Vim Color Editor HOW−TO (Vi Improved with syntax color highlighting)

Table of Contents

Table of Contents

Vim Color Editor HOW−TO (Vi Improved with syntax color highlighting)

Al Dev (Alavoor Vasudevan) alavoor@yahoo.com

v15.0, 09 Dec 2000

This document is a guide to quickly setting up the Vim color editor on Linux or Unix systems. The information here will improve the productivity of programmers because the Vim editor supports syntax color highlighting and bold fonts, improving the "readability" of program code. A programmer's productivity improves 2 to 3 times with a color editor like Vim. The information in this document applies to all operating sytems where Vim works, such as Linux, Windows 95/NT, Apple Mac, IBM OSes, VMS, BeOS and all flavors of Unix like Solaris, HPUX, AIX, SCO, Sinix, BSD, Ultrix etc.. (it means almost all operating systems on this planet!)

---

6. Online VIM help

7. Vim Home page and Vim links

8. Vim Tutorial

9. Vi Tutorial

10. Vim Reference Card

---

1. Introduction

Vim stands for 'Vi Improved'. Vi is the most popular and powerful editors in the Unix world. **Vi** is an abbreviation for "***Vi***sual" editor. One of the first editors was a line editor called 'ed' (and 'ex'). The ***Vi***sual editor like Vi was a vast improvement over line editors like 'ed' (or 'ex'). The editors 'ed' and 'ex' are still available on Linux: see 'man ed' and 'man ex'.

A good editor improves programmer productivity. Vim supports color syntax highlighting of program code and also emphasises text using different fonts like normal, bold or italics. A color editor like Vim can improve the **productivity** of programmers by **2 to 3 times**!! Programmers can read the code much more rapidly as the code syntax is colored and highlighted.

1.1 Before you Install

Before you install Vim, please refer to the OS specific release notes and information about compiling and usage of Vim at −

- Go to this location and look for files os_*.txt http://cvs.vim.org/cgi−bin/cvsweb/vim/runtime/doc

If you do not have the Vim package (RPM, DEB, tar, zip) then download the Vim source code by ftp from the official Vim site

- The home page of vim is at http://www.vim.org
- Mirror site in US is at http://www.us.vim.org
- Ftp site ftp://ftp.vim.org/pub/vim
- Or use one of the mirrors ftp://ftp.vim.org/pub/vim/MIRRORS

1.2 Install Vim on Redhat Linux

To use Vim install the following RPM packages on Redhat Linux −

```
        rpm -i vim*.rpm
OR do this -
        rpm -i vim-enhanced*.rpm
        rpm -i vim-X11*.rpm
        rpm -i vim-common*.rpm
        rpm -i vim-minimal*.rpm
```

You can see the list of files the vim rpm installs by −

```
        rpm −qa | grep ^vim | xargs rpm −ql | less
or
        rpm −qa | grep ^vim | awk '{print "rpm −ql " $1 }' | /bin/sh | less
```

and browse output using j,k, CTRL+f, CTRL+D, CTRL+B, CTRL+U or using arrow keys, page up/down keys. See 'man less'.

Note that the RPM packages for Redhat Linux use a Motif interface. If you have installed the GTK libraries on your system, consider compiling Vim from the source code for a clean GUI interface. For information on compiling Vim from the source code, see "Install Vim on Unixes", below.

1.3 Install Vim on GNU Debian Linux

To install Vim on Debian Linux (GNU Linux), login as root and when connected to internet type −

```
apt−get install vim vim−rt
```

It will download the latest version of vim, install it, configure it. The first package listed is vim, the standard editor, compiled with X11 support, vim−rt is the vim runtime, it holds all the syntax and help files.

1.4 Install Vim on Unixes

For other flavors of unixes like Solaris, HPUX, AIX, Sinix, SCO download the source code file (see Before you Install)

```
        zcat vim.tar.gz | tar −xvf −
        cd vim−5.5/src
        ./configure −−enable−gui=motif
        make
        make install
```

1.5 Install Vim on Microsoft Windows 95/NT

For Windows 95/NT, download the Vim zip file. For Windows 95/NT you must download **TWO** zip files −

- Runtime support file **vim*rt.zip**
- Vim command file **vim*56.zip**. Where Vim version is 5.6.

Get these two zip files (see Before you Install) Unpack the zip files using the Winzip http://www.winzip.com. Both the zip files (vim*rt.zip and vim*56.zip) must be unpacked in the same directory like say **c:\vim**.

For Windows 95/98, set the environment variable VIM in autoexec.bat by adding this line −

```
set VIM=c:\vim\vim56
```

For Windows NT, add the environment variable VIM to the **Control Panel | System | Environment | System Properties** dialog:

```
VIM=c:\vim\vim56
```

The VIM variable should point to whereever you installed the vim56 directory. You can also set your PATH to include the gvim.exe's path.

You may need to logoff and relogin to set your environment. At an MS−DOS prompt type −

```
        set vim
```

And you should see − VIM=c:\vim\vim56

Create a short−cut on to your desktop by click−and−drag from c:\vim\vim56\gvim.exe. Copy the gvimrc_example file to the $VIM_gvimrc. In my case it is c:\vim\vim56_gvimrc.

1.6 Install Vim on VMS

Download files

You will need both the Unix and Extra archives to build vim.exe for VMS. For using Vim's full power you will need the runtime files as well. Get these files (see Before you Install)

You can download precompiled executables from: http://www.polarfox.com/vim

VMS vim authors are −

- zoltan.arpadffy@essnet.se
- arpadffy@altavista.net
- cec@gryphon.gsfc.nasa.gov
- BNHunsaker@chq.byu.edu
- sandor.kopanyi@altavista.net

Compiling

Unpack the Unix and Extra archives together into one directory. In the <.SRC> subdirectory you should find the make file OS_VMS.MMS. By editing this file you may choose between building the character, GUI and debug version. There are also additional options for Perl, Python and Tcl support.

You will need either the DECSET mms utility or the freely available clone of it called mmk (VMS has no make utility in the standard distribution). You can download mmk from http://www.openvms.digital.com/freeware/MMK/

If you have MSS on your system, the command

> mms /descrip=os_vms.mms

will start building your own customised version of Vim. The equivalent command for mmk is:

> mmk /descrip=os_vms.mms

Deploy

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

```
   vim (or wherever)
    |- tmp
    |- vim55
    |----- doc
    |----- syntax
    |- vim56
    |----- doc
    |----- syntax
    vimrc    (system rc files)
    gvimrc

Use:

>       define/nolog device:[leading-path-here.vim]       vim
>       define/nolog device:[leading-path-here.vim.vim56] vimruntime
>       define/nolog device:[leading-path-here.tmp]       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.

Note: Logicals $VIMRUNTIME and $TMP are optional. Read more at :help runtime

Practical 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 all Vim runtime directory structure to the deployment position. Add the following lines to your LOGIN.COM (in SYS$LOGIN directory). Set up logical $VIM as:

```
>       $ define VIM device: <path>
```

Set up some symbols:

```
>       $ ! vi starts Vim in chr. mode.
>       $ vi*m  :== mcr device:<path>VIM.EXE

>       $ !gvi starts Vim in GUI mode.
>       $ gv*im :== spawn/nowait mcr device:<path>VIM.EXE −g
```

Create .vimrc and .gvimrc files in your home directory (SYS$LOGIN).

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 default setup for all users, and for users is enough just to have their own additions or resetting in home directory in files .vimrc and .gvimrc. It should work without problems.

Note: Remember, system rc files (default for all users) do not have the leading "." So, system rc files are:

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

and user's customised rc files are:

```
>       sys$login:.vimrc
>       sys$login:.gvimrc
```

You can check that everything is on 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 mcr VIM:VIM.EXE −g
>       $ set disp/create/node=192.168.5.223/trans=tcpip
```

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

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

as for example:

