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

Al Dev (Alavoor Vasudevan) < alavoor[AT]yahoo.com >

v19.5, 2 Nov 2003

Abstract

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 systems 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!)

Table of Contents

Introduction	3
Before you Install	3
Install Vim on Redhat Linux	3
Install Vim on Debian GNU/Linux	4
Install Vim on Unices	4
Install Vim on Microsoft Windows 95/NT	4
Install Vim on VMS	4
Install Vim on OS/2	7
Install Vim on Apple Macintosh	7
Install Vim on Microsoft Windows 95/NT	8
Install bash shell	9
Edit bash_profile	9
Setup Window colors	10
MS Windows Notepad and Wordpad Imitator in Vim	10
MS Windows EditPlus Features in Vim	11
Setup gvim init files	11
Sample gvimrc file	12
Xdefaults parameters	13
Color Syntax init files	14
Auto source-in method	14
Manual method	15
VIM Usage	15
Remote Vi - MS Windows QVWM Manager	16
Vi companions	17
Directory Tree 'tags'	18
Ctags for ESQl	19
Ctags for JavaScript programs, Korn, Bourne shells	20
Debugger gdb	21
Online VIM help	22
Vim Home page and Vim links	22
Vi Resources and Tips	23
Vim Tutorial	23

Vim Hands-on Tutorial	23
Vi Tutorials on Internet	24
Vi Tutorial	24
Cursor Movement Commands	25
Repeat Counts	27
Deleting Text	27
Changing Text	28
Yanking (Copying) Text	28
Filtering text	29
Marking Lines and Characters	29
Naming Buffers	30
Substitutions	30
Miscellaneous "Colon Commands"	30
Setting Options	31
Key Mappings	32
Editing Multiple Files	32
Final Remarks	33
Vim Reference Card	33
Vi states	33
Shell Commands	34
Setting Options	34
Notations used	34
Interrupting, cancelling	34
File Manipulation	35
Movement	35
Line Positioning	36
Character positioning	36
Words, sentences, paragraphs	37
Marking and returning	37
Corrections during insert	37
Adjusting the screen	38
Delete	38
Insert, change	38
Copy and Paste	39
Operators (use double to affect lines)	39
Search and replace	39
General	40
Line Editor Commands	40
Other commands	40
Vim as XML Editor	41
Matchit.vim & Xmledit.vim	41
Vim and Docbook - Useful key mappings	41
XML Validation	42
Build Your "WYSIWYG" HTML Editor With Vi & Netscape	42
Sample .vimhtmlrc File	43
WYSIWYG	45
Other 'WYSIWYG' uses	46
Source code for atchange	46
HTML Beautifier Inside Vim : Program Tidy	47
Emacs - Old Habits Die Hard !	48
Related URLs	48
Other Formats of this Document	48
Acrobat PDF format	49
Convert Linuxdoc to Docbook format	50

Convert to MS WinHelp format	50
Reading various formats	50
Copyright Notice	51

Introduction

(The latest version of this document is at "<http://milkyway.has.it>" [<http://milkyway.has.it>] and mirror at "<http://www.milkywaygalaxy.freesevers.com>" [<http://www.milkywaygalaxy.freesevers.com>]. You may want to check there for changes).

Vim stands for 'Vi Improved'. Vi is the most popular and powerful editors in the Unix world. Vi is an abbreviation for " Visual" editor. One of the first editors was a line editor called 'ed' (and 'ex'). The Visual 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.

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>" [<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>" [<http://www.vim.org>]
- Mirror site in US is at "<http://www.us.vim.org>" [<http://www.us.vim.org>]
- Ftp site "<ftp://ftp.vim.org/pub/vim>" [<ftp://ftp.vim.org/pub/vim>]
- Or use one of the mirrors "<ftp://ftp.vim.org/pub/vim/MIRRORS>" [<ftp://ftp.vim.org/pub/vim/MIRRORS>]

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.

Install Vim on Debian GNU/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.

On Debian 3.0 (woody) and above, use 'apt-get install vim' instead. The vim-rt package is part of the main vim package.

Install Vim on Unixes

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

```
zcat vim.tar.gz | tar -xvf -
cd vim-5.5/src
./configure --prefix=$HOME/local
make
make install
```

You can exclude prefix option if you want to install in default location in /usr/local. If the graphics version 'gvim' gives trouble then try with

```
./configure --prefix=$HOME/local --enable-gui=motif
```

Install Vim on Microsoft Windows 95/NT

See the section called " Install Vim on Microsoft Windows 95/NT " .

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 the section called " Before you Install ")

You can download precompiled executables from: "<http://www.polarfox.com/vim>" [<http://www.polarfox.com/vim>]

VMS vim authors are -

- zoltan.arpadffy@essnet.se [[mailto: zoltan.arpadffy@essnet.se](mailto:zoltan.arpadffy@essnet.se)]
- arpadffy@altavista.net [[mailto: arpadffy@altavista.net](mailto:arpadffy@altavista.net)]
- cec@gryphon.gsfc.nasa.gov [[mailto: cec@gryphon.gsfc.nasa.gov](mailto:cec@gryphon.gsfc.nasa.gov)]
- BNHunsaker@chq.byu.edu [[mailto: BNHunsaker@chq.byu.edu](mailto:BNHunsaker@chq.byu.edu)]
- sandor.kopanyi@altavista.net [[mailto: sandor.kopanyi@altavista.net](mailto: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.

Install Vim on OS/2

Read the release notes for Vim on OS/2, see the section called " 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
```

Install Vim on Apple Macintosh

Read the release notes for Vim on OS/2, see the section called " 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 [mailto: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 [mailto: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:
```

Install Vim on Microsoft Windows 95/NT

For Windows 95/NT, download the Vim zip file. Double click on the vim.exe and *do "FULL" install and not "Typical"* to get graphics and colors.

If you decide download in parts, then you must download *TWO* zip files -

- Runtime support file *vim*rt.zip*
- Vim command file *vim*60.zip* . Where Vim version is 5.6.

Get one big executable or two zip files from: Goto "<http://www.vim.org>" [<http://www.vim.org>] and click on Download-> download FAQ-> Windows 95/NT or click on these "<ftp://vim.ftp.fu-berlin.de/pc/gvim60.zip>" [<ftp://vim.ftp.fu-berlin.de/pc/gvim60.zip>] and "<ftp://vim.ftp.fu-berlin.de/pc/vim60rt.zip>" [<ftp://vim.ftp.fu-berlin.de/pc/vim60rt.zip>] . (see also the section called " Before you Install ")

Unpack the zip files using the Winzip "<http://www.winzip.com>" [<http://www.winzip.com>] . Both the zip files (vim*rt.zip and vim*60.zip) must be unpacked in the same directory like say *c:\vim* .

