Autocompletion Support in Vim

Last modified: September 19, 2022

by baeldung (https://www.baeldung.com/linux/author/baeldung)

File Editing (https://www.baeldung.com/linux/category/files/editing)

vi (https://www.baeldung.com/linux/tag/vi) **vim (https://www.baeldung.com/linux/tag/vim)**

If you have a few years of experience in the Linux ecosystem, and you're interested in sharing that experience with the community, have a look at our **Contribution Guidelines** (/linux/contribution-guidelines).

1. Overview

Vim (/linux/vi-vim-editors) text editor supports autocompletion for the standard text files by default. Also, when configured properly, Vim enables an autocomplete feature for files with code in the languages it recognizes.

In this tutorial, we'll see **how the autocompletion support in Vim works.**

2. Built-in Support

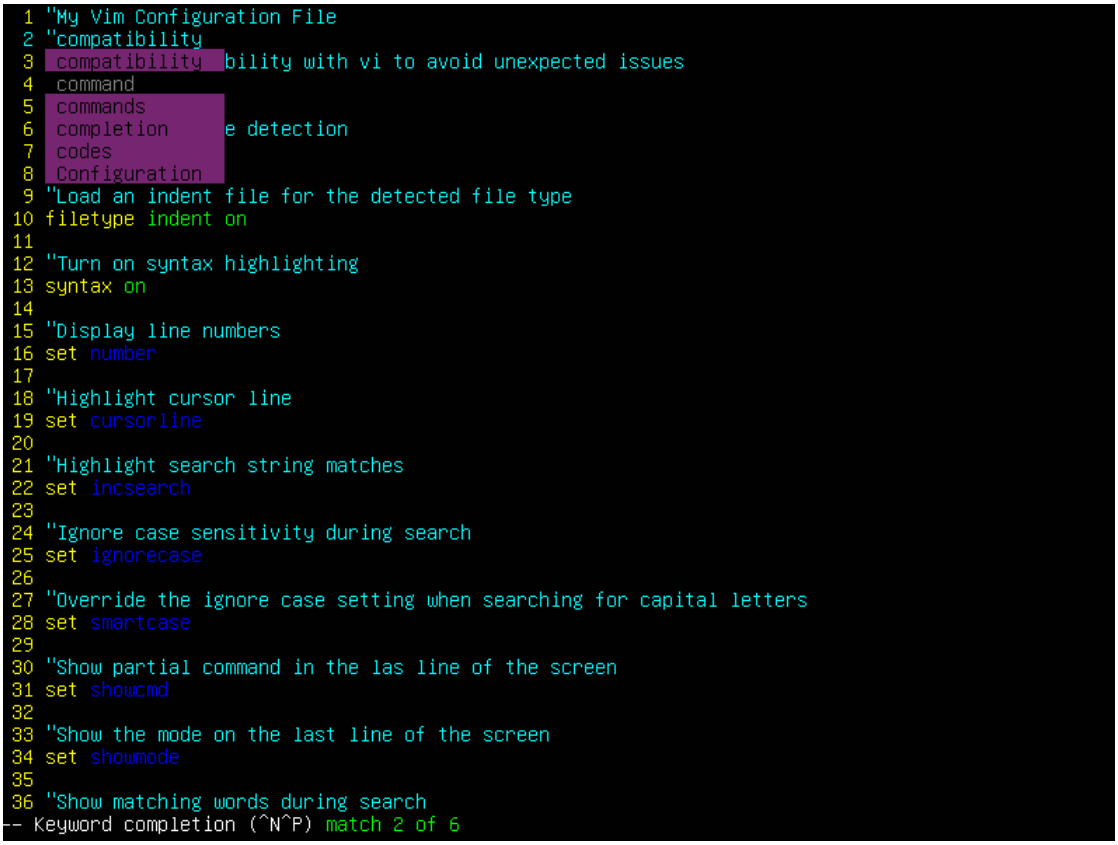
Of course, recent versions of the Vim editor come with a built-in autocomplete feature. **Vim completes words by checking the available terms currently in the buffer**. Importantly, this functionality is case-sensitive.

2.1. *Ctrl-N*

First, we'll see how to use the autocomplete feature in a standard text file.

