file "tools/vim132" is a shell script that can be used to put Vim in 132
column mode on a vt100 and lookalikes.

 vim:tw=78:ts=8:ft=help:norl:
*os_vms.txt*    For Vim version 8.0.  Last change: 2014 Aug 29


                  VIM REFERENCE MANUAL


                                                        *VMS* *vms*
This file contains the particularities for the VMS version of Vim.
You can reach this information file by typing :help VMS in Vim command
prompt.

 1. Getting started     |vms-started|
 2. Download files      |vms-download|
 3. Compiling           |vms-compiling|
 4. Problems            |vms-problems|
 5. Deploy              |vms-deploy|
 6. Practical usage     |vms-usage|
 7. GUI mode questions  |vms-gui|
 8. Useful notes        |vms-notes|
 9. VMS related changes |vms-changes|
10. Authors             |vms-authors|

==============================================================================

1. Getting started                                      *vms-started*

Vim (Vi IMproved) is a Vi-compatible text editor that runs on nearly every
operating system known to humanity.  Now use Vim on OpenVMS too, in character
or X/Motif environment.  It is fully featured and absolutely compatible with
Vim on other operating systems.

==============================================================================

2. Download files                                       *vms-download*

You can download the Vim source code by ftp from the official Vim site:
        ftp://ftp.vim.org/pub/vim/
Or use one of the mirrors:
        ftp://ftp.vim.org/pub/vim/MIRRORS
```

```
You can download precompiled executables from:
        http://www.polarhome.com/vim/
        ftp://ftp.polarhome.com/pub/vim/

To use the precompiled binary version, you need one of these archives:

  vim-XX-exe-ia64-gui.zip       IA64 GUI/Motif executables
  vim-XX-exe-ia64-gtk.zip       IA64 GUI/GTK executables
  vim-XX-exe-ia64-term.zip      IA64 console executables
  vim-XX-exe-axp-gui.zip        Alpha GUI/Motif executables
  vim-XX-exe-axp-gtk.zip        Alpha GUI/GTK executables
  vim-XX-exe-axp-term.zip       Alpha console executables
  vim-XX-exe-vax-gui.zip        VAX GUI executables
  vim-XX-exe-vax-term.zip       VAX console executables

and of course (optional)
  vim-XX-runtime.zip            runtime files

The binary archives contain: vim.exe, ctags.exe, xxd.exe files.

For GTK executables you will need GTKLIB that is available for
Alpha and IA64 platform.

==============================================================================

3. Compiling                                            *vms-compiling*

See the file [.SRC]INSTALLVMS.TXT.

==============================================================================

4. Problems                                             *vms-problems*

The code has been tested under Open VMS 6.2 - 8.2 on Alpha, VAX and IA64
platforms with the DEC C compiler.  It should work without big problems.
If your system does not have some include libraries you can tune up in
OS_VMS_CONF.H file.

If you decided to build Vim with +perl, +python, etc. options, first you need
to download OpenVMS distributions of Perl and Python.  Build and deploy the
libraries and change adequate lines in MAKE_VMS.MMS file.  There should not be
a problem from Vim side.

Also GTK, XPM library paths should be configured in MAKE_VMS.MMS

Note: Under VAX it should work with the DEC C compiler without problems.  The
VAX C compiler is not fully ANSI C compatible in pre-processor directives
semantics, therefore you have to use a converter program that will do the lion
part of the job.  For detailed instructions read file INSTALLvms.txt

MMS_VIM.EXE is build together with VIM.EXE, but for XXD.EXE you should
change to a subdirectory and build it separately.

CTAGS is not part of the Vim source distribution anymore, however the OpenVMS
specific source might contain CTAGS source files as described above.
You can find more information about CTAGS on VMS at
http://www.polarhome.com/ctags/

Advanced users may try some acrobatics in FEATURE.H file as well.

It is possible to compile with +xfontset +xim options too, but then you have
```

```
to set up GUI fonts etc. correctly.  See :help xim from Vim command prompt.

You may want to use GUI with GTK icons, then you have to download and install
GTK for OpenVMS or at least runtime shareable images - LIBGTK from
polarhome.com

For more advanced questions, please send your problem to Vim on VMS mailing
list <vim-vms@polarhome.com>
More about the vim-vms list can be found at:
http://www.polarhome.com/mailman/listinfo/vim-vms

==============================================================================

5. Deploy                                                   *vms-deploy*

Vim uses a special directory structure to hold the document and runtime files:

   vim (or wherever)
    |- tmp
    |- vim57
    |----- doc
    |----- syntax
    |- vim62
    |----- doc
    |----- syntax
    |- vim64
    |----- doc
    |----- syntax
    vimrc    (system rc files)
    gvimrc

Use: >

        define/nolog VIM        device:[path.vim]
        define/nolog VIMRUNTIME device:[path.vim.vim60]
        define/nolog TMP        device:[path.tmp]

To get vim.exe to find its document, filetype, and syntax files, and to
specify a directory where temporary files will be located.  Copy the "runtime"
subdirectory of the Vim distribution to vimruntime.

Logicals $VIMRUNTIME and $TMP are optional.