For Windows 95/98, set the environment variable VIM (all caps no lowercase) in autoexec.bat by adding this line -

```
set VIM=c:\vim\vim60
```

For Windows NT, add the environment variable VIM (all caps no lowercase) to the *Start | Control Panel | System | Environment | System Properties* dialog. For Windows 2000, click on *Start | Control Panel | System | Advanced | Environment Variable* dialog and add variable VIM (all caps no lowercase):

```
VIM=c:\vim\vim60
```

The VIM variable should point to wherever you installed the vim60 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. Bring up a MS-DOS window by click on Start->Programs->MSDOS (for Windows 95/98) and Start->Run->cmd (for Windows NT/2000). At an MS-DOS prompt type -

```
c:\> set vim
c:\> cd vim\vim60
c:\> install.exe
```

For 'set vim' command, you should see - VIM=c:\vim\vim60 and start the install program which will setup the environment.

Create a short-cut on to your desktop by click-and-drag from c:\vim\vim60\gvim.exe.

Color Syntax Highlighting: To enable color syntax highlighting and other nice features you must copy the gvimrc_example file to the \$VIM_gvimrc. In my case it is c:\vim\vim60_gvimrc.

```
c:\> copy c:\vim\vim60\gvimrc_example $VIM\_gvimrc
```

Useful Tips :

- Just double click on gvim icon on desktop and click MyComputer->C:drive->Select a file and drag and drop it into the gvim window. The file is automatically opened by gvim window!!
- To automatically maximize the Vim window in MS Windows, you can use "<http://www.southbaypc.com/AutoSizer>" [<http://www.southbaypc.com/AutoSizer>] or you can right click on Gvim shortcut and select properties and pick maximize the window on startup.

Install bash shell

In order make MS Windows 95/98/NT/2000/XP even more user-friendly, install the bash shell (Bourne Again Shell). Install "<http://sources.redhat.com/cygwin/setup.exe>" [<http://sources.redhat.com/cygwin/setup.exe>] (Cygwin-setup program) and select bash and other common utilities. The CygWin main site is at "<http://sources.redhat.com/cygwin>" [<http://sources.redhat.com/cygwin>]. With CygWin the Windows 2000 computer will look like Linux/Unix box!! And combined with gvim editor, the Windows 2000 gives programmers more power. The cygwin home is at "<http://cygwin.com>" [<http://cygwin.com>].

You may also want to install MKS in case you are planning to use Java and Java compilers. Get MKS from "<http://www.mks.com>" [<http://www.mks.com>].

Edit bash_profile

After installing the Cygwin, insert some useful aliases in ~/.bash_profile file. Open a cygwin window and at bash prompt -

```
bash$ cd $HOME
bash$ gvim .bash_profile
set -o vi
alias ls='ls --color '
alias cp='cp -i '
alias mv='mv -i '
alias rm='rm -i '
alias vi='gvim '
alias vip='gvim ~/.bash_profile & '
alias sop='. ~/.bash_profile '
alias mys='mysql -uroot -p '
```

```
PATH=$PATH: "/cygdrive/c/Program Files/mysql/bin"
```

With color ls, when you do ls you will see all the directory names and files in different colors (it looks great!). With set -o vi, you can use the command line history editing just as in linux.

Setup Window colors

The default background color of MS DOS prompt window is black and white text. You must change the color, fontsize and window size to make it more pleasing. On MS Windows 2000, click on button Start->Run and type "cmd" and hit return. On MS Windows 95/98/NT click on Start->Programs->MSDOS Prompt which will bring up MSDOS window. Right click on the top left corner of the MSDOS prompt window and select properties. Select color background and enter R=255, G=255, B=190 (red, green, blue) for lightyellow background and text foreground color to black (R=0, G=0, B=0). This sets background to light yellow and text foreground to black and this combination is most pleasing to human eyes. If you have problems with colors in cygwin bash window when doing 'man ls', set the text color to "marune".

For Windows95 see Color for MS-DOS prompt window [<http://www.elementkjournals.com/w95/9711/w9597b3.htm>].

MS Windows Notepad and Wordpad Imitator in Vim

Is there a MS Windows Notepad or Wordpad for Linux or Unix? Yes there is!! For those non-technical users of MS Windows who extensively use the Notepad and Wordpad, there is a command 'evim' which imitates the Notepad and Wordpad. The 'evim' is exactly like Notepad and Wordpad and has all their functionalities. You need to install the package vim-X11 package to enable evim.

```
bash$ evim <filename>
```

On Unixes like HPUX, Solaris, AIX do these:

```
ln -s /usr/local/bin/vim /usr/local/bin/gvim
ln -s /usr/local/bin/vim /usr/local/bin/evim
```

Create a command file 'notepad' in /usr/local/bin/notepad with these 2 lines below

```
#!/bin/sh
/usr/local/bin/evim $@
```

If evim is linked to vim it automatically launches evim. Or you can set up alias for evim and gvim as below:

```
alias evim='vim -g -U /usr/local/vim/share/vim/vim61/evim.vim '
alias gvim='vim -g '
```

Wordpad for Unix/Linux? Look at TED, AbiWord and OpenOffice. Visit TED - A Wordpad for Linux [<http://www.nlgg.nl/Ted>] or AbiWord - Light weight, small software [<http://www.abiword.org>] or OpenOffice - Heavy weight, big software [<http://www.openoffice.org>].

See also Wine project which has notepad and wordpad WINE project [<http://www.winehq.org>].

MS Windows EditPlus Features in Vim

Vim has many of the features of EditPlus editor and in near future will try incorporate many good features of EditPlus editor :

- Left hand side File Navigator - In Vim (gvim) click on menu Window-> 'Split File Explorer' and adjust the width of file explorer window.
- Connect direct to remote host via ftp and handle automatically the edits and uploads (looks as if the remote files and folders are in local host).
- Many other features

The EditPlus is at [EditPlus \[http://www.editplus.com\]](http://www.editplus.com) . See also [TextPad \[http://www.textpad.com\]](http://www.textpad.com) and [UltraEdit \[http://www.ultraedit.com\]](http://www.ultraedit.com) .

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
set autoindent
"
" You may want to turn off the beep sounds (if you want quite) with visual bell
" set vb
" Source in your custom filetypes as given below -
" so $HOME/vim/myfiletypes.vim
```

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 the section called " 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
```

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 nogupty
" 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
"
" You may want to turn off the beep sounds (if you want quite) with visual bell
" set vb
" Source in your custom filetypes as given below -
" so $HOME/vim/myfiletypes.vim
```

```
" 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 <word> " . 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`.

