The 'Vi Improved' Text Editor

"*annoying but excellent*" DAVID RAYNER

Contents

Vim is an editor which proves frustrating to use at first (and later...) but which is capable of editing text rapidly with very few keystrokes. For example the keystrokes 'dap' deletes the current paragraph (and copies it to the clipboard, so it can be pasted later). Vim is especially designed for people who can type without looking at the keyboard....

Web Resources
Section 1

`www.vim.org`
the official site

`http://en.wikibooks.org/wiki/Learning_the_vi_editor/`
another book

1.1 Tips

`http://vim.runpaint.org/toc/`
good advanced tips, with explanations

`www.vim.org/tips/index.php`
lots of tips

`http://vim.wikia.com/wiki/Main_Page`
more tips

`http://rayninfo.co.uk/vimtips.html`
advanced tips

1.2 Cheat Sheets

`http://www.eec.com/business/vi.html`

The Crash Course
Section 2

The most important thing to know about vim, is that it has 2 (main) modes: 'Normal' mode and 'Insert' mode. In normal mode you can't type anything!!! Thereby leading the novice to a great deal of confusion, perplexity and desperation. In order to start entering text in the editor, you have to type 'i', (which puts the editor in Insert Mode). Hitting the Esc key, goes back to Normal Mode (in which you can enter commands). If you can overcome this initial hurdle in using Vim, you may (one day) appreciate its qualities.
One important hint: If vim seems to have gone completely mad ... check that the caps lock key is not on!

Get me the hell out of here!!, (save and exit)

```
[esc] ZZ
```

```
[esc] :wq! [enter]    ~(if the previous didnt work!)
```

Enter some text

```
i                  ~(and then start typing)
```

Undo the change which you just made (your gonna need this!).

```
[esc] u
```

```
:u   ~(the same)
```

Any command beginning with ':' must be terminated by pressing the Enter key.

Repeat the thing you just did

```
.
```

Redo the change which you just undid

```
[esc] :red [enter]
```

Stop inserting text

```
[esc]
```

Find the word 'big' in the file

```
[esc] /big
```

View the helpful user guides to vim. These are more understandable Than the reference manuals

```
:help user
```

View the help for the command ':w' (the save file command)

```
:help :w
```

Getting help for the Vim editor

```
:help
```

```
:help commandname
```

The help obtained through typing ':help topic' can be very cryptic for the same reason that unix man pages and javadoc files are cryptic ... there are virtually no example commands. Look on the Internet for the countless tutorials and cheat-sheets for vim.

2.1 Tips

Absolutely essential tips

`* # g* g#`	Find word under cursor (<cword>) (forwards/backwards)
`%`	Match brackets {}[]()
`.`	Repeat last modification
`@:`	Repeat last : command (then @@)
`matchit.vim`	% now matches tags <tr><td><script> <?php etc
`<C-N><C-P>`	Word completion in insert mode
`<C-X><C-L>`	Line complete SUPER USEFUL
`/<C-R><C-W>`	Pull <cword> onto search/command line
`/<C-R><C-A>`	Pull <CWORD> onto search/command line
`:set ignorecase`	You nearly always want this
`:set smartcase`	Overrides ignorecase if uppercase used in search string
`:syntax on`	Colour syntax in Perl,HTML,PHP etc
`:set syntax=perl`	Force syntax (usually taken from file extension)
`:h regexp<C-D>`	Type control-D and get a list all help topics containing

Getting Help

help for help (USE TAB)

`:h quickref`	VIM Quick Reference Sheet (ultra)
`:h tips`	Vim's own Tips Help
`:h visual<C-D><tab>`	Obtain list of all visual help topics
	: Then use tab to step thru them
`:h ctrl<C-D>`	List help of all control keys
`:helpg uganda`	Grep HELP Files use :cn, :cp to find next
`:helpgrep edit.*director`	Grep help using regexp
`:h :r`	Help for :ex command
`:h CTRL-R`	Normal mode
`:h /\r`	What's \r in a regexp (matches a <CR>)
`:h \\zs`	Double up backslash to find \zs in help ???
`:h i_CTRL-R`	Help for say <C-R> in insert mode
`:h c_CTRL-R`	Help for say <C-R> in command mode
`:h v_CTRL-V`	Visual mode
`:h tutor`	VIM Tutor
`<C-[>, <C-T>`	Move back & Forth in HELP History
`gvim -h`	VIM Command Line Help
`:cabbrev h tab h`	Open help in a tab

The Vim Annoyances

In order to go into 'normal' mode (where you can enter commands such as 'D' delete the rest of the line) you have to press the [escape] key which is quite difficult to find without looking at the keyboard. But it may be possible to assign another key(s) to perform this job

4.1 Traps

Vim traps

`In regular expressions you must backslash + (match 1`
`In regular expressions you must backslas`
`In regular expressions you must backslash (`
`In regular expressions you must backslash {`
`/fred\+/ matches fred/freddy but`
`/\(fred\)\{2,3}/ note what you have`

Opening Files To Edit

Edit the text file 'poem.txt' in the current folder

```
vim poem.txt
```

Edit the file which was just mentioned on the command line

```
less ~/docs/poem.txt
```

```
vim !$          ~(this opens the file '~/docs/poem.txt' for editing)
```

Choose a file from the current directory to edit

```
:e .            ~(the dot IS necessary)
```

Choose a file to edit

```
:E              ~(no dot is necessary)
```

Edit the file most recently edited in the current session

```
:e #
```

Reload the current file which has been modified in another editor

```
:e!
```

Edit a compressed file (.Z, .gz .bz2)

```
vim file  ~(same as editing a normal file, thanks to the 'gzip' plugin)
```

Edit a script that's somewhere in your path.

```
vim `which <scriptname>`
```

Its possible to edit the standard input stream (usually called 'stdin') This might also be useful for search multiple files.

Edit (a new file) containing lines which contain 'tree' from 'doc.txt'

```
grep tree doc.txt | vim - ~(Im not sure why you would want to do this)
```

exploring files to edit

`:e .`	File explorer
`:Exp(lore)`	File explorer note capital Ex
`:Sex(plore)`	File explorer in split window
`:browse e`	Windows style browser
`:ls`	List of buffers
`:cd ..`	Move to parent directory
`:args`	List of files
`:args *.php`	Open list of files (you need this!)
`:lcd %:p:h`	Change to directory of current file
`:autocmd BufEnter * lcd %:p:h`	Change to directory of current file automatica lly (put in _vimrc)

Section 6

Editing Multiple Files

Edit all filenames ending with '.txt' in the current folder

```
vim *.txt
```

Edit all files on the computer ending with 'tree' in the filename

```
vim $(find / -name "*tree")     ~('find' may take a while to finish)
```

```
vim `find / -name "*tree"`     ~(the same)
```

See all the files currently being edited

```
:args
```

Open a new file and edit it

```
:arge file
```

Close a file ???

```
:bd
```

Edit lots of files which contain a certain string (including subfolders)

```
vim $(grep -ril text *)
```

Substitute yes with no, in all the files, confirming each substitution

```
:argdo %s/yes/no/gc | w   ~(the 'w' saves each file)
```

Write all files in the argument list (all files being edited)

```
:wa
```

Jump to the file (or url) under the cursor

```
gf
```

Open a new file (without closing the last) and edit it

```
arge newfile
```

Toggle between 2 buffers (files) which are open in vim

```
[CTRL] ^  ~(this switches back and forth between 2 files)
```

Sessions (Open a set of files)

```
gvim file1.c file2.c lib/lib.h lib/lib2.h : load files for "session"
```

```
:mksession ~( Make a Session file (default Session.vim))
```

Execute multiple commands on a group of files

```
vim -c "argdo %s/ABC/DEF/ge | update" *.c
```

Remove blocks of text from a series of files

```
vim -c "argdo /begin/+1,/end/-1g/^/d | update" *.c
```

Operate a command over multiple files

`:argdo %s/foo/bar/e`	Operate on all files in :args
`:bufdo %s/foo/bar/e`	
`:windo %s/foo/bar/e`	
`:argdo exe '%!sort'|w!`	Include an external command
`:bufdo exe ''normal @q'' | w`	Perform a recording on open files
`:silent bufdo !zip proj.zip %:p`	Zip all current files

Multiple Files Management (Essential)

`:bn`	Goto next buffer
`:bp`	Goto previous buffer
`:wn`	Save file and move to next (super)
`:wp`	Save file and move to previous
`:bd`	Remove file from buffer list (super)
`:bun`	Buffer unload (remove window but not from list)
`:badd file.c`	File from buffer list
`:b3`	Go to buffer 3
`:b main`	Go to buffer with main in name eg main.c (ultra)
`:sav php.html`	Save current file as php.html and "move" to php.html
`:sav! %<.bak`	Save Current file to alternative extension (old way)
`:sav! %:r.cfm`	Save Current file to alternative extension
`:sav %:s/fred/joe/`	Do a substitute on file name
`:sav %:s/fred/joe/:r.bak2`	Do a substitute on file name & ext.
`:!mv % %:r.bak`	Rename current file (DOS use Rename or DEL)
`:help filename-modifiers`	Get help
`:e!`	Return to unmodified file
`:w c:/aaa/%`	Save file elsewhere
`:e #`	Edit alternative file (also cntrl-ˆ)
`:rew`	Return to beginning of edited files list (:args)
`:brew`	Buffer rewind
`:sp fred.txt`	Open fred.txt into a split
`:sball,:sb`	Split all buffers (super)
`:scrollbind`	In each split window
`:map <F5> :ls<CR>:e #`	Pressing F5 lists all buffer, just type number
`:set hidden`	Allows to change buffer w/o saving current buffer

Vim Tabs

Section 7

Recent versions of vim can use tabs to edit multiple files. I had no idea about this. Tabs are similar to the tabs used in modern web browsers, allowing you to have more that one file open and easily switch between the open files.

View help for using tabs

```
help :tab
help tabpage    ~(better help)
```

Show all open tab files

```
:tabs
```

7.1 Opening Files In Tabs

Open 2 files in vim using tabs

```
vim -p fred.php joe.php
```

Vim 7 tabs

```
:tabe fred.php                ~( open fred.php in a new tab )
```

```
:tab ball                     ~( tab open files )
```

7.2 Closing Tabs

Close the current tab

```
:tabc
```

```
:tabclose  ~(the same)
```

Close all tabs except the current one

```
:tabo
```

Exit all tabs and vim in one go

```
:qa
```

7.3 Switching Tabs

Switch to the next tab. Goes back to the first if on the last tab

```
:tabn
```

```
:tabnext                  ~(the same)
```

```
[control] [pagedown]      ~(the same again)
```

Switch to the previous tab

```
:tabp
```

```
:tabN   ~(the same)
```

Go to the last tab page

```
:tabl
```

Go to the first tab page

```
:tabr
```

```
:tabfir
```

Vim 7 forcing use of tabs from .vimrc

```
:nnoremap gf <C-W>gf
```

```
:cab      e  tabe
```

```
:tab sball ~( retab all files in buffer (repair))
```

7.4 Editing In Multiple Windows

Vim windows are not like 'desktop' windows; they are panes into which the text editing area is split.
Open the file in a split window above the current file

```
:sp file       ~(the original file remains open)
```

```
:split file    ~(the same)
```

Close the current window

```
:q
```

Jump from one window to another

```
[control]+w [control]+w
```

7.5 Transfering Text From One File To Another

Open both files in vim

```
vim fileA fileB
```

Use '[control] ˆ' to switch between the 2 files
Delete a paragraph in one file and paste it in the second

```
dap (press [control] ^) p
```

Copy a paragraph to the clipboard and paste it into the second

```
yap (press [control] ^) p
```

Editing Remote Files

Section 8

Use the netrw plugin, but the version which comes with vim 7.0 for windows does not work well.
To access sourceforge from vim on MS Windows put in the vimrc file

```
let g:netrw_cygwin = 0
```

```
let g:netrw_scp_cmd = "E:\\tools\\putty\\pscp.exe -pw some -batch
```

Edit a file on a sourceforge site with vim

```
:e scp://user,project@web.sourceforge.net/htdocs/file.txt
```

8.1 Editing A Wiki With Lynx And Vim

`http://c2.com/cgi/wiki?UsingWikiWithLynx`
instructions on how to use vim and lynx to edit a wiki.