If $VIMRUNTIME is not set, Vim will guess and try to set up automatically.
Read more about it at :help runtime

If $TMP is not set, you will not be able to use some functions as CTAGS,
XXD, printing etc. that use temporary directory for normal operation.
The $TMP directory should be readable and writable by the user(s).
The easiest way to set up $TMP is to define a logical: >

        define/nolog TMP SYS$SCRATCH
or as: >
        define/nolog TMP SYS$LOGIN

==============================================================================

6. Practical usage                                          *vms-usage*

Usually, you want to run just one version of Vim on your system, therefore
it is enough to dedicate one directory for Vim.
Copy the whole Vim runtime directory structure to the deployment position.
```

```
Add the following lines to your LOGIN.COM (in SYS$LOGIN directory).
Set up the logical $VIM as: >

        $ define VIM device:<path>

Set up some symbols: >

        $ ! vi starts Vim in chr. mode.
        $ vi*m  :== mcr VIM:VIM.EXE

        $ !gvi starts Vim in GUI mode.
        $ gv*im :== spawn/nowait mcr VIM:VIM.EXE -g

Please, check the notes for customization and configuration of symbols.

You may want to create .vimrc and .gvimrc files in your home directory
(SYS$LOGIN) to overwrite default settings.

The easiest way is just rename example files.  You may leave the menu file
(MENU.VIM) and files vimrc and gvimrc in the original $VIM directory.  It will
be the default setup for all users, and for users it is enough to just have
their own additions or resetting in their home directory in files .vimrc and
.gvimrc.  It should work without problems.

Note: Remember, system rc files (default for all users) don't have a leading
".".  So, system rc files are: >

        $VIM:vimrc
        $VIM:gvimrc
        $VIM:menu.vim

and user customized rc files are: >

        sys$login:.vimrc
        sys$login:.gvimrc

You can check that everything is at the right place with the :version command.

Example LOGIN.COM: >

        $ define/nolog VIM RF10:[UTIL.VIM]
        $ vi*m :== mcr VIM:VIM.EXE
        $ gv*im:== spawn/nowait/input=NLA0 mcr VIM:VIM.EXE -g -GEOMETRY 80x40
        $ set disp/create/node=192.168.5.223/trans=tcpip

Note: This set-up should be enough, if you are working on a standalone server or
clustered environment, but if you want to use Vim as an internode editor in
DECNET environment, it will satisfy as well.
You just have to define the "whole" path: >

        $ define VIM "<server_name>[""user password""]::device:<path>"
        $ vi*m :== "mcr VIM:VIM.EXE"

For example: >

        $ define VIM "PLUTO::RF10:[UTIL.VIM]"
        $ define VIM "PLUTO""ZAY mypass""::RF10:[UTIL.VIM]" ! if passwd required

You can also use the $VIMRUNTIME logical to point to the proper version of Vim
if you have installed more versions at the same time.  If $VIMRUNTIME is not
defined Vim will borrow its value from the $VIM logical.  You can find more
information about the $VIMRUNTIME logical by typing :help runtime as a Vim
```

```
command.

System administrators might want to set up a system wide Vim installation,
then add to the SYS$STARTUP:SYLOGICALS.COM >

        $ define/nolog/sys VIM device:<path>
        $ define/nolog/sys TMP SYS$SCRATCH

And to the SYS$STARTUP:SYLOGIN.COM >

        $ vi*m :== mcr VIM:VIM.EXE
        $ gv*im:== spawn/nowait/input=NLA0 mcr VIM:VIM.EXE -g -GEOMETRY 80x40


It will set up a normal Vim work environment for every user on the system.

IMPORTANT: Vim on OpenVMS (and on other case insensitive system) command line
parameters are assumed to be lowercase. In order to indicate that a command
line parameter is uppercase "/" sign must be used.

Examples:
  >
        vim -R  filename  ! means: -r List swap files and exit
        vim -/r filename  ! means: -R Readonly mode (like "view")
        vim -u  <vimrc>   ! means: -u Use <vimrc> instead of any .vimrc
        vim -/u <gvimrc>  ! means: -U Use <gvimrc> instead of any .gvimrc

==============================================================================

7. GUI mode questions                                        *vms-gui*

OpenVMS is a real mainframe OS, therefore even if it has a GUI console, most
of the users do not use a native X/Window environment during normal operation.
It is not possible to start Vim in GUI mode "just like that".  But anyhow it
is not too complicated either.

First of all: you will need an executable that is built with the GUI enabled.

Second: you need to have installed DECW/Motif on your VMS server, otherwise
you will get errors that some shareable libraries are missing.

Third: If you choose to run Vim with extra features such as GUI/GTK then you
need a GTK installation too or at least a GTK runtime environment (LIBGTK
can be downloaded from http://www.polarhome.com/vim/).

1) If you are working on the VMS X/Motif console:
   Start Vim with the command: >

        $ mc device:<path>VIM.EXE -g
<
   or type :gui as a command to the Vim command prompt.  For more info :help
   gui

2) If you are working on some other X/Window environment like Unix or a remote
   X VMS console.  Set up display to your host with: >

        $ set disp/create/node=<your IP address>/trans=<transport-name>
<
   and start Vim as in point 1.  You can find more help in VMS documentation or
   type: help set disp in VMS prompt.
   Examples: >
```

```
        $ set disp/create/node=192.168.5.159             ! default trans is DECnet
        $ set disp/create/node=192.168.5.159/trans=tcpip ! TCP/IP network
        $ set disp/create/node=192.168.5.159/trans=local ! display on the same node

Note: you should define just one of these.
For more information type $help set disp in VMS prompt.

3) Another elegant solution is XDM if you have installed on OpenVMS box.
   It is possible to work from XDM client as from GUI console.

4) If you are working on MS-Windows or some other non X/Window environment
   you need to set up one X server and run Vim as in point 2.
   For MS-Windows there are available free X servers as MIX, Omni X etc.,
   as well as excellent commercial products as eXcursion or ReflectionX with
   built-in DEC support.