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
```

Color Syntax init files

Auto source-in method

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

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

Click on the 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 you want to use is not detected, then there are two ways to add it.

Method 1: You can modify the `$VIMRUNTIME/filetype.vim` file, but this is not recommended as it will be overwritten when you install a new version of Vim.

Method 2: Create a file in `$HOME/vim/myfiletypes.vim` and put these lines in it -

```
"
" *****
" Filename : $HOME/vim/myfiletypes.vim
" See the document by typing :help autocmd within vim session
" see also the doc at /usr/share/vim/doc/autocmd.txt
" This file will setup the autocommands for new filetypes
" using the existing syntax-filetypes.
" For example when you open foo.prc it will use syntax of plsql
" Basically does :set filetype=prc inside vim
" Add a line in $HOME/.gvimrc as below:
" so $HOME/vim/myfiletypes.vim
"
" *****
augroup filetype
    au!
```

```
    au! BufRead,BufNewFile *.phc      set filetype=php
    au! BufRead,BufNewFile *.mine     set filetype=mime
    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 source in the file "myfiletypes.vim". (*CAUTION*: You MUST put this in both vimrc and gvimrc files in order for this to work) Example:

```
so $HOME/vim/myfiletypes.vim
```

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

See the documentation on autocommand at -

- `:help autocmd` (within a vim editing session)
- See also the doc at `/usr/share/vim/doc/autocmd.txt`

Your file will then be sourced in 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 "myfiletypes.vim" are used before checking the contents of the file.

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
```

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. The `vim` also gives colors but cannot set the background colors and `gvim` can set the background and foreground colors.

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, Syntax 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.

Remote Vi - MS Windows QVWM Manager

QVWM Window Manager is the best as it is very similar in appearance to Microsoft Windows. If you want to use Vi and Vim remotely from a MS Windows PC client, then you should use VNC + QVWM manager. Servers are generally located in remote Data Centers and to edit the files, you should first login to remote servers from MS Windows or Linux desktop PCs. After starting VNC server and QVWM manager on remote server, you should fire up `vncviewer` on your client desktop and edit remote files with `gvim`.

To use graphical editor like `gvim` for remote operations, use the following techniques below:

You can use the VNC to display remote machines on your local display.

- The VNC is at "<http://www.uk.research.att.com/vnc>" [<http://www.uk.research.att.com/vnc>] and commercial VNC is at "<http://www.realvnc.com>" [<http://www.realvnc.com>]
- Get VNC rpms from `rpmfind` [<http://rpmfind.net/linux/rpm2html/search.php?query=vnc>] or from commercial VNC at "<http://www.realvnc.com>" [<http://www.realvnc.com>]
- The best Window manager for VNC is QVWM which is like MS Windows 98/NT/2000 interface, get it from "<http://www.qvwm.org>" [<http://www.qvwm.org>].
- After starting `vncserver`, you can start the `vncviewer` program on clients like MS Windows, Mac or Linux.
- See also the List of X11 Windows Managers [http://www.dlhoffman.com/publiclibrary/RPM/X11_Window_Managers.html].

Compiling qvwm on Solaris : On Solaris you should install the following packages which you can get from "<http://sun.freeware.com>" [<http://sun.freeware.com>] - `xpm`, `imlib`, `jpeg`, `libungif`, `giflib` (`giftran`), `libpng`, `tiff`. And you can download the binary package for solaris from "<http://www.qvwm.org>" [<http://www.qvwm.org>].

Or you can download the `qvwm` source for solaris from "<http://www.qvwm.org>" [<http://www.qvwm.org>] and compile it using `gcc`. If mainsite is busy use the mirror sites listed there. Click on the link "The latest version of `qvwm` is [<ftp/http>]" to download the source code.

Troubleshooting the compile:

- You must install all the packages listed - xpm, imlib, jpeg, libungif, giflib, libpng, tiff. Otherwise src will not compile
- Must edit the src/util.cc file and change snprintf to printf to compile the program to get rid of the compile errors.
- You should put unsigned long before arg in usleep() usleep((unsigned long) 10000)
- Still problems then see "<http://www.milkywaygalaxy.freesevers.com/vnc>" [<http://www.milkywaygalaxy.freesevers.com/vnc>] for compile instructions.

For transferring files from Unix to MS Windows use the ftp clients like

- Commercial :
 1. The ftp voyager "<http://cws.internet.com/ftp-ftpvoyag.html>" [<http://cws.internet.com/ftp-ftpvoyag.html>]
 2. Cute ftp "<http://cws.internet.com/ftp-cuteftp.html>" [<http://cws.internet.com/ftp-cuteftp.html>]
- Free ratings:
 1. WSFTP : "http://www.csra.net/junodj/ws_ftp32.htm" [http://www.csra.net/junodj/ws_ftp32.htm]
 2. LeechFTP "<http://cws.internet.com/ftp-leechftp.html>" [<http://cws.internet.com/ftp-leechftp.html>]
"<http://stud.fh-heilbronn.de/~jdebis/leechftp/>" [<http://stud.fh-heilbronn.de/~jdebis/leechftp/>]
 3. FTP Control "<http://cws.internet.com/ftp-ftpcontrol.html>" [<http://cws.internet.com/ftp-ftpcontrol.html>]
 4. GetRight "<http://cws.internet.com/ftp-getright.html>" [<http://cws.internet.com/ftp-getright.html>]

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++.

Directory Tree 'tags'

To recursively process the tags file for the entire directory :

```
$ cd $HOME
$ ctags -R
```

This will recurse the directory underneath and create a tag file all the files under the directory and beneath. But to use this tag file you must set the following in the vim session or modify `.gvimrc` file

```
$ vi ~/.gvimrc
" Set the tag file search order
set tags=./tags,tags,~/tags,/home/john/ccplus/tags
```

Or in the Vim session you set the tags with colon command:

```
$ vi somefile.cpp
```

```
:set tags=./tags,tags,~/tags,/home/john/ccplus/tags
```

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
```

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/Perl 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 "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
```

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`.

See also gdb interface to Vim at "<http://www.linux.com/gdbvim.tgz>" [<http://www.linux.com/gdbvim.tgz>].

Memory leak tools -

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

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 the section called " Vim Tutorial " To see the settings type `:set all` or `:set`. To see list of options type `:options`. To see topics on set type `:help set`.

