

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

Table of Contents

<u>Vim Color Editor HOW-TO (Vi Improved with syntax color highlighting)</u>	1
<u>Al Dev (Alavor Vasudevan) alavor@yahoo.com</u>	1
<u>1.Introduction</u>	1
<u>2.Setup gvim init files</u>	1
<u>3.Color Syntax init files</u>	1
<u>4.VIM Usage</u>	1
<u>5.Vi companions</u>	2
<u>6.Online VIM help</u>	2
<u>7.Vim Home page and Vim links</u>	2
<u>8.Vim Tutorial</u>	2
<u>9.Vi Tutorial</u>	2
<u>10.Vim Reference Card</u>	2
<u>11.Other Formats of this Document</u>	3
<u>1. Introduction</u>	3
<u>1.1 Install Vim on Redhat Linux</u>	3
<u>1.2 Install Vim on GNU Debian Linux</u>	4
<u>1.3 Install Vim on Unixes</u>	4
<u>1.4 Install Vim on Microsoft Windows 95/NT</u>	4
<u>10. Vim Reference Card</u>	5
<u>10.1 Vi states</u>	5
<u>10.2 Shell Commands</u>	5
<u>10.3 Setting Options</u>	5
<u>10.4 Notations used</u>	5
<u>10.5 Interrupting, cancelling</u>	6
<u>10.6 File Manipulation</u>	6
<u>10.7 Movement</u>	6
<u>10.8 Line Positioning</u>	7
<u>10.9 Character positioning</u>	7
<u>10.10 Words, sentences, paragraphs</u>	7
<u>10.11 Marking and returning</u>	8
<u>10.12 Corrections during insert</u>	8
<u>10.13 Adjusting the screen</u>	8
<u>10.14 Delete</u>	8
<u>10.15 Insert, change</u>	9
<u>10.16 Copy and Paste</u>	9
<u>10.17 Operators (use double to affect lines)</u>	9
<u>10.18 Search and replace</u>	9
<u>10.19 General</u>	10
<u>10.20 Line Editor Commands</u>	10
<u>10.21 Other commands</u>	10
<u>11. Other Formats of this Document</u>	11
<u>2. Setup gvim init files</u>	12
<u>2.1 Sample gvimrc file</u>	13
<u>2.2 Xdefaults parameters</u>	14
<u>3. Color Syntax init files</u>	15
<u>4. VIM Usage</u>	15
<u>5. Vi companions</u>	16

Table of Contents

5.1 Ctags for ESOL	17
5.2 Ctags for Javascript programs, Korn, Bourne shells	18
5.3 Debugger gdb	19
6. Online VIM help	20
7. Vim Home page and Vim links	21
8. Vim Tutorial	22
8.1 Vim Hands-on Tutorial	22
8.2 Vi Tutorials on Internet	22
9. Vi Tutorial	23
9.1 Cursor Movement Commands	23
9.2 Repeat Counts	26
9.3 Deleting Text	26
9.4 Changing Text	27
9.5 Yanking (Copying) Text	27
9.6 Filtering text	28
9.7 Marking Lines and Characters	28
9.8 Naming Buffers	29
9.9 Substitutions	29
9.10 Miscellaneous "Colon Commands"	30
9.11 Setting Options	30
9.12 Key Mappings	31
9.13 Editing Multiple Files	31
9.14 Final Remarks	32

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

AI Dev (Alavor Vasudevan) alavor@yahoo.com

v4.0, 31 December 1999

This document is a guide to very quickly setup Vim color editor on Linux or Unix systems. The information here will improve the productivity of programmer since Vim editor supports syntax color highlighting and bold fonts which improves the "readability" of program code. Programmer's productivity improves 2 to 3 times with color editor like Vim. The information in this document applies to all operating systems where Vim works that is – Windows 95/NT and all flavors of Unix like Linux, Solaris, HPUX, AIX, SCO, Sinix, BSD, SCO, etc..

