reference section for more brilliant intro articles.

Welcome to the intermediate users. These are some cool capabilities of VI(m) that I wasn't aware of. But now I use them daily to be more productive.

For those of you who prefer TL;DR:

- tab-pages
- sessions
- line numbers (+ marks) and copy/paste
- folds
- indention with `=`
- insert-completion
- netrw
- splits/windows
- `:!` and a little bit about `:make`

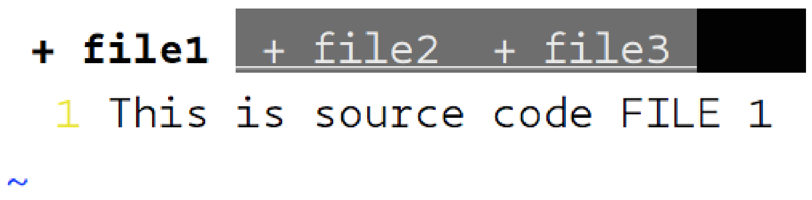
Vim tab-pages

Did you mention tabs in Vim? I didn't know that existed!

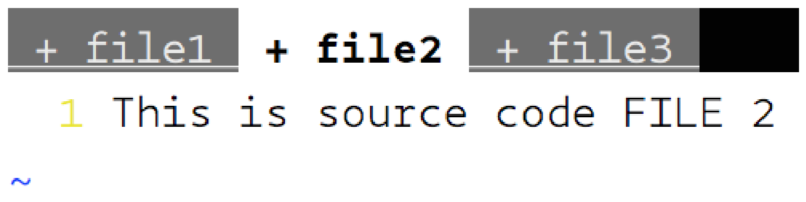
I know, right! A tab page is a page with one or more windows with a label (aka tab) at the top.

If you are interested in knowing more about windows, buffers, tab pages: technical details

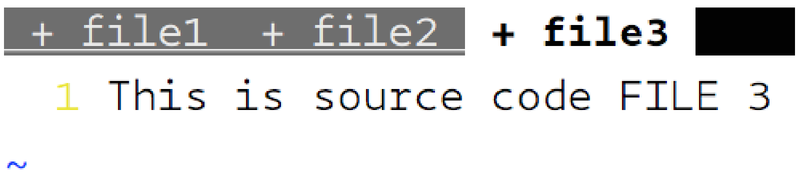
Have a look:



```
+ file1  + file2  + file3
1 This is source code FILE 1
~
```



```
+ file1  + file2  + file3
1 This is source code FILE 2
~
```


```
 + file1  + file2  + file3
  1 This is source code FILE 3
~
```

Vim tabs in action

Steps:

- Open Vim with any file or just Vim: `$ vim file1`
- Type the contents of file and get into command mode (Press `Esc`)
- `:tabedit file2` , will open a new tab and take you to edit `file2`
- `:tabedit file3` , will open a new tab and take you to edit `file3`
- To navigate between these tabs, you can be in normal mode and type : `gt` or `gT` to go to next tab or previous tab respectively. You can also navigate to a particular index tab (indexed from 1) using `{i}gt` where, i is the index of your tab. Example: `2gt` takes you to 2nd tab
- To directly move to first tab or last tab, you can enter the following in command mode: `:tabfirst` or `:tablast` for first or last tab respectively. To move back and forth : `:tabn` for next tab and `:tabp` for previous tab
- You can list all the open tabs using : `:tabs`
- To open multiple files in tabs: `$ vim -p source.c source.h`
- To close a single tab: `:tabclose` and to close all other tabs except the current one: `:tabonly` . Use the suffix `!` to override changes of unsaved files

[DEMO] Tabs in VIM
*VIM supports tabs to open multiple files and work with them*asciinema.org

I think this feature enables us to effectively save time by sharing the buffer between tabs and enabling us to copy paste between tabs and keep multiple sessions of different tab set for category of work. Example: You can have a terminal tab with all Vim tabs of source code C files only and you can have another terminal tab with all Vim tabs of header files (.h).

Tabs provide so much convenience to keep all my files open and access them when I want. However, isn't it a pain to open all tabs every time I reboot or close and open the terminal?

Steps:

- Open any number of tabs you wish to work with
- From any tab, press `Esc` and enter the command mode
- Type `:mksession header-files-work.vim` and hit enter
- Your current session of open tabs will be stored in a file `header-files-work.vim`
- To see restore in action, close all tabs and Vim
- Either start vim with your session using : `$ vim -S header-files-work.vim` or open vim with any other file and enter command mode to type: `:source header-files-work.vim` and BOOM! All your tabs are opened for you just the way you saved it!
- If you change any session tabs (close/open new), you can save that back using : `:mks!` while you are in the session

[DEMO] Sessions in VIM
*VIM allows users to store their work sessions separately based on the projects they are working on. Users can easily…*asciinema.org

Can I copy/cut paste without having to know line numbers?

Oh Yes! Earlier I used to see the line numbers (`:set nu`) of the functions I wanted to copy/cut. Let's say I want to copy/cut lines 34 to 65. I used `:34,65y` (Copy/Yank) or `:34,65d` (Cut/Delete).

> Of course counting the lines and using `{n}yy` or `{n}dd` (where `n` is number of lines) is not an option for hundreds of lines ?

There can be some functions that span multiple pages and you don't want to go down only to forget what was the first line number. There's a simple way to achieve this without worrying anything about line numbers!

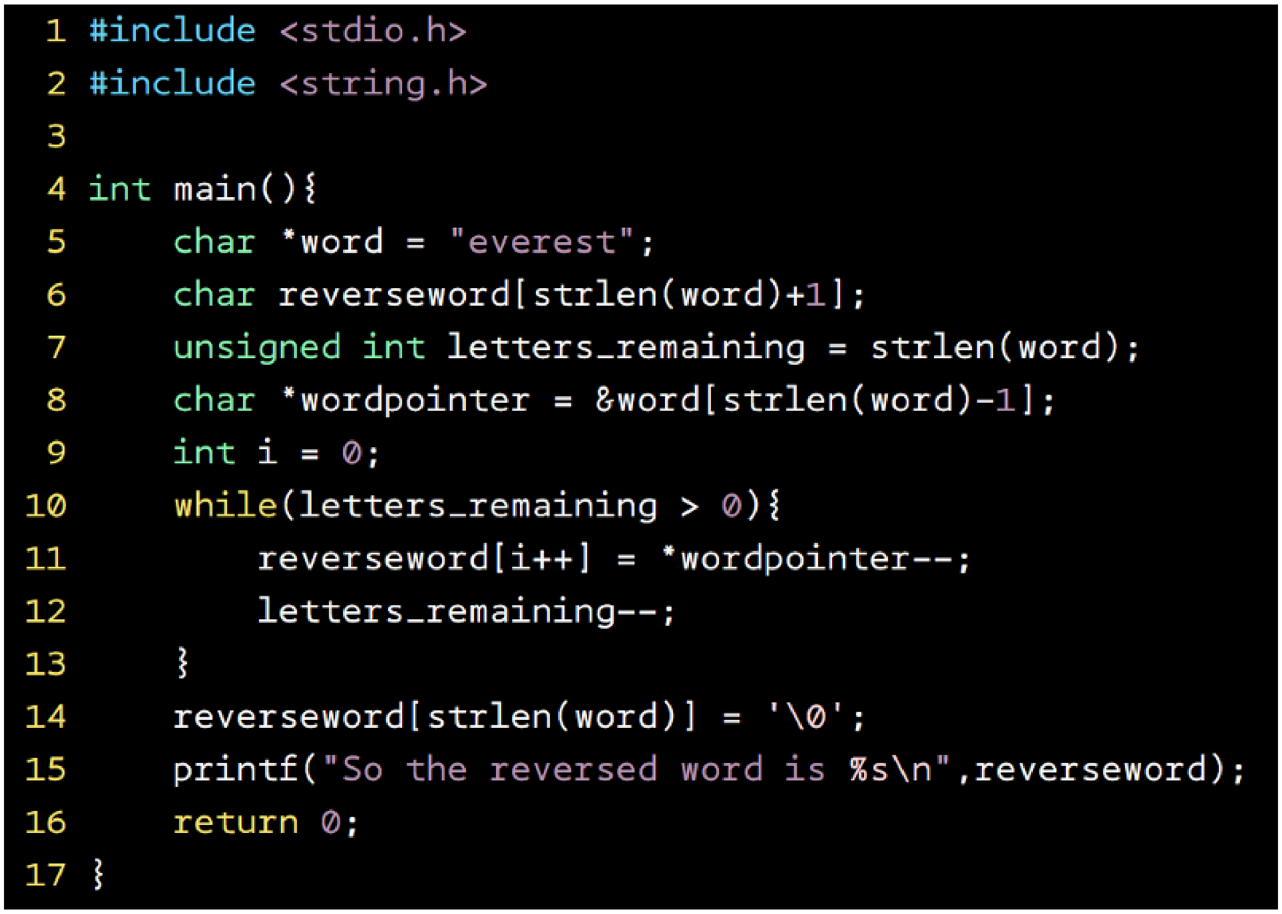
Steps:

- Enter normal mode, go to the start line
- Type `mk` (Mark point with alphabet 'k' or use any other alphabet)
- Move down (page down or whatever) and move to the end line
- `y'k` will yank/copy all the lines from start to end
- `d'k` will cut/delete all the lines from start to end

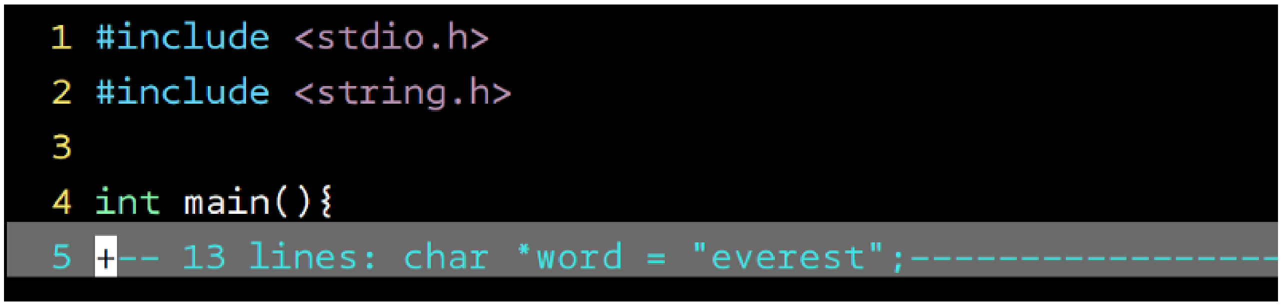
times. This may be a lot to ask because this is not an IDE but, by any chance can we fold the code blocks?