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

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

GUI mode questions

VMS is not a native X window environment, so you can not start Vim in GUI mode "just like that". But it is not too complicated to get a running Vim.

```
1) If you are working on the VMS X 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 other X window environment as Unix or some 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.

1.7 Install Vim on OS/2

Read the release notes for Vim on OS/2, see Before you Install .

At present there is no native PM version of the GUI version of vim: The OS/2 version is a console application. However, there is now a Win32s−compatible GUI version, which should be usable by owners of Warp 4 (which supports Win32s) in a Win−OS/2 session. The notes in this file refer to the native console version.

To run Vim, you need the emx runtime environment (at least rev. 0.9b). This is generally available as (ask Archie about it):

```
    emxrt.zip     emx runtime package
```

1.8 Install Vim on Apple Macintosh

Read the release notes for Vim on OS/2, see Before you Install .

The author of Vim on Mac (old version vim 3.0) is

```
Eric Fischer
5759 N. Guilford Ave
Indianapolis IN 46220 USA
```

Email to enf@pobox.com

Mac Bug Report When reporting any Mac specific bug or feature change, makes sure to include the following address in the "To:" or "Copy To:" field.

dany.stamant@sympatico.ca

Vim compiles out of the box with the supplied CodeWarrior project when using CodeWarrior 9. If you are using a more recent version (e. g. CW Pro) you have to convert the project first. When compiling Vim for 68k Macs you have to open the "size" resource in ResEdit and enable the "High level events aware" button to get drag and drop working. You have to increase the memory partition to at least 1024 kBytes to prevent Vim from crashing due to low memory.

```
 vim:ts=8:sw=8:tw=78:
```

---

2. Setup gvim init files

To enable the syntax color highlighting you MUST copy the gvimrc file to your home directory. This will also put the "Syntax" Menu with gvim command. You can click on Syntax Menu and select appropriate languages like C++, Perl, Java, SQL, ESQL etc..

```
cd $HOME
cp /usr/doc/vim-common-5.3/gvimrc_example  ~/.gvimrc
```

Comment lines in .gvimrc begin with double−quotes ("). You can customize gvim by editing the file $HOME/.gvimrc and put the following lines −

```
" This line is a comment .... one which begins with double-quotes
" The best is the bold font, try all of these and pick one....
set guifont=8x13bold
"set guifont=9x15bold
"set guifont=7x14bold
"set guifont=7x13bold
```

```
"
" Highly recommended to set tab keys to 4 spaces
set tabstop=4
set shiftwidth=4
"
" The opposite is 'set wrapscan' while searching for strings....
set nowrapscan
"
" The opposite is set noignorecase
set ignorecase
```

It is **very strongly** recommended that you set the *tabstop* to 4 and *shiftwidth* to 4. The *tabstop* is the number of spaces the TAB key will indent while editing with gvim. The *shiftwidth* is the number of spaces the lines will be shifted with ">>" or "<<" vi commands. Refer to Vi tutorials Vim Tutorial for more details.

To see the list of available fonts on Linux/Unix see the command **xlsfonts**. Type −

```
        bash$ xlsfonts | less
        bash$ xlsfonts | grep -i bold | grep x
        bash$ man xlsfonts
```

2.1 Sample gvimrc file

You can change the settings like color, bold/normal fonts in your $HOME/.gvimrc file. It is **very strongly** recommended that you set the background color to *lightyellow* or *white* with *black* foreground. Ergonomics says that best background color is *lightyellow* or *white* with *black* foreground. Hence change the variable 'guibg' in your $HOME/.gvimrc file as follows:

```
        highlight Normal guibg=lightyellow
```

The sample gvimrc from /usr/doc/vim−common−5.3/gvimrc_example is as follows:

```
" Vim
" An example for a gvimrc file.
" The commands in this are executed when the GUI is started.
"
" To use it, copy it to
"     for Unix and OS/2:  ~/.gvimrc
"             for Amiga:  s:.gvimrc
"  for MS-DOS and Win32:  $VIM\_gvimrc

" Make external commands work through a pipe instead of a pseudo-tty
"set noguipty

" set the X11 font to use. See 'man xlsfonts' on unix/linux
" set guifont=-misc-fixed-medium-r-normal--14-130-75-75-c-70-iso8859-1
set guifont=8x13bold
"set guifont=9x15bold
"set guifont=7x14bold
```

```
"set guifont=7x13bold
"
" Highly recommended to set tab keys to 4 spaces
set tabstop=4
set shiftwidth=4
"
" The opposite is 'set wrapscan' while searching for strings....
set nowrapscan
"
" The opposite is set noignorecase
set ignorecase

" Make command line two lines high
set ch=2

" Make shift-insert work like in Xterm
map <S-Insert> <MiddleMouse>
map! <S-Insert> <MiddleMouse>

" Only do this for Vim version 5.0 and later.
if version >= 500

  " I like highlighting strings inside C comments
  let c_comment_strings=1

  " Switch on syntax highlighting.
  syntax on

  " Switch on search pattern highlighting.
  set hlsearch

  " For Win32 version, have "K" lookup the keyword in a help file
  "if has("win32")
  "  let winhelpfile='windows.hlp'
  "  map K :execute "!start winhlp32 -k <cword> " . winhelpfile <CR>
  "endif

  " Hide the mouse pointer while typing
  set mousehide

  " Set nice colors
  " background for normal text is light grey
  " Text below the last line is darker grey
  " Cursor is green
  " Constants are not underlined but have a slightly lighter background
  highlight Normal guibg=grey90
  highlight Cursor guibg=Green guifg=NONE
  highlight NonText guibg=grey80
  highlight Constant gui=NONE guibg=grey95
  highlight Special gui=NONE guibg=grey95

endif
```

See also sample vimrc used for console mode vim command from /usr/doc/vim−common−5.3/vimrc_example.

2.2 Xdefaults parameters

You can set some of the Vim properties in Xdefaults file.

WARNING: *Do not set **Vim*geometry** as it will break the gvim menu, use **Vim.geometry** instead.*

Edit the $HOME/.Xdefaults file and add the following lines:

```
! GVim great Colors.
Vim*useSchemes:         all
Vim*sgiMode:            true
Vim*useEnhancedFSB:     true
Vim.foreground:         Black
!Vim.background:        lightyellow2
Vim*background:         white
! Do NOT use Vim*geometry , this will break the menus instead
! use Vim.geometry. Asterisk between Vim and geometry is not allowed.
! Vim.geometry: widthxheight
Vim.geometry:           88x40
!Vim*font:              -misc-fixed-medium-r-normal--20-200-75-75-c-100-iso8859-15-*5
Vim*menuBackground: yellow
Vim*menuForeground: black
```

In order for this change to take effect, type −

```
        xrdb −merge $HOME/.Xdefaults
        man xrdb
```

You can also edit the ~/.gvimrc file to change the background colors

```
        gvim $HOME/.gvimrc
The best background color is lightyellow or white, with black foreground.
        highlight Normal guibg=lightyellow
```

3. Color Syntax init files

3.1 Auto source−in method

This section below is obtained from gvim session by typing 'help syntax' −

```
bash$ gvim some_test
:help syntax
```

Click on Menu Window=>Close_Others to close other Window. And then do CTRL+] on 'Syntax Loading Procedure' menu which will take you there. (Use CTRL+T to rewind and go back).

If a file type that you want to use is not detected yet, there are two ways to add it. It's better not modify the *$VIMRUNTIME/filetype.vim* file. It will be overwritten when installing a new version of Vim. Create a file in $HOME/vim/myfiletypes.vim and put these line in it −