Start lynx; press "o"; in the "Editor" field write "usr/bin/vim" or the path to the "vim" executable (if you dont know this, type "which vim" on the command line). At the bottom of the page activate the "accept changes" link.
Go to a wiki-page in lynx and "click" on "edit". Go to the edit field and type "control X v" or "control V v". After editing type ":wq" and then click on the save link

8.2 Editing With Encryption

The Vim encryption algorithm is stated to be weak. But it'll keep the honest people out.
Start editing 'file.txt' using encryption

```
vim -x file.txt  ~(a password prompt will appear)
```

Remove the password for a file (and file encryption)

```
set key=
```

Change or add a password for a file

```
:X     ~(an 'enter password' prompt will appear)
```

Saving Files

Get help for filename modifiers (for saving files with new names)

```
:help filename-modifiers
```

Set vim to automatically save files on exit or when switching buffers

```
:set autowriteall
```

Save a copy of the current file named 'copy.txt'

```
:w copy.txt    ~(continue to edit the original file)
```

Save a file with a new name ("save as") and edit the new file

```
:sav copy.txt
```

Save the current file changing the extension to bak

```
:w %:r.bak
```

Save a file you edited in vim without the needed permissions

```
:w !sudo tee %     ~(the file must be reloaded)
```

```
:w !pfexec tee %  ~(another way, but requires 'pfexec' to be installed)
```

9.1 Saving Chunks Of Files

Save highlighted text to a new file

```
[shift]+v , highlight text with j or k, then
```

```
:w newfile
```

Save the text between bookmarks 'a' and 'b' to newfile

```
'a,'b w newfile
```

Save the lines 200-300 to the end of 'textfile' (append the text)

```
:200,300w >> textfile
```

Save the current paragraph to the end of 'textfile'

```
!ap<backspace>w >> textfile     ~(<backspace> means press that key)
```

Write everything from the current line to the bookmark 'x' to 'file.txt'

```
!'x<backspace>w file.txt        ~(the file is created)
```

```
!'x<backspace>w! file.txt       ~(if the file exists, it is overwritten)
```

Filename Modifiers

manipulating file names

`:h filename-modifiers`	Help
`:w %`	Write to current file name
`:w %:r.cfm`	Change file extention to .cfm
`:!echo %:p`	Full path & file name
`:!echo %:p:h`	Full path only
`:!echo %:t`	Filename only
`:reg %`	Display filename
`<C-R>%`	Insert filename (insert mode)
`"%p`	Insert filename (normal mode)
`/<C-R>%`	Search for file name in text

Vim Patterns

Find '+3354.43' and replace at the end of a line

```
:%s/+\(\d\|\.\)*$//gc    ~(looks for a '+' followed by digits or a '.')
```

Ex Commands

Ex commands are derived from an ancient text editor known as 'ed' or 'ex'. These commands are usually written after a ':' colon. These commands are often overlooked when using 'vim' but in some circumstances have a zen-like power to achieve low-keystroke edits. These ex and ed commands are well explained in the classic unix treatise 'The Unix Programming Environment' by Kernighan et al.

Show a help listing of all vim ex commands

```
help holy-grail
```

```
help ex-cmd-index
```

Show help about the ex 'p' (print) command

```
:help :p
```

Show every line in the file which begins with a hash '#'

```
:g/^\s*#/p  ~(the lines are displayed one screen at a time)
```

View the whole file with unprintable characters made visible

```
:%l       ~(good for seeing pesky tab characters)
```

View all blank lines with the invisible characters visible

```
:g/^\s*$/l
```

simple ex commands

`m`	Move text
`d`	Delete text
>	Shift one shiftwidth to the right
<	Shift text one shift-width to the left
`co`	Copy text
`p`	Print (the file is unchanged)
`l`	Print text with invisible characters made visible
`a`	Append text
`i`	Insert text

12.1 Inserting Text With Ex

Insert the word 'tree' after the next line starting with a '#'

```
:/^ *#/a<cr>tree<cr>.<cr>    ~(but doesnt work as a command)
```

12.2 Moving Text With Ex

Move the current line to line 100 (inserting not overwriting)

```
:m100
```

Move the current line to the bottom of the file

```
:m$
```

```
:.m$  ~(the same)
```

Move the line number 315 to the top of the file

```
:315m0
```

Move lines 316 to 330 inclusive to line 10

```
:316,330m10    ~(this inserts the lines at line 10, not overwrites)
```

Move all lines which begin with 'doc' to the bottom of the file

```
:g/^\s*doc/m$
```

Move the next line which starts with a '#' to the end of the file

```
:/^\s*#/m$
```

12.3 Deleting With Ex

Delete all lines which start with '#' (which are script comments)

```
:g/^\s*#/d
```

Delete all empty lines from a file with vim

```
:g!/\S/d
```

Delete all lines from the current line to the end of the file

```
:.,$d
```

Delete the next 10 lines after the current line

```
:.,+10d
```

12.4 Copying With Ex

Copy and insert the current line 4 lines below the current line

```
:co+3
```

```
:.co+3      ~(the same)
```

Copy all lines from the current to the next blank line to line 10

```
:.,/^\s*$/co10      ~(the copied lines are inserted, not overwritten)
```

Copy all lines from the current to the end of the file into the clipboard

```
:,$y
```

```
:.,$y      ~(the same)
```

```
:.,$ya     ~(the same)
```

```
:.,$yank   ~(the same again)
```

Paste the contents of the clipboard after line 100

```
:100pu
```

*Insert the contents of the clipboard after every line which starts with **

```
:g/^\s*\*/pu
```

12.5 Reformatting Lines With Ex

On all lines beginning with '>>' join the next line to it

```
:g/^\s*>>/j
```

Shift one 'shiftwidth' to the right all lines not starting with a '#'

```
:g!/^\s*#/>
```

examples of ex line addresses and ranges

123	Line number 123
/doc	The next line containing the pattern 'doc'
.	The current line
?pat	Previous with pat
$	The last line of the file
.-10	Ten lines before the current line
+	The line after the current line
123,234	From line 123 to line 234
	, the line before the current line
'x	Marked with x
+n	N forward
''	Previous context
%	All lines in the file

Move all comments the top of the file in vim

```
:g:^\s*#.*:m0
```

```
:g/^\s*#.*/m0            ~(the same)
```

```
:g/^[:space:]*#.*/m0     ~(also the same)
```

12.6 The Ex Global Command

Chain an external command

```
:.g/^/ exe ".!sed 's/N/X/'" | s/I/Q/
```

Combining g// with normal mode commands

```
:g/|/norm 2f|r*            ~(replace 2nd | with a star)
```

Send output of previous global command to a new window

```
:nmap <F3>  :redir @a<CR>:g//<CR>:redir END<CR>:new<CR>:put! a<CR><CR>
```

Global combined with substitute (power editing)

```
:'a,'bg/fred/s/joe/susan/gic :  can use memory to extend matching
```

```
:/fred/,/joe/s/fred/joe/gic :  non-line based (ultra)
```

```
:/biz/,/any/g/article/s/wheel/bucket/gic:  non-line based
```

Find fred before beginning search for joe

```
:/fred/;/joe/-2,/sid/+3s/sally/alley/gIC
```

Create a new file for each line of file eg 1.txt,2.txt,3,txt etc

```
:g/^/exe ".w ".line(".").".txt"
```

Save results to a register/paste buffer

```
:g/fred/y A                   : append all lines fred to register a
```

```
:g/fred/y A | :let @*=@a      : put into paste buffer
```

```
:let @a=''|g/Barratt/y A |:let @*=@a
```

Filter lines to a file (file must already exist)

```
:'a,'bg/^Error/ . w >> errors.txt
```

Duplicate every line in a file wrap a print " around each duplicate

```
:g/./yank|put|-1s/'/"/g|s/.*/Print '&'/
```

Replace string with contents of a file, -d deletes the "mark"

```
:g/^MARK$/r tmp.txt | -d
```

Display prettily

```
:g/<pattern>/z#.5           : display with context
```

```
:g/<pattern>/z#.5|echo "=========="  : display beautifully
```

Perform a substitute on every other line

```
:g/^/ if line('.')%2|s/^/zz /
```

Match all lines containing "somestr" between markers a & b   Copy after line containing "otherstr"

```
:'a,'bg/somestr/co/otherstr/ : co(py) or mo(ve)
```

As above but also do a substitution

```
:'a,'bg/str1/s/str1/&&&/|mo/str2/
```

```
:%norm jdd                  : delete every other line
```

Incrementing numbers (type <c-a> as 5 characters)

```
:.,$g/^\d/exe "norm! \<c-a>": increment numbers
```

```
:'a,'bg/\d\+/norm! ^A       : increment numbers
```

global command display

`:g/gladiolli/#`	Display with line numbers (YOU WANT THIS!)
`:g/fred.*joe.*dick/`	Display all lines fred,joe & dick
`:g/`$<fred$ >$`/`	Display all lines fred but not freddy
`:g/^\s*$/d`	Delete all blank lines
`:g!/^dd/d`	Delete lines not containing string
`:v/^dd/d`	Delete lines not containing string
`:g/joe/,/fred/d`	Not line based (very powerfull)
`:g/fred/,/joe/j`	Join Lines between 'fred' and 'joe'
`:g/-------/.-10,.d`	Delete string & 10 previous lines
`:g/{/ ,/}/- s/\n\+/\r/g`	Delete empty lines but only between {...}
`:v/\S/d`	Delete empty lines (and blank lines ie whitespace)
`:v/./,/./-j`	Compress empty lines
`:g/^$/,/./-j`	Compress empty lines
`:g/<input|<form/p`	ORing
`:g/^/put_`	Double space file (pu = put)
`:g/^/m0`	Reverse file (m = move)
`:g/^/m$`	No effect!
`:'a,'bg/^/m'b`	Reverse a section a to b
`:g/^/t.`	Duplicate every line
`:g/fred/t$`	Copy(transfer) lines matching fred to EOF
`:g/stage/t'a`	Copy (transfer) lines matching stage to marker a (cannot use .)
`:g/^Chapter/t.|s/./-/g`	Automatically underline selecting headings
`:g/\(^I[^^I]*\)\{80}/d`	Delete all lines containing at least 80 tabs

12.7 The Command History

some command history commands

	`:ju(mps)`	List of your movements
:help jump-motions	`:history`	List of all your commands
	`:his c`	Commandline history
	`:his s`	Search history
	`q/`	Search history Window (puts you in full edit mode) (exit CTRL-C)
	`q:`	Commandline history Window (to full edit mode) (exit CTRL-C)
	`:<C-F>`	History Window (exit CTRL-C)

Searching For Text

Search for the word 'big'

```
/big
```

Repeat the last search

```
n
```

Repeat the last search in the opposite direction

```
N
```

searching and moving

`/joe/e`	Cursor set to End of match
`/joe/e+1`	Cursor set to End of match plus 1
`/joe/s-2`	Cursor set to Start of match minus 2
`/joe/+3`	Find joe move cursor 3 lines down
`/^joe.*fred.*bill/`	Find joe AND fred AND Bill (Joe at start of line)
`/^[A-J]/`	Search for lines beginning with one or more A-J
`/begin\_.*end`	Search over possible multiple lines
`/fred\_s*joe/`	Any whitespace including newline
`/fred|joe`	Search for FRED OR JOE
`/.*fred\&.*joe`	Search for FRED AND JOE in any ORDER!
/$<*fred*$ >/	Search for fred but not alfred or frederick
/$<*ndndndnd*$ >	Search for exactly 4 digit numbers
`/\D\d\d\d\d\D`	Search for exactly 4 digit numbers
/$<*ndn*{4}$ >	Same thing
`/\([^0-9]|^\)%.*%`	Search for absence of a digit or beginning of line

Summary of repeated searches

;	F, t, F or T
,	F, t, F or T in opposite direction
`n`	Last / or ? search
`N`	Last / or ? search in opposite direction

Match word boundaries

```
/\<all\>/      ~(matches 'all' but not 'balls')
```

Find the next, previous word under the cursor

```
g*  g#
```

Make all searches case insensitive

```
:set ignorecase
```

```
:set ic        ~(the same)
```

(this affects substitutions with s/// as well as searches)

Make all searches case sensitive (the default)

```
:set noignorecase
```

VIM has an its own 'grep' command (different from the bash one)

```
:grep some_keyword *.c   ~(get list of all c-files containing keyword)
```

```
:cn                      ~(go to next occurrence)
```

Multiple file search

```
:bufdo /searchstr/  ~( use :rewind to recommence search )
```

Multiple file search better but cheating

```
:bufdo %s/searchstr/&/gic ~( say n and then a to stop)
```

finding empty lines

/^\n\{3}	Find 3 empty lines
/^str.*\nstr	Find 2 successive lines starting with str
/\(^str.*\n\)\{2}	Find 2 successive lines starting with str

