UltiSnips: Snippets Guide

Sanjyot Shenoy

2021

Table of Contents

1 Introduction

We use the UltiSnips Plugin for Vim to write our custom snippets for not just LaTeX files but also, any file extensions.
For a complete and detailed use of UltiSnips and its syntax, you can check the official documentation here:
`https://github.com/SirVer/ultisnips/blob/master/doc/UltiSnips.txt.`
We will use this documentation as a reference, so do keep it handy or open. The section of the official documentation on which these sections are based on are mentioned in the parentheses.

2 Keywords and Syntax

We now see some of the basic syntax required for writing own snippets, or understanding them, most of this is directly taken from the official documentation given above.

2.1 Comments (Based on §4.1.2)

All lines starting with a # character areconsidered comments. Comments are ignored by UltiSnips. Use them to document snippets.

2.2 `extends` (Based on §4.1.2)

This is basically used to 'extend' or 'activate' snippets of a different file type to the current file type. An example would be to extend the snippets of .c to .cpp file types. Multiple 'extends' lines are permitted in a snippet file, and they can be included anywhere in the file. When the 'extends' directive is included in a snippet file, it instructs UltiSnips to include all snippets from the indicated filetypes.
Syntax: `extends ft1, ft2, ft3`
Example: In cpp.snippets we have `extends c`

2.3 `priority` (Based on §4.1.2)

A line beginning with the keyword 'priority' sets the priority for all snippets defined in the current file after this line. The default priority for a file is always 0. When a snippet should be expanded, UltiSnips will collect all snippet definitions from all sources that match the trigger and keep only the ones with the highest priority. For example, all shipped snippets have a priority < 0, so that user defined snippets always overwrite shipped snippets.
Syntax: `priority n`
Example: `priority 1000`

2.4 `snippet` (Based on §4.1.2)

A line beginning with the keyword 'snippet' marks the beginning of a snippet definition and a line starting with the keyword 'endsnippet' marks the end. The snippet definition is placed between the lines.
Syntax: If trigger word has no spaces:

```
snippet trigger_word [ "description" [ options ] ]
snippet definition
endsnippet
```

If trigger word has spaces:

```
snippet "trigger word" [ "description" [ options ] ]
snippet definition
endsnippet
```

Note: It is not technically necessary to use quotes to wrap a trigger with spaces. Any matching characters will do. For example, '! !' or '? ?'.
Example: This is a TEX-snippet for \ldots

```
snippet ...  "ldots" iA
\ldots
endsnippet
```

2.4.1 `description` (Based on §4.1.2)

The 'description' is a string describing the trigger. It is helpful for documenting the snippet and for distinguishing it from other snippets with the same tab trigger. When a snippet is activated and more than one tab trigger match, UltiSnips displays a list of the matching snippets with their descriptions. The user then selects the snippet they want.

2.4.2 Snippet Options (Based on §4.1.3)

The 'options' control the behavior of the snippet. Options are indicated by single characters. The 'options' characters for a snippet are combined into a word without spaces.
There are a lot of options, which can be checked in §4.1.3 of UltiSnips Documentation. We will only mention the widely used ones.

`b` : Beginning of line - A snippet with this option is expanded only if the tab trigger is the first word of the line. In other words, if only whitespace precedes the tab trigger, expand.

`i` : In-word expansion - A snippet with this option is expanded regardless of the preceding character. In other words, the snippet can be triggered in the middle of a word.

`A` : Snippet will be triggered automatically, when condition matches.

`e` : With this option expansion of snippet can be controlled not only by previous characters in line, but by any given python expression. This option can be specified along with other options, like 'b'.

`r` : Regular Expression - With this option, the tab trigger is expected to be a python regular expression. The snippet is expanded if the recently typed characters match the regular expression. Note: The regular expression MUST be quoted (or surrounded with another character) like a multi-word tab trigger (see above) whether it has spaces or not. A resulting match is passed to any python code blocks in the snippet definition as the local variable "match".

2.5 Character Escaping (Based on §4.1.4)

In snippet definitions, the characters '`', '{', '$' and '\' have special meaning. If you want to insert one of these characters literally, escape them with a backslash, '\'.

2.6 Visual Placeholder (Based on §4.3)

Snippets can contain a special placeholder called ${VISUAL}. The ${VISUAL} variable is expanded with the text selected just prior to expanding the snippet.
The selected text is deleted, and you are dropped into Insert mode. Now type the snippet tab trigger and press the key to trigger expansion. As the snippet expands, the previously selected text is printed in place of the ${VISUAL} placeholder.
The ${VISUAL} placeholder can contain default text to use when the snippet has been triggered when not in Visual mode.
Syntax: `${VISUAL:default text}`
The ${VISUAL} placeholder can also define a transformation (see |UltiSnips-transformations|). For more on this, check section on transformations.
Syntax: `${VISUAL:default/search/replace/option}`
If this seems confusing, check the documentation §4.3 example or §4.7.

