Emacs

Configuring Emacs for C++

3 years ago • by Zeeman Memon

C++ is a programming language that is known for its immaculate performance and powerful features that offer users greater control over the actual hardware without dropping to the level of assembly language. Having a huge ecosystem of libraries, frameworks, and tools along with a widely developed community and a large number of tutorials and guides alongside its excellent documentation to back it up, C++ has become one of the most popular programming languages, being used almost everywhere.

Its versatility and compactness have led to it developing a huge name among game developers and, thus, is often used to develop games, game engines, and desktop applications. Having such a powerful core sitting behind it, it is extremely important to have an editor that provides the best features and facilitates the needs of its users.

Emacs is one such handy editor that, due to its flexible and adaptable nature, has quickly risen to become an extremely efficient and powerful tool to use. Features like git integration, multiple editing modes, and regex search and replacement show the commanding presence it has among the wide set of text editors.

Being extremely customizable, it can thus easily be configured to be used as a C++ IDE. Hence today, we will be looking at how one can configure Emacs for C++ Development and turn it into a C++ IDE.

Integration of C++ with Emacs

To integrate C++ with Emacs, we will be using several packages such as auto-complete, flycheck, magit, etc. Let us now move onto the process.

1) Adding the Initial Commands to the Initialization File

On starting Emacs, the first thing that gets processed is the initialization file (init file), which contains commands allowing users to customize and set up Emacs according to their preferences.

So, to make Emacs act as a C++ IDE, we have to add some commands to this file.

To do this, open the terminal and enter the following command to open the initialization file:

```
$ emacs ~/.emacs
```

Now we have to add the following lines of code:

```
(require 'package)
(add-to-list 'package-archives
             '("melpa" . "http://melpa.org/packages/") t)
(package-initialize)
(unless (package-installed-p 'use-package)
  (package-refresh-contents)
  (package-install 'use-package))
(use-package try :ensure t)
(use-package which-key :ensure t :config (which-key-mode))
```

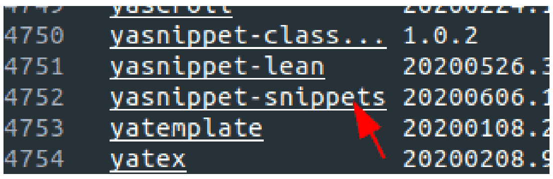
The code given above basically adds the Melpa archive to the list of package repositories found in Emacs and permits it to use these packages.

Furthermore, it installs the use-package, which can be used to install other packages, set up key bindings, and configure various Emacs modes.

2) Adding the Yasnippet Package

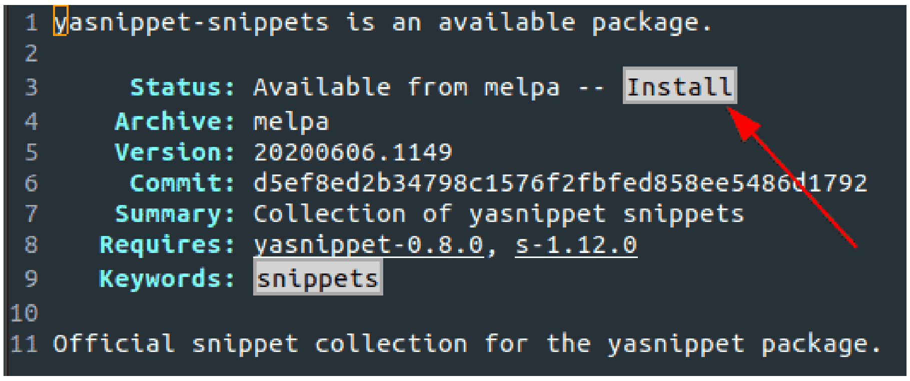
Once we're done setting up our Emacs with the initial content, we now will be adding the yasnippet package, which is a snippet extension for Emacs and hence provides function templates for multiple languages, including C++.

To add this into our initialization file, we first have to install it. To do so, open Emacs and hit **Alt + x** and enter **package-list-packages**. Now hit **Ctrl + s** and search for yasnippet. Continue clicking Ctrl + s till you find the package: **yasnippet-snippets**.



```
4750   yasnippet-class...  1.0.2
4751   yasnippet-lean      20200526.3
4752   yasnippet-snippets  20200606.1
4753   yatemplate          20200108.2
4754   yatex               20200208.9
```

You can either click on it and select install or hit **i** followed by **x** to install the yasnippet package.



