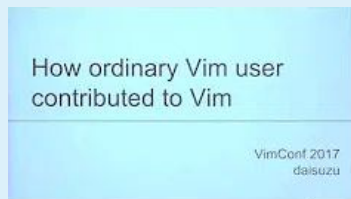
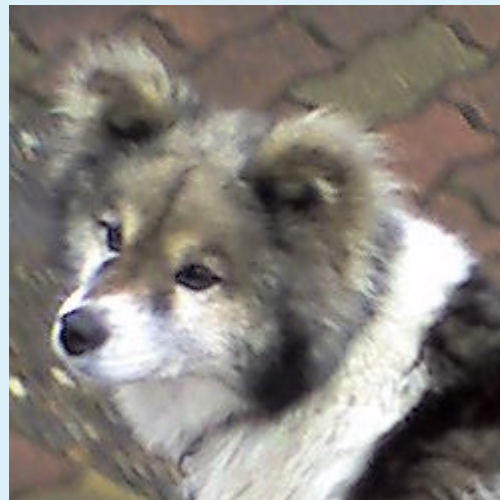


Usage and manipulation of the tag stack

VimConf 2019
daisuzu

About me

- daisuzu(Daisuke Suzuki)
 -  https://twitter.com/dice_zu
 -  <https://github.com/daisuzu>
 -  <https://daisuzu.hatenablog.com>
- Job
 - Server side software engineer
 - Go Language Committee(in-company organization)
- VimConf



2017



2018

Motivation

- I had to submit for CFP, but there was no content to talk.
- By the way, I love [vim-lsp](#) and wanted to use built-in tag jump with vim-lsp.
 - [prabirshrestha/vim-lsp#435](#) by @kailin4u (Merged on July 18)
 - But there is a problem that go back too much
- I investigated the internal processing of the tag stack and fixed it.
 - [prabirshrestha/vim-lsp#449](#) by @daisuzu (Created on July 26)
 - Tag stack is really deep, so I decided to talk about this

Presentation Driven Development

Agenda

- Basics of the tags feature
- Deep dive into the tag stack

Basics of the tags feature

What is a tag?

- It is a location where an identifier is defined
 - That's news to you? Please see ***:help***

```
help.txt      For Vim version 8.1.  Last change: 2019 Jul 21

                VIM- main help file

                k
Move around:   Use the cursor keys, or "h" to go left,      h  l
                "j" to go down, "k" to go up, "l" to go right.  j
Close this window: Use ":q<Enter>".
Get out of Vim: Use ":qa!<Enter>" (careful, all changes are lost!).

Jump to a subject: Position the cursor on a tag (e.g.bars) and hit CTRL-].
```

- A list of tags is kept in a tags file
 - The tags file has to be generated before the tag commands can be used
 - ctags
 - :helptags

Using tags

- Preparations

1. Generate the tags file

```
# Supported languages like C/C++  
$ ctags -R
```

2. Set tags option (If need to change)

- `:set tags=./tags,tags`

- Jump to tag

CTRL-]

- Go back

CTRL-T

Example) Jump to tag

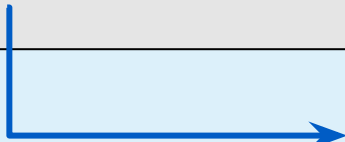
```
void write_block(char **s; int cnt)
{
    int i;
    for (i = 0; i < cnt; ++i)
        write_line(s[i]);
}
```

CTRL-]

```
void write_line(char *s)
{
    while (*s != 0)
        write_char(*s++);
}
```

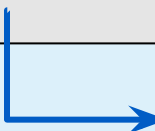

Example) Jump to tag

```
void write_block(char **s; int cnt)
{
    int i;
    for (i = 0; i < cnt; ++i)
        write_line(s[i]);
}
```



```
void write_line(char *s)
{
    while (*s != 0)
        write_char(*s++);
}
```

CTRL-]



```
void write_char(char c)
{
    putchar((int) (unsigned char) c);
}
```

Example) Show the contents of the tag stack

```
void write_block(char **s; int cnt)
{
    int i;
    for (i = 0; i < cnt; ++i)
        write_line(s[i]);
}
```

```
void write_line(char *s)
{
    while (*s != 0)
        write_char(*s);
}
```

```
void write_char(char c)
{
    putchar((int) (unsigned char) c);
}
```