2.7 Tabstops and Placeholders (Based on §4.5)

Snippets are used to quickly insert reused text into a document. Often the text has a fixed structure with variable components. Tabstops are used to simplify modifying the variable content. With tabstops you can easily place the cursor at the point of the variable content, enter the content you want, then jump to the next variable component, enter that content, and continue until all the variable components are complete.
Syntax: The syntax for a tabstop is the dollar sign followed by a number, for example, '$1'. Tabstops start at number 1 and are followed in sequential order. The '$0' tabstop is a special tabstop. It is always the last tabstop in the snippet no matter how many tabstops are defined. If there is no '$0' defined, '$0' tabstop will be defined at the end of the snippet.
Note (Jumping forward and backward): You can use <c-j> to jump to the next tabstop, and <c-k> to jump to tothe previous. The <Tab> key was not used for jumping forward because it is commonly used for completion.
Syntax for default text: It is often useful to have some default text for a tabstop. The default text may be a value commonly used for the variable component, or it may be a word or phrase that reminds you what is expected for the variable component. To include default text, the syntax is **`'$1:value'`**.
Note (Nested Tabstops): Sometimes it is useful to have a tabstop within a tabstop. To do this, simply include the nested tabstop as part of the default text.
Typing any text at the first tabstop replaces the default value, including the second tabstop, with the typed text. So the second tabstop is essentially deleted. When a tabstop jump is triggered, UltiSnips

moves to the next remaining tabstop '$0'. This feature can be used intentionally as a handy way for providing optional tabstop values to the user.
Examples: Can be seen in the official documentation.

3 Interpolation (Based on §4.4)

Interpolation, in simple terms means the use of external programming/scripting languages in order to make the snippets more dynamic and make the snippets have the power to process simple logical/processing tasks. There are few interpolation methods which UltiSnips has the access to, which we will now discuss.

3.1 Shellcode (i.e bash/zsh/etc.)(Based on §4.4.1)

Snippets can include shellcode. Put a shell command in a snippet and when the snippet is expanded, the shell command is replaced by the output produced when the command is executed.
The syntax for shellcode is simple: wrap the code in backticks, '''. When a snippet is expanded, UltiSnips runs shellcode by first writing it to a temporary script and then executing the script. The shellcode is replaced by the standard output.
Anything you can run as a script can be used in shellcode. Include a shebang line, for example, `#!/usr/bin/perl`, and your snippet has the ability to run scripts using other programs, perl, for example.
Example: This example insertes current date using shellcode.

```
snippet today
Today is the `date +%d.%m.%y`.
endsnippet
```

3.2 VimScript (Based on §4.4.2)

You can also use Vim scripts (sometimes called VimL) in interpolation. The syntax is similar to shellcode. Wrap the code in backticks and to distinguish it as a Vim script, start the code with '!v'. Here is an example that counts the indent of the current line:
Example: Used to indent.

```
snippet indent
Indent is: `!v indent(".")`.
endsnippet
```

3.3 Python (Based on §4.4.3)

We recommend the reader to directly refer the documentation §4.4.3, as the explanation is clear and concise there for snippets involving basic python evaluations.
Also, check this video for some neat examples/explanation of basic python snippets:
`http://vimcasts.org/episodes/ultisnips-python-interpolation/`
Python Regular Expression(Regex) Snippets require a good amount of expertise in understanding Regexes and their syntax. A Regex snippet works by making the trigger word as the Regex itself, and then snip.rv will use the match keyword for its completion.
Here are some good resources on Regex:

1. An example-based guide which is much better to follow:
   `https://realpython.com/regex-python/`
2. Official Python Documentation: `https://docs.python.org/3/library/re.html#`
3. W3Schools for testing with examples: `https://www.w3schools.com/python/python_regex.asp`
4. A great tool for visual breakdown of the Regex: `https://regexper.com/`
5. A great tool for testing Regex with explanation of all matches: `https://pythex.org/`

3.4 Global Snippets (Based on §4.4.4)

Global snippets provide a way to reuse common code in multiple snippets. Currently, only python code is supported. The result of executing the contents of a global snippet is put into the globals of each python block in the snippet file.
Syntax: To create a global snippet, use the keyword 'global' in place of 'snippet', and for python code, you use '!p' for the trigger.
For example, the following snippet produces the same output as just a normal python snippet. However, with this syntax the 'upper_right' snippet can be reused by other snippets.
Example:

```
global !p
def upper_right(inp):
return (75 - 2 * len(inp))*' ' + inp.upper()
endglobal
```

4 Transformations (Based on §4.7)

This is an advanced topic which is powerful but not frequently used, but we encourage you to study it through the documentation itself, in §4.7.