```
" Filename : $HOME/vim/myfiletypes.vim
" myfiletypefile
augroup filetype
        au! BufRead,BufNewFile *.mine   set filetype=mine
        au! BufRead,BufNewFile *.xyz    set filetype=drawing
        au! BufRead,BufNewFile *.prc    set filetype=plsql
augroup END
```

Then add a line in your $HOME/.vimrc and $HOME/.gvimrc file to set the "myfiletypefile" variable to the name of this file. (**CAUTION:** You MUST put this in both vimrc and gvimrc files in order for this to work) Example:

```
        let myfiletypefile = "~/vim/myfiletypes.vim"
```

NOTE: Make sure that you set "myfiletypefile" before switching on file type detection. This is must be before any ":filetype on" or ":syntax on" command.

Your file will then be sourced after the default FileType autocommands have been installed. This allows you to overrule any of the defaults, by using ":au!" to remove any existing FileType autocommands for the same pattern. Only the autocommand to source the scripts.vim file is given later. This makes sure that your autocommands in "myfiletypefile" are used before checking the contents of the file.

3.2 Manual method

Instead of using "Syntax" menu you can also manually source in the syntax file. Edit the file with gvim and at : (colon) command give 'so' command. For example −

```
        gvim foo.pc
        :so $VIM/syntax/esqlc.vim
```

The syntax source files are at /usr/share/vim/syntax/*.vim. Vim supports more than 120 different syntax files for different languages like C++, PERL, VHDL, JavaScript,...and so on!!

Each syntax file supports one or more default file name extensions, for example, JavaScript syntax file supports the *.js extension. If you happen to use an extension that conflicts with another default syntax file (such as adding JavaScript to a *.html file) than you can source in the additional syntax file with the command :so $VIM/syntax/javascript.vim. To avoid all of this typing, you can create a soft link like −

```
        ln −s $VIM/syntax/javascript.vim js
        gvim foo.html  (... this file contains javascript functions and HTML)
        :so js
```

4. VIM Usage

You can use Vim in two modes − one with GUI and other without GUI. To use GUI use command −

```
        gvim foo.cpp
```

To use non−gui mode give −

```
        vim foo.cpp
OR plain vanilla mode
        vi foo.cpp
```

It is very strongly recommended that you always use gvim instead of vim, since GUI mode with colors will definitely improve your productivity.

GUI mode gvim provides the following −

- You can mark the text using the mouse to do cut, copy and paste.
- You can use the Menu bar which has − File, Edit, Window, Tools, Synatx and Help buttons.
- Also in near future in gvim − a second menu bar will display the list of files being edited, and you can switch files by clicking on the filenames, until then you can use vi commands − :e#, :e#1, :e#2, :e#3, :e#4, ...so on to select the files.

5. Vi companions

Generally Vim is used in conjunction with other powerful tools like **ctags** and **gdb**. **ctags** is for very rapid navigation through millions of lines of "C/C++" code and **gdb** is for debugging the "C/C++" code. A brief introduction of these two indispensable commands will be given in this chapter.

ctags is the most powerful command available for coding C, C++, Java, Perl, Korn/Bourne shell scripts or Fortran. Developers very extensively use **ctags** to navigate through thousands of functions within C/C++ programs. See 'man ctags' on Unix. It is **very important** that you learn how to use ctags to develop programs in C or C++, Java, etc.. Navigation is the single most important task while doing development of C or C++ code. Using ctags you can very quickly read the code by jumping from a calling line to the called function, drill down deeper into nested function calls, and unwind back all the way up to the top. You can go back and forth from function to function very quickly.

Without NAVIGATION you will be completely lost! **ctags** is like the magnetic COMPASS needle for the programmers.

Usage of **ctags** :

```
        ctags *.cpp
        gvim −t foo_function
        gvim −t main
```

This will edit the C++ program file which contains the function foo_function() and will automatically place the cursor on the first line of the function foo_function(). The second command takes you to the line with the main() function definition.

Inside the Vim editor, you can jump to a function by typing : (colon) tag < function name >as below −

```
        :tag sample_function
```

This will place the cursor on first line of sample_function()

If you want to jump into the function from a line in file which contains the function name, place the cursor just before the function name and press **CTRL+]** (press control key and left−square−bracket key simultaneously).

```
                // example code
                switch(id_number) {
                        Case 1:
                                if ( foo_function( 22, "abcef") == 3 )
                                    ^
                                    |
                                    |
                                    |
                  Place the cursor here (just before foo_function) and press CTRL+]
                  This takes you to the function named "foo_function".
                  To come back to this line press CTRL+t
```

To go back to the calling line press **CTRL+t** (Control key and letter 't' together). Keep pressing **CTRL+t** to unwind and go to the first line where you started the navigation. That is you can keep pressing **CTRL+]** and then keep pressing **CTRL+t** to go back. You can repeat these as many times as you want to have complete navigation through all the functions of C or C++.

5.1 Ctags for ESQL

Since ctags does not directly support the Embedded SQL/C (ESQL) language, the following shell script can be used to create tags for esql. ESQL/C is database SQL commands embedded inside the "C" programs. Oracle's ESQL/C is called Pro*C and Sybase, Informix have ESQL/C and PostgreSQL has product "ecpg".

Save this file as "sqltags.sh" and do chmod a+rx tags_gen.sh.

```
#!/bin/sh

# Program to create ctags for ESQL, C++ and C files
ESQL_EXTN=pc
tag_file1=tags_file.1
tag_file2=tags_file.2

which_tag=ctags

rm −f $tag_file1 $tag_file2 tags

aa=`ls *.$ESQL_EXTN`
#echo $aa
for ii in $aa
do
        #echo $ii
        jj=`echo $ii | cut −d'.' −f1`
        #echo $jj

        if [ ! −f $jj.cpp ]; then
                echo " "
                echo " "
                echo "***********************************************"
                echo "ESQL *.cpp files does not exist.. "
                echo "You must generate the *.cpp from *.pc file"
                echo "using the Oracle Pro*C pre−compiler or Sybase"
                echo "or Informix esql/c pre−compiler."
                echo "And then re−run this command"
                echo "***********************************************"
                echo " "
                exit
        fi

        rm −f tags
        $which_tag $jj.cpp
        kk=s/$jj\.cpp/$jj\.pc/g

        #echo $kk > sed.tmp
        #sed −f sed.tmp tags >> $tag_file1

        #sed −e's/sample\.cpp/sample\.pc/g' tags >> $tag_file1
        sed −e $kk tags >> $tag_file1
done

# Now handle all the C++/C files − exclude the ESQL *.cpp files
rm −f tags $tag_file2
bb=`ls *.cpp *.c`
aa=`ls *.$ESQL_EXTN`
for mm in $bb
do
        ee=`echo $mm | cut −d'.' −f1`
        file_type="NOT_ESQL"
        # Exclude the ESQL *.cpp and *.c files
        for nn in $aa
        do
                dd=`echo $nn | cut −d'.' −f1`
                if [ "$dd" = "$ee" ]; then
                        file_type="ESQL"
                        break
                fi
        done
```

```
        if [ "$file_type" = "ESQL" ]; then
                continue
        fi

        rm -f tags
        $which_tag $mm
        cat tags >> $tag_file2
done

mv -f $tag_file2 tags
cat  $tag_file1 >> tags
rm -f $tag_file1

# Must sort tags file for it work properly ....
sort tags > $tag_file1
mv $tag_file1 tags
```

5.2 Ctags for JavaScript programs, Korn, Bourne shells

The shell script given below can be used to generate tags for a very large variety of programs written in JavaScript, PHP/FI scripts, Korn shell, C shell, Bourne shell and many others. This is a very generic module.

Save this file as tags_gen.sh and do chmod a+rx tags_gen.sh.