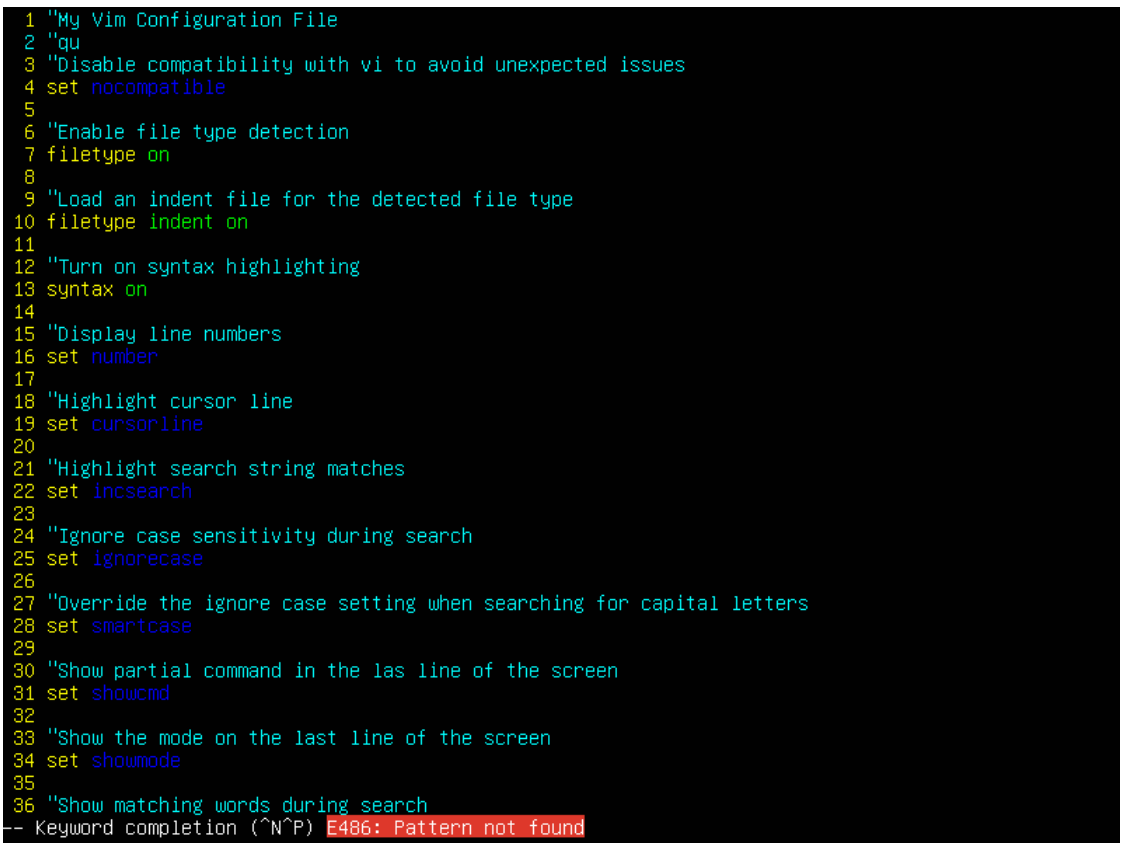
For this purpose, let's open an example *.vimrc* configuration file. Next, we'll enter a few characters of a word we need to write and **press *Ctrl-N* to trigger autocompletion**.

If Vim finds only one match, it will automatically complete the term. However, **if Vim finds more than one match for the text, a list of words will appear**:



We can select the preferable term from the list.

Also, **we can use *Ctrl*-P to make the list cycle backward**. Alternatively, we can use the arrow keys to move up and down the list. **If the program finds no match to complete a word, we'll see an error stating *Pattern not Found***:



Also, to see Vim's documentation on the autocomplete functionality, we can use *:help i_CTRL-N*.

2.2. Omnicompletion

Omnicompletion (https://www.vim.org/scripts/script.php?script_id=3172) provides an intelligent autocompletion feature for programs in Vim. When called, **Omnicompletion examines the text before the cursor to predict the intended word**.

Ordinarily, Omnicompletion is not available by default. Now, to enable Omnicompletion features on Vim, let's add the following to our *~/.vimrc:*

```
" Enable plugins and load plugin for the detected file type.
filetype plugin on
" Enable Omnicomplete features
set omnifunc=syntaxcomplete#Complete
```

Here, the double quotation marks begin comment lines. Below each comment is the line with the described function.

Finally, **we can press *Ctrl-X* and *Ctrl-O* to start Omnicompletion**. To check whether Omnicompletion is working, **we can use the command *:echo &omnifunc* in a file**.