```
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
```

Vim Home page and Vim links

The home page of vim is at "<http://www.vim.org>" [<http://www.vim.org>] and mirror site in US is at "<http://www.us.vim.org>" [<http://www.us.vim.org>]

Vim FAQ is at "<http://www.grafnetix.com/~laurent/vim/faq.html>" [<http://www.grafnetix.com/~laurent/vim/faq.html>] and at "<http://www.vim.org/faq>" [<http://www.vim.org/faq>]

Eli's Vim Page at "<http://www.netusa.net/~eli/src/vim.html>" [<http://www.netusa.net/~eli/src/vim.html>]

Vi Lovers home page "<http://www.thomer.com/thomer/vi/vi.html>" [<http://www.thomer.com/thomer/vi/vi.html>]

Vim Reference Guide at "<http://scisun.sci.cny.cuny.edu/~olrcc/vim/>" [<http://scisun.sci.cny.cuny.edu/~olrcc/vim/>]

Vim mailing list at "<http://www.findmail.com/listsaver/vimannounce.html>" [<http://www.findmail.com/listsaver/vimannounce.html>] and "<http://www.vim.org/mail.html>" [<http://www.vim.org/mail.html>]

Mailing list archives are kept at:

- "<http://www.egroups.com/group/vim>" [<http://www.egroups.com/group/vim>]
- "<http://www.egroups.com/group/vimdev>" [<http://www.egroups.com/group/vimdev>]
- "<http://www.egroups.com/group/vimannounce>" [<http://www.egroups.com/group/vimannounce>]

Vim macros "<http://www.grafnetix.com/~laurent/vim/macros.html>" [<http://www.grafnetix.com/~laurent/vim/macros.html>]

Vi Resources and Tips

The following Vi resources are available on internet:

- O'Reilly "Learning the Vi Editor" at "<http://www.eyetap.org/ece385/oreilly/unix/vi/index.htm>" [<http://www.eyetap.org/ece385/oreilly/unix/vi/index.htm>]
- Vi Google directory at Google-Vi [<http://directory.google.com/Top/Computers/Software/Editors/Vi/?tc=1>]
- Resources, Tips, News about Vim "<http://vim.sourceforge.net>" [<http://vim.sourceforge.net>]
- Vi Cheatsheet "<http://www.geekcheat.com/Merchant2/merchant.mv>" [<http://www.geekcheat.com/Merchant2/merchant.mv>]
- Vim and vi article "<http://www.troubleshooters.com/lpm/200212/200212.htm>" [<http://www.troubleshooters.com/lpm/200212/200212.htm>]
- Vim Outliner - An outline processor is a software program enabling the user to quickly construct outlines, and better yet, to correct and rearrange the outline. "<http://www.troubleshooters.com/vimoutliner>" [<http://www.troubleshooters.com/vimoutliner>]

Vim Tutorial

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
```

Vi Tutorials on Internet

- Purdue University "<http://ecn.www.ecn.purdue.edu/ECN/Documents/VI/>" [<http://ecn.www.ecn.purdue.edu/ECN/Documents/VI/>]
- Advanced Vi tutorial "<http://www.yggdrasil.com/bible/bible-src/user-alpha-4/guide/node171.html>" [<http://www.yggdrasil.com/bible/bible-src/user-alpha-4/guide/node171.html>]
- Tutorials "http://www.cfm.brown.edu/Unixhelp/vi_.html" [http://www.cfm.brown.edu/Unixhelp/vi_.html]
- Univ of Hawaii tutorial "<http://www.eng.hawaii.edu/Tutor/vi.html>" [<http://www.eng.hawaii.edu/Tutor/vi.html>]
- InfoBound "<http://www.infobound.com/vi.html>" [<http://www.infobound.com/vi.html>]
- Vi Lovers home page "<http://www.thomer.com/thomer/vi/vi.html>" [<http://www.thomer.com/thomer/vi/vi.html>]
- vi Help file "<http://www.vmunix.com/~gabor/vi.html>" [<http://www.vmunix.com/~gabor/vi.html>]
- These are dead links:

Quick Vi tutorial "<http://linuxwww.db.erau.edu/LUG/node165.html>" [<http://linuxwww.db.erau.edu/LUG/node165.html>]
- Tutorials "http://www.linuxbox.com/~taylor/4ltrwr/section3_4.html" [http://www.linuxbox.com/~taylor/4ltrwr/section3_4.html]
- Unix world online vi tutorial "<http://www.networkcomputing.com/unixworld/unixhome.html>" [<http://www.networkcomputing.com/unixworld/unixhome.html>]
- Cornell Univ "<http://www.tc.cornell.edu/Edu/Tutor/Basics/vi/>" [<http://www.tc.cornell.edu/Edu/Tutor/Basics/vi/>]
- Beginner's Guide to vi "<http://www.cs.umar.edu/unixinfo/general/packages/viguide.html>" [<http://www.cs.umar.edu/unixinfo/general/packages/viguide.html>]
- vim FAQ "<http://www.math.fu-berlin.de/~guckes/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.

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.

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 keyboards, 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 *hjkl* 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 current 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 separated 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.

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 eighth 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

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 current 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.

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 current line to and including last line
- `c1G` : Change from current line to and including Line 1

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

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.

Marking Lines and Characters

You can mark lines and characters to be used as target 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).

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
- `"bdIG` : Delete from the current 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*)

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 occurs
- `:'a,.s/*/ha ha/` From the line marked a to the current line (line .), substitute for everything on the line the text ha ha

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 current 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 name for the current `vi` buffer to `abc`
- `:q` Quit, unless unsaved changes 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.)

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 in a 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'
```


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 retying 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
```

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.

Vim Reference Card

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"

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

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*

Notations used

Notations:

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

Interrupting, cancelling

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

File Manipulation

1. *ZZ* Save the file and exit vi
2. *:wq* Save the file and exit vi
3. *:w* Write the current file
4. *:w!* Force write the current file, if file is read-only
5. *:w name* Write to file *name*
6. *:q* Exit from vi
7. *:q!* Force exit from vi (discarding changes)
8. *:e name* Edit file *name*
9. *:e!* reedit, discard changes
10. *:e + name* edit file *name* , starting at end
11. *:e + n* edit starting at line *n*
12. *:e #* edit alternate file
13. *:n* edit next file in *arglist*
14. *:args* list files in current filelist
15. *:rew* rewind current filelist and edit first file
16. *:n args* specify new arglist
17. *:f* show current file and line
18. *CTRL-G* synonym for *:f* , show current file and line
19. *:ta tag* to tag file entry *tag*
20. *CTRL-J* *:ta*, following word is tag

Movement

1. *Arrows* Move the cursor
2. *CTRL-d* Scroll half page down
3. *CTRL-u* Scroll half page up
4. *CTRL-f* Scroll a full page down
5. *CTRL-b* Scroll a full page up
6. *:0* Move to start of file
7. *:n* Move to line number *n*

8. *:\$* Move to end of file
9. *0* Move to start of line
10. *^* Move to first non-blank character
11. *.\$* Move to end of line
12. *CR* Move to the start of next line
13. *-* Move to the start of previous line
14. *%* Find matching bracket
15. *G* goto line (last line default)
16. *]]* next section/function
17. *[[* previous section/function

Line Positioning

1. *H* Home window line
2. *L* Last window line
3. *M* Middle window line
4. *+* Next line, at first non-white
5. *-* Previous line, at first non-white
6. *CR* return, same as *+*
7. *j* next line, same column
8. *k* previous line, same column

Character positioning

1. *0* beginning of line
2. *\$* end of line
3. *h* forward
4. *l* backwards
5. *SPACE* same as *l*
6. *fx* find x forward
7. *Fx* find x backward
8. *;* repeat last *f* *F*
9. *,* inverse of *;*

10./ to specified column

11.% find matching { or }

Words, sentences, paragraphs

1. *w* Word forward
2. *b* Word backward
3. *e* End of word
4. *)* To next sentence
5. *(* Back sentence
6. *}* To next paragraph
7. *{* Back paragraph
8. *W* Blank delimited word
9. *B* Back W
10. *E* To end of W