```
#!/bin/sh

tmp_tag=tags_file
tmp_tag2=tags_file2

echo " "
echo " "
echo " "
echo " "
echo " "
echo "Generate tags for ...."
while :
do
        echo "    Enter file extension for which you want to generate tags."
        echo -n "    File-extension should be like sh, js, ksh, etc... : "
        read ans

        if [ "$ans" == "" ]; then
                echo " "
                echo "Wrong entry. Try again!"
        else
                break
        fi
done

\rm -f $tmp_tag

aa=`ls *.$ans`

for ii in $aa
do
        jj=`echo $ii | cut -d'.' -f1`
        #echo $jj
```

```
        cp $ii $jj.c
        ctags $jj.c
        echo "s/$jj.c/$ii/g" > $tmp_tag2
        sed −f $tmp_tag2 tags >> $tmp_tag
        \rm −f tags $jj.c
done

sort $tmp_tag > tags

\rm −f $tmp_tag $tmp_tag2
```

5.3 Debugger gdb

You would be using gdb extensively along with Vi. Debugging is the most important aspect of programming as the major cost of software projects goes into debugging and testing.

To debug C++/C programs use 'gdb' tool. See **'man gdb'**. You must compile your programs with −g3 option like

```
        gcc −g3 foo.c foo_another.c sample.c
```

To set up easy aliases do −

```
Setup an alias in your ~/.bash_profile
      alias gdb='gdb −directory=/home/src −directory=/usr/myname/src '
Give −
      gdb foo.cpp
      gdb> dir /hom2/another_src
      This will add to file search path
      gdb> break 'some_class::func<TAB><TAB>
This will complete the function name saving you typing time... and will output like −
      gdb> break 'some_class::function_foo_some_where(int aa, float bb)'
```

Pressing TAB key twice is the command line completion, which will save you lots of typing time. This is one of the most important technique of using gdb.

To get online help do −

```
      gdb> help
Gives online help
      gdb> help breakpoints
Gives more details about breakpoints.
```

To set breakpoints and do debugging

```
      unixprompt> gdb exe_filename
      gdb> b main
This will put breakpoint in main() function
      gdb> b 123
This will put breakpoint in line 123 of the current file
      gdb> help breakpoints
Gives more details about breakpoints.
```

To analyze the core dumps do

```
          unixprompt> gdb exe_filename  core
          gdb> bt
    Gives backtrace of functions and line numbers where the program failed
          gdb> help backtrace
    Gives more details about backtrace.
```

You can also use GUI version of gdb called xxgdb.

Memory leak tools −

- Freeware Electric Fence on linux cd,
- Commercial tools Purify http://www.rational.com
- Insure++ http://www.insure.com

---

6. Online VIM help

See the online man pages. At unix shell prompt type **'man vim'** and **'man gvim'**.

Or inside the gvim session type :help to get the help page. See also Vim Tutorial

```
                               VIM − main help file

          Move around:  Use the cursor keys, or "h" to go left,
                        "j" to go down, "k" to go up, "l" to go right.
                        ":1" takes you to 1st line of page
                        ":n" takes you to nth line of page
                        "<SHIFT>g" takes you to bottom of page
                        ":/someword/ will search for "someword" in doc

    Close this window:  Use ":q<Enter>".

    Jump to a subject:  Position the cursor on a tag between |bars| and hit CTRL−].

       With the mouse:  ":set mouse=a" to enable the mouse (in xterm or GUI).
                        Double−click the left mouse button on a tag between |bars|.

            jump back:  Type CTRL−T or CTRL−O.

    Get specific help:  It is possible to go directly to whatever you want help
                        on, by giving an argument to the ":help" command |:help|.
                        It is possible to further specify the context:
                              WHAT                  PREPEND    EXAMPLE      ~
                          Normal mode commands     (nothing)   :help x
                          Visual mode commands        v_       :help v_u
                          Insert mode commands        i_       :help i_<Esc>
                          command−line commands       :        :help :quit
                          command−line editing        c_       :help c_<Del>
                          Vim command arguments       −        :help −r
                          options                     '        :help 'textwidth'

    list of documentation files:

    |howto.txt|     how to do the most common things
    |intro.txt|     introduction to Vim
    |index.txt|     alphabetical index for each mode
    |autocmd.txt|   automatically executing commands on an event
    |change.txt|    delete and replace text
```

7. Vim Home page and Vim links

The home page of vim is at http://www.vim.org and mirror site in US is at http://www.us.vim.org

Vim FAQ is at http://www.grafnetix.com/~laurent/vim/faq.html and at http://www.vim.org/faq

Eli's Vim Page at http://www.netusa.net/~eli/src/vim.html

The Vi Lovers Home Page http://www.cs.vu.nl/~tmgil/vi.html

Vim Reference Guide at http://scisun.sci.ccny.cuny.edu/~olrcc/vim/

Vim mailing list at http://www.findmail.com/listsaver/vimannounce.html and http://www.vim.org/mail.html

Mailing list archives are kept at:

- http://www.egroups.com/group/vim
- http://www.egroups.com/group/vimdev
- http://www.egroups.com/group/vimannounce

Vim macros

8. Vim Tutorial

8.1 Vim Hands−on Tutorial

On Linux system see the tutorial at /usr/doc/vim−common−5.*/tutor, on other unix systems go to directory where vim is installed and look for doc directory.