Please note, that executables without GUI are slightly faster during startup
than with enabled GUI in character mode. Therefore, if you do not use GUI
features, it is worth to choose non GUI executables.

==============================================================================

8. Useful notes                                          *vms-notes*

8.1 Backspace/delete
8.2 Filters
8.3 VMS file version numbers
8.4 Directory conversion
8.5 Remote host invocation
8.6 Terminal problems
8.7 Hex-editing and other external tools
8.8 Sourcing vimrc and gvimrc
8.9 Printing from Vim
8.10 Setting up the symbols
8.11 diff and other GNU programs
8.12 diff-mode
8.13 Allow '$' in C keywords
8.14 VIMTUTOR for beginners
8.15 Slow start in console mode issue
8.16 Common VIM directory - different architectures

8.1 Backspace/delete

There are backspace/delete key inconsistencies with VMS.
:fixdel doesn't do the trick, but the solution is: >

        :inoremap ^? ^H         " for terminal mode
        :inoremap <Del> ^H      " for gui mode

Read more in ch: 8.6 (Terminal problems).
(Bruce Hunsaker <BNHunsaker@chq.byu.edu> Vim 5.3)


8.2 Filters

Vim supports filters, i.e., if you have a sort program that can handle
input/output redirection like Unix (<infile >outfile), you could use >

        :map \s 0!'aqsort<CR>

(Charles E. Campbell, Jr. <cec@gryphon.gsfc.nasa.gov> Vim 5.4)
```

8.3 VMS file version numbers

Vim is saving files into a new file with the next higher file version
number, try these settings. >

```
:set nobackup        " does not create *.*_ backup files
:set nowritebackup   " does not have any purpose on VMS.  It's the
                     " default.
```

Recovery is working perfectly as well from the default swap file.
Read more with :help swapfile

(Claude Marinier <ClaudeMarinier@xwavesolutions.com> Vim 5.5, Zoltan Arpadffy
Vim 5.6)

8.4 Directory conversion

Vim will internally convert any unix-style paths and even mixed unix/VMS
paths into VMS style paths. Some typical conversions resemble:

```
/abc/def/ghi            -> abc:[def]ghi.
/abc/def/ghi.j          -> abc:[def]ghi.j
/abc/def/ghi.j;2        -> abc:[def]ghi.j;2
/abc/def/ghi/jkl/mno    -> abc:[def.ghi.jkl]mno.
abc:[def.ghi]jkl/mno    -> abc:[def.ghi.jkl]mno.
  ./                    -> current directory
  ../                   -> relative parent directory
  [.def.ghi]            -> relative child directory
   ./def/ghi            -> relative child directory
```

Note: You may use <,> brackets as well (device:<path>file.ext;version) as
rf10:<user.zay.work>test.c;1

(David Elins <delins@foliage.com>, Jerome Lauret
<JLAURET@mail.chem.sunysb.edu> Vim 5.6)

8.5 Remote host invocation

It is possible to use Vim as an internode editor.
1. Edit some file from remote node: >

```
vi "<server>""username passwd""::<device>:<path><filename>;<version>"
```

Example: >

```
vi "pluto""zay passwd""::RF10:<USER.ZAY.WORK>TEST.C;1"
```

Note: syntax is very important, otherwise VMS will recognize more parameters
instead of one (resulting with: file not found)

2. Set up Vim as your internode editor. If Vim is not installed on your
host, just set up your IP address, the full Vim path including the server name
and run the command procedure below: >

```
$ if (p1 .eqs. "") .OR. (p2 .eqs. "") then goto usage
$ set disp/create/node=<your_IP_here>/trans=tcpip
$ define "VIM "<vim_server>""''p1' ''p2'""::<device>:<vim_path>"
$  vi*m :== "mcr VIM:VIM.EXE"
$ gv*im :== "spawn/nowait mcr VIM:VIM.EXE -g"
$ goto end
```

```
        $ usage:
        $ write sys$output " Please enter username and password as a parameter."
        $ write sys$output " Example: @SETVIM.COM username passwd"
        $ end:
```

Note: Never use it in a clustered environment (you do not need it), loading
could be very-very slow, but even faster than a local Emacs. :-)

(Zoltan Arpadffy, Vim 5.6)

8.6 Terminal problems

If your terminal name is not known to Vim and it is trying to find the default
one you will get the following message during start-up:

```
---
Terminal entry not found in termcap
'unknown-terminal' not known.  Available built-in terminals are:
    builtin_gui
    builtin_riscos
    builtin_amiga
    builtin_beos-ansi
    builtin_ansi
    builtin_vt320
    builtin_vt52
    builtin_pcansi
    builtin_win32
    builtin_xterm
    builtin_iris-ansi
    builtin_debug
    builtin_dumb
defaulting to 'vt320'
---
```

The solution is to define the default terminal name: >

```
        $ ! unknown terminal name.  Let us use vt320 or ansi instead.
        $ ! Note: it's case sensitive
        $ define term "vt320"
```

Terminals from VT100 to VT320 (as V300, VT220, VT200) do not need any extra
keyboard mappings. They should work perfectly as they are, including arrows,
Ins, Del buttons etc., except Backspace in GUI mode. To solve it, add to
.gvimrc: >

```
        inoremap <Del> <BS>
```

Vim will also recognize that they are fast terminals.

If you have some annoying line jumping on the screen between windows add to
your .vimrc file: >

```
        set ttyfast     " set fast terminal
```

Note: if you're using Vim on remote host or through a very slow connection, it's
recommended to avoid the fast terminal option with: >