Using regular expression memory (back references) in a search

```
/\(fred\).*\(joe\).*\2.*\1
```

Repeating the Regexp (rather than what the Regexp finds)

```
/^\([^,]*,\)\{8}
```

How to search for a URL without backslashing

```
?http://www.vim.org/ ~( (first) search BACKWARDS!!! clever huh!)
```

Specify what you are NOT searching for (vowels)

```
/\c\v([^aeiou]&\a){4}    ~( search for 4 consecutive consonants)
```

```
/\%>20l\%<30lgoat        ~( Search for goat between lines 20 and 30)
```

```
/^.\{-}home.\{-}\zshome/e ~( match only the 2nd occurence in a line of
  ⇒ "home" *)
```

```
:%s/home.\{-}\zshome/alone ~( Substitute only the occurrence of home in
  ⇒ any line)
```

Find str but not on lines containing tongue

```
^\(.*tongue.*\)\@!.*nose.*$
```

```
\v^((tongue)@!.)*nose((tongue)@!.)*$
```

```
.*nose.*\&^\%(\%(tongue\)\@!.\)*$
```

```
:v/tongue/s/nose/&/gic
```

Section 14

Search And Replace

These these character classes dont work within [...] character classes.
Using a character class such as '\a' for an alphabetc character may be better than using the character class '[A-Za-z] because the \w class may also handle international word characters, such as n tilde (the spanish enye) for example.

substitution

:%s/fred/joe/igc	General substitute command
:%s//joe/igc	Substitute what you last searched for
:%s/~/sue/igc	Substitute your last replacement string
:%s/\r//g	Delete DOS returns ^M

deleting empty lines

:%s/^\n\{3}//	Delete blocks of 3 empty lines
:%s/^\n\+/\r/	Compressing empty lines
:%s#<[^>]\+>##g	Delete html tags, leave text (non-greedy)
:%s#<_.\{-1,}>##g	Delete html tags possibly multi-line (non-greedy)
:%s#.*\(\d\+hours\).*#\1#	Delete all but memorised string (\1)

VIM Power Substitute

```
:'a,'bg/fred/s/dick/joe/ ~( VERY USEFUL)
```

See a long descrition of patterns usable with s///

```
:help pattern
```

Before doing a search and replace on an important file it is a good idea to use the 'c' modifier to the 's' command, to check what changes will be made

```
:%s/\s\+$//gc ~(remove trailing spaces, with confirmation each time)
```

```
:%s/\s\+$//g  ~(remove trailing whitespace, no confirmation)
```

Replace the word tall with small in the file, asking for confirmation

```
:%s/\<tall\>/small/gc ~(this will replace 'big' but not 'stall' etc)
```

```
:%s/\<tall\>/small/g  ~(the same but with no confirmation prompt)
```

Multiple commands on one line

```
:%s/\f\+\.gif\>/\r&\r/g | v/\.gif$/d | %s/gif/jpg/
```

```
:%s/a/but/gie|:update|:next : then use @: to repeat
```

Remove trailing whitespace on every line

```
:%s/\s\+*$//g
```

```
:1,$s/\s\+*$//g              ~(the same)
```

```
:1,$s/[[:space:]]\+*$//g    ~(the same again)
```

```
:1,$s/[[:blank:]]\+*$//g    ~(almost the same again)
```

Turn DOS ˆM line breaks into real line breaks

```
:%s/\r/\r/g
```

Replace all colon ':' characters with semicolons ';'

```
:%s,:,;,g       ~(this shows that any char can be used as the delimiter)
```

```
:%s/:/;/g       ~(the same)
```

Delete all empty lines from a file with vim

```
:g!/\S/d
```

Make multiple consecutive blank lines into only one

```
:g/^\s*$/,/\S/-j|s/.*//
```

Insert the line number at the beginning of each line

```
:%s/^/\=line('.').' '
```

Move all comments the top of the file in vim

```
:g:^\s*#.*:m0
```

```
:g/^\s*#.*/m0    ~(the same)
```

Get rid of tab characters in the whole file

```
:%! expand
```

Replace all slash '/' characters with bar '—' characters

```
:%s@/@|@g
```

Delete all lines which contain the text 'big'

```
:g/big/d
```

```
:g/RE/cmd      ~(the general form of the command)
```

Memory

```
:%s#.*\(tbl_\w\+\).*#\1# "( produce a list of all strings tbl_*)
```

```
:s/\(.*\):\(.*\)/\2 : \1/ "( reverse fields separated by :)
```

```
:%s/^\(.*\)\n\1$/\1/ "( delete duplicate lines)
```

Non-greedy matching \{-}

```
:%s/^.\{-}pdf/new.pdf/ "( delete to 1st pdf only)
```

Use of optional atom \?

```
:%s#\<[zy]\?tbl_[a-z_]\+\>#\L&#gc "( lowercase with optional leading
  ⇒ characters)
```

Over possibly many lines

```
:%s/<!--\_.\{-}-->// "( delete possibly multi-line comments)
```

```
:help /\{-} "( help non-greedy)
```

Substitute using a register

```
:s/fred/<c-r>a/g "( sub "fred" with contents of register "a")
```

```
:s/fred/<c-r>asome_text<c-r>s/g
```

```
:s/fred/\=@a/g "( better alternative as register not displayed)
```

Search for alternate patterns 'goat' or 'cow' and replace with 'sheep'

```
:%s/goat\|cow/sheep/gc
```

Insert a blank line every 5 lines

```
:%s/\v(.*\n){5}/&\r/    "(???)
```

Using a vim function within a substitution

```
:s/<today>/\=strftime("%c")/    "(insert a date instead of <today>)
```

special character classes for vim patterns

`\s`	Whitespace character: <Space> and <Tab>
`\S`	Non-whitespace character; opposite of \s
`\d`	Digit: eg [0-9]
`\D`	Non-digit: eg [ˆ0-9]
`\x`	Hex digit: eg [0-9A-Fa-f]
`\X`	Non-hex digit: eg [ˆ0-9A-Fa-f]
`\o`	Octal digit: eg
`\O`	Non-octal digit: eg [ˆ0-7]
`\w`	Word character: eg [0-9A-Za-z_]
`\W`	Non-word character: eg [ˆ0-9A-Za-z_]
`\h`	Head of word character: eg [A-Za-z_]
`\H`	Non-head of word character: eg [ˆA-Za-z_]
`\a`	Alphabetic character: eg [A-Za-z]
`\A`	Non-alphabetic character: eg [ˆA-Za-z]
`\l`	Lowercase character: eg [a-z]
`\L`	Non-lowercase character: eg [ˆa-z]
`\u`	Uppercase character: eg [A-Z]
`\U`	Non-uppercase character: eg [ˆA-Z]
`\i`	Identifier character (see 'isident' option)
`\I`	Like "\i", but excluding digits
`\k`	Keyword character (see 'iskeyword' option)
`\K`	Like "\k", but excluding digits
`\f`	File name character (see 'isfname' option)
`\F`	Like "\f", but excluding digits
`\p`	Printable character (see 'isprint' option)
`\P`	Like "\p", but excluding digits
`\_x`	Where x is any of the above character classes with end-of-line included

14.1 Uppercase And Lowercase

Warning: :s///... etc is affected by the setting of 'ignorecase' in the vim settings. Type :set and if 'ignorecase' appears the all searches and replaces are going to be case insensitive.

Turn off and on case insensitivity in searches with :s///

```
:set ic
```

```
:set ignorecase ~(the same)
```

```
:set noic
```

Show lines which contain only uppercase letters or spaces

```
g/^\(\u\| \)\+$/p
```

Make all text in a file lowercase

```
:%s/[A-Z]/\L&/g    ~(only works for 'ascii' text)
```

```
:%s/\a/\L&/g       ~(the same, but should work for international text)
```

Make all words 'capital case', that is with the first letter in uppercase

```
:%s/\<\a\+/\u&/gc     ~(this turns 'hello you' into 'Hello You')
```

Note that \u \U \l \L have different meanings depending on context. In the right hand side of a s/// expression they mean ...

escape codes

`\l`	Turn only the next character in the match to lower case
`\L`	Turn all the characters in the match to lower case
`\u`	Convert only the first character in the match to upper case
`\u`	Convert all the characters in the match to upper case

Changing Case

guu	Lowercase line
gUU	Uppercase line
Vu	Lowercase line
VU	Uppercase line
g~~	Flip case line
vEU	Upper Case Word
vE~	Flip Case Word
ggguG	Lowercase entire file

Title-ise Visually Selected Text (map for .vimrc)

```
vmap ,c :s/\<\(.\)\(\k*\)\>/\u\1\L\2/g<CR>
```

Title-ise a line

```
nmap ,t :s/.*/\L&/<bar>:s/\<./\u&/g<cr>
```

Uppercase first letter of sentences

```
:%s/[.!?]\_s\+\a/\U&\E/g
```

Text Indentation

Section 15

Vim has various options and commands for automatically indenting (that is adding spaces or tabs at the beginning of a line) as you type or for using with commands like 'gqap' (reformat a paragraph). In fact, there are so many indentation options, that one can sometimes become frustated when vim constantly indents lines in an unexpected way.

See the value of the current shift-width (used to in/decrease indentation)

```
:set sw
```

```
:set shiftwidth   ~(the same)
```

View help for the 'shiftwidth' configuration option

```
:help 'sw'
```

Convert all tabs in a document to an equivalent number of spaces

```
:retab   ~(this may change indentation)
```

15.1 Automatic Indentation

View help for the 'autoindent' configuration option

```
:help 'ai'
```

Make the next line after the current line have the same indentation

```
:set ai           ~(when you are typing, the next indent is 'aligned')
```

```
:set autoindent   ~(the same)
```

Make sure that vim uses spaces and not annoying tabs for indentations

```
:set expandtab
```

```
:set et           ~(the same)
```

Reformat a paragraph while preserving the current indentation of the text

```
:set ai [esc] gqap :set noai  ~(or just leave 'autoindent' set to on)
```

(reformatting means making each line of a similar length)

Turn off automatic indentation of the next line when typing

```
:set noai
```

15.2 Manual Indentation

Increase the indentation for the current paragraph by one shift-width

```
>ap  ~(you can repeat this operation by just typing .)
```

Decrease the indentation for this paragraph and the next 3 paragraphs as well

```
<4ap ~(the indent is decreased by a 'shiftwidth')
```

Decease the indentation by 1 shiftwidth for all lines above the current

```
<gg
```

Increase the shift-width of the current line by one 'shiftwidth'

```
>>
```

some indentation options

`shiftwidth`	(sw) the number of tabs or spaces for an increase or decrease
`cindent`	Automatically indent c code files
`smartindent`	Automatically indent other types of files
`autoindent`	Align the indent of the next line to the current
`expandtab`	Use spaces not tabs for indentations

indentation commands

`retab`	Replace all tabs in the file with an equivalent number of spaces
>>	Shift the current line
>	Shift a range of lines

15.3 My Favourite Settings

Set the shiftwidth to 1, using spaces not tabs, automatically align next line

```
:set shiftwidth=1 expandtab autoindent
```

```
:set sw=1 et ai    ~(the same, but terse)
```

Section 16

Working With Text Data

Duplicating the columns (fields) of a space delimited file

```
:%s= [^ ]\+$=&&= - duplicate end column
```

```
:%s= \f\+$=&&= - same thing
```

```
:%s= \S\+$=&& - usually the same
```

Working with Columns sub any str1 in col3

```
:%s:\(\(\w\+\s\+\)\{2}\)str1:\1str2:
```

Swapping first & last column (4 columns)

```
:%s:\(\w\+\)\(.*\s\+\)\(\w\+\)$:\3\2\1:
```

Format a mysql query

```
:%s#\<from\>\|\<where\>\|\<left join\>\|\<\inner join\>#\r&#g
```

Substitute string in column 30

```
:%s/^\(.\{30\}\)xx/\1yy/
```

Decrement numbers by 3

```
:%s/\d\+/\=(submatch(0)-3)/
```

Increment numbers by 6 on certain lines only

```
:g/loc\|function/s/\d/\=submatch(0)+6/
```

Better

```
:%s#txtdev\zs\d#\=submatch(0)+1#g
```

```
:h /\zs
```

Increment only numbers gg\d\d by 6 (another way)

```
:%s/\(gg\)\@<=\d\+/\=submatch(0)+6/
```

```
:h zero-width
```

Rename a string with an incrementing number

```
:let i=10 | 'a,'bg/Abc/s/yy/\=i/ |let i=i+1 # convert yy to 10,11,12
 ⇒ etc
```

