Personal Learning File
Author: Lucas Gouveia Belon, a brazilian guy.
<leader> default key is \
Saving Vim as a file (may ask for a filename)
:w
we can also do
:saveas (it will make a copy)
Quit VIM without saving changes
:q!
Quit Saving Changes
:wq
Alternative way:
ZZ (zz does a different thing, commands are case sensitive)
When some text has a link you can let the cursor over
the link and press "Ctrl +]"
When you want to go back to where you were "Ctrl + T"
There are 3 basic modes on vim:
Normal mode - You can use lots of key bindings
Insert mode - You can insert text
Visual mode - it works like a text selection
(visual block mode : Ctrl + V ["o" other end])
If you continually press <Esc> you'll go back to normal mode
After writing in the command mode, press Ctrl+D to see completion
possibilities, or <TAB> to change beetween them
MOTION KEYS - Moving around with VIM k
h (Left) j (Down) k (Up) l (Right) h-+-l
Tip: J looks like a downarrow. Left and Right are at extremities. j
{
} Jumps blocks of codes
w,W (words) goes foward jumping words
e Goes forward but the cursor stops in a different
position (try and find out)
b,B (backwords) goes backwards jumping words
0 (zero) goes to the begining of the line
$ goes to the end of the line
^ goes to the begining of the line that's not a empty space
Ctrl + f Page Down
Ctrl + b Page Up
Ctrl + d Half Page Down
Ctrl + u Half Page Up
Ctrl + e One line Down
Ctrl + y one line Up
(left hand always goes down)
zz - Put your cursor at the middle of the screen (it scrolls
the page to fit)
zt - Put your cursor at the top of the screen
zb - Put your cursor at the bottom of the screen
INSERT MODE
Figure it how these are different
a, i
A Goes to the end of the line and inits insert mode
I Goes to the beggining of the line and inits insert mode
o Inputs a new line below and starts the insert mode
O Inputs a new line above and starts the insert mode
s Erases the character and inits insert mode
S Erases the line and inits insert mode
MORE COMMANDS
~ - Change case a->A
u - undo
Ctrl R - Redo
</> - shift text
J - Join lines (turn it in a monoline)
<del> - erases as x does
x - delete the actual character

d - Used with motion commands, deletes characters
D - delete from here to the end of the line (the same as d$)
C - delete posterior word then inits insert mode
% Jumps in beetween two logical parenthesis "([{" or ")]}"
:n Goes to the 'n' line
t! Goes to the next ! (can be used with any character)
(forward 'til ! character)
d% Erases everything inside a parenthesis
c - change
c w (change word) Deletes a word and inits insert mode
d t } - Delete until the } sign
c t } - Delete until the } and inits insert mode
:! [command] - Executes the command on terminal
When writing, for example, python code, you can run the present buffer with:
:! python3 %
/ Starts the search mode (jump beetween terms with n and N)
// Repeats last search
? Starts the search, but backwards
?? Repeats last search, but backwards
* Search next ocurrencie of the word under the cursor
# Search previous ocurrencie of the word under the cursor
:vimgrep Word *.txt : Search for "Word" in all files of the working directory
Ctrl + O goes to previous positions
Ctrl + I goes to newer positions (the oposite of Ctrl+O)
You can save parts of the file in a new file by using visual
mode (pressing v), selecting the wanted piece of text, then running
:w [fileName]
You can concatenate a external file on your vim's buffer by doing
:r [file or filepath]
Ctrl + G - Shows file stat and cursor position
G goes to the end of the file
gg Goes to the beginning of the file
y (yank) it copies the text under the cursor (used on visual mode)
yy copies the line (Y also does) can be used with numbers
Example:
3yy - copies 3 lines.
p (paste) the paste command
R (replace) enter in substitution mode
To replace "old" in a line for "new" type :s/old/new
To replace all "old" words in a line for "new" type :s/old/new/g
To replace "old" in a number of lines :#,#s/old/new
To replace in the whole file :%s/old/new/g
To replace in the whole file but asking before :%s/old/new/gc
To ask for help press:
:help
It will open a new window. To switch beetween windows use
(Ctrl + W) + (Ctrl + W). Yes, twice
The above command can be used with splitted windows

MARKS

You can navigate through your marks on the text. To set a mark
:ma
We inserted a mark named "a", we can have 26 of them [a-z]
`a
If you want to go to the beggining of the line of that mark
you can go as follows
'a
If you have two marks, you can switch beetween them with
''
you can list all your marks with
:marks
Since there is an access file history you can navigate through
some files with those marks. But you won't use the [a-z] pattern

MULTIFILE EDITING – A powerfull tool

There are 3 kinds of windows that you can change in between

Buffers (everything is a buffer when we're editing)
they behave like another vim that's opened.
Tabs (Like windows tabs that shows up on the top of the window)
Splitted windows (we already seen these with the :help command)
Always, when we create or open a new file, a new buffer is created.
If we close with :q any of them, they will not be erased from the
buffer stack, they will only become hidden. Keep it in mind.