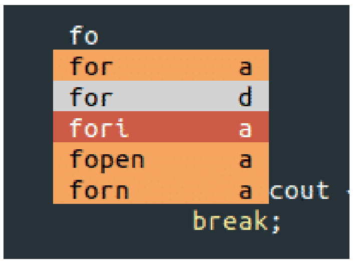
```
yasnippet-snippets is an available package.

     Status: Available from melpa -- Install
    Archive: melpa
    Version: 20200606.1149
     Commit: d5ef8ed2b34798c1576f2fbfed858ee5486d1792
    Summary: Collection of yasnippet snippets
   Requires: yasnippet-0.8.0, s-1.12.0
   Keywords: snippets

Official snippet collection for the yasnippet package.
```

After installing yasnippet, hit **Ctrl + x,** followed by **Ctrl + f,** and open the ~/.emacs file. Inside this add the following lines of code:

```
(require 'yasnippet)
(yas-global-mode 1)
```

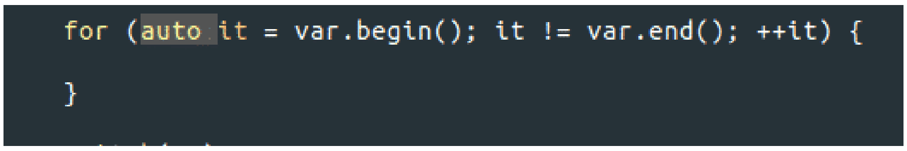
Simply hit **Ctrl + X,** followed by **Ctrl + S** to save your file and restart Emacs. Now open some C++ files, and you can see drop-down options like these, which provide you with a snippet upon clicking.

Drop Down Options:



```
fo
for        a
for        d
fori       a
fopen      a
forn       a cout
        break;
```

The snippet is given after clicking on one of the Options:



```
for (auto it = var.begin(); it != var.end(); ++it) {

}
```

3) Adding some Additional Packages

Now we are going to add some additional packages that would give us more functionalities in Emacs for C++ development.

First of all, we will add the **auto-complete mode,** which, as the name implies, is an auto-complete extension. Add the following lines of code to your initialization file:

```
(use-package auto-complete
  :ensure t
  :init
  (progn
    (ac-config-default)
    (global-auto-complete-mode t)
    ))
```

Next, we will add a **flycheck,** which helps in syntax checking by reporting errors and warnings. To do this, the following needs to be added to our init file:

```
(use-package auto-complete
(use-package flycheck
  :ensure t
  :init
  (global-flycheck-mode t))
```

Finally, we will add the **modern c++ font** package, which provides syntax highlight of the latest C++ language.

```
(use-package modern-cpp-font-lock
:ensure t)
```

4) Git Integration using Magit

Emacs also allows integration with Git using Magit, a git tool that acts as an interface to Git. We can include this in Emacs by adding the following lines of code to our initialization file:

```
(use-package magit
  :ensure t
  :init
  (progn
    (bind-key "C-x g" 'magit-status)))
```

Over here, we set up a key bind using the keys **Ctrl + x** followed by **g**, which allows us to see the status of our files (untracked, staged, commits).

To see the commands of Magit, hit the question mark (?) key. Some commands that will be shown are:

Using these, we can integrate our files with the version control Git.

5) Compiling and running C++ Code

For compiling and running the C++ code on Emacs, we will be making use of a custom function and the GDB debugger. This makes the process much easier compared to making make-files and then compiling and building them. To do this, first of all, add the following code to your initialization file:

```
(defun code-compile ()
  (interactive)
  (unless (file-exists-p "Makefile")
    (set (make-local-variable 'compile-command)
     (let ((file (file-name-nondirectory buffer-file-name)))
       (format "%s -o %s %s"
           (if  (equal (file-name-extension file) "cpp") "g++" "gcc" )
           (file-name-sans-extension file)
           file)))
    (compile compile-command)))

(global-set-key [f9] 'code-compile)
```

The code above allows us to compile a C++ file using the **f9** key.

After compiling, to run the C++ file, hit Alt + x and enter gdb. You will get something like this:

Simply click enter, and gdb will start. Now click on the **Run** button found on top of Emacs to run the C++ program.

The following image shows the C++ Code being run:

Why use Emacs for C++?

Emacs is a highly powerful text editor that offers so many amazing features to its users. Being easily customizable, Emacs can easily be set up by users according to their needs and preferences. They can change themes, font styles, and so much more. It is a must-have editor for users wanting more freedom in their work.

ABOUT THE AUTHOR

Zeeman Memon

Hi there! I'm a Software Engineer who loves to write about tech. You can reach out to me on LinkedIn.

View all posts

RELATED LINUX HINT POSTS

Emacs Org Mode Tutorial

Copying and Pasting in Emacs

How to Comment Multiple Lines at Once in Emacs

5 Best emacs plugins for web development

Emacs vs Vim

Emacs Cheatsheet

Emacs Commands Comprehensive Overview