As above but more precise

```
:let i=10 | 'a,'bg/Abc/s/xx\zsyy\ze/\=i/ |let i=i+1 # convert xxyy to
 ⇒ xx11,xx12,
```

xx13

16.1 Sorting Text

Sorting with external sort

`:%!sort -u`	Sort the whole file using external sort
`:'a,'b!sort -u`	Sort all text between bookmarks 'a' and 'b'
`!} sort -u`	Sorts paragraph (note normal mode!!)
`!2} sort -u`	Sort 2 paragraphs
`:g/^$/;,/^$/-1!sort`	Sort each block (note the crucial ;)

Sorting with internal sort

```
:sort /.*\%2v/   : sort all lines on second column
```

16.2 Csv Data Files

Highlight a particular csv column (put in .vimrc)

```
    function! CSVH(x)
        execute 'match Keyword /^\([^,]*,\)\{'.a:x.'}\zs[^,]*/'
        execute 'normal ^'.a:x.'f,'
    endfunction
    command! -nargs=1 Csv :call CSVH(<args>)
```

Columnise a csv file for display only as may crop wide columns

```
    :let width = 20
    :let fill=' ' | while strlen(fill) < width | let fill=fill.fill |
     ⇒ endwhile
    :%s/\([^;]*\);\=/\=strpart(submatch(1).fill, 0, width)/ge
    :%s/\s\+$//ge
```

Call with

```
:Csv 5   ~( highlight fifth column)
```

Section 17

Filtering Text

Remove all consecutive blank lines in the current file

```
:%!cat -s     ~(type 'u' undo to see how many lines would be deleted)
```

```
:1,$!cat -s    ~(the same)
```

```
gg!G cat -s    ~(the same, using motions instead of ranges)
```

Remove all extra spaces (consecutive spaces) in the current file

```
:%! tr -s ' '
```

Reformat all lines after the current one, wrapping lines

```
:.,$!par   ~(par knows how to reformat 'bash comments' for example)
```

Insert the result of the 'ls' bash command at the cursor position

```
:r !ls | tr '\n' ' '
```

Reformatting Text

Section 18

In this context 'formatting' refers to the length of each text line and the amount of indentation of each line. The vim 'gq' command takes into account the current screen size and font size when wrapping the lines.
For reformatting source code see: "working with source code"

Remove extra spaces between words in the current paragraph

```
!ap tr -s ' '
```

```
!ap tr -s '[:blank:]'   ~(the same but handles tabs)
```

```
!ap s/[ ]\+/[ ]/g       ~(the same but harder to type)
```

Format the next paragraph of text (fill and break lines)

```
!}fmt       ~(uses external program 'fmt')
```

```
!}par       ~(the same, but 'par' can do more than format)
```

Turn on autoindenting

```
:set ai     ~(useful with 'gq', since it preserves the current indent)
```

Format the until the end of the paragraph with the vim formatter

```
gq}         ~(formats only after the cursor)
```

Format the current paragraph with the vim formatter

```
gqap              ~(this also formats before the cursor)
```

Format the text until the bookmark 'a'

```
gq'a
```

Format the entire file

```
gggqG
```

Centre align the whole file with a width of 40

```
:%center 40
```

```
:%left 40         ~(left alignment)
```

```
:%right 40
```

Center the current line

```
:ce
```

Is your Text File jumbled onto one line? use following

`:%s/\r/\r/g`	Turn DOS returns ^M into real returns
`:%s= *$==`	Delete end of line blanks
`:%s= \+$==`	Same thing
`:%s#\s*\r\?$##`	Clean both trailing spaces AND DOS returns
`:%s#\s*\r*$##`	Same thing

Moving Around

This section is about various movement commands available in Vim. Movement commands can follow other command (such as deleting, or copying) to provide a range for that command

the basic movement

`j`	One line down
`k`	One line up
`h`	One character left
`l`	Once character right

Go to the beginning of the file

```
gg
```

Go to the end of the file

```
G
```

Go to line 33

```
33G
```

Jump to the last modification line

```
'.
```

Jump back to where you just were (before jumping)

```
''      ~(i find this very handy)
```

Cycle through recent modifications, forwards and backwards

```
g; g,
```

Set a bookmark named 'a' at the current position

```
ma        ~(or mb, mc, etc)
```

Jump to the position indicated by the bookmark a

```
'a
```

Go to the end of the paragraph

```
}
```

Go to the next instance of the character 'g' on the line

```
fg
```

Repeat this operation

```
;
```

Go BACK to the next instance of the character 'g' on the line

```
Fg
```

Move to the matching parenthesis {,[,(

```
%
```

Scroll scroll the current line to the top of the window

```
zt
```

Jump to the next instance of the word under the cursor

```
*
```

Return to last edit position (You want this!)

```
autocmd BufReadPost *
     \ if line("'\"") > 0 && line("'\"") <= line("$") |
     \   exe "normal! g`\"" |
     \ endif
```

Refering To Chunks Of Text

Chunks of text can be referred to by line numbers, motion commands, or 'text objects'.

Delete lines 4 to 9

```
:4,9d
```

Delete until the next word 'stop' (not including the word)

```
d/stop
```

20.1 Text Objects

Text objects are very useful because they allow you to refer to a chunk of text without knowing the line numbers, or having to position the cursor at the start of the chunk.

Get help about the available 'text objects'

```
:help text-objects
```

Delete the current paragraph (before and after the cursor)

```
dap            ~('ap' refers to a paragraph)
```

Change the current sentence (before and after the cursor)

```
cas
```

Change the current sentence and the next one

```
2cas
```

Change the current single quotes quotation

```
ci'            ~(leaves the quotation marks)
```

```
ca'            ~(deletes the quotation marks)
```

available textobjects

`aw`	A word
`aW`	A long word (including punctuation)
`iW`	A long word without white space
`as`	A sentence
`ap`	A sentence
`ip`	A sentence, without the blank lines
`a>`	A angle braket block
`a} or aB`	A brace block
`i} or iB`	A brace block
`a]`	A square bracket block
`a)`	A bracket block
`i)`	A bracket block not including the brackets
`a" or a'`	A quotation (one line only)
`i" or i'`	A quotation not including the quotes

text objects

`:h text-objects`	Get help
`daW`	Delete contiguous non whitespace
`di< yi< ci<`	Delete/Yank/Change HTML tag contents
`da< ya< ca<`	Delete/Yank/Change whole HTML tag
`dat dit`	Delete HTML tag pair
`diB daB`	Empty a function {}
`das`	Delete a sentence

`http://vim.wikia.com/wiki/Indent_text_object`

How to define a new text object

Inserting

Insert the current date in a new line

```
:r !date
```

```
:.r !date      ~(the same)
```

```
:read !date    ~(the same)
```

Insert the current date in at the end of the document

```
:$r !date
```

Insert the current date after the next blank line

```
:/^ *$/r !date      ~(doesnt work with tab characters)
```

```
:/^\s*$/r !date     ~(handles any kind of whitespace)
```

```
:/^[:space:]*$/r !date       ~(the same)
```

```
:/^[[:space:]]*$/r !date     ~(the same, again)
```

Insert todays date after every blank line in the file

```
:g/^\s*/r !date     ~(this seems a bit slow)
```

Insert all the text files in the current directory

```
:r *.txt                    ~(but why would you do this?!)
```

```
cat *.txt >> big.txt    ~(similar but not the same, from bash)
```

Insert a file listing of the current folder at the top of the file

```
:1r !ls
```

The following doesnt work because vim is running as a different user

```
:r !history
```

Insert after the cursor, lines 200-300 of the file 'tree.txt'

```
:r !sed -n 200,300p tree.txt
```

Insert an attachment within an email message in vim

```
:r !uuencode binfile binfile
```

Inserting From The Web

Insert the contents of a webpage as plain text at the end of the file

```
:$r !lynx -dump http://rayninfo.co.uk/vimtips.html
```

22.1 Inserting A File Into Itself

Double the current file (append itself to the end of itself)

```
:$r !cat %
```

Insert all comment lines in the file after the cursor

```
:r !grep '^ *#' %
```

```
:r !cat % | grep '^ *#'      ~(the same)
```

In the command above, lines beginning with the '#' comment character are copied in the same order to the current cursor position. Exactly why you would want to do this remain unclear but I thought that it was interesting.

22.2 Random Text

Make a command 'Rand' which inserts some random words in the document

```
:com! -nargs=1 Rand r !shuf -n <args> /usr/share/dict/words | tr '\n' '
 ⇒  '
```

This command can be executed with 'Rand 200' which inserts 200 random words for the dictionary file into the current file after the cursor.

Deleting

Section 23

Delete the current paragraph (deletes before and after the cursor)

```
dap            ~('ap' refers to a paragraph)
```

Delete the paragraph after the cursor

```
d}
```

Delete the current line and the next line

```
dj
```

Delete the current line and the line above it

```
dk
```

Delete the rest of the line after the cursor

```
D
```

Delete several lines, using visual mode

```
[shift]+v, move down (j) or up (k) to select text, press d
```

Delete up to the line number 234

```
d234G
```

Delete from current line until the bookmark 'c'

```
d'c
```

Editing

Section 24

Change the rest of the line after the cursor

```
C
```

Shift the next paragraph right

```
>}             ~(change the shift width with :set sw=4)
```

Lower case a line

```
guu
```

Upper case a line

```
gUU
```

Make the current paragraph uppercase

```
gUap
```

Make the next word uppercase

```
gUw
```

24.1 Repeating Edits

Summary of editing repeats

`.`	Last edit (magic dot)
`:&`	Last substitute
`:%&`	Last substitute every line
`:%&gic`	Last substitute every line confirm
`g%`	Normal mode repeat last substitute
`g&`	Last substitute on all lines
`@@`	Last recording
`@:`	Last command-mode command
`:!!`	Last :! command
`:~`	Last substitute
`:help repeating`	

24.2 The Edit Changes List

managing changes

`'.`	Jump to last modification line (SUPER)
`` `. ``	Jump to exact spot in last modification line
`g;`	Cycle thru recent changes (oldest first)
`g,`	Reverse direction
`:changes`	
`:h changelist`	Help for above
`<C-O>`	Retrace your movements in file (starting from most recent)
`<C-I>`	Retrace your movements in file (reverse direction)

Cut Copy And Paste

Section 25

In vim, copying text to the clipboard is known as “yanking”. This text can then be pasted into the text with the ”p” command.

Copy the current line to the clipboard.

```
yy
```

```
""yy   ~(the same, but unnecessary)
```

Paste text into the file after the current position

```
p
```

```
""p    ~(the same, naming explicitly the default register "")
```

Paste text into the file before the current position

```
P
```

Paste text after current position and adjust the indent

```
]p
```

Copy to the clipboard all text until the next occurrence of “stop”

```
y/stop  ~(the word "stop" is not included in the copied text)
```

Copy all text of the current line until the next ”:” character

```
yf:
```

Registers

Section 26

Registers are buffers or ’clipboards’ which can hold text or commands. A register can be pasted or executed. Registers have one letter names from a to z. A macro can be recorded and and stored in a register with ’q’;

Display contents of all registers

```
:reg
```

```
:di   ~(the same ??)
```

Display contents of register b

```
:reg b
```

Copy the current line into register 'd'

```
"dyy
```

Delete the current line and store in register d and the default reg.

```
"cdd
```

Store text that is to be changed or deleted in register a

```
"act<   ~(change until the '<' character)
```

Change the word under the cursor and store a copy of the old one in reg b

```
"bcaw    ~(a copy of the old word is also stored in the default register
  ⇒ )
```

Strip leading '>>' from a line, copy it into a register and execute it

```
s/[ ]*>>// | normal u"ayy@a
```

(useful for executing shell commands in the document which have some prefix)

Copy (yank) from the cursor to the second instance of ' on the line

```
y2f'    ~(the text is copied in the default register "")
```

Copy all text on the line up to the next quote ' and put it in register 'k'

```
"kyf'
```

List your Registers

`:reg`	Display contents of all registers
`:reg a`	Display content of register a
`:reg 12a`	Display content of registers 1,2 & a
``5p``	Retrieve 5th "ring"
`"1p....`	Retrieve numeric registers one by one
`:let @y='yy@"'`	Pre-loading registers (put in .vimrc)
`qqq`	Empty register "q"
`qaq`	Empty register "a"
`:reg .-/%:*"`	The seven special registers
`:reg 0`	What you last yanked, not affected by a delete
`"_dd`	Delete to blackhole register "_ , don't affect any register

manipulating registers

`:let @a=@_`	Clear register a
`:let @a=""`	Clear register a
`:let @a=@"`	Save unnamed register
`:let @*=@a`	Copy register a to paste buffer
`:let @*=@:`	Copy last command to paste buffer
`:let @*=@/`	Copy last search to paste buffer
`:let @*=@%`	Copy current filename to paste buffer

26.1 Appending Text To The Registers

Append the current line to the register 'a'

```
"Ayy
```

Delete the current line and append it to the register 'x'

```
"Xdd
```

Change the word under the cursor and append it to the register 'q'

```
"Qcaw
```

26.2 Pasting Registers

Paste the contents of the register 'k' into the document after the cursor

```
"kp
```

```
:put k    ~(the same)
```

```
:pu k     ~(the same)
```

Paste the current line into the default register and execute it

```
yy@"  ~(watch out this could have very strange results- user 'u' to
  ⇒ undo)
```

Paste the contents of register 'b' into the document after the cursor

```
"bp
```

Paste the contents of register 'j' into the document before the cursor

```
"jP
```

26.3 The Numeric Registers

The numeric registers ("0 "1 "2 etc) hold the text which was previously in the default register ("") before being replaced with a new yank into the default register.