```
bash$ cd /usr/doc/vim-common*/tutor
bash$ less README.txt
bash$ cp tutor $HOME
bash$ cd $HOME
bash$ less tutor
```

8.2 Vi Tutorials on Internet

- Purdue University http://ecn.www.ecn.purdue.edu/ECN/Documents/VI/
- Quick Vi tutorial http://linuxwww.db.erau.edu/LUG/node165.html
- Advanced Vi tutorial http://www.yggdrasil.com/bible/bible−src/user−alpha−4/guide/node171.html
- Tutorials http://www.cfm.brown.edu/Unixhelp/vi_.html

- Tutorials http://www.linuxbox.com/~taylor/4ltrwrd/section3_4.html
- Unix world online vi tutorial http://www.networkcomputing.com/unixworld/unixhome.html
- Univ of Hawaii tutorial http://www.eng.hawaii.edu/Tutor/vi.html
- InfoBound http://www.infobound.com/vi.html
- Cornell Univ http://www.tc.cornell.edu/Edu/Tutor/Basics/vi/
- Vi Lovers home page: http://www.cs.vu.nl/~tmgil/vi.html
- After Sept 2000, will moveto http://www.thomer.com/thomer/vi/vi.html
- Beginner's Guide to vi http://www.cs.umr.edu/unixinfo/general/packages/viguide.html
- vi Help file http://www.vmunix.com/~gabor/vi.html
- vim FAQ http://www.math.fu−berlin.de/~guckes/vim/faq/

There are many Vi Tutorials on internet. In Yahoo (Lycos, excite or Hotbot) enter "Vi Tutorial" in search field and search engine will return many pointers.

---

9. Vi Tutorial

In this tutorial, we describe some "advanced" **vi** concepts and commands, so you can appreciate the power of **vi** and so you decide how to build your knowledge of **vi** commands. Nearly all **vi** references list the available commands, but many don't bother to discuss how the commands interrelate; this topic is the main purpose of this tutorial.

9.1 Cursor Movement Commands

The **vi** cursor movement commands allow you to position the cursor in the file and/or on the screen efficiently, with a minimum number of keystrokes. There are oodles of cursor movement commands − don't try memorizing them all at once! Later, we'll see that much of the power of **vi** comes from mixing cursor movement commands with other commands to delete, change, yank (copy), and filter text.

Please edit a large text file (say, **wknight**) so you can experiment with each command as it is described. Keep in mind these commands will only work in Command Mode, not Insert Mode; if you start getting your "commands" in your text, press the ESC key to return to Command Mode.

- **cursor keys** : As we've seen, cursor keys move by single character amounts left, down, up, and right. Movement above the top of the file, below the bottom, to the right of the end of a line, or left of the beginning is not allowed (no line wrapping).
- **hjkl** : When **vi** was written (around 1978), many terminals on UNIX systems did not have cursor keys! **h, j, k,** and **l** were chosen as commands to move left, down, up, and right, respectively. Try them! Most **vi** diehards prefer these to the cursor keys because
    - ♦ **(a)** they are in the same place on all keyborads, and
    - ♦ **(b)** they fit nicely under the fingers, unlike most cursor keys, which are arranged in a box or "T" or some other nonlinear shape.

Why h, j, k, and l? Well, in the ASCII character set, CTRL−H is backspace (moves left), CTRL−J is linefeed (moves down), and, of course, k and l are next to h and j, so you see, they're mnemonic.

- **0** : ("zero", not "oh") Move to the beginning of current line. (To try this and the next few commands, use the cursor keys or **h j k l** to move to an indented text line that contains few "e" characters. If you can't find an indented line in your file, create one by inserting a few space characters at the beginning of a line.)
- **^** : Move to first non−white character of current line. (For indented line, 0 and ^ are different.)
- **$** : Move to last character of current line.
- **tC** : Move to (but not on) next character c in current line. (Press 0, then press te. This will move to the first e in the curent line.)
- **fC** : Find (move on top of) next character c in current line. (Press fe, and the cursor will find − that is, move on top − the next e in the current line.)
- **TC** : Move to (but not on) the previous character c in current line (Press $, then Te.)
- **FC** : Find (move on top of) the previous character c in current line. (Press Fe.)
- **n|** : Move to column n in current line. (Try 20 |. The digits 2 and 0 will not be displayed as you type them, but when you press | the cursor will move to column 20.) Try some experiments with t f T F | . When you do something illegal, **vi** will beep your terminal.
- **w** : Forward to beginning of next "small" word (a "small" word consists of unbroken alphanumeric characters or punctuation characters, but not mixed alphanumeric and punctuation). Try tapping w a dozen times or so − note what happens at punctuation.
- **W** : Forward to beginning of next "big" word (alphanumeric and punctuation mixed). Try W a dozen times or so.
- **b** : Backward to beginning of "small" word.
- **B** : Backward to beginning of "big" word.
- **e** : Forward to end of "small" word.
- **E** : Forward to end of "big" word.
- **+ Return** : Move to first non−white space character on next line. (+ and the Return key have the same effect.)
- **−** : Move to first non−white space character on previous line.
- **)** : Move to the end of sentence. (A sentence ends either at a blank line or at a period or examination mark followed by two space characters or at the end of a line. A period or exclamation mark followed by one space character does not end a sentence; this is correct behaviour, according to traditional rules of how sentences should appear in typed documents, but often appears wrong to those who have never suffered through a formal typing class.)

- **(** : Move to beginning of sentence.
- **}** : Move to end of paragraph. (Paragraphs are seperated with blank lines, by **vi**'s definition.)
- **{** : Move to beginning of paragraph.
- **H** : Move to home position (top line) on the screen
- **M** : Move to middle line on the screen.
- **L** : Move to last line on the screen.
- **nG** : Move to line n. If n is not given, move to the last line in the file. (Try 15G to move to line 15, for example. The CTRL−G command displays the name of the file, some status information, and the current line number. To move to the top of the file: 1G)
- **CTRL−d** : Scroll down half−screen (see note).
- **CTRL−u** : Scroll up half−screen (see note).
- **CTRL−f** : Move forward one−screen (see note).
- **CTRL−b** : Move backward one−screen (see note).
- **Note** : These four scrolling/paging commands cannot be used with the delete, change, yank, or filter commands.
- **/reg_exp** : Move to next occurrence of the regular expression reg_exp When you press /, the cursor drops to the lower left corner of the screen and waits for you to type in the regular expression. Press the Return key to finish; **vi** then searches forward for the next occurrence of the regular expression. For example, press /the followed by Return. This moves forward to the next occurrence of the, perhaps imbedded in the middle of some longer word (other, weather, etc.). If you just press / and then Return, **vi** searches for the next occurrence of whatever the last regular expression was that you searched for.
- **n** : Has the same effect as pressing / and then Return; i.e., searches for the next occurrence of whatever the last regular expression was that you searched for.
- **?reg_exp** : Searches backward, rather than forward. If no reg_exp is given, it searches for the last regular expression that was entered. Both / and ? wrap around, so searching "below" the bottom or "above" the top of the file is legal.
- **N** : Same as pressing ? and then Return.

9.2 Repeat Counts

Many of the movement commands discussed above can be preceded with a repeat count; the movement is simply repeated the given number of times:

- **3w** : Move forward three words
- **5k** : Move up four characters
- **3fa** : Find the third succeeding a in current line
- **6+** : Move down six lines

For some commands, the "repeat counts" has special meaning:

- **4H** : Move to Line 4 on the screen (home plus 3)
- **8L** : Move to the eigth line from the bottom of the screen
- **3$** : Move to the end of the third line down

For some commands (e.g., ^) the repeat count is ignored; for others (e.g., / and ?) it is illegal

9.3 Deleting Text

We've seen that **dd** deletes the current line. This can be used with a repeat count: 3dd deletes three lines, the current line, and the two following lines.

The d command can be used as a "prefix" on most of the movement commands above to delete nearly arbitrary chunks of text. When used with d, the movement commands are called target specifiers. d can be given a repeat count. (As you try these experiments, remember to press u after each command to undo the deletion).

- **dw** : Delete "small" word forward
- **d3w** : Delete three "small" words forward
- **3dw** : Three times, delete "small" word forward
- **3d3w** : Three times, delete three "small" words forward (that is, delete nine "small" words forward)
- **d+** : Delete current line and next line down
- **d/the** : Delete from current character up to but not including the next occurrence of the pattern the.
- **d$** : Delete to end of line
- **d0** : Delete to beginning of line
- **d30G** : Delete from the curent line to and including Line 30
- **dG** : Delete from current line to and including last line
- **d1G** : Delete from current line to and including Line 1

To delete single characters, use x. x can be given a repeat count:

- **15x** : Delete current and 14 following characters

x is actually just an abbreviation of d1; that is, delete one character right.

9.4 Changing Text

The c command is similar to d, except it toggles **vi** into Insert Mode, allowing the original (unwanted) text to be changed to something else.

For example, put the cursor on the beginning of a word (press w to get to the beginning of the next word). Then, press cw to change that word. On the screen, the last character in the word being changed will be replaced with a **$** symbol indicating the boundary of the change; type in a new word (you will overwrite the original word on the screen) and press the ESC key when done. Your input may be longer or shorter than the word being changed.