BUFFERS (it gets a little confusing)

We can open many files to edit. The first way to do it is with
:edit <file> <file> <file>
It will open a buffer with all those files within.
An alternative is
:e <file1> <file2> <file3>
It's the same to run
:vim <file1> <file2> <file3>
If you want to edit another file, but not write the changes in the current file yet, you can make it hidden:
:hide edit foo.txt
To see all current buffers
:buffers (:ls is an alternative way)
To open a new Buffer
:badd - stands for buffer add
To close the current Buffer
:bdelete (:bd is an alternative way)
To change to next Buffer
:bnext (:bn is an alternative way)
To change to previous Buffer
:bprevious (:bp is an alternative way)
To change to the first/last Buffer
:bfirst (:bf is an alternative way)
:blast (:bl is an alternative way)
To open all buffers (it will split the windows)
:ball
(When into splitted screen after :ball you can quit the windows with
:q1, or :q2, etc. The number will be the number of the buffer, you can
see them with :buffers or :ls. Yet, the same can be done with :bdelete)
you can see what files you're editing by
:args
and can choose another collection of files to edit without exiting vim with
:args <file1> <file2> <file3>

REGISTERS

If we want to copy several pieces of text we can use registers to do it
"fyas - "f: register f; yas: yank a sentence
or
"l3Y - "l: register l; 3Y: yank 3 lines (same as 3yy)
(it works with visual mode, don't worry)
and then just copy the pieces
"fp
"lp
We can also append to a register. To append to a "f" register we use the uppercase letter
Ctrl+V jjlll "Fy
To paste we use the same commands
"fp
As we're talking about copying files, lets see a tip with :write command
:write >> logfile
It will append to the end of the file your whole file or the selected text

TABS (they're easy once you know how to manage buffers)

To understand what they are, you can think on it like a way to overlap windows.
You can open a new tab with
:tabnew
And close the current tab with
:tabclose
And change beetween tabs with
:tabnext

:tabprevious
And also go to the first and last tabs with
:tabfirst
:tablast

SPLITTED WINDOWS (they're useful af)

If you want to split screen with a new file just go with
:new
or
:vnew
It will open the new file on top position or on left position
To split the screen in two:
:split <filename>
Or as I rather do, let Ctrl + D show options
If only :split is given, it will copy the actual buffer.
a Vertical split is possible
:vert split
Or an alternative
:vsplit
To open as split an existing buffer:
:sbuffer 1 (:sb1 is an alternative way)
Or open vertically:
:vert sbuffer 2
You can also specifie where you want to split your screen
here are 8 combinations:

window	horizontal	up	-->	:topleft	split
window	horizontal	down	-->	:botright	split
window	vertical	left	-->	:topleft	vsplit
window	vertical	right	-->	:botright	vsplit
buffer	horizontal	up	-->	:leftabove	split
buffer	horizontal	down	-->	:rightbelow	split
buffer	vertical	left	-->	:leftabove	vsplit
buffer	vertical	right	-->	:rightbelow	vsplit

MULTISPLIT MANIPULATION

Ctrl+W J moves the current window to fill the bottom of the screen
Ctrl+W H moves the current window to fill left corner of the screeni
Ctrl+W K move current window to top
Ctrl+W L move current window to far right
Ctrl+W x exchange current window with its neighbour
Ctrl+W r rotates the windows (imagine with 3 opened windows)
Ctrl+W _ maximise height of current window
Ctrl+W | maximise width of current window
Ctrl+W w cycle between the open windows
Ctrl+W h focus the window to the left
Ctrl+W j focus the window to the down
Ctrl+W k focus the window to the up
Ctrl+W l focus the window to the right
Ctrl+W t focus the window on top
Ctrl+W b focus the window on bottom
What if we want to change the size of our splitted screen?
We can do it with the binding:
Ctrl+W + to make the screen grow
Ctrl+W - to make the screen shrink
Ctrl+W = to make screens equally sized
We can do it also with
:resize <new size>
or
:vertical resize <new size>
GoldTip: You can open a split with a terminal inside it
:term
Once you're at the terminal, to do the : commands you'll need to press "Ctrl+W :"
When we're doing some version of our code often we'll need to see the difference
$ vimdiff <file> <file>
or
$ vim -d <file> <file>

then we can perform
:diffsplit
or even
:vert diffsplit
They will be with scroll binded, to remove it you can
:set noscrollbind
or you can put it back with
:set scrollbind
We can update the diff
:diffupdate
And jump beetween changes with
[c - Jumps to previous change
]c - Jumps to next change
We can apply changes in current diff window with
:diffget
and aply changes from current pane to another with
:diffput

FOLD ACTION

When we're programming sometimes we want to make that portion of code
a little smaller. We could fold these line up to a single line, and
since it's a very cool function to have on a text editor we'll have it
here on vim.
We make it enable
:set foldenable
We choose the fold method
:set foldmethod=indent
Other acceptable values are: marker, manual, expr, syntax, diff, instead of indent.
And if we don't want this funcionality anymore we can run
:set nofoldenable
The common keys to manage folds are:
zc - close fold
zo - open fold
zM - close all folds
zR - open all folds
za - open/close folds