Paste what was formerly in the default register into the document

```
"0p
```

26.4 Executing Registers

The registers may also be executed, if they contain a series of valid vim or shell commands. One can record a macro into a register and then execute it.

Record a macro to be stored in the register 'y'

```
qy    ~(type 'q' to stop recording the macro)
```

Execute the contents of the register 'y'

```
@y
```

Copy the current line into a register and execute it

```
"ayy@a
```

Section 27

Text Information

Show information for the character under the cursor

```
ga
```

Count words in a text file

```
g<C-G>
```

Section 28

Spell Checking

Get help on spell checking

```
:help spell
```

Turn on spell checking with the default language

```
:set spell
```

Turn on spell checking, with US english

```
:setlocal spell spelllang=en_us
```

```
:setlocal spell spelllang=en_gb    ~(United Kingdom spelling)
```

Find out what language is currently being spell checked

```
:set spelllang
```

Turn off spell checking

```
:set nospell
```

```
:setlocal nospell    ~(not really sure what the diff is)
```

Move to the next badly spelled word

```
]s
```

Add the current word to personal dictionary as good

```
zg
```

Suggest alternatives to a badly spelled word

```
z=
```

Configuring Vim

Section 29

Configurations made with :set ... only apply for the current vim editing session. To make them permanent these set's need to be placed one of the vim configuration files (either for the current user or for all users).

View help on setting

```
:help options
```

See a brief description of all vim configuration options

```
:help option-list   ~(this also show the long and short name of an
  ⇒ option)
```

See a very detailed description of every single configuration option

```
:help option-summary
```

See detailed help for the 'autoindent' configuration option

```
:help 'autoindent'    ~(note the quote characters. they are often needed
  ⇒ )
```

```
:help 'ai'            ~(the same)
```

```
:help 'noai'          ~(the same again)
```

```
:help ai              ~(also works, but sometimes doesnt)
```

See detailed help for the 'cindent' configuration option

```
:help 'cindent'
```

```
:help 'cin'     ~(the same)
```

```
:help 'nocin'   ~(the same)
```

```
:help nocin     ~(also works)
```

```
:help cindent   ~(No!!: shows documentation for the 'cindent' function)
```

See what settings for configuration options are currently in force

```
:set       ~(only shows the modified settings)
```

```
:set all   ~(show all settings, modified and unmodified)
```

See what the current value of the 'shiftwidth' option is

```
:set sw
```

See what the value for the autoindent option value is

```
:set ai?        ~(note: ':set ai' doesnt work, because it turns on
 ⇒ autoindent)
```

```
:set autoindent?   ~(the same)
```

Dont insert any tabs

```
:set expandtab
```

```
set expandtab [in the 'vimrc' file] ~(note there is no colon ':')
```

Set more than one option at a time

```
:set expandtab autoindent  ~(turns tabs into spaces, and auto-indents
 ⇒ text)
```

Put commands and configurations in the file, or somewhere else

```
~/.vimrc  ~(this only affects one user, not everybody)
```

Edit the global vim configuration file with the needed permissions

```
!sudo vim /etc/vim/vimrc
```

Make a command to edit the global vim configuration file (but not reload it)

```
com! Vimrc !sudo vim /etc/vim/vimrc      ~(executed with ':Vimrc')
```

```
command! Vimrc !sudo vim /etc/vim/vimrc  ~(the same)
```

A command 'So' to reload the vimrc file easily

```
command! So so /etc/vim/vimrc   ~(execute with ':So')
```

Reload the vim configuration file

```
:so $MYVIMRC
```

```
:source $MYVIMRC   ~(the same)
```

```
:source ~/.vimrc   ~(reload the users personal config file)
```

Find the location of the start up configuration files

```
:version
```

Change the number of spaces for a shift to 4 (with '>>' for example)

```
:set sw=4
```

```
:set shiftwidth=4     ~(the same)
```

Show line numbers for the whole file

```
:set number
```

```
:set nu       ~(the same)
```

Show tabs and ends of lines

```
:set list      ~(':set nolist' to turn it off)
```

Show partial commands in the status bar

```
:set showcmd
```

Dont show the matching parenthesis when moving around the file

```
:NoMatchParen
```

(for some reason, this is not working in my "vimrc" file)

DO show the matching parenthesis when moving around the file

```
:DoMatchParen
```

see also :set showmatch

Some mappings to make editing the .vimrc file easier

```
:nmap ,s :source $MYVIMRC
```

```
:nmap ,v :arge $MYVIMRC
```

```
:command Vimrc arge $MYVIMRC
```

Save your sessions in vim to resume later

```
:mksession! <filename>
```

29.1 The Configuration File

In vim, the configuration file (or files) is called 'vimrc'. There maybe several configuration files, one for the current user, and one global file for all users.

_vimrc configuration file essentials

`:set incsearch`	Jumps to search word as you type (annoying but excellent)
`:set wildignore=*.o,*.obj,*.bak,*.exe`	Tab complete now ignores these
`:set shiftwidth=3`	For shift/tabbing
`:set vb t_vb=".`	Set silent (no beep)
`:set browsedir=buffer`	Maki GUI File Open use current directory

Find out where your vim configuration file is

```
:echo $MYVIMRC
```

```
:!find / -name '*vimrc*'  ~(if the above produces no information)
```

Folding Text

Section 30

http://www.linux.com/archive/articles/114138

'Folds' in text are a way to hide sections of text in a big document to make the document easier to read. When the text is hidden the fold is said to be 'closed' and when the text in the fold is visible then it is 'open'.

30.1 Getting Help

View some help for folding

```
:help folding
```

30.2 Basic Usage

Create a fold which hides the next 2 lines

```
zf2j  ~(the fold is created and the text is hidden)
```

Open the current fold (where the cursor is)

```
zo   ~(the fold is still there and can be closed again with 'zc')
```

Open all the nested folds on the current line

```
zO
```

Make the current paragraph a fold

```
zfap
```

Open all folds in a document

```
zR
```

Create a fold from line 1 to 10

```
:1,10 fo
```

Close a fold (hide the contents of the fold)

```
zc
```

Delete the fold at the cursor

```
zd
```

Close all folds in the document

```
zM
```

Delete all folds

```
zE
```

Delete all the text in a fold (when the fold is closed)

```
dd
```

folding : hide sections to allow easier comparisons

`zf}`	Fold paragraph using motion
`v}zf`	Fold paragraph using visual
`zf'a`	Fold to mark
`zo`	Open fold
`zc`	Re-close fold

30.3 Moving Between Folds

Move to the previous fold (even if it is open)

```
zk
```

Move to the next fold

```
zj
```

30.4 Creating Folds Automatically

Creates a fold at each indent level in the document

```
:set foldmethod=indent
```

Create folds in a document one by one with the 'zf' command

```
:set foldmethod=manual
```

30.5 Advanced Folds

A folding method

```
    vim: filetype=help foldmethod=marker foldmarker=<<<,>>>
    A really big section closed with a tag <<<
    --- remember folds can be nested ---
    Closing tag >>>
```

Section 31

Working With Source Code

Adjusts the indent level of pasted code to the file

```
]p
```

Go to the definition or declaration of the function/var under the cursor

```
gd
```

Align braces in source code

```
[shift]+v, select lines, =    ~([shift]+v enters visual mode)
```

Use the brace-block text objects to work with blocks of code

```
diB       ~(deletes everything between { and }, not including braces)
```

```
di}       ~(the same)
```

```
da)        ~(deletes everything between ( and ), including braces)
```

Delete between quotation marks and enter new text

```
ci"        ~(the cursor can be anywhere within the quotation)
```

Another way to get rid of tabs

```
:set expandtab|retab
```

Toggle source code folds

```
zi    ~(a 'fold' temporarily hides the content of a function or method)
```

Fold code on indented sections

```
:set foldmethod=indent    ~(each indentation level becomes a 'fold')
```

Execute 'make', and jump to the first error found

```
:make     ~(make is usually used for compiling c code)
```

Display the next or previous error

```
:cn :cp
```

31.1 Indenting And Formatting

View which program will perform indenting with the '=' command

```
:set equalprg
```

(if the response is 'equalprg=' then an internal formatter will be used)

Indent the whole file using the '=' formatter

```
gg=G
```

Indent the current paragraph using the built in formatter

```
ap=
```

Indent the current brace {} block

```
a}=
```

Reformat (break and fill lines) a bash comment (starting with #)

```
!ap par   ~(the par formatter leaves the '#' at the start of the lines)
```

31.2 Syntax Highlighting

See what type of source code Vim thinks the file is

```
:set filetype
```

(if the response is 'filetype=' then Vim cant guess)

Tell Vim that the file is java source code

```
:set filetype=java    ~(this normally shouldnt be necessary)
```

Turn on syntax highlighting

```
:syntax enable
```

```
:syntax on       ~(doesnt respect the current colourscheme)
```

Turn off syntax highlighting (its pretty annoying really)

```
:syntax off
```

An example of setting your own highlighting

```
:syn match DoubleSpace "  "
```

```
:hi def DoubleSpace guibg=#e0e0e0
```

Force Syntax coloring for a file that has no extension .pl

```
:set syntax=perl
```

Change coloring scheme (any file in ∼vim/vim??/colors)

```
:colorscheme blue
```

Force HTML Syntax highlighting by using a modeline

```
# vim:ft=html:
```

Force syntax automatically (for a file with non-standard extension)

```
au BufRead,BufNewFile */Content.IE?/* setfiletype html
```

31.3 Converting Highlighted Syntax To Html

View the help for converting highlighted syntax

```
:help TOhtml
```

Convert highlighted syntax to html (html opens in a new window)

```
:TOhtml
```

```
:runtime! syntax/2html.vim  ~(the same, more or less)
```

Convert the current line to syntax highlighted html

```
:.TOhtml
```

Convert syntax to HTML using style sheets not <font> tags

```
:let html_use_css = 1
```

```
:TOhtml
```

Convert with no line numbers, xhtml and css

```
:let html_number_lines = 0
```

```
:let use_xhtml = 1
```

```
:let html_use_css = 1
```

Convert code to syntax-coloured HTML

```
:source $VIM/syntax/2html.vim     ~(untested)
```

31.4 C And Cpp

Format the next paragraph of code with the 'indent' c code formatter

```
!}indent
```

```
:%!indent   ~(format the whole file)
```

31.5 Java

Although 'indent' is designed for 'c' code it does a reasonable job with Java code as well

Format the whole file with the external program 'astyle'

```
:%!astyle ...
```

(astyle appears to go crazy with anonymous inner java classes)

Format the current code block (between matching {}) with 'indent'

```
!i}indent   ~(see:"text objects" for an explanation of 'i}')
```

Increase the indent of a code block

```
:set sw=1; >i}
```

Decrease the indent of a code block

```
<i}
```

Set up the '=' command to format with 'astyle'

```
:set equalprg=astyle\ --option=yes\ -q
```

Use the lynx or 'links' browser to view javadoc

```
...
```

Vim And Html

Section 32

Filter all html form elements into paste register

```
:redir @*|sil exec 'g#<\(input\|select\|textarea\|/\=form\)\>#p'|redir
 ⇒  END
:nmap ,z :redir @*<Bar>sil exec 'g@<\(input\<Bar>select\<Bar>textarea
 ⇒ \<Bar>/\=fo
rm\)\>@p'<Bar>redir END<CR>
```

Vim And Latex

Section 33

`http://www.vmunix.com/vim/howto/latex.html`

Set up a mapping to run LaTeX on the current file

```
:map ,rl :!latex %
```

Using The Shell

Section 34

Get output from other commands (requires external programs)

`:r!ls -R`	Reads in output of ls
`:put=glob('**')`	Same as above
`:r !grep ''^ebay'' file.txt`	Grepping in content
`:20,25 !rot13`	Rot13 lines 20 to 25
`!!date`	Same thing (but replaces/filters current line)

Execute the previous shell command

```
:!!
```

Simple Shell script to rename files w/o leaving vim

```
 $ vim
 :r! ls *.c
 :%s/\(.*\).c/mv & \1.bla
 :w !sh
 :q!
```

34.1 Vim In Batch Mode

Batch mode means running vim from a bash shell non-interactively.