Marking and returning

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

Corrections during insert

1. *CTRL-h* Erase last character
2. *CTRL-w* Erase last word
3. *erase* Press DELETE key, same as CTRL-h
4. *kill* Your kill key, erase input this line
5. ** Escapes CTRL-h, DELETE and kill
6. *ESC* Ends insertion, back to command
7. *CTRL-?* Interrupt, terminates insert
8. *CTRL-d* Backtab over *autoindent*

9. *CTRL-v* Quote non-printing character

Adjusting the screen

1. *CTRL-l* Clear and redraw
2. *CTRL-r* retype, eliminate @lines
3. *z-CR* redraw, current line at window top
4. *z-* redraw, current line at window bottom
5. *z.* redraw, current line at window center
6. */pat/z-* *pat* line bottom
7. *tn* Use *n* line window
8. *CTRL-e* Scroll window down 1 line
9. *CTRL-y* Scroll window up 1 line

Delete

1. *x* Delete the character under the cursor
2. *X* Delete the character before the cursor
3. *D* Delete to the end of line
4. *d^* Delete back to start of line
5. *dd* Delete the current line
6. *ndd* Delete *n* lines starting with the current one
7. *dnw* Delete *n* words starting from cursor

Insert, change

1. *i* Enter input mode inserting before the cursor
2. *I* Enter input mode inserting before the first non-blank character
3. *a* Enter input mode inserting after the cursor
4. *A* Enter input mode inserting after the end of the line
5. *o* Open a new line below current line and enter input mode
6. *O* Open a new line above current line and enter input mode
7. *r* Replace the character under the cursor (does NOT enter input mode)
8. *R* Enter input mode replacing characters

9. *C* shift-c. Change rest of line
10. *D* shift-d. Delete rest of line
11. *s* Substitute chars
12. *S* Substitute lines
13. *J* Join lines
14. *J* Join lines

Copy and Paste

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

1. *Y* Copy the current line to the yank buffer
2. *n yy* Copy *n* lines starting from the current to the yank buffer
3. *p* Paste the yank buffer after the cursor (or below the current line)
4. *P* Paste the yank buffer before the cursor (or above the current line)
5. " *x p* Put from buffer *x*
6. " *x y* Yank to buffer *x*
7. " *x d* Delete into buffer *x*

Operators (use double to affect lines)

1. *d* delete
2. *c* change
3. *<* left shift
4. *>* right shift
5. *!* filter through command
6. *=* indent for LISP
7. *y* yank text to buffer

Search and replace

1. */ text* Search forward for *text*
2. *? text* Search backward for *text*
3. *n* Repeat the last search in the same direction
4. *N* Repeat the last search in the reverse direction
5. */* Repeat the last search forward

6. `? Repeat the last search backward`
7. `[addr] s/from/to/ [g]` Search for the occurrence of *from* and replace it with *to* in the current line, or in the range *addr* (two line numbers separated 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".

General

1. `:sh` Forks a shell (to be exited with CTRL-d)
2. `! command` Forks a shell to execute *command*
3. `:set number` Switch on line numbering
4. `:set nonumber` Switch off line numbering

Line Editor Commands

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

Other commands

1. `u` Undo the last change
2. `U` Restore the current line
3. `~` Change case
4. `J` Join the currentline with the next line
5. `.` Repeat last text changing command
6. `CTRL-g` Show file name and line number

Vim as XML Editor

If you do lot of XML editing with Vim, refer to this mini-howto on Vim as XML Editor at <http://www.pinkjuice.com/howto/vimxml> [<http://www.pinkjuice.com/howto/vimxml>] . See also Vim XML Wiki page [<http://www.protocol7.com/svg-wiki/?VimXml>] , Devin Weaver "Script Karma" - xmledit [http://vim.sourceforge.net/scripts/script.php?script_id=301] a filetype plugin to help edit XML, HTML, and SGML documents, Vim.org XML scripts [http://www.vim.org/scripts/script_search_results.php?keywords=xml] , Vim.org XML tips [http://www.vim.org/tips/tip_search_results.php?keywords=xml] , w3.org - Well formed XML doc [<http://www.w3.org/TR/REC-xml.html#sec-well-formed>] .

Matchit.vim & Xmledit.vim

With XML syntax recognition turned on, you can find the matching XML tag by placing the cursor on the XML tag (but within the < and >) and then pressing the % key. Note, however that you should not place the cursor on the < or > since vim will simply match the < or > . Visit http://www.vim.org/scripts/script.php?script_id=39 [www.vim.org/scripts/script.php?script_id=39] For installation instructions enter:

```
vim somefile.txt
:help add-local-help
rpm -qa | grep -i vim
rpm -ql vim-minimal | less
rpm -ql vim-common | less
ls /usr/share/vim/vim61/macros/matchit.*
mkdir -p ~/.vim/plugin
cp /usr/share/vim/vim61/macros/matchit.vim ~/.vim/plugin/
mkdir ~/.vim/doc
cp /usr/share/vim/vim61/macros/matchit.txt ~/.vim/doc/
vim some-xml-file.xml # And test out the % key to match the tags
```

For xmledit.vim download the tar ball from http://www.vim.org/scripts/script.php?script_id=301 [http://www.vim.org/scripts/script.php?script_id=301] and unpack this into ~/.vim for Unix and into \$VIM/vim-files folder for MS DOS or MS Windows.

```
cd ~/.vim
tar xvf xmledit.tar.gz
ln ftplugin/*.xml ~/.vim/plugin
vim some-xml-file.xml # And test out the autocreation of tags in insert mode
```

Vim and Docbook - Useful key mappings

This is from Ashley - gVim and Docbook [<http://supportweb.cs.bham.ac.uk/documentation/tutorials/doc-system/build/tutorials/gvim/gvim.html#gVim-DocBook>] . One can write DocBook documents at an incredibly faster rate if one maps element entry to key bindings. A directory called xml was created in the ftplugins directory of the gVim installation. Into this was placed a vim file that contained macros to map key combinations to element insertions. Comma precedes each mapping, this is convenient because if the user types comma followed by space, nothing happens, but if the user types comma followed by one of the mapped key combinations an element is inserted. Most of the mappings are intuitive, for example, ulink is mapped onto ,u-l. The mappings are very easy to customise and the improvement in document creation speed is amazing.

XML Validation

To check for well-formedness of your XML document just do

```
:!xmllint --valid --noout %
```

And the `--dtdvalid dtd` allows validation of the document(s) against a given DTD.

Build Your "WYSIWYG" HTML Editor With Vi & Netscape

This section was written by Manas K Laha [<http://www.pcquest.com/content/linux/handson/101100102.asp>], Aerospace Engineering Department, IIT Kharagpur, India. This is about a quick and dirty way to create an HTML editor combining vi and Netscape.