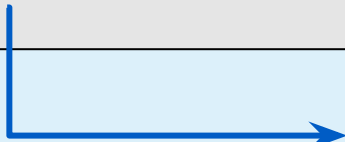


:tags

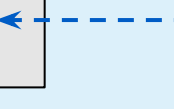
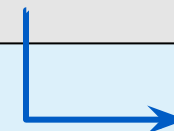
```
# TO tag          FROM line  in file/text
1  1 write_line    8  write_block.c
2  1 write_char    7  write_line.c
>
```

Example) Go back

```
void write_block(char **s; int cnt)
{
    int i;
    for (i = 0; i < cnt; ++i)
        write_line(s[i]);
}
```



```
void write_line(char *s)
{
    while (*s != 0)
        write_char(*s++);
}
```



CTRL-T

```
void write_char(char c)
{
    putchar((int) (unsigned char) c);
}
```

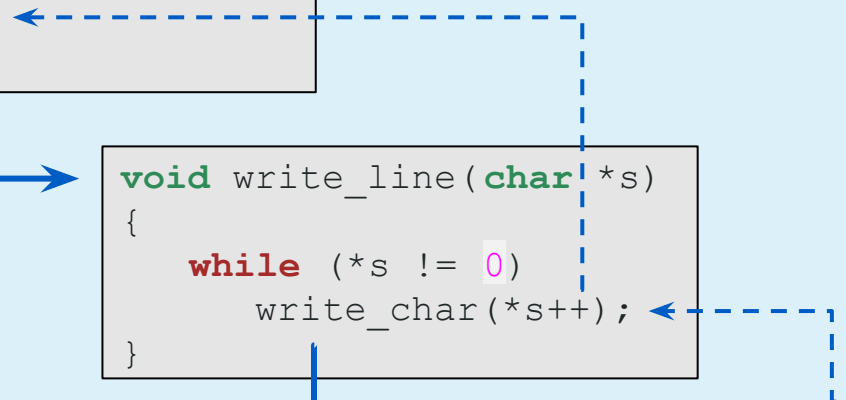
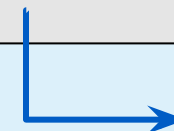
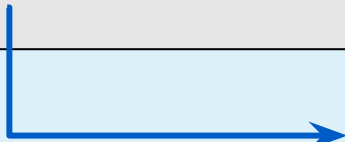
Example) Go back

```
void write_block(char **s; int cnt)
{
    int i;
    for (i = 0; i < cnt; ++i)
        write_line(s[i]);
}
```

```
void write_line(char *s)
{
    while (*s != 0)
        write_char(*s++);
}
```

```
void write_char(char c)
{
    putchar((int)(unsigned char)c);
}
```

CTRL-T



Tag commands

Jump to tag

- in same window

-
- CTRL-], {Visual}CTRL-]
 - `:[count]tag {name}`
 - `g<LeftMouse> , <C-LeftMouse>`

- in split window

-
- `:stag [name]`
 - `CTRL-W CTRL-] , CTRL-W]`

- in preview window

-
- `:ptag [name]`
 - `CTRL-W }`

- and add the matching tags to a new location list

- `:ltag [name]`

Go back

-
- CTRL-T
 - `:[count]pop`
 - `g<RightMouse> , <C-RightMouse>`

-
- `:[count]ppop`

Tag match list

Select a tag to jump

- in same window

- `:tselect [name]`
- `g] , {Visual}g]`

- in split window

- `:stselect [name]`
- `CTRL-W g]`

- in preview window

- `:ptselect [name]`

Jump directly when there is only one match, otherwise select a tag to jump

- `:tjump [name]`
- `g CTRL-] , {Visual}g CTRL-]`

- `:stjump [name]`
- `CTRL-W g CTRL-]`

- `:ptjump [name]`
- `CTRL-W g }`

Difference from the jump-motions

The jump-motions jumps around the jump list with **CTRL-I** or **CTRL-O**.