Absolutely! Let's say you want to skip remembering those line numbers and walk around with your new found love *the markers*. Go to the beginning of the function body and type `mb` . Now, just go to the end of the function body using `%` (brace matching) or any other convenient technique and press `zf'b` and you're done!

Before and after:



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

int main(){
    char *word = "everest";
    char reverseword[strlen(word)+1];
    unsigned int letters_remaining = strlen(word);
    char *wordpointer = &word[strlen(word)-1];
    int i = 0;
    while(letters_remaining > 0){
        reverseword[i++] = *wordpointer--;
        letters_remaining--;
    }
    reverseword[strlen(word)] = '\0';
    printf("So the reversed word is %s\n",reverseword);
    return 0;
}
```



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

int main(){
+-- 13 lines: char *word = "everest";-----------------
```

Before-After

If you are comfortable using the line numbers, the command is even easier to remember: `:5,16fo` (fo stands for code **fo**ld). Once you have folded your code, it's easy to toggle between open and closed views using `zo` (Open the code fold) and `zc` (Close the code fold). Don't stress it so much. Just use `za` to toggle between open and closed folds ?

Let's say you spent considerable time folding your functions in a large file, you would obviously want to retain those folds every time you open that file right? (If not, why did you waste your energy folding them!?), so there's a solution right in your `~/.vimrc` . Insert the following lines in `~/.vimrc` and your code folds are saved and restored:

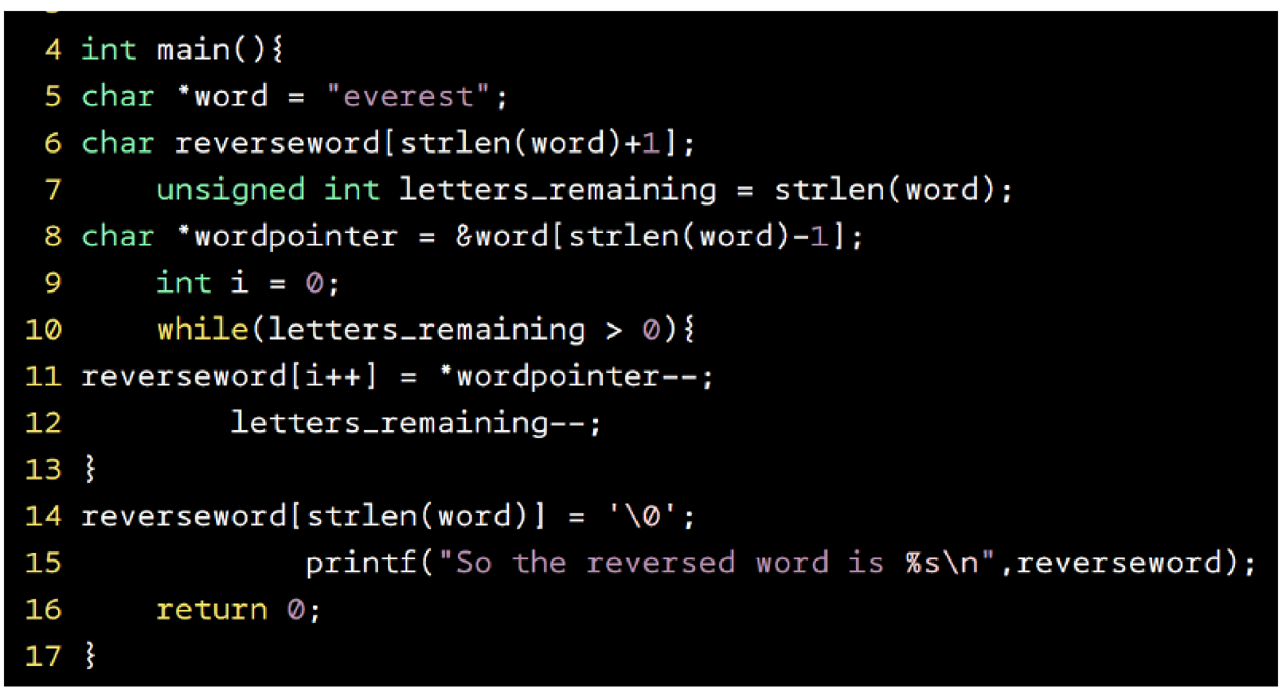
```
autocmd BufWinLeave *.* mkview
autocmd BufWinEnter *.* silent loadview
```

I'm usually careful with my indentation but sometimes, I have to edit some other idiot's source code and it bugs me to edit his/her code without indentation. Are there any magical keystrokes to make that happen?

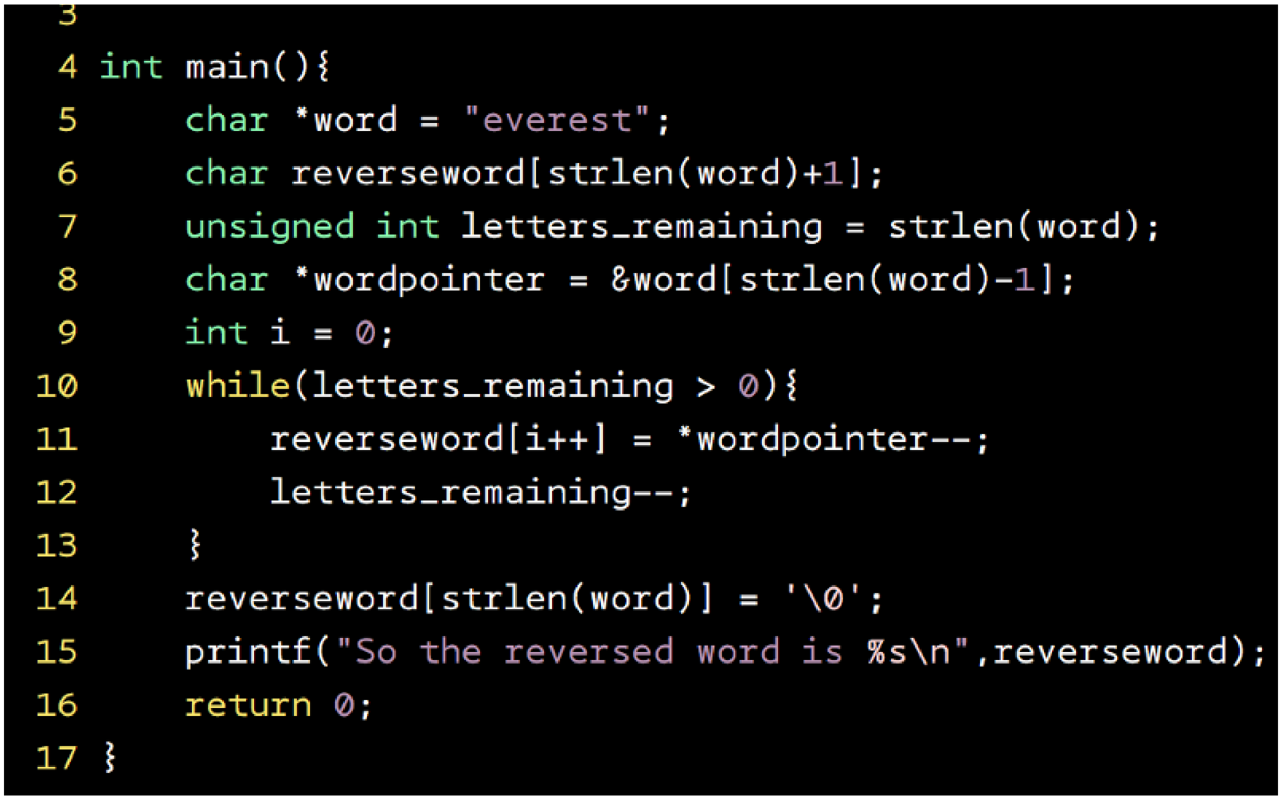
Sure! It's as simple as: `=i{` . Really that's all ! (***i*** is inner object)

[DEMO] Indentation in VIM
*VIM allows blocks of code to be indented with a few keystrokes. All you have to do is place the cursor in a block of...*asciinema.org

Before-after:



```
int main(){
char *word = "everest";
char reverseword[strlen(word)+1];
    unsigned int letters_remaining = strlen(word);
char *wordpointer = &word[strlen(word)-1];
    int i = 0;
    while(letters_remaining > 0){
reverseword[i++] = *wordpointer--;
        letters_remaining--;
}
reverseword[strlen(word)] = '\0';
            printf("So the reversed word is %s\n",reverseword);
    return 0;
}
```



```
int main(){
    char *word = "everest";
    char reverseword[strlen(word)+1];
    unsigned int letters_remaining = strlen(word);
    char *wordpointer = &word[strlen(word)-1];
    int i = 0;
    while(letters_remaining > 0){
        reverseword[i++] = *wordpointer--;
        letters_remaining--;
    }
    reverseword[strlen(word)] = '\0';
    printf("So the reversed word is %s\n",reverseword);
    return 0;
}
```

Before-After

All you have to do is place the cursor anywhere within a block you want to indent, press `Esc` to enter normal mode and then: `=i{` . Boom! Your entire function body (including inner blocks) is indented.