If vi (or one of its friends, such as elvis or vim) is your favorite text editor, as it is mine, you must surely long for a way of creating HTML with it quickly and comfortably. And if with that you could get the convenience of "WYSIWYG", wouldn't you just jump at it? All this is indeed possible, and here we'll see how.

The major hurdles I've found in editing HTML with vi are

1. The need to write HTML tags. It appears that there are more tags to be written than displayable matter. Moreover, some of these tags have a syntax that is hard to remember.
2. The need to keep track of whether an opening tag has been given its proper closing tag at the right place (for example, whether a `` has a matching ``).
3. Difficulty in readily identifying matching pairs of tags.

These can be got around using some of the less used features of vi and friends. In this article I shall use vim for definiteness, but the ideas should apply to classic vi and its other look-alikes as well.

The "abbreviation" feature of vim:

Vim has a feature whereby it is possible to assign a keystroke sequence to represent a string of characters in input mode. This is the `ab` colon command. For example, the command

```
:ab tT <tt> </tt>
```

creates an abbreviation, named `tT`, for the sequence of characters `<tt></tt>`. Then, in input mode, as soon as the characters `tT` are typed, they are replaced by the string `<tt></tt>`. Complicated HTML tags may also be abbreviated. The `ab` definition

```
:ab aH <a href=?http://?><!? { ?>^M  
Comments here^M^D </a><!? } ?>
```

makes `aH` the shorthand for

```
<a href=?http://?><!? { ?>
Comments here
</a><!? } ?>
```

where the two `^M`s cause the two line breaks and the `^D` causes the closing `` tag to be indented back to be in line with the opening `<a>` tag. (Indenting the matter enclosed within a matching pair of tags makes reading and editing the raw HTML easier.) The syntax of the tag is outlined, as an aid to memory. The 'Comments here' line is a placeholder, to be replaced with appropriate text.

What are the `{` and `}` within HTML comments doing there? Aha! Those are for matching the opening and closing tags (in this case `<a>` and ``). The bracket matching feature of vi (using the `%` key) is readily usable for the purpose. This can be really helpful when the opening and closing tags are many lines apart and cannot be readily matched 'by eye', such as can be the case with the `` pair.

This technique can be extended to generate fancier HTML, for example with frames. All one has to do is to define the appropriate abbreviations. For example, the definition

```
:ab fS <frameset scrolling=?no?
frameborder=?0? framespacing=?0?
cols=?20%,80%?><!? { ?>^M </frameset> <!?
} ?>
```

makes the string `fS` a convenient abbreviation for the pair of tags:

```
<frameset scrolling=?no? frameborder=?0?
framespacing=?0? cols=?20%,80%?> <!? { ?>
</frameset><!? } ?>
```

Some tags do not like comments to come in between the opening and closing pair. The ones I have found are `<title></title>` and ``. Luckily, in both these the opening and closing pair are never very far apart, so the braces-within-comments feature is not needed.

Sample .vimhtmlrc File

How do I tell vim about these abbreviations? I put all of them (and a command to set some of vim's variables) in a file, which I call `.vimhtmlrc` and which resides in my home directory, and invoke

```
vim -u ~/.vimhtmlrc index.html
```

where `index.html` is the HTML file I want to edit. This is what my `.vimhtmlrc` file looks like:

```
ab aH <a href=?http://?><!? { ?>^M
Comments here^M^D</a> <!? } ?>

ab aM <a href=?mailto:?>^M Comments
here^M^D </a>

ab b0 <body bgcolor=#e0e0e0
text=#000000><!? { ?>^M </body> <!? } ?>
```

```
ab bR <br>

ab cE <center bgcolor=#e0e0e0
text=#000000><!? { ?>^M </center> <!? } ?>

ab cM <!? ^M ?>

ab cO <code> <!? { ?>^M </code> <!? } ?>

ab dL <dl><!? { ?>^M</dl><!? } ?>

ab dT <dt>

ab fO <font color=#000000>^M </font>

ab h1 <h1><!? { ?>^M Heading size
1^M^D </h1> <!? } ?>

ab h2 <h2><!? { ?>^M Heading size
2^M^D </h2><!? } ?>

ab h3 <h3><!? { ?>^M Heading size
3^M^D </h3> <!? } ?>

ab hD <head> <!? { ?>^M </head> <!? } ?>

ab hR <hr>

ab hT <html> <!? { ?>^M </html> <!? } ?>

ab iM <img src=??>

ab lI <li> <!? { ?>^M </li> <!? } ?>

ab oL <ol> <!? { ?>^M </ol> <!? } ?>

ab pR <pre> <!? { ?>^M </pre> <!? } ?>

ab tD <td> <!? { ?>^M </td> <!? } ?>

ab tL <title>^M Title here^M^D </title>

ab tS <table bgcolor=?#d0d0d0?> <!? {
?>^M </table> <!? } ?>

ab tT <tt> </tt>

ab uL <ul> <!? { ?>^M </ul> <!? } ?>

ab xB <b> </b>

ab xI <i> </i>

ab xP <p> <!? { ?>^M </p> <!? } ?>
```

```
se ai aw sw=4 ts=4 wm=10 showmode
showmatch ruler magic
```

When the ab commands are put in a file, to be read in by vim at startup, then the leading `:is` not needed. The last line is a command to set some of vim's variables. Here is what they mean:

```
se      set: tells vim to activate the options that follow
ai      line in the same column as this one (and not from column 1)
aw      autowrite: automatically write file to disk when it changes on a TAB key
ts=4    characters (and not the normal 8); this is my personal preference
sw=4    number of spaces to use for indentation chars from right margin where
wm=10   line wrapping starts (useful if one is writing running text and not prog
        message on status line to show
showmode current mode (for the novice, actually) briefly jump to matching
        opening '(' or '{' or '[' as
showmatch soon as a closing ')' or '}' or ']' is typed; beep if no match
ruler   show cursor line and column in status line some characters, such as '.'
magic   and '*', have special meanings in search and replace patterns.
```

Typing help in a vim window shows the explanations for these options and many more besides.

WYSIWYG

'WYSIWYG' has two parts to it. To begin with is the fact that Netscape under Unix (and Linux) can be controlled remotely.

That is, you may control the behavior of an already running Netscape through commands of the form

```
netscape -remote -noraize 'openFile(/home/mlaha/html/index.html)'
```

(If no Netscape is running, the command just exits with an error message.) This command causes the Netscape browser window to attempt to open the file `/home/mlaha/html/index.html`. For more on remote controlling Netscape, see ["http://home.netscape.com/newsref/std/x-remote.html"](http://home.netscape.com/newsref/std/x-remote.html) [<http://home.netscape.com/newsref/std/x-remote.html>].