Command line tricks

`gvim -h`	Help
`''ls | gvim -''`	Edit a stream!!
`cat xx | gvim`	-c "v/^\d\d\—^[3-9]/d " - filter a stream
`gvim -o file1 file2`	Open into a split

Execute one command after opening file

```
gvim.exe -c "/main" joe.c  : Open joe.c & jump to "main"
```

Execute multiple command on a single file

```
vim -c "%s/ABC/DEF/ge | update" file1.c
```

Automate editing of a file (Ex commands in convert.vim)

```
vim -s "convert.vim" file.c
```

Load VIM without .vimrc and plugins (clean VIM)

```
gvim -u NONE -U NONE -N
```

Access paste buffer contents (put in a script/batch file)

```
gvim -c 'normal ggdG"*p' c:/aaa/xp
```

Print paste contents to default printer

```
gvim -c 's/^/\=@*/|hardcopy!|q!'
```

34.2 Piping Text Fragments

Pipe lines 22,25 though the 'sed' bash shell command

```
:22,25w !sed 's/e/E/g'   ~(the original lines are not changed)
```

Display all following lines without any '^' carat characters

```
:.,$w !sed 's/^//g'      ~(the lines are piped through 'sed')
```

Save lines after this one in 'new.txt' after removing comment (#) lines

```
:.,$w !grep -v '^#' >> new.txt
```

Pipe all lines from current to next blank line to sed

```
:.,/^ *$/w ! sed 's/e/E/g'
```

```
:.,/^\s$/w ! sed 's/e/E/g'   ~(the same, almost)
```

Pipe all lines from current back to previous blank line to 'sed'

```
:.,/^\s$/w ! sed 's/e/E/g'   ~(prompts the user to swap order)
```

View lines in this and next 2 paragraphs with lines having 'tree'

```
!3ap<backspace>w !grep -v tree | less
```

Compile the current paragraph with pdflatex and save as 'new.pdf'

```
!ap<backspace>w !pdflatex -jobname new
```

Pipe from the line after the 1st previous line having a '' star To the line before the next line containing a ',' comma*

```
?\*?+1,/,/-1w !sed 's/e/E/g'    ~(this is quite a complex range)
```

34.3 Gotchas

In the following recipes, the space before the '!' exclamation mark is very important, since without it the command does something very different and possibly destructive.

Remember the space

```
:22,25w!sed 's/e/E/g'   ~(No!!! wrong, may truncate the current file)
```

Section 35

Using External Programs

Sort the current file

```
:%!sort -u
```

Sort the current paragraph (before and after the cursor)

```
!apsort -u
```

Look up help for the keyword under the cursor

```
K
```

Change the program used to look up keyword help with 'K'

```
:set keywordprg=man
```

Map ',ee' to execute the word under the cursor in bash

```
:map ,ee :! <cword>
```

Execute 2 shell commands (change to parent folder, and show the file tree.txt)

```
:!cd ..; cat tree.txt
```

Switching Between Modes

Section 36

Switch to normal mode from command mode

```
:normal
```

Visual Mode

Section 37

Visual is the newest and usually the most intuitive editing mode

Select a line visually

```
V ~(then use 'k' and 'j' to select lines above and below as well)
```

Reselect last visual mode selection

```
gv
```

Operating a Recording on a Visual BLOCK

```
    1) define recording/register
    qq:s/ to/ from/g^Mq
    2) Define Visual BLOCK
    V}
    3) hit : and the following appears
    :'<,'>
    4)Complete as follows
    :'<,'>norm @q
```

using the visual mode

v	Enter visual mode
V	Visual mode whole line
<C-V>	Enter VISUAL BLOCK mode
gv	Reselect last visual area (ultra)
o	Navigate visual area
``*y or ''+y	Yank visual area into paste buffer
V%	Visualise what you match
V}J	Join Visual block (great)
V}gJ	Join Visual block w/o adding spaces
`[v`]	Highlight last insert
:%s/\%Vold/new/g	Do a substitute on last visual area

Visual searching

```
:vmap // y/<C-R>"<CR> ~( search for visually highlighted text)
```

```
:vmap <silent> //    y/<C-R>=escape(@", '\\/.*$^~[]')<CR><CR> : with
 ⇒ spec chars
```

Pull Visually Highlighted text into LHS of a substitute

```
:vmap <leader>z :<C-U>%s/\<<c-r>*\>/
```

Vim Sessions

Following 4 maps enable text transfer between VIM sessions

`:map <f7> :'a,'bw!  c:/aaa/x`	Save text to file x
`:map <f8> :r c:/aaa/x`	Retrieve text
`:map <f11> :.w!  c:/aaa/xr<CR>`	Store current line
`:map <f12> :r c:/aaa/xr<CR>`	Retrieve current line
`:ab php`	List of abbreviations beginning php
`:map ,`	List of maps beginning ,

Command Mappings

http://vim.wikia.com/wiki/Mapping_keys_in_Vim_-_Tutorial_(Part_1)

A 'mapping' allows the user to use a key or keys to carry out a command or a series of commands. They are like 'shortcuts'. Put mappings in ∼/.vimrc (without the first colon) to make them permanent.

Show the vim help for creating command mappings

```
:help key-mapping
```

Show current macros, or mappings

```
:map
```

Write current mappings and settings to file 'map.txt'

```
:mk map.txt
```

Map [shift]+s to delete a line.

```
:map <S-s> dd
```

```
:map <C-s> dd      ~( [control]+s )
```

```
:map <A-s> dd      ~( [alt]+s )
```

Remove the mapping for 'hh'

```
unmap hh
```

Map the function key f5 to delete a line

```
:map <f5> dd
```

Map ',hh' to insert 'hello' (from insert mode)

```
:map! ,hh hello
```

Create a mapping which calls a command

```
:map! ,test :Test
```

Map a macro in command mode

```
:cmap
```

Make a macro in insert and command line mode

```
:Imap
```

Make a macro in insert mode

```
:imap
```

For use in Maps

`<CR>`	Carriage Return for maps
`<ESC>`	Escape
`<LEADER>`	Normally
`<BAR>`	— pipe
`<BACKSPACE>`	Backspace
`<SILENT>`	No hanging shell window

`http://www.vim.org/htmldoc/intro.html#%3CBar%3E`
all special keys

mapping special keys

`<esc>`	The Esc key
`<enter>` or `<cr>` or `<return>`	The Enter key
`<space>`	The space key
`<left>`	The cursor left
`<right>`	The cursor right

39.1 Using Functions In Mappings

To use a Vim function within a mapping, the idiom <C-R>=functionname(...) needs to be used
Map [f2] to insert the current date and time after the cursor

```
:map <F2> a<C-R>=strftime("%c")<CR><Esc>
```

Map ',,r' to type the start of a replace with the word under the cursor

```
:map ,,r :%s/<C-R>=expand("<cword>")<cr>//gc<left><left><left>
```

Cword, cWORD, cfile

```
<cword> is the word under the cursor
```

```
<cfile> is the path under the cursor
```

39.2 Example Mappings

Map ,ww to write out text from the cursor until 'big' to the file 'junk.txt'

```
map ,ww !/big<enter><backspace>w junk.txt<enter>
```

Mapping two commands, ',mm' saves the file and goes to the next word 'big'

```
map ,mm w <bar> /big<cr>
```

Map ',ll' to underline the current line with dashes '-'

```
map ,ll yyp:s/[ ]*$// \| s/[ ]*$// \| s/[^ ]/-/g \| s/- /--/g<cr>
```

(the dashed underline will have gaps if the words have 2 space gaps)

Abbreviations

Section 40

Abbreviations are very similar to mappings but are activated in 'insert' mode, instead in 'normal' mode Abbreviations are essentially a way to avoid typing repetitive things. Be happy, this is good.
The abbreviation is activated after typing a space or newline after the abbreviation.
Automatically exchange the word 'big' for 'very large and bulky'

```
:abbr big very large and bulky   ~(this is called an abbreviation)
```

Turn off an abbreviation

```
:unab big
```

Show all abbreviations which are currently available

```
:abbr
```

Define an abbreviation which inserts several lines of text

```
:ab poem froth and foam<cr>here alone<cr>
```

Create an abbreviation which inserts text then searche for 'rain'

```
:ab sss snake<esc>/rain<cr>
```

Put an abbreviation in the 'vimrc' file, without the colon ':'

```
ab rrr rain
```

Creating New Commands

A command is anything written after a colon ':'. New commands can be created in Vim. User defined commands must begin with a capital letter.

Define the command 'Boo' to delete the current sentence

```
:com! Boo normal das
```

(normal switches from command to normal mode) (this command can be used by typing ' Esc :Boo')

A command 'Test' which find the next occurance of '=='

```
:com! Look normal /==<cr>
```

```
:com! Look normal /==/<cr>     ~(exactly the same)
```

A command 'TT' which puts 'tree' after the next line starting with '#'

```
:com! TT normal /^ *#<cr>otree<esc>
```

The command 'J' adds 'tree' after next blank line and saves the file

```
com! J normal /^ *$/<cr>otree<esc>:w<cr>
```

The command 'Gr' adds 'grass' 2 lines after the next blank line

```
com! J normal /^ *$/+2<cr>otree<esc>
```

View help on how to make user defined commands

```
:help command
```

See the currently user defined commands

```
:command
```

```
:com  ~(the same)
```

Delete all new commands which have been defined by the user

```
:comclear
```

Create a new command 'Folder' which lists the files in the current folder

```
:com Folder !ls
```

```
:command Folder !ls   ~(the same)
```

```
:command! Folder !ls  ~(the same, but over-rides an existing command)
```

Define command 'squeeze' which combines multiple consecutive empty lines

```
:command! Squeeze g/^\s*$/,/\S/-j|s/.*//
```

Define command 'Nocomment' which lists the current file without comments

```
command! Nocomment !cat % | grep -v '^ *#' | less
```

(this command can be executed with ':Nocomment' within vim)

Define a command which shows lines having only uppercase letters

```
command! Upper !sed -n ';/^ *[[:upper:] ]\{2,\}$/p' % | less
```

Delete everything after the current line to the end of the file

```
:com Ddel +,$d
```

41.1 With Arguments

Rename the current buffer with filename completion

```
:com -nargs=1 -bang -complete=file Ren f <args>|w<bang>
```

Define a command 'Say' with one argument which just echos the given text

```
:com -nargs=1 Say :echo "<args>"
```

```
:command -nargs=1 Say :echo "<args>"    ~(the same)
```

(This command can be invoked with :Say text)

Create a command 'Book bookfile' which edits a text file

```
command! -nargs=1 Book arge ~/<args>.txt
```

So we can use this command by typing ':Book tree' to edit the file ∼/tree.txt

Create a new command 'Sing' which accepts a variable number of arguments

```
com -nargs=* Sing
```

41.2 With Line Ranges

Define command 'Rep' which replaces a range with the contents of a file

```
:com -range -nargs=1 -complete=file Rep <line1>-pu_|<line1>,<line2>d|r
  ⇒ <args>|<line1>d
```

Define command 'Lines' which counts the number of lines in a range

```
:com! -range -nargs=0 Lines  echo <line2> - <line1> + 1 "lines"
```

(this can be executed with ':.,$ Lines' for example)

A command 'Comp' which pipes the given range to the 'sed'

```
:com! -range -nargs=0 Comp <line1>,<line2>w !sed 's/e/E/g'
```

```
:com! -range Comp <line1>,<line2>w !sed 's/e/E/g'     ~(the same)
```

(this could be executed with ':.,$ Comp' for example)

Make a command to compile to pdf the given range and save as 'new.pdf'

```
:com! -range Pdf <line1>,<line2>w !pdflatex -jobname new
```

special command paramters

<line1>	The first line of the range, with '-range' option
<line2>	Another line in the range
<bang>	

41.3 Commands Using Commands

We can define new commands which can then be used in other new commands, for example:

Use a previous user-defined command in another for finding images

```
command! -nargs=1 Findi !find ~ -name '*.<args>'
```

```
command! Findjpg Findi jpg
```

41.4 Commands And Mappings

Commands can use argument specified after the command name but mappings cannot (since they are entered in 'normal' mode) Mappings can also use a previously defined command.