> NOTE: Don't expect indentation of your python files ?. It only works when Vim can identify the start and end using opening and closing parenthesis)

You can also increase/decrease the indentation within a block using : `>` ;i{ to increase a `nd` <i{ to decrease in normal mode.

I may be dreaming but (*quavering voice*), I mean I just want to give it a try, Uhmm, I may be pushing it really far with this one but (*5 second pause*)..never mind, lets move on to my next question

Vim is quite open-minded to take criticism or face the fact that it's not an IDE, go ahead, let's see what you've got.

Uhmmm, sorry but by any chance (*panting*) with any plugin or something, does vim have autocomplete like an IDE?

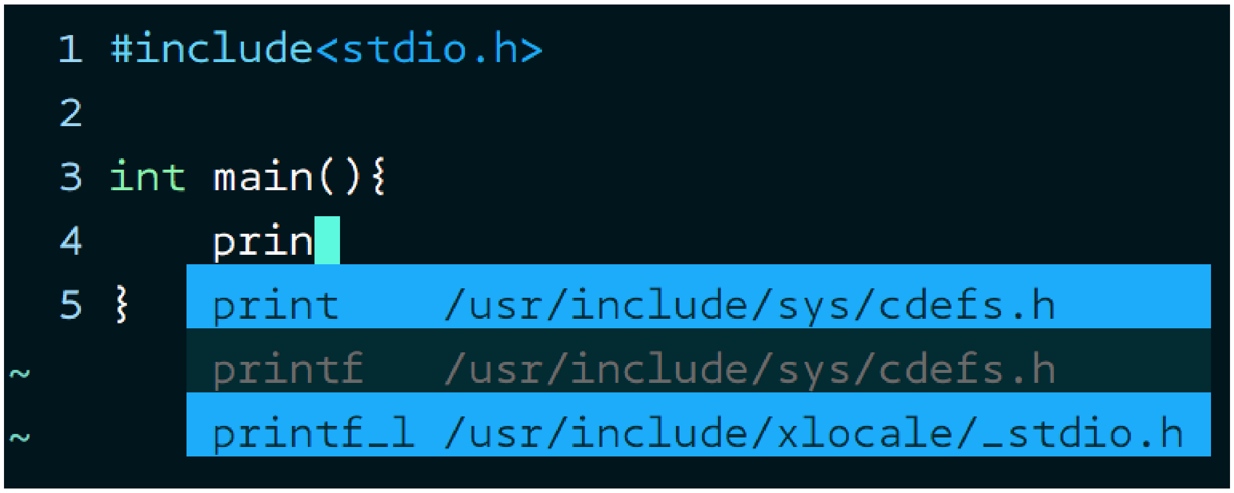
? You may be surprised but yes it does! ? and guess what…

* drum rolls *

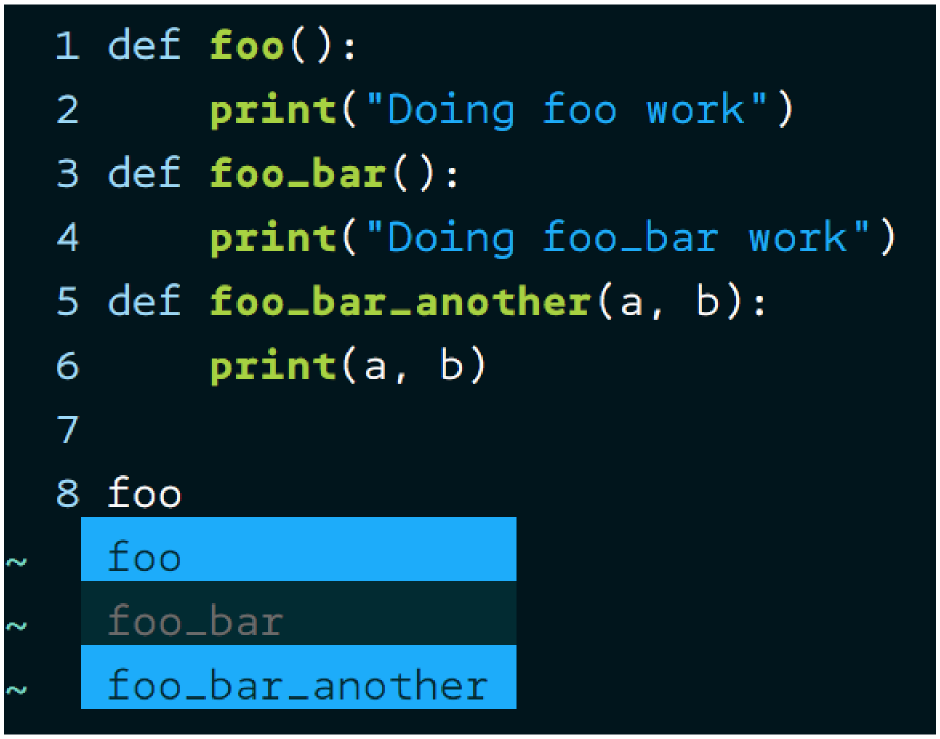
Without a plugin!

You heard me right! The only condition for Vim to show you options is “Vim should know what you’re talking about.” It could be through an included source file or defined functions or variables.

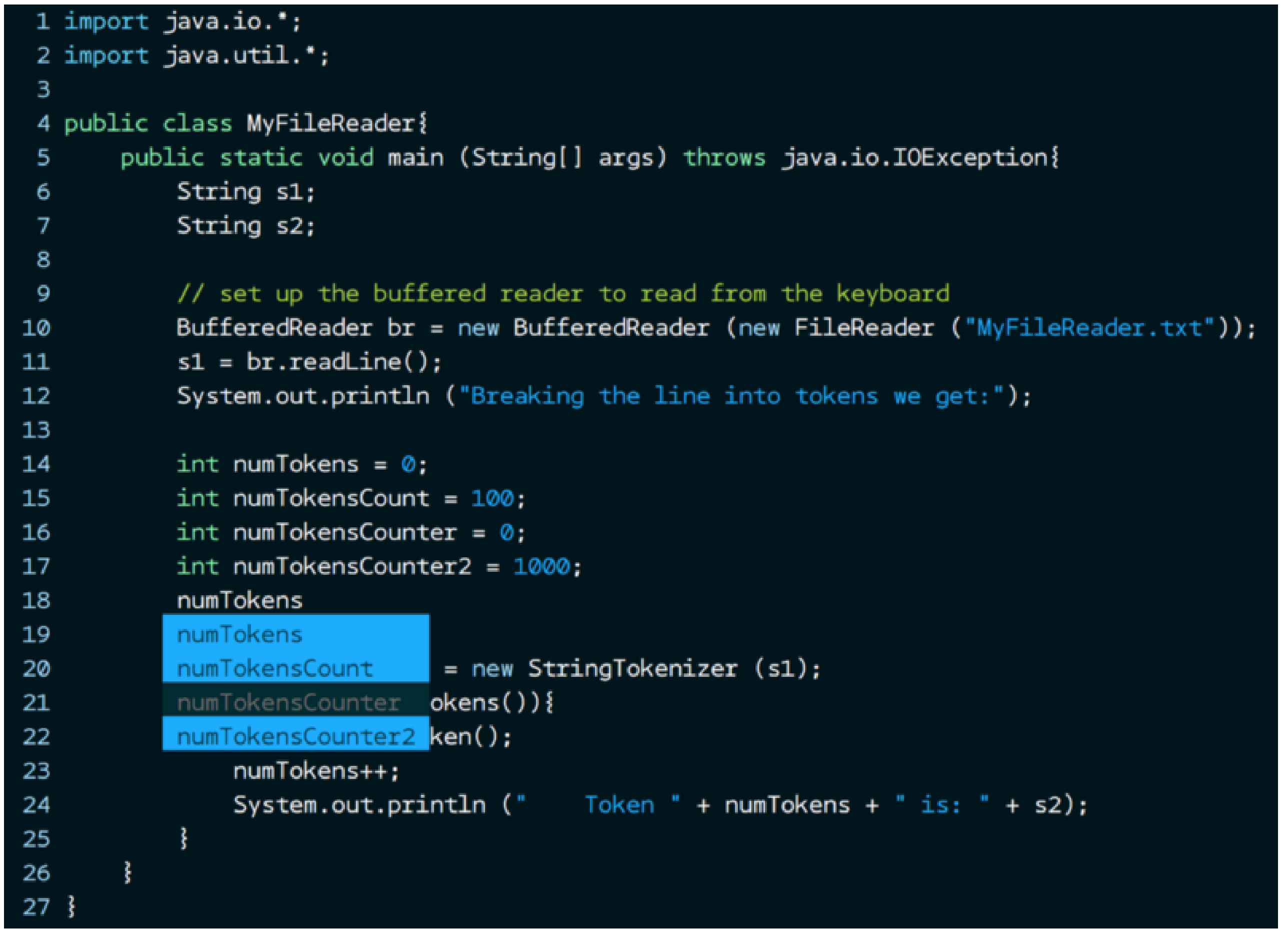
All you have to do is start typing and then press `Ctrl+n` in insert mode.



```
#include<stdio.h>

int main(){
    prin
}   print    /usr/include/sys/cdefs.h
    printf   /usr/include/sys/cdefs.h
    printf_l /usr/include/xlocale/_stdio.h
```



```
def foo():
    print("Doing foo work")
def foo_bar():
    print("Doing foo_bar work")
def foo_bar_another(a, b):
    print(a, b)

foo
foo
foo_bar
foo_bar_another
```