```
        set nottyfast   " set terminal to slow mode
```

(Zoltan Arpadffy, Vim 5.6)

8.7 Hex-editing and other external tools

A very important difference between OpenVMS and other systems is that VMS uses
special commands to execute executables: >

```
        RUN <path>filename
        MCR <path>filename <parameters>
```

OpenVMS users always have to be aware that the Vim command :! "just" drop them
to DCL prompt. This feature is possible to use without any problem with all
DCL commands, but if we want to execute some programs such as XXD, CTAGS, JTAGS,
etc. we're running into trouble if we follow the Vim documentation (see: help
xxd).

Solution: Execute with the MC command and add the full path to the executable.
Example: Instead of :%!xxd command use: >

```
        :%!mc vim:xxd
```

... or in general: >

```
        :!mc <path>filename <parameters>
```

Note: You can use XXD and CTAGS from GUI menu.

To customize ctags it is possible to define the logical $CTAGS with standard
parameters as: >

```
        define/nolog CTAGS "--totals -o sys$login:tags"
```

For additional information, please read :help tagsearch and CTAGS
documentation at http://ctags.sourceforge.net/ctags.html.

(Zoltan Arpadffy, Vim 5.6-70)

8.8 Sourcing vimrc and gvimrc

If you want to use your .vimrc and .gvimrc from other platforms (e.g. Windows)
you can get in trouble if you ftp that file(s): VMS has different end-of-line
indication.
The symptom is that Vim is not sourcing your .vimrc/.gvimrc, even if you say:
>

```
        :so sys$login:.vimrc
```

One trick is to compress (e.g. zip) the files on the other platform and
uncompress it on VMS; if you have the same symptom, try to create the files
with copy-paste (for this you need both op. systems reachable from one
machine, e.g. an Xterm on Windows or telnet to Windows from VMS).

(Sandor Kopanyi, <sandor.kopanyi@mailbox.hu> Vim 6.0a)

8.9 Printing from Vim

To be able to print from Vim (running in GUI mode) under VMS you have to set
up $TMP logical which should point to some temporary directory and logical
SYS$PRINT to your default print queue.
Example: >

```
        $define SYS$PRINT HP5ANSI
```

You can print out the whole buffer or just the marked area.
More info under :help hardcopy

(Zoltan Arpadffy, Vim 6.0c)

8.10 Setting up the symbols

When I use gvim this way and press CTRL-Y in the parent terminal, gvim exits.
I now use a different symbol that seems to work OK and fixes the problem.
I suggest this instead: >

```
        $ GV*IM:==SPAWN/NOWAIT/INPUT=NLA0: MCR VIM:VIM.EXE -G -GEOMETRY 80X40
```

The /INPUT=NLA0: separates the standard input of the gvim process from the
parent terminal, to block signals from the parent window.
Without the -GEOMETRY, the gvim window size will be minimal and the menu
will be confused after a window-resize.

(Carlo Mekenkamp, Coen Engelbarts, Vim 6.0ac)

8.11 diff and other GNU programs

From 6.0 diff functionality has been implemented, but OpenVMS does not use
GNU/Unix like diff therefore built in diff does not work.
There is a simple solution to solve this anomaly. Install a Unix like diff
and Vim will work perfectly in diff mode too. You just have to redefine your
diff program as: >

```
        define /nolog diff <GNU_PATH>diff.exe
```

Another, more sophisticated solution is described below (8.12 diff-mode)
There are other programs such as patch, make etc that may cause the same
problems. At www.polarhome.com is possible to download an GNU package for
Alpha and VAX boxes that is meant to solve GNU problems on OpenVMS.
(Zoltan Arpadffy, Vim 6.1)

8.12 diff-mode

Vim 6.0 and higher supports Vim diff-mode (See |new-diff-mode|, |diff-mode|
and |08.7|). This uses the external program 'diff' and expects a Unix-like
output format from diff. The standard VMS diff has a different output
format. To use Vim on VMS in diff-mode, you need to:
    1 Install a Unix-like diff program, e.g. GNU diff
    2 Tell Vim to use the Unix-like diff for diff-mode.

You can download GNU diff from the VIM-VMS website, it is one of the GNU
tools in http://www.polarhome.com/vim/files/gnu_tools.zip. I suggest to
unpack it in a separate directory "GNU" and create a logical GNU: that
points to that directory, e.g: >

```
   DEFINE GNU    <DISK>:[<DIRECTORY>.BIN.GNU]
```

You may also want to define a symbol GDIFF, to use the GNU diff from the DCL
prompt: >

```
   GDIFF :==     $GNU:DIFF.EXE
```

Now you need to tell Vim to use the new diff program. Take the example
settings from |diff-diffexpr| and change the call to the external diff
program to the new diff on VMS. Add this to your .vimrc file: >

```
     " Set up vimdiff options
       if v:version >= 600
        " Use GNU diff on VMS
        set diffexpr=MyDiff()
        function MyDiff()
           let opt = ""
           if &diffopt =~ "icase"
             let opt = opt . "-i "
           endif
           if &diffopt =~ "iwhite"
             let opt = opt . "-b "
           endif
           silent execute "!mc GNU:diff.exe -a " . opt . v:fname_in . " " .
v:fname_new .
                \  " > " . v:fname_out
        endfunction
       endif
```

You can now use Vim in diff-mode, e.g. to compare two files in read-only mode: >

```
    $ VIM -D/R <FILE1> <FILE2>
```

You can also define new symbols for vimdiff, e.g.: >

```
    $ VIMDIFF     :== 'VIM' -D/R
    $ GVIMDIFF    :== 'GVIM' -D/R
```

You can now compare files in 4 ways: >