1. Introduction

- [1.1 Install Vim on Redhat Linux](#)
- [1.2 Install Vim on GNU Debian Linux](#)
- [1.3 Install Vim on Unices](#)
- [1.4 Install Vim on Microsoft Windows 95/NT](#)

2. Setup gvim init files

- [2.1 Sample gvimrc file](#)
- [2.2 Xdefaults parameters](#)

3. Color Syntax init files

4. VIM Usage

5.Vi companions

- [5.1 Ctags for ESOL](#)
- [5.2 Ctags for Javascript programs, Korn, Bourne shells](#)
- [5.3 Debugger gdb](#)

6.Online VIM help

7.Vim Home page and Vim links

8.Vim Tutorial

- [8.1 Vim Hands-on Tutorial](#)
- [8.2 Vi Tutorials on Internet](#)

9.Vi Tutorial

- [9.1 Cursor Movement Commands](#)
- [9.2 Repeat Counts](#)
- [9.3 Deleting Text](#)
- [9.4 Changing Text](#)
- [9.5 Yanking \(Copying\) Text](#)
- [9.6 Filtering text](#)
- [9.7 Marking Lines and Characters](#)
- [9.8 Naming Buffers](#)
- [9.9 Substitutions](#)
- [9.10 Miscellaneous "Colon Commands"](#)
- [9.11 Setting Options](#)
- [9.12 Key Mappings](#)
- [9.13 Editing Multiple Files](#)
- [9.14 Final Remarks](#)

10.Vim Reference Card

- [10.1 Vi states](#)
- [10.2 Shell Commands](#)
- [10.3 Setting Options](#)
- [10.4 Notations used](#)
- [10.5 Interrupting, cancelling](#)
- [10.6 File Manipulation](#)

- [10.7 Movement](#)
- [10.8 Line Positioning](#)
- [10.9 Character positioning](#)
- [10.10 Words, sentences, paragraphs](#)
- [10.11 Marking and returning](#)
- [10.12 Corrections during insert](#)
- [10.13 Adjusting the screen](#)
- [10.14 Delete](#)
- [10.15 Insert, change](#)
- [10.16 Copy and Paste](#)
- [10.17 Operators \(use double to affect lines\)](#)
- [10.18 Search and replace](#)
- [10.19 General](#)
- [10.20 Line Editor Commands](#)
- [10.21 Other commands](#)

11. Other Formats of this Document

[Next](#) [Previous](#) [Contents](#) [Next](#) [Previous](#) [Contents](#)

1. Introduction

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

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

1.1 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 | 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'.

1.2 Install Vim on GNU Debian Linux

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

```
apt-get install vim vim-rt
```

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

1.3 Install Vim on Unixes

For other flavors of unixes like Solaris, HPUX, AIX, Sinix, SCO download the source code file

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

1.4 Install Vim on Microsoft Windows 95/NT

For Windows 95/NT, download the zip file and install clicking on setup. Get the zip file from –

- The home page of vim is at <http://www.vim.org>
- Mirror site in US is at <http://www.us.vim.org>

[Next](#) [Previous](#) [Contents](#)[Next](#)[Previous](#)[Contents](#)

10. Vim Reference Card

10.1 Vi states

Vi has 3 modes:

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

10.2 Shell Commands

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

10.3 Setting Options

1. **:setoption** Activate *option*
2. **:setoption=value** Assign *value* to *option*
3. **:set nooption** Deactivate *option*
4. **:set** Display options set by user
5. **:set all** Display list of all current options, both default and those set by the user
6. **:set option?** Display values of *option*

10.4 Notations used

Notations:

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

10.5 Interrupting, cancelling

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

10.6 File Manipulation

- **ZZ** Save the file and exit vi
- **:wq** Save the file and exit vi
- **:w** Write the current file
- **:w!** Force write the current file, if file is read-only
- **:wname** Write to file *name*
- **:q** Exit from vi
- **:q!** Force exit from vi (discarding changes)
- **:e name** Edit file *name*
- **:e!** reedit, discard changes
- **:e + name** edit file *name*, starting at end
- **:e + n** edit starting at line *n*
- **:e #** edit alternate file
- **:n** edit next file in *arglist*
- **:args** list files in current filelist
- **:rew** rewind current filelist and edit first file
- **:n args** specify new arglist
- **:f** show current file and line
- **CTRL-G** synonym for **:f**, show current file and line
- **:ta tag** to tag file entry *tag*
- **CTRL-]** :ta, following word is tag