Recording And Using Macros

A 'macro' in vim is a series of commands which a user carries out while 'recording'. The user is then able to replay that series of commands by invoking the macro. In vim, the names of macros are single letters and are invoked with '@n' where 'n' is the name of the macro.

Start recording a macro named 'q'

```
qq
```

Stop recording a macro

```
q
```

Execute (replay) the macro

```
@q
```

Repeat the macro

```
@@
```

macro commands

`@q`	To execute
`@@`	To Repeat
`5@@`	To Repeat 5 times
`qQ@qq`	Make an existing recording q recursive

Editing a register/recording

```
"qp          ~(display contents of register q (normal mode))
```

```
<ctrl-R>q    ~(display contents of register q (insert mode))
```

You can now see recording contents, edit as required

```
"qdd      ~(put changed contacts back into q)
```

```
@q        ~(execute recording/register q)
```

Combining a recording with a map (to end up in command mode)

```
:nnoremap ] @q:update<bar>bd
```

Vim Functions

These functions can be used in mappings, and scripts. User defined functions need to start with a Capital letter.'

Get help on the built-in functions for vim

```
:help functions          ~(an alphabetical function list)
```

```
:help function-list      ~(function list categorized by purpose)
```

See all user defined functions

```
:functions
```

List the code for the Leapyear function

```
:function Leapyear
```

Delete the Leapyear function

```
:delfuncton Leapyear
```

Call a function for a range

```
:10,15call Func(...)
```

Redefine a function

```
:function! Func()
```

vim functions

`getline(".")`	Returns the text of the current line
`line(".")`	Returns the number of the current line

43.1 Creating New Functions

Function to delete duplicate lines

```
function! Del()
 if getline(".") == getline(line(".") - 1)
   norm dd
 endif
endfunction
```

A function to save word under cursor to a file

```
function! SaveWord()
   normal yiw
   exe ':!echo '.@0.' >> word.txt'
endfunction
map ,p :call SaveWord()
```

Vim Programming

Section 44

Vim, like all good overblown software, has its own scripting language built into it, which may be of use if you wish to automate some editing process and you are unable to do it with mappings, macros, abbreviations, user-defined commands, bash shell filters, or any of the other myriad tools available with vim.

View some help for programming with vim

```
eval.txt
```

Insert ip range using vim

```
:for i in range(1,255) | .put='192.168.0.'.i | endfor
```

To combine statements on one line use '—' *Reduce the current line by one*

```
-
```

Increase the current line by one

```
+
```

Put the cursor on the 15th character of the current line)

```
:call cursor(line("."), 15)
```

44.1 Variable Assignment

Set s to the line number of the current line

```
let s = line(".")
```

Subtract 1 from a variable s

```
let s=s-1
```

Vim Functions

Section 45

Get help for the 'line' function

```
help line
```

Display the line number of the last line in the file using 'line'

```
:echo line('$')
```

Insert the line number of the last line into the document.

```
:.put=line('$')
```

values usable with the 'line' function

`.`	The cursor position
`$`	The last line in the current buffer
`'x`	Position of mark x (if the mark is not set, 0 is returned)
`w0`	First line visible in current window
`w$`	Last line visible in current window

String Variables

Set the variable s to be the string 'grass' and display it

```
:let s='grass' | echo s
```

Set s to be 'eat' and display the string length of s (which is '3')

```
:let s='eat' | echo strlen(s)
```

Array Variables

In the vim language, arrays are called 'lists'
Useful list functions

```
    :let r = call(funcname, list)    " call a function with an argument
     ⇒ list
    :if empty(list)                  " check if list is empty
    :let l = len(list)               " number of items in list
    :let big = max(list)             " maximum value in list
    :let small = min(list)           " minimum value in list
    :let xs = count(list, 'x')       " count nr of times 'x' appears in
     ⇒ list
    :let i = index(list, 'x')        " index of first 'x' in list
    :let lines = getline(1, 10)      " get ten text lines from buffer
    :call append('$', lines)         " append text lines in buffer
    :let list = split("a b c")       " create list from items in a string
    :let string = join(list, ', ')   " create string from list items
    :let s = string(list)            " String representation of list
    eval.txt [Help][RO]                                          372,15
```

47.1 Get Help For Lists

View extensive information about list variable in vim

```
:help List    ~(notice the capital 'L' in 'List')
```

47.2 List Information

Display the number of elements in a list

```
echo len(l)
```

47.3 Creating Lists

Create a new list (array) with 4 elements

```
:let mylist = [1, two, 3, "four"]
```

Create a new list with no elements

```
:let emptylist = []
```

47.4 Getting Elements From Lists

Set a variable 'item' to the value of the 1st element of the list

```
:let item = mylist[0]
```

Set a variable to the 3rd element of a list variable

```
:let i = mylist[2]
```

Get the last item from the list

```
:let i = alist[-1]
```

47.5 Joining Lists Together

Join or concatenate 2 lists together

```
:let longlist = mylist + [5, 6]
```

Add 2 new elements to a list

```
:let mylist += [7, 8]
```

Get all elements from a list from the 3rd to the last

```
:let shortlist = mylist[2:-1]
```

```
:let shortlist = mylist[2:]      "(the same)
```

Assign variable a the 1st value and variable b the 2nd value

```
:let [a, b] = mylist
```

47.6 Changing List Items

Set the 5th element of the list to 'grass'

```
:let list[4] = "grass"
```

Dictionaries

Section 48

Dictionaries are what are sometimes known in other languages as 'hashes' or 'associative arrays'. They a set of items, each of which has a key and a value

View good help for the dictionary data type, notice the capital D

```
help Dictionary
```

Loop over a dictionary

```
:for key in keys(mydict)
:   echo key . ': ' . mydict[key]
:endfor
```

If Tests

Section 49

Test if the variable 's' consists entirely of whitespace

```
if s !~ "^\\s*$"
  ...
endif
```

Test if the variable 's' equals 0

```
if s == 0
  ...
endif
```

Exe Or Eval

Section 50

In vim the classic 'eval' statement is called 'exe'. It allows you to exectute some vim script code which has been constructed from a string

Execute a new vimscript command

```
:exe 'let sum = ' . join(nrlist, '+')
```

50.1 Loops

A for loop to insert an ip range using vim

```
:for i in range(1,255) | .put='192.168.0.'.i | endfor
```

Loop through each item in 'list' calling a function

```
:for item in list
```

```
:  call Doit(item)
```

```
:endfor
```

While loop

```
 while p > 0
   ...
 endwhile
```

Other Stuff

Section 51

Edit a script that's somewhere in your path.

```
vim `which scriptfile`
```

```
vim $(which scriptfile)    "(the same)
```

Lazy man's vim

```
function v { if [ -z $1 ]; then vim; else vim *$1*; fi }
```

Download a sequence of vim patches

```
seq -f"ftp://ftp.vim.org/pub/vim/patches/7.1/7.1.%03g" 176 240 | xargs
 ⇒ -I {} wget -c {};
```

Rayninfo

Section 52

This section includes many recipes from rayninfo.co.uk

\zs and \ze regex delimiters :h /\zs

```
/<\zs[^>]*\ze>  "( search for tag contents, ignoring chevrons)
```

zero-width :h /\@=

`/<\@<=[^>]*>\@=`	Search for tag contents, ignoring chevrons
`/<\@<=\_[^>]*>\@=`	Search for tags across possible multiple lines

searching over multiple lines _ means including newline

`/<!--\_p\{-}-->`	Search for multiple line comments
`/fred\_s*joe/`	Any whitespace including newline
`/bugs\(\_.\)*bunny`	Bugs followed by bunny anywhere in file
`-h \_`	Help

Search for declaration of subroutine/function under cursor

```
:nmap gx yiw/^\(sub\<bar>function\)\s\+<C-R>"<CR>
```

Find replacement text, put in memory, then use \zs to simplify substitute

```
:%s/"\([^.]\+\).*\zsxx/\1/
```

Pull word under cursor into LHS of a substitute

```
:nmap <leader>z :%s#\<<c-r>=expand("<cword>")<cr>\>#
```

Substitute singular or plural

```
:'a,'bs/bucket\(s\)*/bowl\1/gic
```

All following performing similar task, substitute within substitution Multiple single character substitution in a portion of line only

```
:%s,\(all/.*\)\@<=/,_,g ~( replace all / with _ AFTER "all/")
```

Same thing

```
:s#all/\zs.*#\=substitute(submatch(0), '/', '_', 'g')#
```

Substitute by splitting line, then re-joining

```
:s#all/#&^M#|s#/#_#g|-j!
```

Substitute inside substitute

```
:%s/.*/\='cp '.submatch(0).' all/'.substitute(submatch(0),'/','_','g')/
```

Storing glob results (note must use APPEND) you need to empty reg a first with qaq.
Operate until string found

```
d/fred/    ~(delete until fred)
```

```
y/fred/    ~(yank until fred)
```

```
c/fred/e   ~(change until fred end)
```

How to have a variant in your .vimrc for different PCs

```
 if $COMPUTERNAME == "NEWPC"
 ab mypc vista
 else
 ab mypc dell25
 endif
```

Vertically split current file with other.php

```
:vsplit other.php
```

VISUAL MODE (easy to add other HTML Tags)
Wrap <b></b> around VISUALLY selected Text

```
:vmap sb "zdi<b><C-R>z</b><ESC>
```

Wrap <?= ?> around VISUALLY selected Text

```
:vmap st "zdi<?= <C-R>z ?><ESC>
```

more g stuff

`gf`	Open file name under cursor (SUPER)
`:nnoremap gF :view <cfile><cr>`	Open file under cursor, create if necessary
`ga`	Display hex,ascii value of char under cursor
`ggVGg?`	Rot13 whole file
`ggg?G`	Rot13 whole file (quicker for large file)
`:8 | normal VGg?`	Rot13 from line 8
`:normal 10GVGg?`	Rot13 from line 8
`<C-A>,<C-X>`	Increment,decrement number under cursor (not win32)
`<C-R>=5*5`	Insert 25 into text (mini-calculator)

Make all other tips superfluous

```
    :h 42  ##( also http://www.google.com/search?q=42 )
    :h holy-grail
    :h!
```

Disguise text (watch out)

```
ggVGg? ~( rot13 whole file (toggles))
```

```
:set rl! ~( reverse lines right to left (toggles))
```

```
:g/^/m0 ~( reverse lines top to bottom (toggles))
```

Display RGB colour under the cursor eg #445588

```
:nmap <leader>c :hi Normal guibg=#<c-r>=expand("<cword>")<cr><cr>
```

```
map <f2> /price only\\|versus/ :in a map need to backslash the \
```

Type table,,, to get <table></table> ### Cool ###

```
imap ,,, <esc>bdwa<<esc>pa><cr></<esc>pa><esc>kA
```

Simple PHP debugging display all variables yanked into register a

```
iab phpdb exit("<hr>Debug <C-R>a  ");
```

Using a register as a map (preload registers in .vimrc)

```
:let @m=":'a,'bs/"
```

```
:let @s=":%!sort -u"
```

Useful tricks

``ayy@a	Execute "Vim command" in a text file
yy@"	Same thing using unnamed register
u@.	Execute command JUST typed in
"ddw	Store what you delete in register d
"ccaw	Store what you change in register c

Number lines

```
:new | r!nl #
```

Quick jumping between splits

```
map <C-J> <C-W>j<C-W>_
```

```
map <C-K> <C-W>k<C-W>_
```

Delete first 2 characters of 10 successive lines

```
0<c-v>10j2ld
```

How to copy a set of columns using VISUAL BLOCK Visual block (AKA columnwise selection) (NOT BY ordinary v command)

```
<C-V> then select "column(s)" with motion commands (win32 <C-Q>)
```

```
then c,d,y,r etc
```

How to overwrite a visual-block of text with another such block Move with hjkl etc

```
Pick the first block: ctrl-v move y
```

```
Pick the second block: ctrl-v move P <esc>
```

Launching Win IE

```
:nmap ,f :update<CR>:silent !start c:\progra~1\intern~1\iexplore.exe
 ⇒ file://%:p<CR>
```

```
:nmap ,i :update<CR>: !start c:\progra~1\intern~1\iexplore.exe <cWORD><
 ⇒ CR>
```