```
    1. VMS  diff:  $ DIFF     <FILE1> <FILE2>
    2. GNU  diff:  $ GDIFF    <FILE1> <FILE2>
    3. VIM  diff:  $ VIMDIFF  <FILE1> <FILE2>
    4. GVIM diff:  $ GVIMDIFF <FILE1> <FILE2>
```

(Coen Engelbarts, Vim 6.1)

8.13 Allow '$' in C keywords

DEC C uses many identifiers with '$' in them. This is not allowed in ANSI C, and Vim recognises the '$' as the end of the identifier. You can change this with the 'iskeyword' option.
Add this command to your .vimrc file: >

```
    autocmd FileType c,cpp,cs  set iskeyword+=$
```

You can also create the file(s) $VIM/FTPLUGIN/C.VIM (and/or CPP.VIM and CS.VIM) and add this command: >

```
        set iskeyword+=$
```

Now word-based commands, e.g. the '*'-search-command and the CTRL-] tag-lookup, work on the whole identifier. (Ctags on VMS also supports '$' in C keywords since ctags version 5.1.)

(Coen Engelbarts, Vim 6.1)

8.14 VIMTUTOR for beginners

The VIMTUTOR.COM DCL script can help Vim beginners to learn/make their first steps with Vim on OpenVMS. Depending of binary distribution you may start it

```
with: >

        @vim:vimtutor

(Thomas.R.Wyant III, Vim 6.1)

8.16 Slow start in console mode issue

As GUI/GTK Vim works equally well in console mode, many administrators
deploy those executables system wide.
Unfortunately, on a remote slow connections GUI/GTK executables behave rather
slow when user wants to run Vim just in the console mode - because of X
environment detection timeout.

Luckily, there is a simple solution for that. Administrators need to deploy
both GUI/GTK build and just console build executables, like below: >

    |- vim73
    |----- doc
    |----- syntax
       vimrc    (system rc files)
       gvimrc
       gvim.exe (the renamed GUI or GTK built vim.exe)
       vim.exe  (the console only executable)

Define system symbols like below in for ex in LOGIN.COM or SYLOGIN.COM: >

        $ define/nolog VIM RF10:[UTIL.VIM73] ! where you VIM directory is
        $ vi*m  :== mcr VIM:VIM.EXE
        $ gvi*m :== mcr VIM:GVIM.EXE
        $ ! or you can try to spawn with
        $ gv*im :== spawn/nowait/input=NLA0 mcr VIM:GVIM.EXE -g -GEOMETRY 80x40


Like this, users that do not have X environment and want to use Vim just in
console mode can avoid performance problems.

(Zoltan Arpadffy, Vim 7.2)

8.15 Common VIM directory - different architectures

In a cluster that contains nodes with different architectures like below:

$show cluster
View of Cluster from system ID 11655  node:
TOR                                                                  18-AUG-2008
11:58:31
+---------------------------------+
\A6        SYSTEMS        \A6 MEMBERS \A6
+-----------------------+---------\A6
\A6  NODE  \A6   SOFTWARE   \A6  STATUS \A6
+--------+--------------+---------\A6
\A6 TOR    \A6 VMS V7.3-2   \A6 MEMBER  \A6
\A6 TITAN2 \A6 VMS V8.3     \A6 MEMBER  \A6
\A6 ODIN   \A6 VMS V7.3-2   \A6 MEMBER  \A6
+---------------------------------+

It is convenient to have a common VIM directory but execute different
executables.
There are several solutions for this problem:

Solution 1.  All executables in the same directory with different names
```

```
This is easily done with the following script that can be added
to the login.com or sylogin.com: >

        $ if f$getsyi("NODE_HWTYPE") .eqs. "VAX"
        $ then
        $       say "VAX platform"
        $       vi*m:== mcr vim:VIM.EXE_VAX
        $ endif
        $ if f$getsyi("NODE_HWTYPE") .eqs. "ALPH"
        $ then
        $       say "ALPHA platform"
        $       vi*m :== mcr vim:VIM.EXE_AXP
        $ endif
        $ if f$getsyi("ARCH_NAME") .eqs. "IA64"
        $ then
        $      say "IA64 platform"
        $      vi*m :== mcr vim:VIM.EXE_IA64
        $ endif

Solution 2.  Different directories: >

        $ if f$getsyi("NODE_HWTYPE") .eqs. "VAX"
        $ then
        $       say "VAX platform"
        $       define/nolog VIM RF10:[UTIL.VAX_EXE] ! VAX executables
        $ endif
        $ if f$getsyi("NODE_HWTYPE") .eqs. "ALPH"
        $ then
        $       say "ALPHA platform"
        $       define/nolog VIM RF10:[UTIL.AXP_EXE] ! AXP executables
        $ endif
        $ if f$getsyi("ARCH_NAME") .eqs. "IA64"
        $ then
        $      say "IA64 platform"
        $      define/nolog VIM RF10:[UTIL.IA64_EXE] ! IA64 executables
        $ endif
        $! VIMRUNTIME must be defined in order to find runtime files
        $ define/nolog VIMRUNTIME RF10:[UTIL.VIM73]

A good example for this approach is the [GNU]gnu_tools.com script from
GNU_TOOLS.ZIP package downloadable from http://www.polarhome.com/vim/

(Zoltan Arpadffy, Vim 7.2)