10.7 Movement

- **Arrows** Move the cursor
- **CTRL-d** Scroll half page down
- **CTRL-u** Scroll half page up
- **CTRL-f** Scroll a full page down
- **CTRL-b** Scroll a full page up
- **:0** Move to start of file
- **:n** Move to line number *n*
- **:\$** Move to end of file
- **0** Move to start of line
- **^** Move to first non-blank character
- **\$** Move to end of line
- **CR** Move to the start of next line
- **-** Move to the start of previous line
- **%** Find matching bracket

- **G** goto line (last line default)
- **]]** next section/function
- **[[** previous section/function

10.8 Line Positioning

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

10.9 Character positioning

- **0** beginning of line
- **\$** end of line
- **h** forward
- **l** backwards
- **SPACE** same as l
- **fx** find x forward
- **Fx** find x backward
- **;** repeat last f F
- **,** inverse of ;
- **|** to specified column
- **%** find matching { or }

10.10 Words, sentences, paragraphs

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

10.11 Marking and returning

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

10.12 Corrections during insert

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

10.13 Adjusting the screen

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

10.14 Delete

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

10.15 Insert, change

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

10.16 Copy and Paste

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

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

10.17 Operators (use double to affect lines)

- **d** delete
- **c** change
- **<** left shift
- **>** right shift
- **!** filter through command
- **=** indent for LISP
- **y** yank text to buffer

10.18 Search and replace

- **/text** Search forward for *text*
- **?text** Search backward for *text*
- **n** Repeat the last search in the same direction

- **N** Repeat the last search in the reverse direction
- **/** Repeat the last search forward
- **?** Repeat the last search backward
- **[addr] s/from/to/ [g]** Search for the 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".

10.19 General

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

10.20 Line Editor Commands

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

10.21 Other commands

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

[NextPreviousContents](#) Next [PreviousContents](#)

11. Other Formats of this Document

This document is published in 10 different formats namely – DVI, Postscript, Latex, LyX, GNU-info, HTML, RTF(Rich Text Format), Plain-text, Unix man pages and SGML.

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

The document is written using a tool called "SGML tool" which can be got from – <http://www.xs4all.nl/~cg/sgmltools/> Compiling the source you will get the following commands like

- `sgml2html vim-howto.sgml` (to generate html file)
- `sgml2rtf vim-howto.sgml` (to generate RTF file)
- `sgml2latex vim-howto.sgml` (to generate latex file)

This document is located at –

- <http://metalab.unc.edu/LDP/HOWTO/VIM-HOWTO.html>

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

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

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

```
To read dvi document give the command -
xdvi -geometry 80x90 howto.dvi
And resize the window with mouse. See man page on xdvi.
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. Ghostscript and gv are also available on other platforms like OS/2, Windows 95 and NT.

```
To read postscript document give the command -
gv howto.ps
```

```
To use ghostscript give -
ghostscript howto.ps
```

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

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

Next [PreviousContentsNextPreviousContents](#)

2. Setup gvim init files

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

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

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

```
" This line is a comment .... one which begins with double-quotes
" The best is the bold font, try all of these and pick one....
set guifont=8x13bold
"set guifont=9x15bold
"set guifont=7x14bold
"set guifont=7x13bold
"
" Highly recommended to set tab keys to 4 spaces
set tabstop=4
set shiftwidth=4
"
" The opposite is 'set wrapscan' while searching for strings....
set nowrapscan
"
" The opposite is set noignorecase
set ignorecase
```

It is **very strongly** recommended that you set the *tabstop* to 4 and *shiftwidth* to 4. The *tabstop* is number spaces TAB key will put while editing with gvim. The *shiftwidth* is the number spaces the lines will be shifted with ">>" or "<<" vi commands (Press twice "greater than" and "less than" keys on keyboard). Refer

to Vi tutorials [Vim Tutorial](#) for more details.

2.1 Sample gvimrc file

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

```
highlight Normal guibg=lightyellow
```

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

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

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

" set the X11 font to use
" set guifont=-misc-fixed-medium-r-normal--14-130-75-75-c-70-iso8859-1

" 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`.

2.2 Xdefaults parameters

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

WARNING!! WARNING: Do NOT put *Vim*geometry* it will break the gvim menu, use *Vim.geometry* instead

Edit the `$HOME/.Xdefaults` and put 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. Asterik 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
```