And then, there is `atchange`. Jeffrey Copeland and Jeffrey Haemer (["http://alumni.caltech.edu/~copeland/work/edit-web.html"](http://alumni.caltech.edu/~copeland/work/edit-web.html) [<http://alumni.caltech.edu/~copeland/work/edit-web.html>] , ["ftp://ftp.ncifcrf.gov/pub/delila/atchange"](ftp://ftp.ncifcrf.gov/pub/delila/atchange) [<ftp://ftp.ncifcrf.gov/pub/delila/atchange>] and ["http://www.lecb.ncifcrf.gov/~toms/atchange.html"](http://www.lecb.ncifcrf.gov/~toms/atchange.html) [<http://www.lecb.ncifcrf.gov/~toms/atchange.html>]) describe a little shell script, called `atchange`, that waits in the background for a named file to change and then invokes a specified command. Thus,

```
atchange index.html 'netscape -noraize -remote 'openFile(/home/mlaha/html/index.ht
```

would cause `atchange` to run in the background, watching the file `index.html` and, as soon as it changed, ask Netscape to display it afresh. If you were editing `index.html` with `vi`, then, when you saved it (with `:w`, say), `atchange` would spring into action and Netscape would update its display.

If you wish to edit another HTML file, you have to quit vim, kill the current invocation of `atchange`, then start it again with the name of the new file in place of `index.html` and begin editing that file with vim.

Other 'WYSIWYG' uses

As you may have guessed, atchange can be used in other instances, too. You can make a handy 'WYSIWYG' LaTeX editor by having atchange monitor your LaTeX source and, when it changed, run the necessary programs to convert it to Postscript. The 'WYSIWYG' capability is provided in this case by invoking Ghostview with the monitoring option (-watch) that causes it to redisplay its current Postscript file whenever that file changes. Thus, every time you saved your LaTeX source file in the editor, the Postscript output with the latest changes would be automatically displayed in the Ghostscript window.

Source code for atchange

```
#!/usr/local/bin/perl
# by Jeff Haemer
#     and a tip o' the hat to Tom Schneider
#     who wrote the original version as a shell script
# version = 2.07 of atchange 1999 Dec 30
# 1999 Dec 18: Added shell call to /bin/csh so that
# atchange works under Linux.
# 1999 Feb 5: By setting the PERLCSH variable, the new shell can tell
#             it has been called by atchange.
# The test inside the .cshrc is:
#if ( (! $?PERLCSH ) && $?prompt) then
#   stty erase '^H'
#   set prompt = "`uname -n` \!% "
#endif
# This is necessary under Sun Solaris 2.6 because otherwise the
# call to stty gives an error message now.
# previous change: 1997 Jan 9
# delay time is 0.25 seconds
# For current version and other information about this program, see:
# http://www.lecb.ncifcrf.gov/~toms/atchange.html
# Tom Schneider
# National Cancer Institute
# Laboratory of Mathematical Biology
# Frederick, Maryland 21702-1201
# toms@ncifcrf.gov
# http://www.lecb.ncifcrf.gov/~toms/
# 1999 Dec 30: James Haefner (jhaefner@biology.usu.edu)
# has found that some changes are needed to make atchange
# work under Linux. See the web site for details.
# This code will be revised when a good solution is found.
$0 =~ s(.*\/)(); # basename
$usage = "usage: $0 filename cmd | $0 command_file";
@ARGV || die $usage; # check for proper invocation
# This allows the .cshrc to know that atchange has called it:
$ENV{'PERLCSH'} = "TRUE";
# Haefner Suggestion 1999 Dec 18:
##if default SHELL is sh or csh or tcsh use the following line
###$shell = $ENV{"SHELL"} ? $ENV{"SHELL"} : "/bin/sh";
##if default SHELL is bash (eg, Linux) use the following line
# 1999 Dec 28 - this is not a good idea - untestable by me
# $shell = "/bin/csh";
```

```
$shell = $ENV{"SHELL"} ? $ENV{"SHELL"} : "/bin/sh";
open(SHELL, "|$shell") || die "Can't pipe to $shell: $!";
select(SHELL); $| = 1;
if (@ARGV > 1) {
    # it's a file and a command
    $file = shift;
    $cmd{$file} = join(" ", @ARGV) . "\n";
    $old{$file} = (stat($file))[9]; # mod time.
} else {
    # it's a program
    open(PGM, shift) || die "Can't open $_: $!";
    $/ = "";
    while(<PGM>) {
        # first read the program
        s/#.*\n/\n/g;
        ($file, $cmd) = /(\S*)\s+([\^\000]+)/;
        $cmd{$file} = $cmd;
        unless ($file) { print $cmd{$file}; next; }
        if ($file && ! $cmd{$file}) { warn "odd line"; next; }
        $old{$file} = (stat($file))[9]; # mod time.
    }
}
while(1) {
    # sleep 1;
    select(undef, undef, undef, 0.25); # wait a quarter second, then
    foreach (keys %cmd) {
        # rip through the whole list
        atchange($_);
    }
}
close(SHELL);
sub atchange {
    # if $file has changed, do $cmd{$file}
    my($file) = @_;
    my($new);
    $new = (stat($file))[9];
    return 0 if ($old{$file} == $new);
    while (1) {
        # wait until it stops changing
        $old{$file} = $new;
        sleep 1;
        $new = (stat($file))[9];
        if ($old{$file} == $new) {
            print $cmd{$file};
            return 1;
        }
    }
}
}
```

HTML Beautifier Inside Vim : Program Tidy

While editing HTML files with Vim, it is possible to automatically check syntax errors and beautify the code with program like Tidy

(Visit Raggett Tidy [<http://www.w3.org/People/Raggett/tidy>], Tidy Project [<http://tidy.sourceforge.net>] and Beatifier HOWTO [<http://www.tldp.org/HOWTO/C-C++Beautifier-HOWTO>]).

Inside Vim (gvim), click on menu Tools->'Set Compiler'->'Tidy' and give command ":make". The :make command will run the tidy and show all errors if any. You can also edit the setting file tidy.vim and customize:

```
ls /usr/share/vim/vim61/compiler
cd /usr/share/vim/vim61/compiler
cp tidy.vim tidy.vim.backup
man tidy      # And the manual page of tidy and see options..
tidy -h | less  # See help about using tidy
vi tidy.vim
```

See also related HTML validators section (HTML Beautifier section) in Beautifier HOWTO [<http://www.tldp.org/HOWTO/C-C++Beautifier-HOWTO>]

Emacs - Old Habits Die Hard !

If you were using Emacs before and now started using Vim & vi, then this mistake is committed very often and will be quite annoying to you.

You hit "control x" "control s" to save the file and Vim stops dead. How to rescue without killing the xwindow containing the vim process, and recovering. You will see that there is no response at the keyboard.

The explanation for this behaviour is : It is not the feature of vim but it is the feature of the xterm 'terminal'. The "control s" means stop feeding to terminal output, so you do not see any output. To get out of it do "control q" (to quit out of stop output). This happens only in vim/vi and not in gvim (graphical Vim).

Also in xterm, at the bash prompt type something and do "control s" and type something you will not see output. Now type "control q" you will see output.

