Vim !

Πίνακας περιεχομένων

vim

Vim is the editor of choice for many developers and power users.

It's a *modal* text editor based on the vi editor.

It inherits the key bindings of vi, but also adds a great deal of functionality and extensibility that are missing from the original vi.

vim modals:

insert mode and **command** mode.

vim **starts** in **command mode**.

- command mode
    - In this mode, characters you type is interpreted as a command
- insert mode
    - in insert mode, the characters you type is just inserted.

To **enter insert** mode, press **i**. To get back in command mode, press **esc**

All editors have a command and an input mode.

Most editors default to the input mode whithin which the text we type is interpretted as a raw string to be appendeed to the cursors position.

To "enter" command mode we the have to use some king of control key/button. Thus the command mode is disguised and we only use it briefly.

Vim on the other hand has **three distinct** modes:

- normal mode: The COMMAND mode. Here we can insert commands that can perform all sorts of tasks just like the buttons we mentioned before. The difference is that vim will stay in normal mode unsless we tell it to enter another mode. This alowes for faster commands (no need for menus and mouse clicks) but also for more complicated commands (combos).
- insert mode: the mode to INSERT text like in any other editor
- visual mode: this mode exists in all editors, although it usually is combined into insert mode. Here we can do visual tasks like selecting text or highlighting regions.

1. A simple workflow example with the vim editor.

Lets suppose we want to create a simple program that will print out "Hello world".

Like any good programmer we want our headder declarations seperatelly from our functions.

So lets create our starting file "myprint.h" using the vim <path to file> command

in our case

```
vim ./myprint."h
```

That should have created the file and opened it inside the vim editor.

We now are in command mode, but since we want to create some code we want to enter edit mode. We can do that by pressing the '**i**' key.

Now lets add some text. Folowing our example we have:

```
#include <string>
#include <stdio.h>

class MyPrint
    {
        public:
            std::string message="Hello World";
            int PrintMessage();
    };
```

Ok our headder is now ready. Now we have to implement the "PrintMessage" function.

Hence, lets create the source file. To do that we could exit vim and simply repeat the above steps but that would be really impracticall. And vim is all about practicallity so there ought to be a simpler way!?

Of course there is! We can simply open a new tab from whithin the existing editor!!!

To do that simply type the command "tabnew <path to file>" in command mode, in our case "tabnew ./myprint.cc".

Remember commands, unlike hotkeys, need a colon (**:**) at the beggining. So litterally speaking our command would be "**:tabnew ./myprint.cc**"

Oops! I think I forgot how I named my function! I will have to switch to the other tab again to look at the header.

To do that i can simply hit "gt" in command mode. This hotkey rotates through the tabs clockwise.

If I wanted to rotate anticlockwize i could use **"Gt"** but since i only have two that would be kinda stupid.

Like in this one, in most cases hitting shift along with the command does the exact opposite.(**gt** moves right, **Gt** moves left)

For example '**o**' inserts an empty line underneath the one we are on.'**O**' will do the the opposite, it will insert a blank line above.

the 'o'/'O' command will also initiate insert mode so we are ready to rock in our new line!

But now lets get back to our program.

I now finally remember the name of the function so I will go ahead and write my implementation as follows:

```
#include <myprint.h>

MyPrint::MyPrint()
    {
    }

int MyPrint::PrintMessage(void)
    {
        printf("%s",this->message.c_str());
    return 0;
    }
```

But I now realized I forgot to declare my constructor in my header file.

Well instead of writing the same thing again we can simply copy it right?

To copy move to the desired line and hit "**yy**" in command mode. NO that is not a typo, hit "y" two times!

Now move to the desired line inside the header file and press "**p**". This will insert the line just below the cursor.

After pasting our header should look something like this:

```
#include <string>
#include <stdio.h>

class MyPrint
    {
        public:
            std::string message="Hello World";
            MyPrint::MyPrint()
            int PrintMessage();
    };
```

YES! This this code was obviously copied and pasted from above using the same hotkey you used right now!!!

Now we can simply delete the parts we dont need and our coding is done for today!

```
#include <string>
#include <stdio.h>

class MyPrint
    {
        public:
            std::string message="Hello World";
            MyPrint()
            int PrintMessage();
    };
```

Lets simply use the "**w**" command for writing the file and the "**q**" command for quitting the app. (To write/quit multiple tabs just append "all" for example ***:wall*** or ***:qall***)

> If you would like to execute a system command (like mv or more likely git) you can do that from whithin vim by adding a "**!**" at the beggining.

For example:

```
:!git commit -m"my program is ready!"
```

2. The vimrc

2.1. settings

Vim, like all editors has different configuration options for the user to use (or not to use!);

These are written and saved whithin the "**vimrc**".

Typically this file resides inside the home directory of a user like so "~/**.vimrc**".

Here we can add all sorts of settings like colorschemes or line numbers, or even more breaking changes like changes to visual or normal mode.

> ~ = home directory [tilde]. This corresponds to the **$HOME** internal variable.
>
> **$HOME** is an environment variable that contains the location of your home directory, usually /home/$USER.
>
> The **$** tells us it’s a variable.

2.2. plugins

The real power of vim shows when we add some basic plugins, like a linter, an autocomplete engine and a good colorscheme!

All of that can be done using a simple tool called pluggin manager!

There are many filemanagers out there. One of the most poppular and the one we use is "**Plugged**".

For a fully loaded vimrc and an activelly maintained and updated repo please visit https://gitlab.com/dianshane/vim

You can also reffer there for any questions if you would like something explained.

To download plug and find out how to install plugins you can follow the instructions found here.