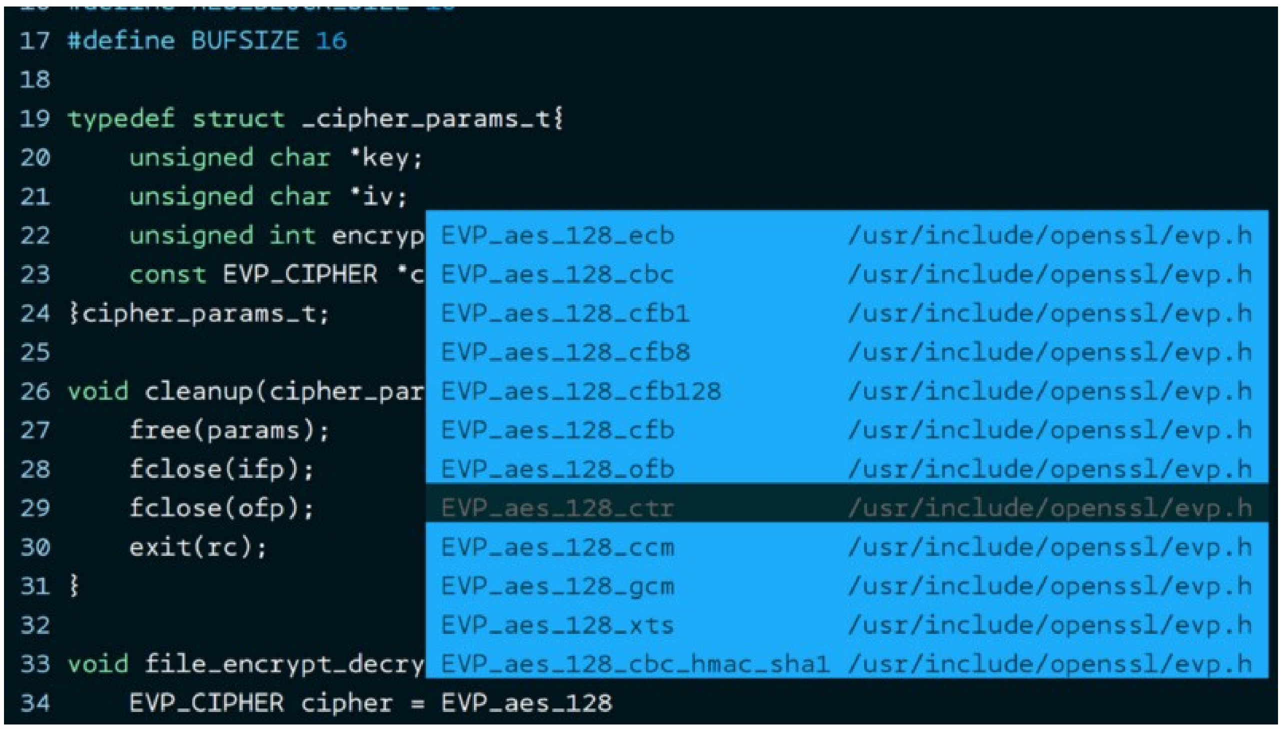


```
import java.io.*;
import java.util.*;

public class MyFileReader{
    public static void main (String[] args) throws java.io.IOException{
        String s1;
        String s2;

        // set up the buffered reader to read from the keyboard
        BufferedReader br = new BufferedReader (new FileReader ("MyFileReader.txt"));
        s1 = br.readLine();
        System.out.println ("Breaking the line into tokens we get:");

        int numTokens = 0;
        int numTokensCount = 100;
        int numTokensCounter = 0;
        int numTokensCounter2 = 1000;
        numTokens
        numTokens
        numTokensCount     = new StringTokenizer (s1);
        numTokensCounter  okens()){
        numTokensCounter2 ken();
            numTokens++;
            System.out.println ("    Token " + numTokens + " is: " + s2);
        }
    }
}
```

Examples in C, Python and Java

Just imagine the uses! Specially if you’re writing C code and you cannot recollect the exact OpenSSL library call, all you have to do is include the header!

[DEMO] Autocomplete feature in VIM
*VIM has autocomplete suggestions for keywords, function names if the appropriate header files are included or if the…*asciinema.org



```
#define BUFSIZE 16

typedef struct _cipher_params_t{
    unsigned char *key;
    unsigned char *iv;
    unsigned int encryp EVP_aes_128_ecb            /usr/include/openssl/evp.h
    const EVP_CIPHER *c EVP_aes_128_cbc            /usr/include/openssl/evp.h
}cipher_params_t;       EVP_aes_128_cfb1           /usr/include/openssl/evp.h
                        EVP_aes_128_cfb8           /usr/include/openssl/evp.h
void cleanup(cipher_par EVP_aes_128_cfb128         /usr/include/openssl/evp.h
    free(params);       EVP_aes_128_cfb            /usr/include/openssl/evp.h
    fclose(ifp);        EVP_aes_128_ofb            /usr/include/openssl/evp.h
    fclose(ofp);        EVP_aes_128_ctr            /usr/include/openssl/evp.h
    exit(rc);           EVP_aes_128_ccm            /usr/include/openssl/evp.h
}                       EVP_aes_128_gcm            /usr/include/openssl/evp.h
                        EVP_aes_128_xts            /usr/include/openssl/evp.h
void file_encrypt_decry EVP_aes_128_cbc_hmac_sha1  /usr/include/openssl/evp.h
    EVP_CIPHER cipher = EVP_aes_128
```

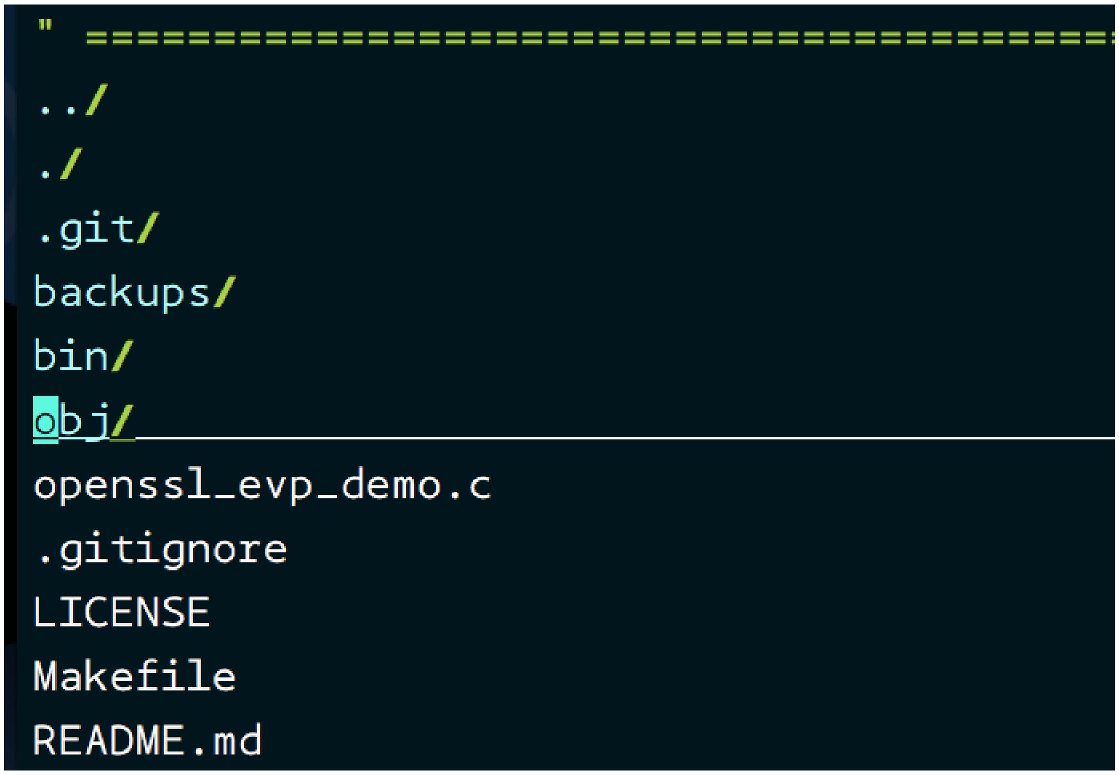
Vim auto-complete helping with OpenSSL functions

> to find them. I just use a Mac to login to a linux machine. So, if you're using Mac, sorry about that.