[NextPreviousContentsNextPreviousContents](#)

3. Color Syntax init files

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 a default file extension names, for example, Javascript syntax file supports *.js extension. If you happen to use different extension conflicting with another default syntax file as in putting Javascript in *.html file than you MUST source in with command :so \$VIM/syntax/javascript.vim. Best method is create a soft link like –

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

[NextPreviousContentsNextPreviousContents](#)

4. VIM Usage

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

```
gvim foo.cpp
```

To use non-gui mode give –

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

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

GUI mode gvim provides the following –

- You can mark the text using the mouse to do cut, copy and paste.
 - You can use the Menu bar which has – File, Edit, Window, Tools, 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.
-

[NextPreviousContentsNextPreviousContents](#)

5. Vi companions

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

The **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 "to and fro" thousands of functions of C/C++ programs. See 'man ctags' on Unix. It is **very important** that you **MUST** learn how to use ctags in order 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 calling line to function and drill down deeper into nested function calls and unwind back all the way upto 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 line having 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)**, this will take you right at the line where the function begins automatically!!

```
// 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 will take you to function by name "foo_function". To come back to this line

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

5.1 Ctags for ESQL

Since ctags directly does not 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 than 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
```

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

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

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

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

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

5.2 Ctags for Javascript programs, Korn, Bourne shells

The shell script given below can be used to generate tags for a very large variety of programs written in JavaScript, PHP/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 " "
echo " "
echo " "
echo " "
echo "Generate tags for ...."
while :
do
    echo "    Enter file extension for which you want to generate tags."
```

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

```
echo -n "      File-extension should be like sh, js, ksh, etc... : "  
read ans  
  
if [ "$ans" == "" ]; then  
    echo " "  
    echo "Wrong entry. Try again!"  
else  
    break  
fi  
done  
  
\rm -f $tmp_tag  
  
aa=`ls *.$ans`  
  
for ii in $aa  
do  
    jj=`echo $ii | cut -d'.' -f1`  
    #echo $jj  
    cp $ii $jj.c  
    ctags $jj.c  
    echo "s/$jj.c/$ii/g" > $tmp_tag2  
    sed -f $tmp_tag2 tags >> $tmp_tag  
    \rm -f tags $jj.c  
done  
  