==============================================================================

9. VMS related changes                                  *vms-changes*

Version 7.4
- Undo: VMS can not handle more than one dot in the filenames use "dir/name" -> "dir/
_un_name"
  add _un_ at the beginning to keep the extension
- correct swap file name wildcard handling
- handle iconv usage correctly
- do not optimize on vax - otherwise it hangs compiling crypto files
- fileio.c fix the comment
- correct RealWaitForChar
- after 7.4-119 use different functions lib$cvtf_to_internal_time because Alpha and
VAX have
  G_FLOAT but IA64 uses IEEE float otherwise Vim crashes
- guard against crashes that are caused by mixed filenames
- [TESTDIR]make_vms.mms changed to see the output files
```

```
- Improve tests, update known issues
- minor compiler warnings fixed
- CTAGS 5.8 +regex included

Version 7.3
- CTAGS 5.8 included
- VMS compile warnings fixed - floating-point overflow warning corrected on VAX
- filepath completion corrected - too many chars were escaped in filename
  and shell commands
- the following plugins are included into VMS runtime:
        genutils 2.4, multiselect 2.2, multvals 3.1, selectbuf 4.3,
        bufexplorer 7.1.7, taglist 4.5
- minor changes in vimrc (just in VMS runtime)
- make_vms.mms - HUGE model is the default
- [TESTDIR]make_vms.mms include as many tests possible
- modify test30 and test54 for VMS
- enable FLOAT feature in VMS port
- os_vms.txt updated

Version 7.2 (2008 Aug 9)
- VCF files write corrected
- CTAGS 5.7 included
- corrected make_vms.mms (on VAX gave syntax error)

Version 7.1 (2007 Jun 15)
- create TAGS file from menu

Version 7 (2006 May 8)
- Improved low level char input (affects just console mode)
- Fixed plugin bug
- CTAGS 5.6 included

Version 6.4 (2005 Oct 15)
- GTKLIB and Vim build on IA64
- colors in terminal mode
- syntax highlighting in terminal mode
- write problem fixed (extra CR)
- ESC and ESC sequence recognition in terminal mode
- make file changed to support new MMS version
- env variable expansion in path corrected
- printing problems corrected
- help text added for case insensitive arguments

Version 6.3 (2004 May 10)
- Improved vms_read function
- CTAGS v5.5.4 included
- Documentation corrected and updated

Version 6.2 (2003 May 7)
- Corrected VMS system call results
- Low level character input is rewritten
- Correction in tag and quickfix handling
- First GTK build
- Make file changes
    - GTK feature added
    - Define for OLD_VMS
    - OpenVMS version 6.2 or older
- Documentation updated with GTK features
- CTAGS v5.5 included
- VMS VIM tutor created

Version 6.1 (2002 Mar 25)
```

```
- TCL init_tcl() problem fixed
- CTAGS v5.4 included
- GNU tools binaries for OpenVMS
- Make file changes
    - PERL, PYTHON and TCL support improved
    - InstallVMS.txt has a detailed description HOWTO build
- VMS/Unix file handling rewritten
- Minor casting and bug fixes

Version 6.0 (2001 Sep 28)
- Unix and VMS code has been merged
        - separated "really" VMS related code
        - included all possible Unix functionality
        - simplified or deleted the configuration files
        - makefile MAKE_VMS.MMS reviewed
- menu changes (fixed printing, CTAGS and XXD usage)
- fixed variable RMS record format handling anomaly
- corrected syntax, ftplugin etc files load
- changed expand_wildcards and expandpath functions to work more general
- created OS_VMS_FILTER.COM - DECC->VAXC pre-processor directive convert
  script.
- Improved code's VAXC and new DECC compilers compatibility
- changed quickfix parameters:
        - errormessage format to suite DECC
        - search, make and other commands to suite VMS system
- updated and renamed MMS make files for Vim and CTAGS.
- CTAGS has been removed from source distribution of Vim but it will remain
  in OpenVMS binary distributions.
- simplified build/configuration procedure
- created INSTALLvms.txt - detailed compiling instructions under VMS.
- updated test scripts.

Version 5.8 (2001 Jun 1)
- OS_VMS.TXT updated with new features.
- other minor fixes.
- documentation updated
- this version had been tested much more than any other OpenVMS version
  earlier

Version 5.7 (2000 Jun 24)
- New CTAGS v5.0 in distribution
- Documentation updated

Version 5.6 (2000 Jan 17)
- VMS filename related changes:
        - version handling (open everything, save to new version)
        - correct file extension matching for syntax (version problem)
        - handle <,> characters and passwords in directory definition
        - handle internode/remote invocation and editing with passwords
        - OpenVMS files will be treated case insensitive from now
        - corrected response of expand("%:.") etc path related functions
        (in one word: VMS directory handling internally)
- version command
        - corrected (+,-) information data
        - added compiler and OS version
        - added user and host information
        - resolving $VIM and $VIMRUNTIME logicals
- VMS port is in MAX_FEAT (maximum features) club with Unix, Win32 and OS/2.
        - enabled farsi, rightleft etc. features
        - undo level raised up to 1000
- Updated OS_VMS.MMS file.
        - maximum features ON is default
```

```
        - Vim is compilable with +perl, +python and +tcl features.
        - improved MMK compatibility
- Created MAKEFILE_VMS.MMS, makefile for testing Vim during development.
- Defined DEC terminal VT320
        - compatibility for VT3*0, VT2*0 and VT1*0 - ANSI terminals
          backwards, but not VT340 and newer with colour capability.
        - VT320 is default terminal for OpenVMS
        - these new terminals are also fast ttys (default for OpenVMS).
        - allowed dec_mouse ttym
- Updated files vimrc and gvimrc with VMS specific suggestions.
- OS_VMS.TXT updated with new features.