I understand Vim is just a text editor but if you want me to work without losing focus and without exiting Vim every now and then, what options do I have if I can't remember all the file names?

Simple, use the file explorer provided by VIM ? Yes, Vim provides a simple file explorer (*without any plugins*). Just type : `:Explore` from any Vim window and you will see an easy to navigate file explorer which can be navigated using ⬆️ and ⬇️ arrow keys. Press E `nter/Return` key to open a file/directory. Use : `q` to exit the explorer and vim. If you do not wish to quit vim and continue working with an open file, you have 3 options:

1. Open the explorer in a horizontal (`:Sexplore`) or vertical (`:Vexplore`) split and exit the explorer using `:q`
2. Open the explorer in another tabpage using `:Texplore` and exit using `:q`
3. Open file explorer in your current window and then unload the current buffer and delete it from the buffer list using `:bdel` (buffer delete).



```
" ======================================
../
./
.git/
backups/
bin/
obj/
openssl_evp_demo.c
.gitignore
LICENSE
Makefile
README.md
```

:Explore from any vim window shows the file explorer

> NOTE: You can also use the short command `:Ex` to open the file explorer

Sometimes I have to repeat same steps on some lines to edit something. I am pretty sure Vim will have some feature that enables me to do this. Am I right?

100% Right! You are talking about macros and Vim supports macros. Repeating the last executed command is simple and can accomplish simple repetitive tasks. However, if the text processing is made up of several steps to achieve a result, macros come in handy.

Consider an example C header file :

```
void encrypt_text(char *text, int bytes)
void decrypt_text(char *text, int bytes)
void process_text(char *text, int bytes)
void another_important_function(int bytes, double precision)
```

Oops! You forgot to put a semicolon at the end of each line and also you just realized that all these functions return an integer error code instead of void.

The steps you need to perform for making change in one line are:

- Place the cursor at the beginning of the word `void`
- Press `cw` in normal mode to delete the word `void` and type `int`
- Press `Esc` , move to the end of line using `Shift+a` to insert `;`
- Press `Esc` and press `^` to return to the beginning of the edited line

Resulting in:

```
void another_important_function(int bytes, double precision)
```

You can just record this sequence of steps and replay it on all 4 lines.

All you have to do is, before you start the sequence, start recording the macro in any alphabet (let's say `a`) by pressing `qa` in normal mode. Now your steps are being recorded in `a`. Once you are done with all your steps, just press `q` in normal mode. This will end the recording. To replay these steps, just keep the cursor at the same place where it was placed during macro. Press `@a` and we're done! BOOM! Vim will repeat the same steps for you on that line! To repeat it on multiple lines, you can also use `@@` after using `@a` command once

I know Vim is nowhere close to an IDE and I may be having some unreasonable hopes but just a quick question: Remote editing of files possible with Vim?

If you think of it considering the available resources:
[1] Vim
[2] openssh-client (Comes installed with most Linux flavors)

You are in luck my friend! Yes, Vim supports remote editing of files ?
Vim just utilizes the secure connection established by scp (secure copy) provided by openssh-client. There are times when you are working with files on multiple remote machines and it's a waste of time to log into a machine just to edit one single file! You can relax in your current machine if you just know your remote machine credentials and path.

```
vim scp://remoteuser@remote_IP_or_hostname/relative/path/of/file
```

For example: I need to edit a file on 10.0.18.12 stored in `/home/dev-john/project/src/main.c` and I have login credentials for `dev-john`, I can access the `main.c` using:

```
$ vim scp://dev-john@10.0.18.12/project/src/main.c
```

I can use the relative path because, I can start looking for the file from the home directory of `dev-john`

TIP: If you access a remote machine frequently, you can create an ssh config file to create a

```
Host remote-dev-machine
    Hostname 10.0.18.12
    User dev-john
    IdentityFile ~/.ssh/id_rsa
```

Now you can access your file using:

```
$ vim scp://remote-dev-machine/project/src/main.c
```

If it's confusing to remember the relative path and not intuitive, you can also specify it with an alternative:

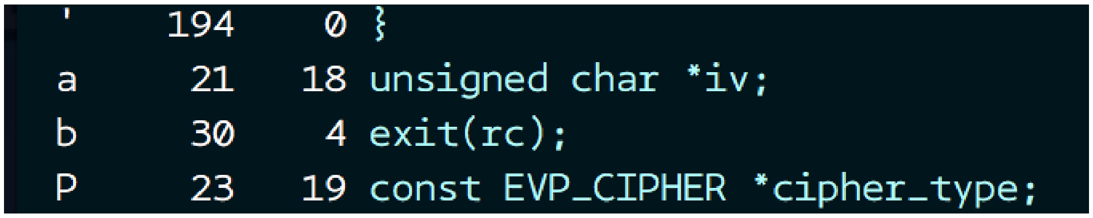
```
$ vim scp://remote-dev-machine/~dev-john/project/src/main.c
```

Awesome! I'm already thrilled at the out-of-the-box capabilities of Vim. Looks like you've a solution to a lot of common editing problems. Let's see. I have a file with over 2000 lines and the functions of my interest are located at line 9, line 768 and line 1898. I know I can jump to a line using line number but I'm not so good at remembering those numbers. Got anything for me?

Hell yeah! What you're looking for is a local bookmark solution in Vim using letters. All you have to do is :

- Place your cursor on any line at any position
- Press `Esc` to make sure you're in normal mode
- Press `m{lowercaseletter}` where `{lowercaseletter}` is any letter from `a-z`
- You just created a local bookmark to navigate in your file

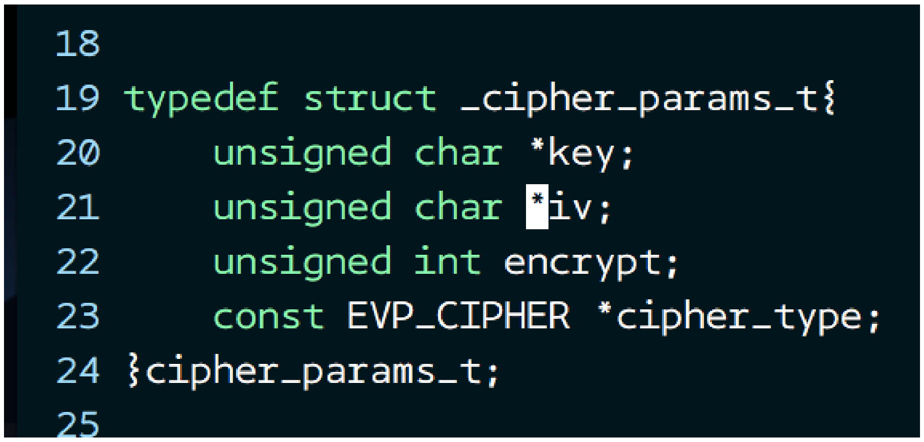
To view all your bookmarks: Press `Esc` and enter command mode, type `:marks` and hit `Enter/Return`. You'll see a list of your bookmarks. To visit any bookmark at any time, just press `Esc` and type `` `{lowercaseletter} ``. Kaboom! You'll arrive at the exact same location with cursor where you bookmarked. Example:



```
 '      194     0 }
 a       21    18 unsigned char *iv;
 b       30     4 exit(rc);
 P       23    19 const EVP_CIPHER *cipher_type;
```

Bookmarks in Vim

I have created a local bookmark to line 21, column 18 using `a` . If I'm editing something on line 1783, I would just press `Esc` and type `` `a `` :



```
18
19 typedef struct _cipher_params_t{
20       unsigned char *key;
21       unsigned char *iv;
22       unsigned int encrypt;
23       const EVP_CIPHER *cipher_type;
24 }cipher_params_t;
25
```

To solve your problem, all you've to do is create 3 local bookmarks and quickly jump to them by looking at `:marks` .

Problem solved ?

What if I told you that you can create global bookmarks too?! ? Yes, it is possible to create global bookmarks too! These are equivalent to your windows or GUI shortcuts (or linux soft/hardlinks) except you don't need to create an actual link. You heard me right! You can literally jump from editing a file in / `dir1` to another file and line in / `project/src/` from your Vim without exiting ! ?

Fret not, it's not a big new thing to remember. All you have to do is:
Use an uppercase letter instead of lower case letter to create a global bookmark. That's all! Really! You navigate to the global bookmark using the same process. Example: If you've created a bookmark using `mP` , all you've to do is press `Esc` and type `` `P `` and BAM! You jump to your global bookmark (Vim remembers the path, so you don't have to type anything about the path)

You can access the global bookmarks in the same way as local : `:marks`

```
P     53    4 ~/project/src/large-file.c
A     11    0 ~/project/README.md
```

NOTE: If you are not interested in the cursor position and just want to be there at the beginning of you bookmarked line, use `'P` instead of `` `P `` (Use a single quote instead of back tick to be positioned at the beginning of the line)

I’ve heard that Vim supports window splitting along with tabs! I understand tabs are great and you get to work with multiple open files at once. But, what about splitting? Why would I want that?