FTPing from VIM

```
cmap ,r  :Nread ftp://209.51.134.122/public_html/index.html
```

```
cmap ,w  :Nwrite ftp://209.51.134.122/public_html/index.html
```

```
gvim ftp://www.somedomain.com/index.html # uses netrw.vim
```

Appending to registers (use CAPITAL)   Yank 5 lines into "a" then add a further 5

```
"a5yy
```

```
10j
```

```
"A5yy
```

```
[I     : show lines matching word under cursor <cword> (super)
```

Conventional Shifting/Indenting

```
:'a,'b>>
```

Visual shifting (builtin-repeat)

```
:vnoremap < <gv
```

```
:vnoremap > >gv
```

Block shifting (magic)

```
>i{
```

```
>a{
```

Also

```
>% and <%
```

Redirection & Paste register *

`:redir @*`	Redirect commands to paste buffer
`:redir END`	End redirect
`:redir >> out.txt`	Redirect to a file

Working with Paste buffer

```
"*yy   ~(yank curent line to paste)
```

```
"*p    ~(insert from paste buffer)
```

Yank to paste buffer (ex mode)

```
:'a,'by*                          : Yank range into paste
```

```
:%y*                              : Yank whole buffer into paste
```

```
:.y*                              : Yank Current line to paster
```

Filter non-printable characters from the paste buffer   Useful when pasting from some gui application

```
:nmap <leader>p :let @* = substitute(@*,'[^[:print:]]','','g')<cr>"*p
```

Re-Formatting text

`gq}`	Format a paragraph
`gqap`	Format a paragraph
`ggVGgq`	Reformat entire file
`Vgq`	Current line

Break lines at 70 chars, if possible after a ;

```
:s/.\{,69\};\s*\|.\{,69\}\s\+/&\r/g
```

*Gvim's use of external grep (win32 or *nix)*

```
:grep somestring *.php     : creates a list of all matching files
```

Use :cn(ext) :cp(rev) to navigate list

```
:h grep
```

Using vimgrep with copen

```
:vimgrep /keywords/ *.php
```

```
:copen
```

GVIM Difference Function (Brilliant)

```
gvim -d file1 file2 ~( vimdiff (compare differences))
```

```
dp ~( "put" difference under cursor to other file)
```

```
do ~( "get" difference under cursor from other file)
```

" complex diff parts of same file :1,2yank a — 7,8yank b :tabedit — put a — vnew — put b :windo diffthis

\v or very magic (usually) reduces backslashing

```
/codes\(\n\|\s\)*where  : normal regexp
```

```
/\vcodes(\n|\s)*where   : very magic
```

pulling objects onto command/search line (SUPER)

`<C-R><C-W>`	Pull word under the cursor into a command line or search
`<C-R><C-A>`	Pull WORD under the cursor into a command line or search
`<C-R>-`	Pull small register (also insert mode)
`<C-R>[0-9a-z]`	Pull named registers (also insert mode)
`<C-R>%`	Pull file name (also #) (also insert mode)
`<C-R>=somevar`	Pull contents of a variable (eg :let sray="ray[0-9]")

find where an option was set

`:scriptnames`	List all plugins, _vimrcs loaded (super)
`:verbose set history?`	Reveals value of history and where set
`:function`	List functions
`:func SearchCompl`	List particular function

Making your own VIM help

```
:helptags /vim/vim64/doc  : rebuild all *.txt help files in /doc
```

```
:help add-local-help
```

Running file thru an external program (eg php)

```
map  <f9>  :w<CR>:!c:/php/php.exe %<CR>
```

```
map  <f2>  :w<CR>:!perl -c %<CR>
```

Capturing output of current script in a separate buffer

```
:new | r!perl # ~( opens new buffer,read other buffer)
```

```
:new! x.out | r!perl # ~( same with named file)
```

```
:new+read!ls
```

Create a new buffer, paste a register "q" into it, then sort new buffer

```
:new +put q|%!sort
```

Inserting DOS Carriage Returns

```
:%s/$/\<C-V><C-M>&/g      ~(that's what you type)
```

```
:%s/$/\<C-Q><C-M>&/g      ~(for Win32)
```

```
:%s/$/\^M&/g              ~(what you'll see where ^M is ONE character)
```

Automatically delete trailing Dos-returns,whitespace

```
autocmd BufRead * silent! %s/[\r \t]\+$//
```

```
autocmd BufEnter *.php :%s/[ \t\r]\+$//e
```

Perform an action on a particular file or file type

```
autocmd VimEnter c:/intranet/note011.txt normal! ggVGg?
```

```
autocmd FileType *.pl exec('set fileformats=unix')
```

Retrieving last command line command for copy & pasting into text

```
i<c-r>:
```

Retrieving last Search Command for copy & pasting into text

```
i<c-r>/
```

" more completions <C-X><C-F> :insert name of a file in current directory
" Substituting a Visual area " select visual area as usual (:h visual) then type :s/Emacs/Vim/ etc :'<,'>s/Emacs/Vim
: REMEMBER you dont type the '<.'> gv : Re-select the previous visual area (ULTRA)
" inserting line number into file :g/^/exec "s/^/".strpart(line(".")." ", 0, 4) :%s/^/\=strpart(line(".")." ", 0, 5)
:%s/^/\=line('.'). ' '

Numbering lines VIM way

```
:set number ~( show line numbers)
```

```
:map <F12> :set number!<CR> ~( Show linenumbers flip-flop)
```

```
:%s/^/\=strpart(line('.')."        ",0,&ts)
```

Numbering lines (need Perl on PC) starting from arbitrary number

```
:'a,'b!perl -pne 'BEGIN{$a=223} substr($_,2,0)=$a++'
```

Produce a list of numbers Type in number on line say 223 in an empty file qqmnYP'n^Aq : in recording q repeat with @q

Increment existing numbers to end of file (type <c-a> as 5 characters)

```
:.,$g/^\d/exe "normal! \<c-a>"
```

Advanced incrementing

```
http://vim.sourceforge.net/tip_view.php?tip_id=150
```

Advanced incrementing (really useful) put following in _vimrc

```
 let g:I=0
 function! INC(increment)
 let g:I =g:I + a:increment
 return g:I
 endfunction
```

Eg create list starting from 223 incrementing by 5 between markers a,b

```
:let I=223
```

```
:'a,'bs/^/\=INC(5)/
```

Create a map for INC

```
cab viminc :let I=223 \| 'a,'bs/$/\=INC(5)/
```

Generate a list of numbers 23-64

```
o23<ESC>qqYp<C-A>q40@q
```

editing/moving within current insert (Really useful)

`<C-U>`	Delete all entered
`<C-W>`	Delete last word
`<HOME><END>`	Beginning/end of line
`<C-LEFTARROW><C-RIGHTARROW>`	Jump one word backwards/forwards
`<C-X><C-E>,<C-X><C-Y>`	Scroll while staying put in insert

Encryption (use with care: DON'T FORGET your KEY)

```
:X      "(you will be prompted for a key)
```

```
:h :X
```

Modeline (make a file readonly etc) must be in first/last 5 lines

```
// vim:noai:ts=2:sw=4:readonly:
```

```
" vim:ft=html:                        : says use HTML Syntax highlighting
```

```
:h modeline
```

Creating your own GUI Toolbar entry amenu Modeline.Insert\a\VIM\modeline <Esc><Esc>ggOvim:ff=unix ts=4 ss=4<CR>v im60:fdm=marker<esc>gg
Use this function with

```
:g/^/ call Del()
```

Digraphs (non alpha-numerics)

:digraphs	Display table
:h dig	Help
i<C-K>e'	Enters
i<C-V>233	Enters (Unix)
i<C-Q>233	Enters (Win32)
ga	View hex value of any character

Deleting non-ascii characters (some invisible)

:%s/[\x00-\x1f\x80-\xff]/ /g	Type this as you see it
:%s/[<C-V>128-<C-V>255]//gi	Where you have to type the Control-V
:%s/[-]//gi	Should see a black square & a dotted y
:%s/[<C-V>128-<C-V>255<C-V>01-<C-V>31]//gi	All pesky non-asciis
:exec "norm /[\x00-\x1f\x80-\xff]/"	Same thing

Pull a non-ascii character onto search bar

```
yl/<C-R>"
```

```
/[^a-zA-Z0-9_[:space:][:punct:]]  : search for all non-ascii
```

All file completions grouped (for example main_c.c)

```
:e main_<tab>     "(tab completes)
```

```
gf                "(open file under cursor  (normal))
```

```
main_<C-X><C-F>   "(include NAME of file in text (insert mode))
```

Complex Vim, swap two words

```
:%s/\<\(on\|off\)\>/\=strpart("offon", 3 * ("off" == submatch(0)), 3)/g
```

Swap two words

```
:vnoremap <C-X> <Esc>`.``gvP``P
```

Swap word with next word

```
nmap <silent> gw    "_yiw:s/\(\%#\w\+\)\(\_W\+\)\(\w\+\)/\3\2\1/<cr><c-
 ⇒ o><c-l>
```

Convert Text File to HTML

```
:runtime! syntax/2html.vim "( convert txt to html)
```

```
:h 2html
```

```
:set noma (non modifiable)        : Prevents modifications
```

```
:set ro (Read Only)               : Protect a file from unintentional
 ⇒ writes
```

tags (jumping to subroutines/functions)

taglist.vim	Popular plugin
:Tlist	Display Tags (list of functions)
<C-]>	Jump to function under cursor

Displaying "non-asciis"

```
:set list
```

```
:h listchars
```

How to paste "normal commands" w/o entering insert mode

```
:norm qqy$jq
```

Delete without destroying default buffer contents

```
"_d ~( what you've ALWAYS wanted)
```

```
"_dw ~( eg delete word (use blackhole))
```

Pull full path name into paste buffer for attachment to email etc

```
nnoremap <F2> :let @*=expand("%:p")<cr>  ~(?? error)
```

```
nnoremap <F2> :let @*=substitute(expand("%:p"), "/", "\\", "g")<cr> ~(
 ⇒ win32)
```

Reproduce previous line word by word

```
imap ]  @@@<ESC>hhkyWjl?@@@<CR>P/@@@<CR>3s
```

```
nmap ] i@@@<ESC>hhkyWjl?@@@<CR>P/@@@<CR>3s
```

Programming keys depending on file type

```
:autocmd bufenter *.tex map <F1> :!latex %<CR>
```

```
:autocmd bufenter *.tex map <F2> :!xdvi -hush %<.dvi&<CR>
```

Just Another Vim Hacker JAVH

```
vim -c ":%s%s*%Cyrnfr)fcbafbe[Oenz(Zbbyranne%|:%s)[[()])-)Ig|norm Vg?"
```

Read Vimtips into a new vim buffer (needs w3m.sourceforge.net)

```
:tabe | :r ! w3m -dump http://zzapper.co.uk/vimtips.html
```

Commands to neutralise < for HTML display and publish   Use yy@" to execute following commands

```
:w!|sav! vimtips.html|:/^__BEGIN__/,/^__END__/s#<#\<#g|:w!|:!vimtipsftp
```

Section 53

Using Microsoft Crippleware

Allow use of F10 for mapping (win32)

```
set wak=no       ~( :h winaltkeys )
```

Make it easy to update/reload _vimrc

```
:nmap ,s :source $VIM/_vimrc
```

```
:nmap ,v :e $VIM/_vimrc
```

```
:e $MYVIMRC   ~(edits your _vimrc whereever it might be )
```

Reading Ms-Word documents, requires antiword

```
:autocmd BufReadPre *.doc set ro
```

```
:autocmd BufReadPre *.doc set hlsearch!
```

```
:autocmd BufReadPost *.doc %!antiword "%"
```

Using gVIM with Cygwin on a Windows PC

```
    if has('win32')
    source $VIMRUNTIME/mswin.vim
    behave mswin
    set shell=c:\\cygwin\\bin\\bash.exe shellcmdflag=-c shellxquote=\"
    endif
```

Vim People

Section 54

`bill`
: joy wrote the vi text editor, which built apon 'ex'

`bram`
: moolenaar created the vim editor

`david`
: rayner david at rayninfo.co.uk wrote some good tips about using the vim editor

Information Sources

Section 55

`http://groups.google.com/group/vim_use`
: a users newsgroup

`comp.editors`
: a text editor newsgroup

`http://vim.wikia.com/`
: the vim wiki

`http://www.newriders.com/books/opl/ebooks/0735710015.html`
: a vim book

`http://vimdoc.sourceforge.net/cgi-bin/vim2html2.pl`
: searchable docs

`http://gav.brokentrain.net/projects/vimtips/vimtips.pdf`
: some printable tips