3. Vim Key Bindings

3.1. Move Bindings

Table 1. Key combinations

Key	Action	Followed by
b	back word	
e	end of word	
f	find character after cursor in current line	
m	mark current line and position mark character	tag (a-z)
n	repeat last search	
t	same as "f" but cursor moves to just before found character	character to find
w	move foreward one word	
z	position current line	CR = top; "." = center; "-"=bottom
B	move back one Word	
E	move to end of Word	
F	backwards version of 'f'	character to find
G	goto line number prefixed, or goto end if none	
H	home cursor - goto first line on screen	
J	join current line with next line	
L	goto last line on screen	
M	goto middle line on screen	

Key	Action	Followed by
N	repeat last search, but in opposite direction of original search	
T	backwards version of "t" character to find	
W	foreward Word	
0	move to column zero	
!	shell command filter cursor motion command, shell command	
$	move to end of line	
%	match nearest [],(),{} on line, to its match (same line or others)	
^	move to first non-whitespace character of line	
(	move to previous sentence	
)	move to next sentence	
|	move to column zero	
-	move to first non-whitespace of previous line	
+	move to first non-whitespace of next line	
[	move to previous "{…}" section	"["
]	move to next "{…}" section	"]"
{	move to previous blank-line separated section	"{"
}	move to next blank-line separated section	"}"
'	move to marked line, first non-whitespace	character tag (a-z)
`	move to marked line, memorized column	character tag (a-z)
/	search forward	search string, ESC or CR
?	search backward	search string, ESC or CR
^D	down half screen	
^J	line down	

Key	Action	Followed by
^T	go to the file/code you were editing before the last tag jump	
^U	up half screen	

3.2. All

Allmost complete key binding reference

Definitions

- **word** - a lower-case word ("w", "b", "e" commands) is defined by a consecutive string of letters, numbers, or underscore, or a consecutive string of characters that is not any of {letters, numbers, underscore, whitespace}
- Word - an upper-case word ("W", "B", "E" commands) is a consecutive sequence of non-whitespace.
- **cursor motion command** - any command which positions the cursor is ok here, including the use of numeric prefixes. In addition, a repeat of the edit command usually means to apply to the entire current line. For example, "<<" means shift current line left; "cc" means replace entire current line; and "dd" means delete entire current line.

Key Bindings in Editing Modes - While in any edit mode (insert, replace, etc.) there are some keys that are used to adjust behaviour, rather than just to insert text.

```
ESC - leave edit mode, return to command mode
^D - move line backwards one shiftwidth. shiftwidth must be set, and
either the line must be newly added, or ^T must have been used.
^T - move all after cursor forwards one shiftwidth
```

Table 2. Key combinations

Key	Action	Followed by
a	enter insertion mode after current character	text, ESC
b	back word	
c	change command cursor	motion command
d	delete command cursor	motion command
e	end of word	
f	find character after cursor in current line	

Key	Action	Followed by
i	enter insertion mode before current character	text, ESC
m	mark current line and position mark character	tag (a-z)
n	repeat last search	
o	open line below and enter insertion mode	text, ESC
p	put buffer after cursor	
r	replace single character at cursor	replacement character expected
s	substitute single character with new text	text, ESC
t	same as "f" but cursor moves to just before found character	character to find
u	undo	
w	move foreward one word	
y	yank command	cursor motion command
z	position current line	CR = top; "." = center; "-"=bottom
A	enter insertion mode after end of line	text, ESC
B	move back one Word	
C	change to end of line	text, ESC
D	delete to end of line	
E	move to end of Word	
F	backwards version of "f"	character to find
G	goto line number prefixed, or goto end if none	
H	home cursor - goto first line on screen	
I	enter insertion mode before first non-whitespace character	text, ESC
J	join current line with next line	
L	goto last line on screen	
M	goto middle line on screen	
N	repeat last search, but in opposite direction of original search	

Key	Action	Followed by
O	open line above and enter insertion mode	text, ESC
P	put buffer before cursor	
Q	leave visual mode (go into "ex" mode)	
R	replace mode - replaces through end of current line, then inserts	text, ESC
S	substitute entire line - deletes line, enters insertion mode	text, ESC
T	backwards version of "t" character to find	
U	restores line to state when cursor was moved into it	
W	foreward Word	
Y	yank entire line	
0	move to column zero	
1-9	numeric precursor to other commands [additional numbers (0-9)] command	
!	shell command filter cursor motion command, shell command	
@	vi eval buffer name (a-z)	
$	move to end of line	
%	match nearest [],(),{} on line, to its match (same line or others)	
^	move to first non-whitespace character of line	
&	repeat last ex substitution (":s … ") not including modifiers	
(	move to previous sentence	
)	move to next sentence	
|	move to column zero	
-	move to first non-whitespace of previous line	
_	similar to '^' but uses numeric prefix oddly	

Key	Action	Followed by
+	move to first non-whitespace of next line	
[	move to previous "{...}" section	"["
]	move to next "{...}" section	"]"
{	move to previous blank-line separated section	"{"
}	move to next blank-line separated section	"}"
;	repeat last "f", "F", "t", or "T" command	
'	move to marked line, first non-whitespace	character tag (a-z)
`	move to marked line, memorized column	character tag (a-z)
:	ex-submode ex command	
"	access numbered buffer; load or access lettered buffer	1-9,a-z
~	reverse case of current character and move cursor forward	
,	reverse direction of last "f", "F", "t", or "T" command	
.	repeat last text-changing command	
/	search forward	search string, ESC or CR
<	unindent command	cursor motion command
>	indent command	cursor motion command
?	search backward	search string, ESC or CR
^D	down half screen	
^G	show status	
^J	line down	
^T	go to the file/code you were editing before the last tag jump	
^U	up half screen	
^Z	suspend program	
^[	(ESC) cancel started command; otherwise UNBOUND	

Key	Action	Followed by
^\	leave visual mode (go into "ex" mode)	