In addition, **we can set up Omnicompletion for specific language syntax in Vim**. For example, to set up autocompletion for JavaScript syntax, we'll enter the following command in the Vim command mode and press *Enter*:

```
:set omnifunc=javascriptcomplete#CompleteJS
```

Similarly, the following commands work for HTML, CSS, and PHP, respectively:

- *:set omnifunc=htmlcomplete#CompleteTags*
- *:set omnifunc=csscomplete#CompleteCSS*
- *:set omnifunc=phpcomplete#CompletePHP*

Furthermore, Vim autocompletion has some excellent substitutes. The command *:h ins-completion* shows the variety of built-in completion options on our system.

3. Plugins

Generally, plugins add extra functionality to any program. Thus, **we can add plugins to Vim to extend its auto-completion features**. Notably, we can employ plugin managers (/linux/vim-install-neovim-plugins) like vim-plug (https://github.com/junegunn/vim-plug) and vundle (https://github.com/gmarik/Vundle.vim) to ease the plugin installation process.

Following, let's look at some autocompletion plugins for vim and their unique features.

3.1. SuperTab

SuperTab (https://github.com/ervandew/supertab) is a Vim autocompletion plugin that allows the use of Tab for all insert completions. **The plugin enables us to hit Tab after a partial entry to get an autocompletion functionality**.

For instance, we might want to type "editors". So, we enter "e" and press Tab. The plugin shows a keyword completion list. Then we can select the desired word from the list.
The SuperTab plugin is similar to the built-in Omnicompletion alternative. However, there are a few differences, as we can:

- configure it to suit our needs
- configure different keys to trigger the completion
- get a completion result when Omnicompletion returns none

To install and set up SuperTab, let's create a directory for the installation:

```
$ mkdir -p ~/.vim/pack/plugins/start
```

Next, let's clone the SuperTab repository via *git* (/git-guide):

```
$ git clone --depth=1 https://github.com/ervandew/supertab.git ~/.vim/pack/plugins/start/supertab
```

Further, we'll add *packloadall* to our *~/.vimrc* file:

```
" Enable packloadall for pack plugins.
packloadall
```

Finally, we can get to the SuperTab documentation (https://github.com/ervandew/supertab/blob/master/doc/supertab.txt) from within Vim using the *:help supertab* command.

3.2. Jedi-Vim

Next, Jedi-Vim (https://github.com/davidhalter/jedi-vim) is a Python autocompletion plugin within Vim. Basically, it's a **Vim binding to the autocompletion library Jedi (https://github.com/davidhalter/jedi)**. Uniquely, Jedi-Vim stands out with its broad support for most of Python's core features.

We can use Vundle (https://github.com/gmarik/Vundle.vim) to install Jedi-Vim. To illustrate, let's add the plugin to our configuration file:

```
Plugin 'davidhalter/jedi-vim'
```

Next, we'll save and source the file. After that, we run the *:PluginInstall* command in Vim. Importantly, **to use Jedi-Vim while coding, we press *Ctrl-Space***.

As with other plugins, we can access the Jedi-Vim documentation (https://github.com/davidhalter/jedi-vim/blob/master/doc/jedi-vim.txt) in *vim* with the *:help jedi-vim* command.

3.3. YouCompleteMe

YouCompleteMe (https://github.com/ycm-core/YouCompleteMe) is a fast and intelligent suggest-as-you-type code-completion and refactoring engine for Vim. It has several completion engines built in. Hence, **YouCompleteMe can work with almost any language**.

Indeed, the YouCompleteMe plugin is unique and has several advantages over other autocompletion plugins:

- works with any programing language
- we don't have to press any keyboard combination to trigger the autocompletion
- the filtering is not based on the input being a string prefix of the completion (but that works too)

Similar to other plugins, we use Tab to go through any valid suggestions.

The YouCompleteMe installation (https://github.com/ycm-core/YouCompleteMe#installation) is outside the scope of this article. However, an important step is to install CMake (/linux/compile-32-bit-binary-on-64-bit-os#using-cmake) and Python. In Debian-based systems, we can do that via *apt* (/linux/yum-and-apt#apt-advanced-packaging-tool):

```
$ sudo apt install build-essential cmake python3-dev
```

Further, we'll install mono-complete, go, node, java, and npm

```
$ sudo apt install mono-complete golang nodejs default-jdk npm
```

After having the necessary packages, we follow the installation guide to set up and compile YouCompleteMe.

4. Summary

In conclusion, autocompletion is a great feature in the Vim editor. We have seen how to use the built-in autocompletion features in Vim but also learned about plugins with the same function.

If you have a few years of experience in the Linux ecosystem, and you're interested in sharing that experience with the community, have a look at our **Contribution Guidelines** (/linux/contribution-guidelines).