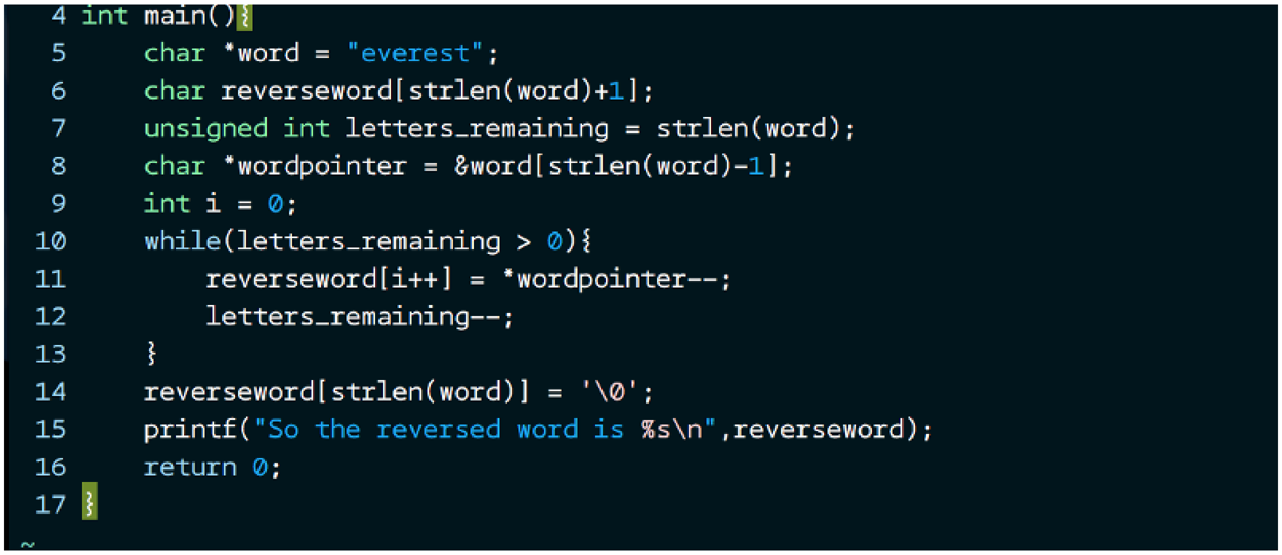
Scenarios:

- You may want to edit a file by looking at another file simultaneously (May be you are defining a C function by looking at it’s declaration in a header file)
- You may want to edit some portion of a file by looking at the top/bottom portion of the same file simultaneously
- Your work may require you to edit a file by looking at different portions of different files simultaneously

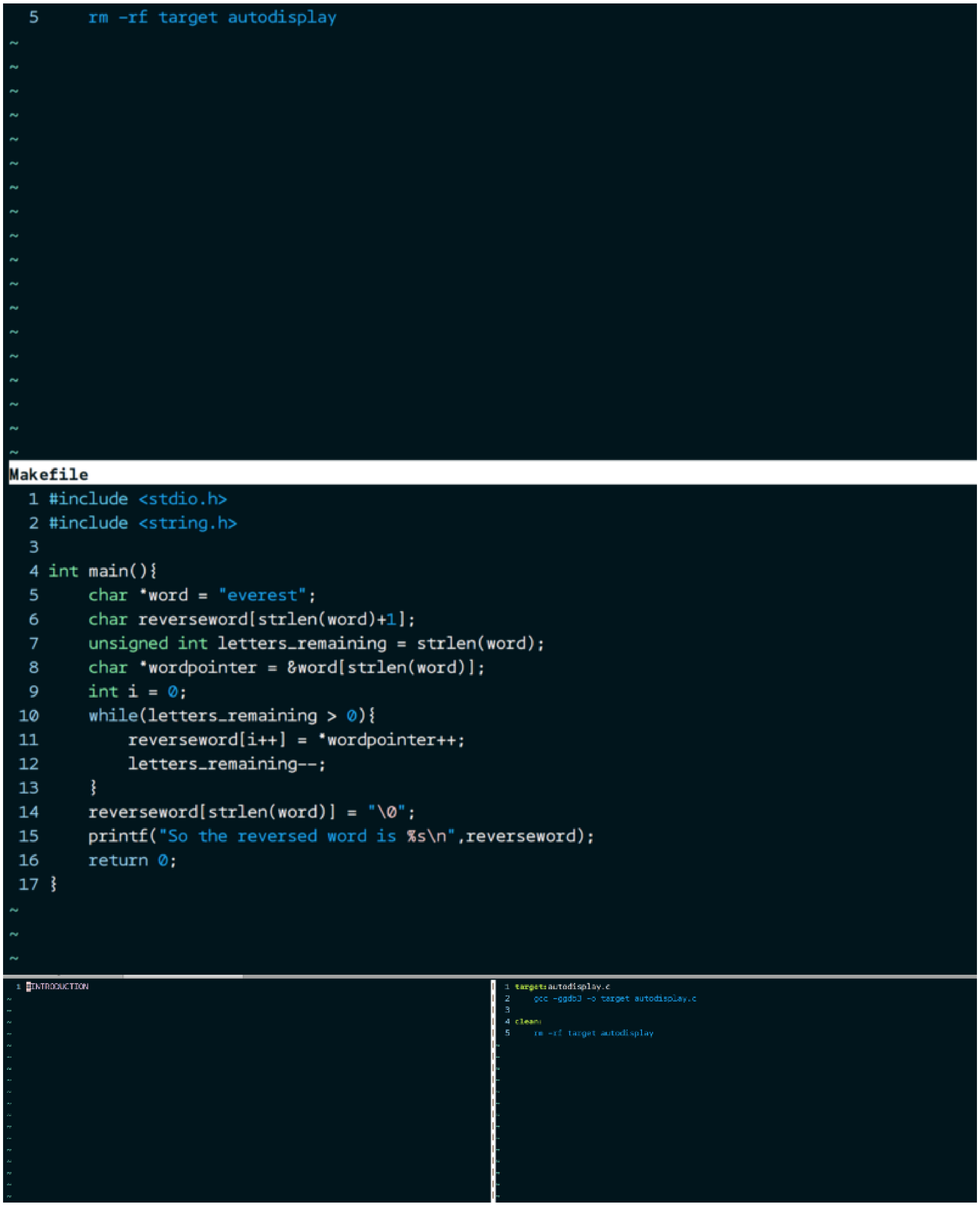
Vim supports splitting of screen both horizontally and vertically. Even better, you can even browse file system to open a file when you split your screen.

Here are the available options:

```
:split filename  - split window horizontally and load filename
:vsplit file     - vertical split and open file
ctrl-w up arrow  - move cursor up a window
ctrl-w ctrl-w    - move cursor to another window (cycle)
ctrl-w _         - maximize current window vertically
ctrl-w |         - maximize current window horizontally
ctrl-w =         - make all equal size
:sview file      - same as split, but readonly
:close           - close current window
```



```
int main(){
    char *word = "everest";
    char reverseword[strlen(word)+1];
    unsigned int letters_remaining = strlen(word);
    char *wordpointer = &word[strlen(word)-1];
    int i = 0;
    while(letters_remaining > 0){
        reverseword[i++] = *wordpointer--;
        letters_remaining--;
    }
    reverseword[strlen(word)] = '\0';
    printf("So the reversed word is %s\n",reverseword);
    return 0;
}
~
```



```
    rm -rf target autodisplay
```

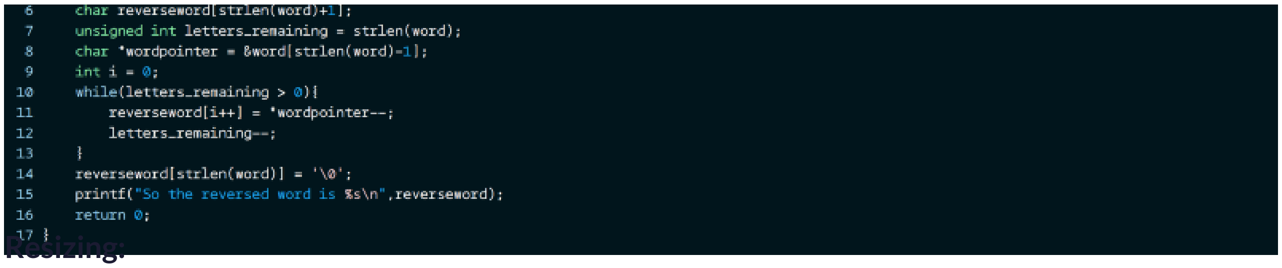
Makefile

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

int main(){
    char *word = "everest";
    char reverseword[strlen(word)+1];
    unsigned int letters_remaining = strlen(word);
    char *wordpointer = &word[strlen(word)];
    int i = 0;
    while(letters_remaining > 0){
        reverseword[i++] = *wordpointer++;
        letters_remaining--;
    }
    reverseword[strlen(word)] = "\0";
    printf("So the reversed word is %s\n",reverseword);
    return 0;
}
```

```
INTRODUCTION
```

```
target:autodisplay.c
    gcc -ggdb3 -o target autodisplay.c

clean:
    rm -rf target autodisplay
```

Maximizing a window for work:



```c
    char reverseword[strlen(word)+1];
    unsigned int letters_remaining = strlen(word);
    char *wordpointer = &word[strlen(word)-1];
    int i = 0;
    while(letters_remaining > 0){
        reverseword[i++] = *wordpointer--;
        letters_remaining--;
    }
    reverseword[strlen(word)] = '\0';
    printf("So the reversed word is %s\n",reverseword);
    return 0;
}
```

Resizing:

The pane needs to be maximized vertically and horizontally to occupy the entire window