Put the cursor at the beginning of a line containing at least three words, and try c3w to change three words. Try c$ to change to the end of the current line. In all cases where the change affects only the current line, the boundary of the change is indicated with $.

When a change affects more than just the current line, **vi** deletes the original text from the screen and toggles into Insert Mode. For example, try c3+ to change the current and the next three lines; **vi** deletes the four original lines from the screen and toggles into Insert Mode in a new blank line. As usual, press the ESC key when you have finished entering your new text.

Some other change commands:

- **cc** : Change current line
- **5cc** : Change five lines (current and next four)
- **c/the** : Change from current character up to but not including the next occurrence of the pattern the
- **c$** : Change to end of line
- **c30G** : Change from the current line to and including Line 30
- **cG** : Change from curernt line to and including last line
- **c1G** : Change from curernt line to and including Line 1

9.5 Yanking (Copying) Text

The y command yanks a copy of text into a buffer; the yanked text can then be put (or pasted) elsewhere in the file using p or P.

The simplest form of yank is yy to yank the current line; after yy, try p to put a copy of the yanked line after the cursor. Following yy, you can make as many copies of the yanked line as you want by moving up and down in the file and pressing p.

To copy multiple lines, try, for example, 5yy (yank the current and next four lines). p puts a copy of the yanked lines after the cursor; the sequence 5yyp "works" but it probably doesn't do what you would like. The P command is like p, but puts a copy of the yanked text ahead of the cursor; try the sequence 5yyP.

Other yank commands:

- **y3w** : Yank three words
- **y$** : Yank to end of current line
- **y1G** : Yank from current line to and including Line 1

9.6 Filtering text

The filter command **!**, prompts for the name of a UNIX command (which should be a filter), then passes selected lines through the filter, replacing those selected line in the **vi** buffer with the output of the filter command. **vi**'s ability to pass nearly arbitrary chunks of text through any UNIX filter adds incredible flexibility to **vi**, at no "additional cost" in size or performance to **vi** itself.

Some examples will help illustrate. Create a line in your file containing just the word who and absolutely no other text. Put the cursor on this line, and press **!!** This command is analogous to dd, cc, or yy, but instead of deleting, changing, or yanking the current line, it filters the current line. When you press the second !, the cursor drops down to the lower left corner of the screen and a single ! is displayed, prompting you to enter the name of a filter. As the filter name, type sh and press the Return key. **sh** (the Bourne shell) is a filter! It reads standard input, does some processing of its input (that is, executes commands), and sends its output (the output of those commands) to standard output. Filtering the line containing who through sh causes the line containing who to be replaced with a list of the current users on the system − right in your file!

Try repeating this process with **date**. That is, create a line containing nothing but the word **date**, then put the cursor on the line, and press **!!sh** and the Return key. The line containing **date** is replaced with the output of the **date** command.

Put your cursor on the first line of the output of who. Count the number of lines. Suppose, for example, the number is six. Then select those six lines to be filtered through sort; press **6!!sort** and the Return key. The six lines will be passed through sort, and sort's output replaces the original six lines.

The filter command can only be used on complete lines, not on characters or words.

Some other filter commands (here, < CR > means press Return):

- **!/the < CR > sort < CR >** : Sort from the current line up to and including the next line containing the
- **!1Ggrep the < CR >** : Replace from the current line to and including Line 1 with just the lines that contain the
- **!Gawk '{print $1}' < CR >** : From the current line to the end of file, replace every line with just its first word.

9.7 Marking Lines and Characters

You can mark lines and characters to be used as targest for movement, deletion, change, yanking, and filtering using the command mc, where c is a lowercase letter.

For example, put the cursor in the middle of some word and press ma. This marks the character under the cursor as mark a.

Now, move the cursor off the marked character and to a different line (use the cursor keys, CTRL−u, or whatever). To return to the marked line, press 'a (that is, single quote, then a). This moves to the first non−white space character on the line containing mark a.

Move off that line again. To return to the marked character, press `a (that is, backquote, then a). This moves on top of the character marked with a.

Marking is usually used with deleting, changing, yanking or filtering. For example, move the cursor to a line other than the one containing mark a, and then press d'a (d, single quote, a). This deletes from the current line to and including the line marked with a.

Put the cursor in the middle of a different word and press mb to set mark b. Now, move the cursor away from that word (but only a few lines, so you can see what we're about to do more easily), and then press d`b (d, backquote, b). This deletes from the current CHARACTER to and including the CHARACTER marked with b.

As another example, to sort the output of who, mark the first line (ma), then move the cursor to the last line and press !'asort and the Return key.

If you jump to a mark and decide you want to jump back to whatever you jumped from, you can press '' (jump back to line) or `` (jump back to character).

9.8 Naming Buffers

When you delete, change, or yank text, the original text is stored (until the next delete, change, or yank) in an unnamed buffer from which it can be put using p or P. Using the unnamed buffer, only the most recently deleted, changed or yanked text may be recovered.

If you wish to delete, change, or yank multiple sections of text and remember them all (up to a maximum of

26), you can give a buffer name ahead of the delete change or yank command. A buffer name has the form "c (double quote, lowercase c).

For example, press "ayy to yank the current line into buffer a, then move to a different line and press "byy to yank that line into buffer b. Now, move elsewhere in the file and press "ap and "bp to put copies of the text stored in buffers a and b.

Some other named buffer commands:

- **"a6yy** : Yank six lines (current and next five) into buffer a
- **"bd1G** : Delete from the curernt line to and including Line 1, storing the deleted lines in buffer b
- **"cy'c** : Yank from the current line to the line marked c into buffer c (marks and buffers are distinct, and may have the same name without confusing **vi**)

9.9 Substitutions

To substitute one chunk of text for another in lines throughout your file, use the :s command. Some substitute examples:

- **:1,$s/the/THE/g** From Line 1 to the last line (line $), substitute for the text THE; do this globally in each line where the occurrs
- **:'a,.s/.*/ha ha/** From the line marked a to the current line (line .), substitute for everything on the line the text ha ha

9.10 Miscellaneous "Colon Commands"

All colon commands begin with a colon; when you press the colon, the cursor drops to the lower left corner of the screen, and a colon prompt is displayed waiting for you to finish your colon command.

Some important examples:

- **:w** Write the buffer contents to the file without quitting from **vi**
- **:w abc** Write the buffer contents to the file abc (creating abc if it doesn't exist, or overwriting current contents if it does exist) without quitting from **vi**
- **:1,10w abc** Write lines 1 through 10 to file abc
- **:'a,$w abc** Write from the line marked a to the last line into file abc
- **:e abc** Edit file abc, instead of the current file. **vi** prints an error message if changes have been made to the curernt file that have not been saved with :w
- **:e! abc** Edit file abc, throwing away any changes that may have been made to the current file
- **:e #** Edit the prior file edited (successive :e# commands toggle back and forth between two files)
- **:f abc** Change the file anme for the current **vi** buffer to abc
- **:q** Quit, unless unsaved chanegs have been made
- **:q!** Quit, throwing away any changes that may have been made
- **:r abc** Read the file abc into current **vi** buffer, after the line the cursor is on (try :r croc to read in a copy of the croc file)
- **:!cmd** Execute command cmd (who, sort, ls, etc.)

9.11 Setting Options

Various options affect the "feel" of **vi**. You can display all the various options that can be set using the colon command :set all. You can also use set to change options.

For example, if you want to see line numbers for the lines in the file you're editing, use the command :set number. To turn off line numbering, use the command :set nonumber. Most options can be abbreviated; :set nu turns on line numbering and :set nonu turns off line numbering.

If you :set nomagic, the special meanings of regular expression characters (period, asterisk, square bracket, etc.) are switched off. Use :set magic to restore the special meanings.

Some options take a value. For example, :set tabstop=4 causes tabs to be displayed as four space characters, rather than the usual eight.

If you find you always want certain options set certain ways, you can put the set commands you want ina file .exrc, or you can set up the environment variable EXINIT to specify the options you want.

For example, if your login shell is Bourne shell, this line could go in your .profile file:

```
        EXINIT='set nomagic nu tabstop=4'; export EXINIT
```

If your login shell is a C shell, this line could go in your .login file:

```
        setenv EXINIT 'set nomagic nu tabstop=4'
```

9.12 Key Mappings

If you find you're performing a series of simple commands over and over, you can map the command series to an unused command key using the :map command. If your mapping must include control characters such as Return key (CTRL−M in ASCII) or the ESC (CTRL−[in ASCII) key, precede such characters with CTRL−v to suppress their usual special meaning.

For example, this command maps CTRL−A to move the cursor forward 55 lines, then back up to the most recent blank line, then change that blank line to a formfeed (CTRL−L) and three blank lines. That is, each CTRL−A will paginate the next page, without splitting paragraphs across pages.

Note: In this command, each control character is shown as ^C, where C is some uppercase letter. For example, CTRL−M is shown as ^M. Also, when you enter this command you will not see the CTRL−v characters as shown: each CTRL−v merely suppresses the usual special meaning of the following control character, so when you press the sequence ^V^M, all you will see on the screen is ^M. In this command, ^M is the Return key and ^[is the ESC key.

```
        :map ^A  55+?^$^V^Mcc^V^L^V^M^V^M^V^M^V^[
```

9.13 Editing Multiple Files

You can edit multiple files with **vi** by giving multiple file names as command line arguments:

```
vi croc fatherw  wknight
```

Three colon commands are used to move through the multiple files:

- **:n** Move to the next file in the argument list (you must save changes with :w or **vi** will print an error message)
- **:N** Move to the previous file in the argument list (you must save changes with :w or **vi** will print an error message)
- **:rew** Rewind and start over with the first file in the argument list

The :n, :N, and :rew commands are somewhat clumsy, but there are some important benefits: the contents of named buffers ("a, "b, "c, etc.) are remembered across files, so you can use :n and :rew with p and P to copy text back and forth between files. Also, the most recent search string for the / and ? commands remembered across files, so you can do repetitive searches in multiple files rather easily.

For example, try the following experiment: First get out of **vi**, then execute **vi** with croc and wknight as arguments:

```
$ vi croc wknight
```

In croc, search for the

/the < CR >

Yank this line into buffer a:

"ayy

Now go to the next file (you've made no change to croc, so this will work):

:n < CR >

Search for the "next" line containing the, without retyping the search string:

n

Put a copy of buffer a after the current line in wknight:

"ap

Move down two lines, and yank the current line into buffer b:

jj"byy

Save the changes to wknight

:w < CR >

Now, rewind to croc

:rew < CR >

Search again, and put a copy of buffer b after the found line:

n"bp

Save the changes, and exit **vi**

ZZ

9.14 Final Remarks

This tutorial was intended to introduce some of the **vi** capabilities that you might overlook in your system's **vi** manual or that might not be mentioned in the manual (different systems have manuals of widely varying quality).

You will not be a **vi** expert after reading this tutorial, but you will have a good appreciation of **vi**'s capabilities. Only time and effort can make a **vi** expert. But the efficiency and universality of **vi** make this effort pay off in the long run.

You may have decided you hate **vi**. So be it! But be aware that **vi** remains the standard UNIX text editor − the one editor you can count on being available on every UNIX system you'll use − so even if you prefer to use something else day−to−day, you'd be well advised to know the bare minimum **vi** material covered in this tutorial.

---

10. Vim Reference Card

10.1 Vi states

Vi has 3 modes:

1. ***command mode*** − Normal and initial state; others return here (use **ESC** to abort a partially typed command)
2. ***input mode*** − entered by specific commands **a i A I o O c C s S R** and ended by **ESC** or abnormally with interrupt
3. ***line mode*** − i.e. waiting for input after a **:** , **/** , **?** or a **!** command (end with **CR**, abort with **CTRL−c**). **CTRL** is the control key: **CTRL−c** means "control c"

10.2 Shell Commands

1. **TERM=** *code* Puts a code name for your terminal into the variable **TERM**
2. **export TERM** Conveys the value of **TERM** (the terminal code) to any UNIX system program that is terminal dependant.
3. **tput init** Initializes the terminal so that it will function properly with various UNIX system programs.
4. **vi** *filename* Accesses the **vi** screen editor so that you can edit a specified file.
5. **vi** *file1 file2 file3* Enters three files into the **vi** buffer to be edited. Those files are *file1, file2,* and *file3*.
6. **view** *file* Invoke vi editor on *file* in read−only mode
7. **vi −R** *file* Invoke vi editor on *file* in read−only mode
8. **vi −r** *file* Recover *file* and recent edits after system crash
9. **vi −r** *file* Recover *file* and recent edits after system crash

10.3 Setting Options

1. **:set** *option* Activate *option*
2. **:set** *option=value* Assign *value* to *option*
3. **:set no** *option* Deactivate *option*
4. **:set** Display options set by user
5. **:set all** Display list of all current options, both default and those set by the user
6. **:set** *option*? Display values of *option*

10.4 Notations used

Notations:

1. **CTRL−c CTRL** is the control key: **CTRL−c** means "control c"
2. **CR** is Carriage return (ENTER key)

10.5 Interrupting, cancelling

- **ESC** end insert or incomplete command
- **CTRL−? CTRL** is the control key: **CTRL−?** means "control ?" delete or rubout interrupts
- **CTRL−l** reprint/refresh screen if CTRL−? scrambles it

10.6 File Manipulation

- **ZZ** Save the file and exit vi
- **:wq** Save the file and exit vi
- **:w** Write the current file
- **:w!** Force write the current file, if file is read−only
- **:w*name*** Write to file *name*
- **:q** Exit from vi
- **:q!** Force exit from vi (discarding changes)
- **:e name** Edit file *name*
- **:e!** reedit, discard changes
- **:e + name** edit file *name*, starting at end
- **:e + n** edit starting at line *n*

- **:e #** edit alternate file
- **:n** edit next file in *arglist*
- **:args** list files in current filelist
- **:rew** rewind current filelist and edit first file
- **:n args** specify new arglist
- **:f** show current file and line
- **CTRL−G** synonym for :f , show current file and line
- **:ta tag** to tag file entry *tag*
- **CTRL−]** :ta, following word is tag

10.7 Movement

- **Arrows** Move the cursor
- **CTRL−d** Scroll half page down
- **CTRL−u** Scroll half page up
- **CTRL−f** Scroll a full page down
- **CTRL−b** Scroll a full page up
- **:0** Move to start of file
- **:n** Move to line number n
- **:$** Move to end of file
- **0** Move to start of line
- **^** Move to first non−blank character
- **$** Move to end of line
- **CR** Move to the start of next line
- **−** Move to the start of previous line
- **%** Find matching bracket
- **G** goto line (last line default)
- **]]** next section/function
- **[[** previous section/function

10.8 Line Positioning

- **H** Home window line
- **L** Last window line
- **M** Middle window line
- **+** Next line, at first non−white
- **−** Previous line, at first non−white
- **CR** return, same as +
- **j** next line, same column
- **k** previous line, same column

10.9 Character positioning

- **0** beginning of line
- **$** end of line
- **h** forward
- **l** backwards
- **SPACE** same as l
- **fx** find x forward

- **Fx** find x backward
- **;** repeat last f F
- **,** inverse of ;
- **|** to specified column
- **%** find matching { or }

10.10 Words, sentences, paragraphs

- **w** Word forward
- **b** Word backward
- **e** End of word
- **)** To next sentence
- (Back sentence
- **}** To next paragraph
- **{** Back paragraph
- **W** Blank delimited word
- **B** Back W
- **E** To end of W

10.11 Marking and returning

- **``** (press twice the back−quote ` key) Previous context
- **''** (press twice the single−quote ` key) Previous context at first non−white in line
- **mx** mark position with letter x
- **`x** (back quote key and letter x) goto mark x
- **'x** goto mark x at first non−white in line

10.12 Corrections during insert

- **CTRL−h** Erase last character
- **CTRL−w** Erase last word
- **erase** Press DELETE key, same as CTRL−h
- **kill** Your kill key, erase input this line
- **** Escapes CTRL−h, DELETE and kill
- **ESC** Ends insertion, back to command
- **CTRL−?** Interrupt, terminates insert
- **CTRL−d** Backtab over *autoindent*
- **CTRL−v** Quote non−printing character

10.13 Adjusting the screen

- **CTRL−l** Clear and redraw
- **CTRL−r** retype, eliminate @lines
- **z−CR** redraw, current line at window top
- **z−** redraw, current line at window bottom
- **z.** redraw, current line at window center
- **/pat/z−** *pat* line bottom
- **tn** Use n line window
- **CTRL−e** Scroll window down 1 line

- **CTRL−y** Scroll window up 1 line

10.14 Delete

- **x** Delete the character under the cursor
- **X** Delete the charater before the cursor
- **D** Delete to the end of line
- **d^** Delete back to start of line
- **dd** Delete the current line
- **ndd** Delete *n* lines starting with the current one
- **dnw** Delete *n* words starting from cursor

10.15 Insert, change

- **i** Enter input mode inserting before the cursor
- **I** Enter input mode inserting before the first non−blank character
- **a** Enter input mode inserting after the cursor
- **A** Enter input mode inserting after the end of the line
- **o** Open a new line below current line and enter input mode
- **O** Open a new line above current line and enter input mode
- **r** Replace the character under the cursor (does NOT enter input mode)
- **R** Enter input mode replacing characters
- **C** shift−c. Change rest of line
- **D** shift−d. Delete rest of line
- **s** Substitute chars
- **S** Substitute lines
- **J** Join lines
- **J** Join lines

10.16 Copy and Paste

The "yank buffer" is filled by *EVERY* delete command, or explicitely by **Y** and **yy**.

- **Y** Copy the current line to the yank buffer
- ***n*yy** Copy *n* lines starting from the current to the yank buffer
- **p** Paste the yank buffer after the cursor (or below the current line)
- **P** Paste the yank buffer before the cursor (or above the current line)
- **"*x*p** Put from buffer x
- **"*x*y** Yank to buffer x
- **"*x*d** Delete into buffer x

10.17 Operators (use double to affect lines)

- **d** delete
- **c** change
- **<** left shift
- **>** right shift
- **!** filter through command
- **=** indent for LISP

- **y** yank text to buffer

10.18 Search and replace

- ***/text*** Search forward for *text*
- **?*text*** Search backward for *text*
- **n** Repeat the last search in the same direction
- **N** Repeat the last search in the reverse direction
- **/** Repeat the last search forward
- **?** Repeat the last search backward
- **[addr] s/from/to/ [g]** Search for the occurence of *from* and replace it with ***to*** in the current line, or in the range ***addr*** (two line numbers seperated by command; 1,$ is the whole file). Replaces one occurrence per line, or all occurrences if ***g*** is specified. For example, :3,20s/someword/anotherword/g Will replace "someword" with "anotherword" starting from line 3 to line 20. 'g' is global means replace all occurrences of "someword".

10.19 General

- **:sh** Forks a shell (to be exited with CTRL−d)
- **:!*command*** Forks a shell to execute *command*
- **:set number** Switch on line numbering
- **:set nonumber** Switch off line numbering

10.20 Line Editor Commands

- **:** Tells **vi** that the next commands you issue will be line editor commands.
- **:sh** Temporarily returns to the shell to perform some shell commands without leaving **vi**.
- **CTRL−d** Escapes the temporary return to the shell and returns to **vi** so you can edit the current window.
- **:***n* Goes to the *n*th line of the buffer.
- **:***x,z***w** *filename* Writes lines from the numbers *x* through the number *z* into a new file called *filename*.
- **:$** Moves the cursor to the beginning of the last line in the buffer.
- **:.,$d** Deletes all the lines from the current line to the last line
- **:r** *filename* Inserts the contents of the file *filename* under the current line of the buffer.
- **:s***/text/new_text/* Replaces the first instance of *text* on the current line with *new_text*
- **:s***/text/new_text/g* Replaces the every occurrence of *text* on the current line with *new_text*
- **:g***/text/s//new_text/g* Changes every occurrence of *text* on the buffer to *new_text*.

10.21 Other commands

- **u** Undo the last change
- **U** Restore the current line
- **~** Change case
- **J** Join the currentline with the next line
- **.** Repeat last text changing command
- **CTRL−g** Show file name and line number

---

11. Related URLs

Related VIM URLs are at −

- C and C++ Beautifer http://www.metalab.unc.edu/LDP/HOWTO/C−C++Beautifier−HOWTO.html
- Linux goodies http://www.aldev.8m.com or at http://www.aldev.webjump.com

---

12. Other Formats of this Document

This document is published in 11 different formats namely − DVI, Postscript, Latex, Adobe Acrobat PDF, LyX, GNU−info, HTML, RTF(Rich Text Format), Plain−text, Unix man pages and SGML.

- You can get this HOWTO document as a single file tar ball in HTML, DVI, Postscript or SGML formats from − ftp://sunsite.unc.edu/pub/Linux/docs/HOWTO/other−formats/
- Plain text format is in: ftp://sunsite.unc.edu/pub/Linux/docs/HOWTO
- Translations to other languages like French, German, Spanish, Chinese, Japanese are in ftp://sunsite.unc.edu/pub/Linux/docs/HOWTO Any help from you to translate to other languages is welcome.

The document is written using a tool called "SGML−Tools" which can be got from − http://www.sgmltools.org Compiling the source you will get the following commands like

- sgml2html CVS−HOWTO.sgml (to generate html file)
- sgml2rtf CVS−HOWTO.sgml (to generate RTF file)
- sgml2latex CVS−HOWTO.sgml (to generate latex file)

LaTeX documents may be converted into PDF files simply by producing a Postscript output using **sgml2latex** (and dvips) and running the output through the Acrobat **distill** (http://www.adobe.com) command as follows:

```
bash$ man sgml2latex
bash$ sgml2latex filename.sgml
bash$ man dvips
bash$ dvips -o filename.ps filename.dvi
bash$ distill filename.ps
bash$ man ghostscript
bash$ man ps2pdf
bash$ ps2pdf input.ps output.pdf
bash$ acroread output.pdf &
```

Or you can use Ghostscript command **ps2pdf**. ps2pdf is a work−alike for nearly all the functionality of Adobe's Acrobat Distiller product: it converts PostScript files to Portable Document Format (PDF) files. **ps2pdf** is implemented as a very small command script (batch file) that invokes Ghostscript, selecting a special "output device" called **pdfwrite**. In order to use ps2pdf, the pdfwrite device must be included in the makefile when Ghostscript was compiled; see the documentation on building Ghostscript for details.

This howto document is located at −

- http://sunsite.unc.edu/LDP/HOWTO/CVS−HOWTO.html

Also you can find this document at the following mirrors sites −

- http://www.caldera.com/LDP/HOWTO/CVS−HOWTO.html
- http://www.WGS.com/LDP/HOWTO/CVS−HOWTO.html
- http://www.cc.gatech.edu/linux/LDP/HOWTO/CVS−HOWTO.html
- http://www.redhat.com/linux−info/ldp/HOWTO/CVS−HOWTO.html
- Other mirror sites near you (network−address−wise) can be found at http://sunsite.unc.edu/LDP/hmirrors.html select a site and go to directory /LDP/HOWTO/CVS−HOWTO.html

In order to view the document in dvi format, use the xdvi program. The xdvi program is located in tetex−xdvi*.rpm package in Redhat Linux which can be located through ControlPanel | Applications | Publishing | TeX menu buttons. To read dvi document give the command −

```
xdvi −geometry 80x90 howto.dvi
man xdvi
```

And resize the window with mouse. To navigate use Arrow keys, Page Up, Page Down keys, also you can use 'f', 'd', 'u', 'c', 'l', 'r', 'p', 'n' letter keys to move up, down, center, next page, previous page etc. To turn off expert menu press 'x'.

You can read postscript file using the program 'gv' (ghostview) or 'ghostscript'. The ghostscript program is in ghostscript*.rpm package and gv program is in gv*.rpm package in Redhat Linux which can be located through ControlPanel | Applications | Graphics menu buttons. The gv program is much more user friendly than ghostscript. Also ghostscript and gv are available on other platforms like OS/2, Windows 95 and NT, you view this document even on those platforms.

- Get ghostscript for Windows 95, OS/2, and for all OSes from http://www.cs.wisc.edu/~ghost

To read postscript document give the command −

```
gv howto.ps
ghostscript howto.ps
```

You can read HTML format document using Netscape Navigator, Microsoft Internet explorer, Redhat Baron Web browser or any of the 10 other web browsers.

You can read the latex, LyX output using LyX a X−Windows front end to latex.

---

13. Copyright Notice