Version 5.5 (1999 Dec 3)
- Popup menu line crash corrected.
- Handle full file names with version numbers.
- Directory handling (CD command etc.)
- Corrected file name conversion VMS to Unix and v.v.
- Correct response of expand wildcards
- Recovery is working from this version under VMS as well.
- Improved terminal and signal handing.
- Improved OS_VMS.TXT

Version 5.4 (1999 Sep 9)
- Cut and paste mismatch corrected.
- Motif directories during open and save are corrected.

Version 5.3 (1998 Oct 12)
- Minor changes in the code
- Standard distribution with +GUI option

Version 5.1 (1998 Apr 21)
- Syntax and DEC C changes in the code
- Fixing problems with the /doc subdirectory
- Improve OS_VMS.MMS

Version 4.5 (1996 Dec 16)
- First VMS port by Henk Elbers <henk@xs4all.nl>

==============================================================================

10. Authors                                             *vms-authors*

OpenVMS documentation and executables are maintained by:
Zoltan Arpadffy <arpadffy@polarhome.com>
OpenVMS Vim page: http://www.polarhome.com/vim/

This document uses parts and remarks from earlier authors and contributors
of OS_VMS.TXT:
        Charles E. Campbell, Jr. <cec@gryphon.gsfc.nasa.gov>
        Bruce Hunsaker <BNHunsaker@chq.byu.edu>
        Sandor Kopanyi <sandor.kopanyi@mailbox.hu>

 vim:tw=78:ts=8:ft=help:norl:
*os_win32.txt*  For Vim version 8.0.  Last change: 2017 Mar 21


                  VIM REFERENCE MANUAL    by George Reilly


                                                *win32* *Win32* *MS-Windows*
This file documents the idiosyncrasies of the Win32 version of Vim.
```

The Win32 version of Vim works on Windows XP, Vista, 7, 8 and 10. There are
both console and GUI versions.

The 32 bit version also runs on 64 bit MS-Windows systems.

Credits:
The Win32 version was written by George V. Reilly <george@reilly.org>.
The original Windows NT port was done by Roger Knobbe <RogerK@wonderware.com>.
The GUI version was made by George V. Reilly and Robert Webb.

For compiling see "src/INSTALLpc.txt".   *win32-compiling*

==
1. Known problems   *win32-problems*

When doing file name completion, Vim also finds matches for the short file
name. But Vim will still find and use the corresponding long file name. For
example, if you have the long file name "this_is_a_test" with the short file
name "this_i~1", the command ":e *1" will start editing "this_is_a_test".

==
2. Startup   *win32-startup*

Current directory   *win32-curdir*

If Vim is started with a single file name argument, and it has a full path
(starts with "x:\"), Vim assumes it was started from the file explorer and
will set the current directory to where that file is. To avoid this when
typing a command to start Vim, use a forward slash instead of a backslash.
Example: >

        vim c:\text\files\foo.txt

Will change to the "C:\text\files" directory. >

        vim c:/text\files\foo.txt

Will use the current directory.

Term option   *win32-term*

The only kind of terminal type that the Win32 version of Vim understands is
"win32", which is built-in. If you set 'term' to anything else, you will
probably get very strange behavior from Vim. Therefore Vim does not obtain
the default value of 'term' from the environment variable "TERM".

$PATH *win32-PATH*

The directory of the Vim executable is appended to $PATH. This is mostly to
make "!xxd" work, as it is in the Tools menu. And it also means that when
executable() returns 1 the executable can actually be executed.

Quotes in file names *win32-quotes*

Quotes inside a file name (or any other command line argument) can be escaped
with a backslash. E.g. >
        vim -c "echo 'foo\"bar'"

Alternatively use three quotes to get one: >
        vim -c "echo 'foo"""bar'"

The quotation rules are:

1. A `"` starts quotation.
2. Another `"` or `"""` ends quotation. If the quotation ends with `"""`, a `"`
   is produced at the end of the quoted string.

Examples, with [] around an argument:
        "foo"   -> [foo]
        "foo""   -> [foo"]
        "foo"bar   -> [foobar]
        "foo" bar   -> [foo], [bar]
        "foo""bar   -> [foo"bar]
        "foo"" bar   -> [foo"], [bar]
        "foo"""bar"   -> [foo"bar]

==
3. Restore screen contents *win32-restore*

When 'restorescreen' is set (which is the default), Vim will restore the
original contents of the console when exiting or when executing external
commands. If you don't want this, use ":set nors". |'restorescreen'|

==
4. Using the mouse *win32-mouse*

The Win32 version of Vim supports using the mouse. If you have a two-button
mouse, the middle button can be emulated by pressing both left and right
buttons simultaneously - but note that in the Win32 GUI, if you have the right
mouse button pop-up menu enabled (see 'mouse'), you should err on the side of
pressing the left button first. |mouse-using|

When the mouse doesn't work, try disabling the "Quick Edit Mode" feature of
the console.

==
5. Running under Windows 95 *win32-win95*
                                    windows95 *windows98* *windowsme*
Windows 95/98/ME support was removed in patch 8.0.0029 If you want to use it
you will need to get a version older than that.

```
==============================================================================
6. Running under Windows 3.1                                *win32-win3.1*

                                          *win32s* *windows-3.1* *gui-w32s*
There was a special version of gvim that runs under Windows 3.1 and 3.11.
Support was removed in patch 7.4.1363.

==============================================================================
7. Win32 mini FAQ                                            *win32-faq*

Q. How do I change the font?
A. In the GUI version, you can use the 'guifont' option.  Example: >
        :set guifont=Lucida_Console:h15:cDEFAULT
<  In the console version, you need to set the font of the console itself.
   You cannot do this from within Vim.