Related URLs

Related VIM URLs are at -

- C and C++ Beautifier "<http://www.metalab.unc.edu/LDP/HOWTO/C-C++Beautifier-HOWTO.html>" [<http://www.metalab.unc.edu/LDP/HOWTO/C-C++Beautifier-HOWTO.html>]
- Linux goodies main site is at "<http://milkyway.has.it>" [<http://milkyway.has.it>] and mirror at "<http://www.milkywaygalaxy.freesevers.com>" [<http://www.milkywaygalaxy.freesevers.com>] Mirror sites are at - [angelfire](http://www.angelfire.com/country/aldev0) [<http://www.angelfire.com/country/aldev0>] , [geocities](http://www.geocities.com/alavoor/index.html) [<http://www.geocities.com/alavoor/index.html>] , [virtualave](http://aldev0.virtualave.net) [<http://aldev0.virtualave.net>] , [Fortunecity](http://members.fortunecity.com/aldev) [<http://members.fortunecity.com/aldev>] , [Freewebsites](http://aldev.freewebsites.com) [<http://aldev.freewebsites.com>] , [Tripod](http://members.tripod.lycos.com/aldev) [<http://members.tripod.lycos.com/aldev>] , [101xs](http://www.101xs.com/101xs/aldev) [<http://www.101xs.com/101xs/aldev>] , [50megs](http://aldev0.50megs.com) [<http://aldev0.50megs.com>] ,

Other Formats of this Document

This document is published in 14 different formats namely - DVI, Postscript, Latex, Adobe Acrobat PDF, LyX, GNU-info, HTML, RTF(Rich Text Format), Plain-text, Unix man pages, single HTML file, SGML (Linuxdoc format), SGML (Docbook format), MS WinHelp format.

This howto document is located at -

- "<http://www.linuxdoc.org>" [<http://www.linuxdoc.org>] and click on HOWTOs and search for howto document name using CTRL+f or ALT+f within the web-browser.

You can also find this document at the following mirrors sites -

- "<http://www.caldera.com/LDP/HOWTO>" [<http://www.caldera.com/LDP/HOWTO>]
- "<http://www.linux.ucla.edu/LDP>" [<http://www.linux.ucla.edu/LDP>]
- "<http://www.cc.gatech.edu/linux/LDP>" [<http://www.cc.gatech.edu/linux/LDP>]
- "<http://www.redhat.com/mirrors/LDP>" [<http://www.redhat.com/mirrors/LDP>]
- Other mirror sites near you (network-address-wise) can be found at "<http://www.linuxdoc.org/mirrors.html>" [<http://www.linuxdoc.org/mirrors.html>] select a site and go to directory `/LDP/HOW-TO/xxxxx-HOWTO.html`
- You can get this HOWTO document as a single file tar ball in HTML, DVI, Postscript or SGML formats from - "<ftp://www.linuxdoc.org/pub/Linux/docs/HOWTO/other-formats/>" [<ftp://www.linuxdoc.org/pub/Linux/docs/HOWTO/other-formats/>] and "<http://www.linuxdoc.org/docs.html#howto>" [<http://www.linuxdoc.org/docs.html#howto>]
- Plain text format is in: "<ftp://www.linuxdoc.org/pub/Linux/docs/HOWTO>" [<ftp://www.linuxdoc.org/pub/Linux/docs/HOWTO>] and "<http://www.linuxdoc.org/docs.html#howto>" [<http://www.linuxdoc.org/docs.html#howto>]
- Single HTML file format is in: "<http://www.linuxdoc.org/docs.html#howto>" [<http://www.linuxdoc.org/docs.html#howto>]

Single HTML file can be created with command (see `man sgml2html`) - `sgml2html -split 0 xxxxhowto.sgml`

- Translations to other languages like French, German, Spanish, Chinese, Japanese are in "<ftp://www.linuxdoc.org/pub/Linux/docs/HOWTO>" [<ftp://www.linuxdoc.org/pub/Linux/docs/HOWTO>] and "<http://www.linuxdoc.org/docs.html#howto>" [<http://www.linuxdoc.org/docs.html#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>" [<http://www.sgmltools.org>] Compiling the source you will get the following commands like

- `sgml2html xxxxhowto.sgml` (to generate html file)
- `sgml2html -split 0 xxxxhowto.sgml` (to generate a single page html file)
- `sgml2rtf xxxxhowto.sgml` (to generate RTF file)
- `sgml2latex xxxxhowto.sgml` (to generate latex file)

Acrobat PDF format

PDF file can be generated from postscript file using either acrobat *distill* or *Ghostscript* . And postscript file is generated from DVI which in turn is generated from LaTeX file. You can download distill software from "<http://www.adobe.com>" [<http://www.adobe.com>] . Given below is a sample session:

```
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.

Convert Linuxdoc to Docbook format

This document is written in linuxdoc SGML format. The Docbook SGML format supercedes the linuxdoc format and has lot more features than linuxdoc. The linuxdoc is very simple and is easy to use. To convert linuxdoc SGML file to Docbook SGML use the program *ld2db.sh* and some perl scripts. The *ld2db* output is not 100% clean and you need to use the *clean_ld2db.pl* perl script. You may need to manually correct few lines in the document.

- Download *ld2db* program from "<http://www.dcs.gla.ac.uk/~rrt/docbook.html>" [<http://www.dcs.gla.ac.uk/~rrt/docbook.html>] or from "<http://milkyway.has.it>" [<http://milkyway.has.it>] and mirror at AI Dev site [<http://www.milkywaygalaxy.freesevers.com>]
- Download the *cleanup_ld2db.pl* perl script from "<http://milkyway.has.it>" [<http://milkyway.has.it>] and mirror at AI Dev site [<http://www.milkywaygalaxy.freesevers.com>]

The *ld2db.sh* is not 100% clean, you will get lots of errors when you run

```
bash$ ld2db.sh file-linuxdoc.sgml db.sgml
bash$ cleanup.pl db.sgml > db_clean.sgml
bash$ gvim db_clean.sgml
bash$ docbook2html db.sgml
```

And you may have to manually edit some of the minor errors after running the perl script. For e.g. you may need to put closing tag `</Para>` for each `<Listitem>`

Convert to MS WinHelp format

You can convert the SGML howto document to Microsoft Windows Help file, first convert the sgml to html using:

```
bash$ sgml2html xxxxhowto.sgml      (to generate html file)
bash$ sgml2html -split 0 xxxxhowto.sgml (to generate a single page html file)
```

Then use the tool *HtmlToHlp* [<http://javadocs.planetmirror.com/htmltohlpe.html>] . You can also use *sgml2rtf* and then use the RTF files for generating winhelp files.

Reading various formats

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>" [<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.

Copyright Notice

Copyright policy is GNU/GPL as per LDP (Linux Documentation project). LDP is a GNU/GPL project. Additional restrictions are - you must retain the author's name, email address and this copyright notice on all the copies. If you make any changes or additions to this document then you should notify all the authors of this document.