```
CTRL-W [N] -    Decrease current window height by N (default 1)
CTRL-W [N] +    Increase current window height by N (default 1)
CTRL-W [N] <    Decrease current window width by N (default 1)
CTRL-W [N} >    Increase current window width by N (default 1)
```

Is there a way to use file explorer while I split panes? (I can't remember and type the file names always!)

Of course, all you have to do is type : `:Sexplore` for horizontal file explorer and `:Vexplore` for vertical file explorer. You can also use `:Vexplore!` to open the file explorer on right side (instead of default left)

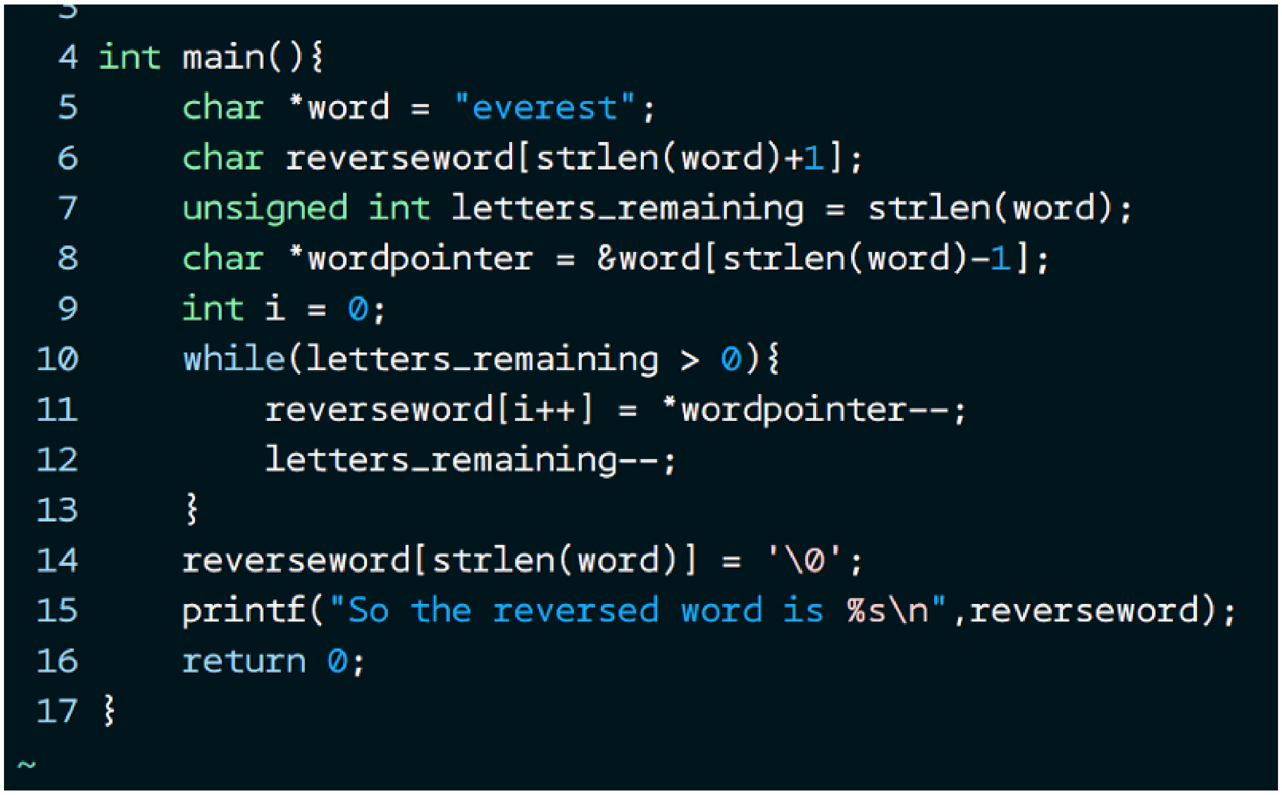
Again, all this works *without any extra plugins* ?

I am in the middle of editing some code and I quickly need to run a shell command. Should I save my work, exit Vim and run my commands? I bet there is a better way out with Vim

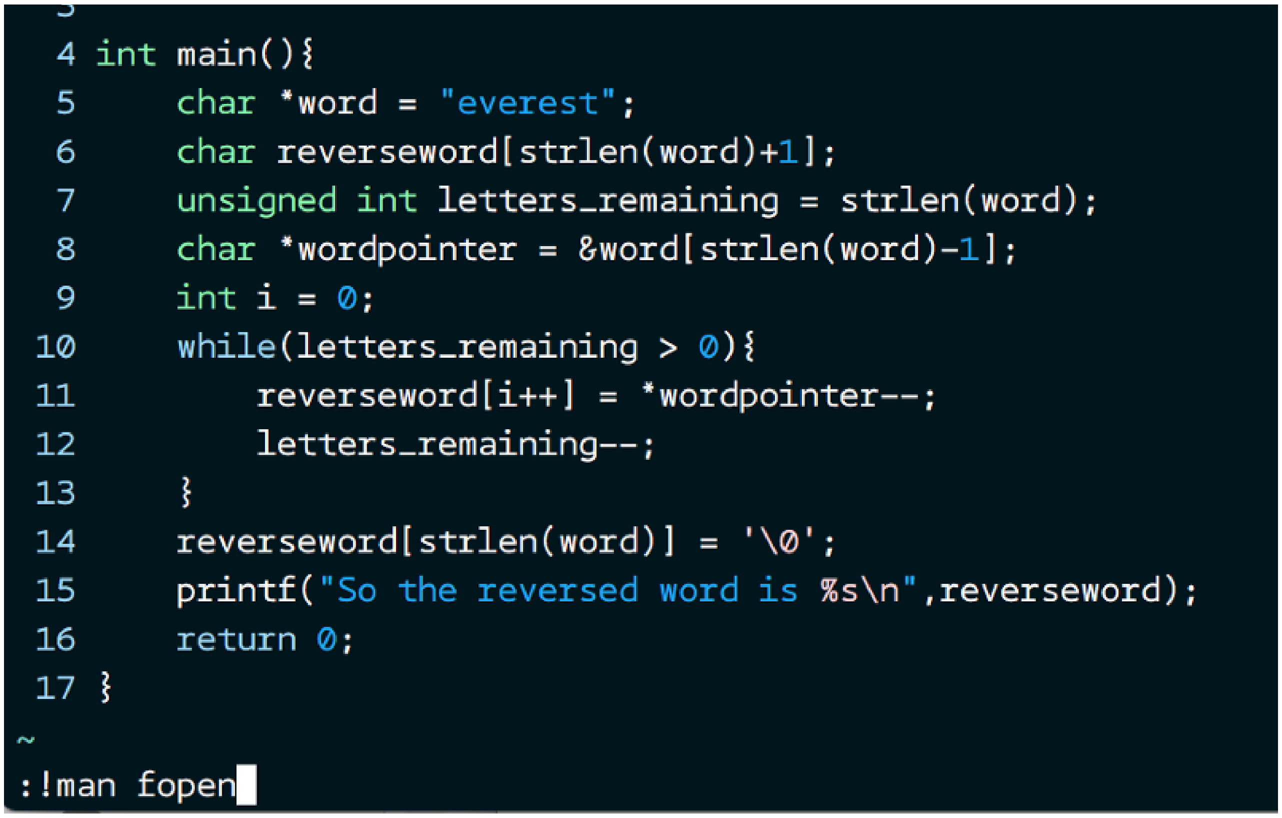
You bet! Vim just doesn't want you to leave Vim and wants you to continue focussing on your work. Hence the option to execute shell commands from within your Vim. Don't worry, all your unsaved work is not discarded, you just execute your command and BAM! you are back in your unsaved/saved file safely!

Let's say you are in the middle of a coding session and you quickly need to head out to take a look at man page of file operations because you forgot the signature! You don't have to save your work, exit Vim and then check man pages or you don't have to open another tab just for the man page. You can issue the command from right within the Vim editor.

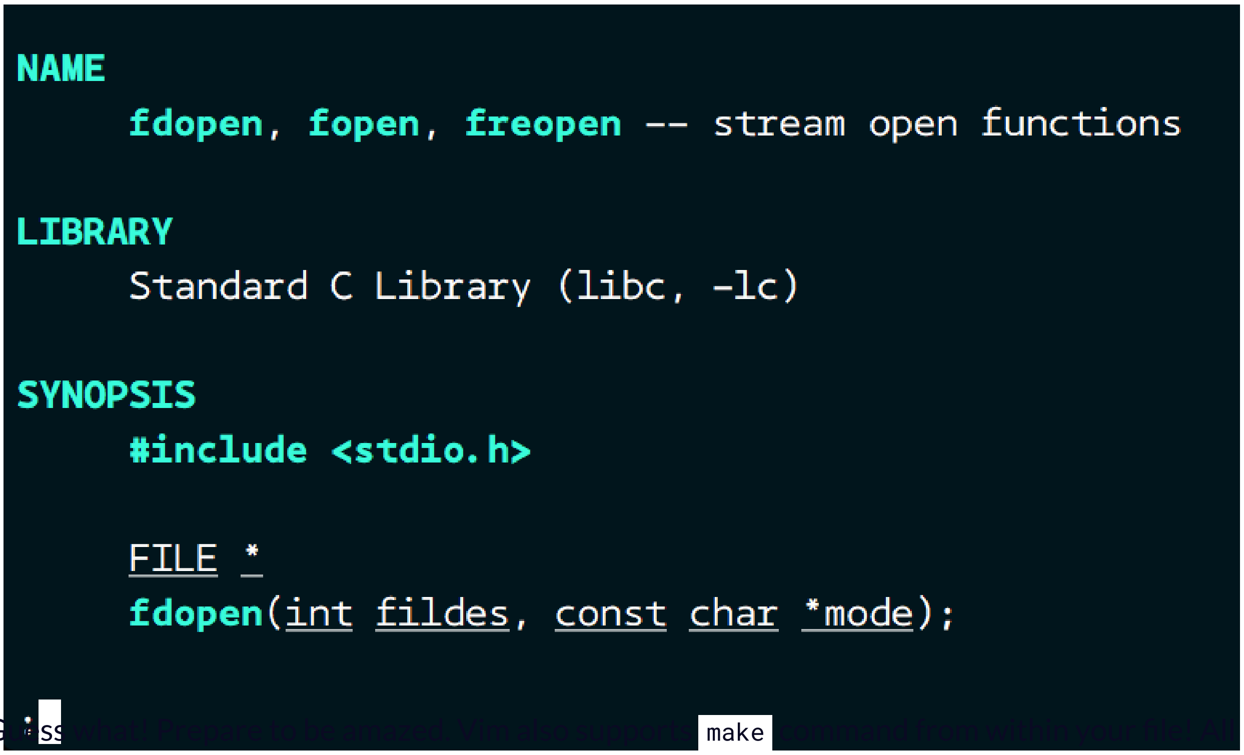
[DEMO] Unix commands from VIM
*VIM allows users to execute shell commands from within VIM without exiting. All you have to do is enter the command…*asciinema.org