Q. How do I type dead keys on Windows NT?
A. Dead keys work on NT 3.51.  Just type them as you would in any other
   application.
   On NT 4.0, you need to make sure that the default locale (set in the
   Keyboard part of the Control Panel) is the same as the currently active
   locale.  Otherwise the NT code will get confused and crash!  This is a NT
   4.0 problem, not really a Vim problem.

Q. I'm using Vim to edit a symbolically linked file on a Unix NFS file server.
   When I write the file, Vim does not "write through" the symlink.  Instead,
   it deletes the symbolic link and creates a new file in its place.  Why?
A. On Unix, Vim is prepared for links (symbolic or hard).  A backup copy of
   the original file is made and then the original file is overwritten.  This
   assures that all properties of the file remain the same.  On non-Unix
   systems, the original file is renamed and a new file is written.  Only the
   protection bits are set like the original file.  However, this doesn't work
   properly when working on an NFS-mounted file system where links and other
   things exist.  The only way to fix this in the current version is not
   making a backup file, by ":set nobackup nowritebackup"     |'writebackup'|

Q. I'm using Vim to edit a file on a Unix file server through Samba.  When I
   write the file, the owner of the file is changed.  Why?
A. When writing a file Vim renames the original file, this is a backup (in
   case writing the file fails halfway).  Then the file is written as a new
   file.  Samba then gives it the default owner for the file system, which may
   differ from the original owner.
   To avoid this set the 'backupcopy' option to "yes".  Vim will then make a
   copy of the file for the backup, and overwrite the original file.  The
   owner isn't changed then.

Q. How do I get to see the output of ":make" while it's running?
A. Basically what you need is to put a tee program that will copy its input
   (the output from make) to both stdout and to the errorfile.  You can find a
   copy of tee (and a number of other GNU tools) at
   http://gnuwin32.sourceforge.net or http://unxutils.sourceforge.net
   Alternatively, try the more recent Cygnus version of the GNU tools at
   http://www.cygwin.com  Other Unix-style tools for Win32 are listed at
   http://directory.google.com/Top/Computers/Software/Operating_Systems/Unix/Win32/
   When you do get a copy of tee, you'll need to add >
        :set shellpipe=\|\ tee
<  to your _vimrc.

Q. I'm storing files on a remote machine that works with VisionFS, and files
   disappear!
A. VisionFS can't handle certain dot (.) three letter extension file names.
   SCO declares this behavior required for backwards compatibility with 16bit
```

```
   DOS/Windows environments.  The two commands below demonstrate the behavior:
>
        echo Hello > file.bat~
        dir > file.bat
<
   The result is that the "dir" command updates the "file.bat~" file, instead
   of creating a new "file.bat" file.  This same behavior is exhibited in Vim
   when editing an existing file named "foo.bat" because the default behavior
   of Vim is to create a temporary file with a '~' character appended to the
   name.  When the file is written, it winds up being deleted.

   Solution: Add this command to your _vimrc file: >
        :set backupext=.temporary

Q. How do I change the blink rate of the cursor?
A. You can't!  This is a limitation of the NT console.  NT 5.0 is reported to
   be able to set the blink rate for all console windows at the same time.

                                                              *:!start*
Q. How can I asynchronously run an external command or program, or open a
   document or URL with its default program?
A. When using :! to run an external command, you can run it with "start". For
   example, to run notepad: >
        :!start notepad
<   To open "image.jpg" with the default image viewer: >
        :!start image.jpg
<   To open the folder of the current file in Windows Explorer: >
        :!start %:h
<   To open the Vim home page with the default browser: >
        :!start http://www.vim.org/
<
   Using "start" stops Vim switching to another screen, opening a new console,
   or waiting for the program to complete; it indicates that you are running a
   program that does not affect the files you are editing.  Programs begun
   with :!start do not get passed Vim's open file handles, which means they do
   not have to be closed before Vim.
   To avoid this special treatment, use ":! start".
   There are two optional arguments (see the next Q):
       /min  the window will be minimized
       /b    no console window will be opened
   You can use only one of these flags at a time.  A second one will be
   treated as the start of the command.

Q. How do I avoid getting a window for programs that I run asynchronously?
A. You have two possible solutions depending on what you want:
   1) You may use the /min flag in order to run program in a minimized state
      with no other changes. It will work equally for console and GUI
      applications.
   2) You can use the /b flag to run console applications without creating a
      console window for them (GUI applications are not affected). But you
      should use this flag only if the application you run doesn't require any
      input.  Otherwise it will get an EOF error because its input stream
      (stdin) would be redirected to \\.\NUL (stdout and stderr too).

   Example for a console application, run Exuberant ctags: >
        :!start /min ctags -R .
<  When it has finished you should see file named "tags" in your current
   directory.  You should notice the window title blinking on your taskbar.
   This is more noticeable for commands that take longer.
   Now delete the "tags" file and run this command: >
        :!start /b ctags -R .
<  You should have the same "tags" file, but this time there will be no
```

```
   blinking on the taskbar.
   Example for a GUI application: >
        :!start /min notepad
        :!start /b notepad
<  The first command runs notepad minimized and the second one runs it
   normally.

                                                    *windows-icon*
Q. I don't like the Vim icon, can I change it?
A. Yes, place your favorite icon in bitmaps/vim.ico in a directory of
   'runtimepath'.  For example ~/vimfiles/bitmaps/vim.ico.


 vim:tw=78:fo=tcq2:ts=8:ft=help:norl:
```