sort $tmp_tag > tags  
  
\rm -f $tmp_tag $tmp_tag2
```

5.3 Debugger gdb

You would be using gdb extensively along with Vi. Debugging is the most important aspect of programming as major cost of software project 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 setup easy aliases do –

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

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

To get online help do –

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

To set breakpoints and do debugging

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

To analyze the core dumps do

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

You can also use GUI version of gdb called xxgdb.

Memory leak tools –

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

[NextPreviousContentsNextPreviousContents](#)

6. Online VIM help

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

Or inside the gvim session type :help to get the help page. See also [Vim Tutorial](#)

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

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_	
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

[NextPreviousContentsNextPreviousContents](#)

7. Vim Home page and Vim links

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

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

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

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

Vim Reference Guide at <http://scisun.sci.ccnycuny.edu/~olrcc/vim/>

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

Mailing list archives are kept at:

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

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

[NextPreviousContentsNextPreviousContents](#)

8. Vim Tutorial

8.1 Vim Hands-on Tutorial

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

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

8.2 Vi Tutorials on Internet

- Purdue University <http://ecn.www.ecn.purdue.edu/ECN/Documents/VI/>
- Quick Vi tutorial <http://linuxwww.db.erau.edu/LUG/node165.html>
- Advanced Vi tutorial <http://www.yggdrasil.com/bible/bible-src/user-alpha-4/guide/node171.html>
- Tutorials http://www.cfm.brown.edu/Unixhelp/vi_.html
- Tutorials http://www.linuxbox.com/~taylor/4ltrwr/section3_4.html
- Unix world online vi tutorial <http://www.networkcomputing.com/unixworld/unixhome.html>

- Univ of Hawaii tutorial <http://www.eng.hawaii.edu/Tutor/vi.html>
- InfoBound <http://www.infobound.com/vi.html>
- Cornell Univ <http://www.tc.cornell.edu/Edu/Tutor/Basics/vi/>
- Vi Lovers home page: <http://www.cs.vu.nl/~tmgil/vi.html>
- Ater Sept 2000, will moveto <http://www.thomer.com/thomer/vi/vi.html>
- Beginner's Guide to vi <http://www.cs.umd.edu/unixinfo/general/packages/viguide.html>
- vi Help file <http://www.vmunix.com/~gabor/vi.html>
- vim FAQ <http://www.math.fu-berlin.de/~guckes/vim/faq/>

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

[NextPreviousContentsNextPreviousContents](#)

9. Vi Tutorial

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

9.1 Cursor Movement Commands

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

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

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

- **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 **h j k l** to move to an indented text line that contains few "e" characters. If you can't find an indented line in your file, create one by inserting a few space characters at the beginning of a line.)
- **^** : Move to first non-white character of current line. (For indented line, 0 and ^ are different.)
- **\$** : Move to last character of current line.
- **tC** : Move to (but not on) next character c in current line. (Press 0, then press te. This will move to the first e in the 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 \$, than 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.

9.2 Repeat Counts

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

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

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

- **4H** : Move to Line 4 on the screen (home plus 3)
- **8L** : Move to the 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

9.3 Deleting Text

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

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

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

9.4 Changing Text

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

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

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

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

Some other change commands:

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

9.5 Yanking (Copying) Text

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

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

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

Other yank commands:

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

9.6 Filtering text

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

Some examples will help illustrate. Create a line in your file containing just the word **who** and absolutely no other text. Put the cursor on this line, and press **!!**. This command is analogous to **dd**, **cc**, or **yy**, but instead of deleting, changing, or yanking the current line, it filters the current line. When you press the second **!**, the cursor drops down to the lower left corner of the screen and a single **!** is displayed, prompting you to enter the name of a filter. As the filter name, type **sh** and press the Return key. **sh** (the Bourne shell) is a filter! It reads standard input, does some processing of its input (that is, executes commands), and sends its output (the output of those commands) to standard output. Filtering theline 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 numebr 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
- **!Ggrep the < CR >** : Replace from the current line to and including Line 1 with just the lines that contain the
- **!Gawk '{print \$1}' < CR >** : From the current line to the end of file, replace every line with just its first word.

9.7 Marking Lines and Characters

You can mark lines and characters to be used as targest for movement, deletion, change, yankidn, 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, than **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, than **a**). This moves on top of the character marked with **a**.

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

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

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

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

9.8 Naming Buffers

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

If you wish to delete, change, or yank multiple sections of text and remember them all (up to a maximum of 26), you can give a buffer name ahead of the delete change or yank command. A buffer name has the form "c (double quote, lowercase c).

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

Some other named buffer commands:

- **"a6yy** : Yank six lines (current and next five) into buffer a
- **"bd1G** : Delete from the 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)

9.9 Substitutions

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

- **:1,\$s/the/THE/g** From Line 1 to the last line (line \$), substitute for the text THE; do this globally in each line where the 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

9.10 Miscellaneous "Colon Commands"

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

Some important examples:

- **:w** Write the buffer contents to the file without quitting from **vi**
- **:w abc** Write the buffer contents to the file abc (creating abc if it doesn't exist, or overwriting current contents if it does exist) without quitting from **vi**
- **:1,10w abc** Write lines 1 through 10 to file abc
- **:'a,\$w abc** Write from the line marked a to the last line into file abc
- **:e abc** Edit file abc, instead of the current file. **vi** prints an error message if changes have been made to the 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.)

9.11 Setting Options

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

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

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

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

If you find you always want certain options set certain ways, you can put the set commands you want 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'
```

Yank this line into buffer a:

```
"ayy
```

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

```
:n < CR >
```

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

```
n
```

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

```
"ap
```

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

```
jj"byy
```

Save the changes to wknight

```
:w < CR >
```

Now, rewind to croc

```
:rew < CR >
```

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

```
n"bp
```

Save the changes, and exit vi

```
ZZ
```

9.14 Final Remarks

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

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

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

[NextPreviousContents](#)