```
int main(){
    char *word = "everest";
    char reverseword[strlen(word)+1];
    unsigned int letters_remaining = strlen(word);
    char *wordpointer = &word[strlen(word)-1];
    int i = 0;
    while(letters_remaining > 0){
        reverseword[i++] = *wordpointer--;
        letters_remaining--;
    }
    reverseword[strlen(word)] = '\0';
    printf("So the reversed word is %s\n",reverseword);
    return 0;
}
~
```



```
int main(){
    char *word = "everest";
    char reverseword[strlen(word)+1];
    unsigned int letters_remaining = strlen(word);
    char *wordpointer = &word[strlen(word)-1];
    int i = 0;
    while(letters_remaining > 0){
        reverseword[i++] = *wordpointer--;
        letters_remaining--;
    }
    reverseword[strlen(word)] = '\0';
    printf("So the reversed word is %s\n",reverseword);
    return 0;
}
~
:!man fopen
```



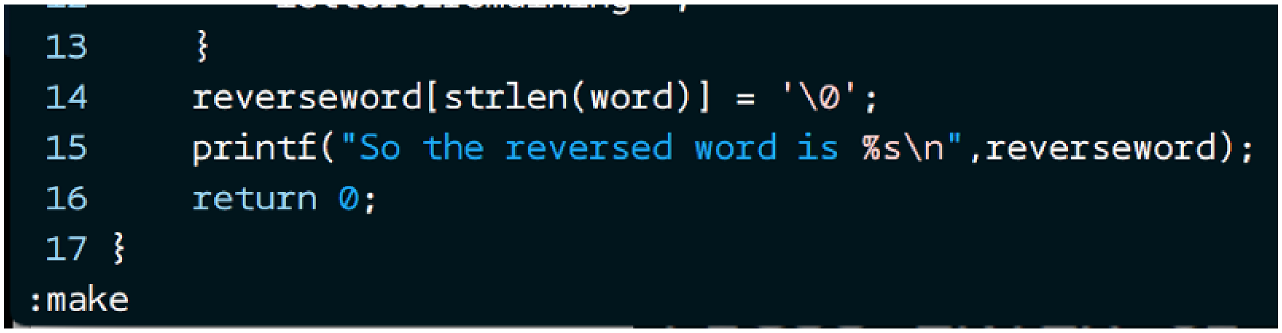
```
NAME
     fdopen, fopen, freopen -- stream open functions

LIBRARY
     Standard C Library (libc, -lc)

SYNOPSIS
     #include <stdio.h>

     FILE *
     fdopen(int fildes, const char *mode);
:
```

Left to Right (Execute shell commands from Vim and jump back to editor)

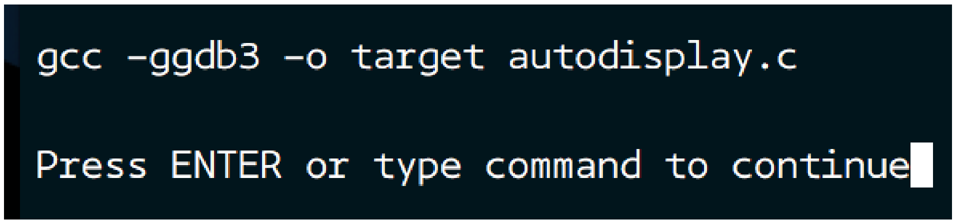
Guess what! Prepare to be amazed. Vim also supports `make` command from within your file! All you have to do is navigate to a directory with `Makefile`. Open any file (Could be your source code) and make all the changes and save it. Wait, there's no need to exit to see the compilation result. You can trigger your make build from right within Vim:

[DEMO] Trigger make builds from vim

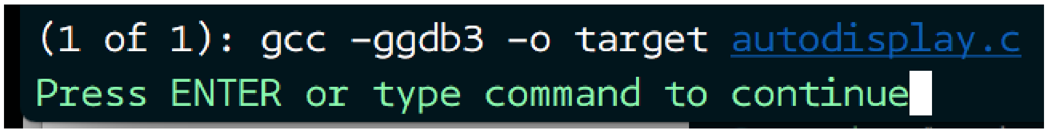
*VIM allows users to trigger make builds without exiting VIM. All we have to do is enter the command mode and type :make*asciinema.org



```
    }
    reverseword[strlen(word)] = '\0';
    printf("So the reversed word is %s\n",reverseword);
    return 0;
}
:make
```



```
gcc -ggdb3 -o target autodisplay.c

Press ENTER or type command to continue
```

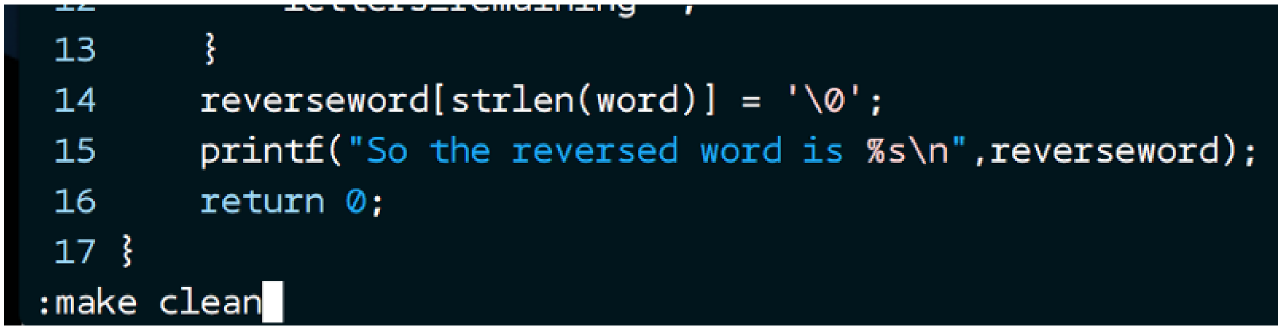


```
(1 of 1): gcc -ggdb3 -o target autodisplay.c
Press ENTER or type command to continue
```

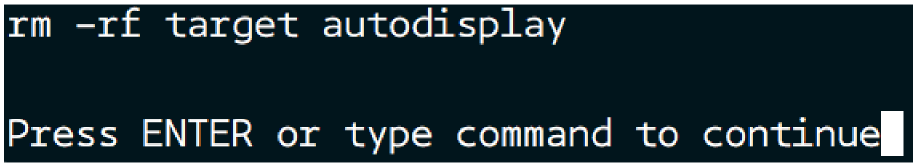
Triggering make builds from Vim

Similarly you can build other targets in your Makefile!

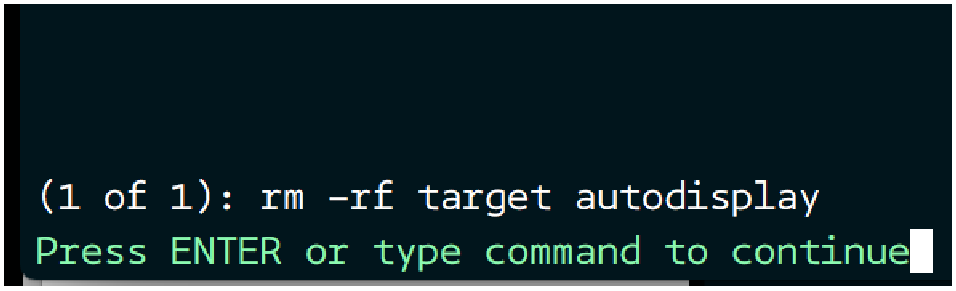
Example: Build directory clean up



```
    }
    reverseword[strlen(word)] = '\0';
    printf("So the reversed word is %s\n",reverseword);
    return 0;
}
:make clean
```



```
rm -rf target autodisplay

Press ENTER or type command to continue
```



```
(1 of 1): rm -rf target autodisplay
Press ENTER or type command to continue
```

Cleaning directory using make command from VIM

I hope these cool features will help you to use Vim more productively.

Your feedback is always welcome.

Feel free to comment, criticize or applaud ?