MACROS

To start recording a macro just push the q key with some letter, and start doing stuff.
It will record every move and you'll be able to redo every step.
To stop recording just press q again:
qa
Ctrl+V jjjy
q
How to execute the saved commands:
@a
You can combine it with numbers to re-run, and save up to 26 collections of commands.
Be careful, the same registers used to save and delete lines are used to save macros

SOME SETTINGS

You should see the
:options
to get some configuration examples
After ":set", sometimes it will be needed to do
:source %
to make these settings to take effect
Every :set will only work on this run of vim's editor, and if we want to make it
permanent we can configure a file .vimrc and let it on our ~/ directory
:set spell - make the word correction active
:set spelllang=en - makes the language english (pt-br for brazillian portuguese)
If we take a look at the already setup .vimrc, we'll find that we can remap our key
bindings with nnoremap, and also make some commonly used commands to work with
these
key bindings, like :nohlsearch
The vim runtime directory is in:
It comes from "echo $VIMRUNTIME"
/usr/share/vim/vim82

Where can you find plugins?
You should ask yourself, does vim needs plugins?
$VIM/vimfiles/pack/dist/opt
$VIMRUNTIME/plugin
$VIMRUNTIME/macros

Standard plugins ~

|pi_getscript.txt|Downloading latest version of Vim scripts
|pi_gzip.txt| Reading and writing compressed files
|pi_logipat.txt| Logical operators on patterns
|pi_netrw.txt| Reading and writing files over a network
|pi_paren.txt| Highlight matching parens
|pi_spec.txt| Filetype plugin to work with rpm spec files
|pi_tar.txt| Tar file explorer
|pi_vimball.txt| Create a self-installing Vim script
|pi_zip.txt| Zip archive explorer

THE TRUE POWER OF MARKS

We saw that they are used to store text, our macros, but here's a thing:
It can store our sessions.
When we exit a vim it creates a mark, it gets stored on the 0-8 mark names
We can see some of the old files with
:oldfiles
and edit them with
:e #<1
And it works with even older files
You can use another way
:browse oldfiles
and it will lead you to choose the file you want

WORKING WITH SESSIONS

Sessions keeps file list, windows layout, global variables, options and others.
How to create a session?
:mksession vimbook.vim
How to restore a session?
:source vimbook.vim
If you want to start vim and restore a specific session?
vim -S vimbook.vim
It can be used to save the layout (uhul)

WORKING WITH VIEW

Imagine it as a smaller version of sessions. It stores single windows.
:mkview
:loadview
There's more about views, they can store up to 10 versions of a file, and can be stored with specific names.

MODELINES

To make a option to run only on a specific file you can setup modelines
/* vim:set shiftwidth=4: */
Put this on the first or last five lines in the file.
You'll also want to put on your .vimrc the line
:set modelines=5
to make vim inspect the first and last 5 lines to find modelines.
This is the format for modelines:
any-text vim:set {option}={value} ... : any-text

VIM HAS A BROWSING TOOL

Forget nerdtree, we will use
:edit .
It opens up a file browser that shows you many things.
Press <F1> to get help on the things you can do in the netrw file browser.
When on the cursor atop a filename you can:

<enter>	Open the file in the current window.	|netrw-cr|
O	Horizontally split window and display file	|netrw-o|
V	Vertically split window and display file	|netrw-v|
p	Use the |preview-window|	|netrw-p|
P	Edit in the previous window	|netrw-P|
T	Open file in a new tab	|netrw-t|
X	Executes the file under the cursor	

COMMANDS DIRECTLY FROM VIM
Its exaustive to keep writing paths from our local directory
we can make it easier with
:cd
:pwd
THERE ARE ENCRYPTION ON VIM
Just acknowledge that it exists
COMPLETION
When on insert mode press Ctrl+P and it will sugest you some words
YANK ABOVE AND BELOW
Ctrl + Y duplicates the characters to a line above
Ctrl + E duplicates the characters to a line below
Example: Create a line like

this is a line of text

Then move to the first character, press "o" to create a new line and, on insert mode, press Ctrl + Y
ABBREVIATIONS
You can create abrr's to you text
:iabbrev ad advertisement
You can unabbreviate
:unabbreviate ad
You can remove all abbr's
:abclear

MY PERSONAL KEY BINDINGS

- <Space> opens and closes folds
- \<Space> turns off the highlight after a search
- .. and ,, changes the vertical size of splitted windows
- \\ and ;; changes the horizontal size of splitted windows
- Macro @i predefined to start a basic project window
- \c creates a new tab :tabnew
- \x goes to the next tab :tabnext
- \s goes to the previous tab :tabprevious
- \z closes a tab :tabclose
- \f goes to the first tab :tabfirst
- \l goes to the last tab :tablast
- \, changes the position of the tab to the left
- \. changes the position of the tab to the right