- CTRL-I ... Go to newer cursor position in jump list
 - Cannot move to a new location alone
 - In other words, the location must be changed using other commands
 - tag jump
 - search
 - goto line
- CTRL-O ... Go to older cursor position in jump list
 - It will go back including other commands as above

Deep dive into the tag stack

Tag stack structure

Store the following data for each window:

items		List of items in the stack.
	tagname	Name of the tag.
	from	Cursor position before the tag jump.
	matchnr	Current matching tag number.
	bufnr	Buffer number of the current jump.
	user_data	Custom data string for <code>'tagfunc'</code> .
curidx		Current index in the stack. Index of bottom of the stack is 1.
length		Number of entries in the stack.

Jump

length = 0

curidx = 1

CTRL-]

length = 1

curidx = 2

1	tagname	'vim_main2'
	from	[1, 444, 12, 0]
	matchnr	1
	bufnr	1

Go back

length = 1

curidx = 2

1	tagname	'vim_main2'
	from	[1, 444, 12, 0]
	matchnr	1
	bufnr	1

CTRL-T

length = 1

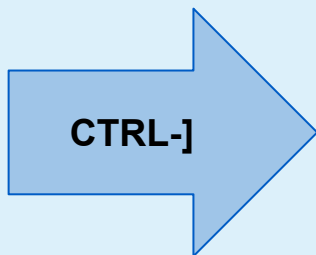
curidx = 1

1	tagname	'vim_main2'
	from	[1, 444, 12, 0]
	matchnr	1
	bufnr	1

Jump again

length = 1

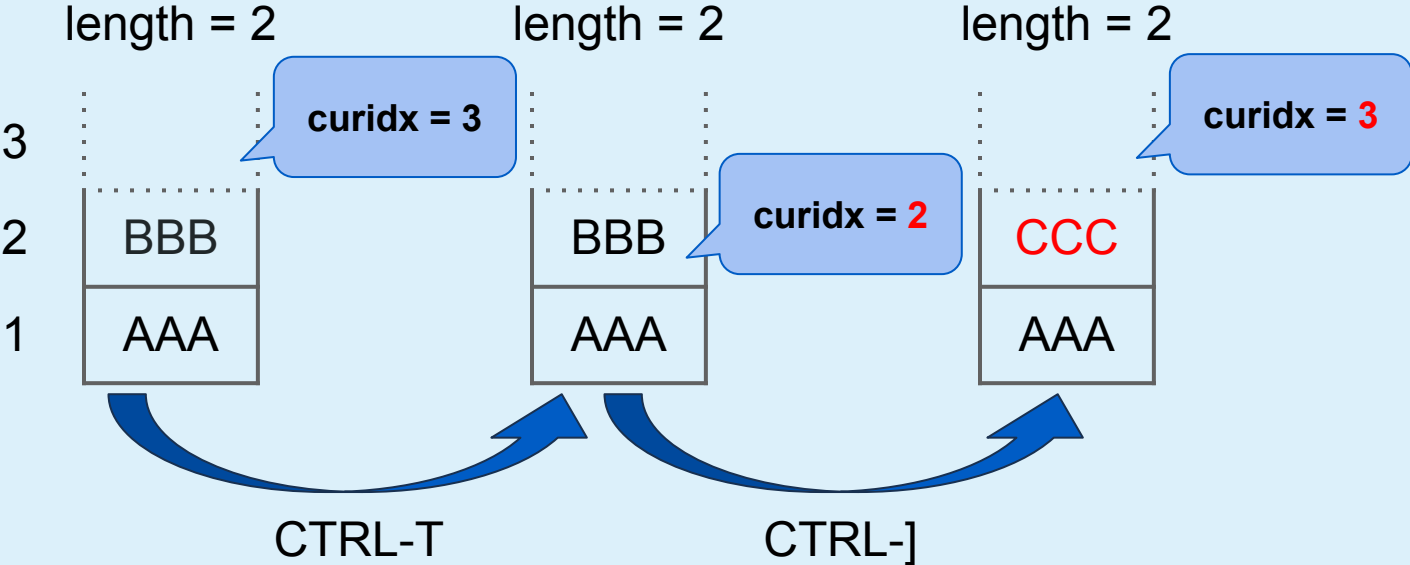
1	tagname	'vim_main2'
	from	curidx = 1
	matchnr	
	bufnr	1



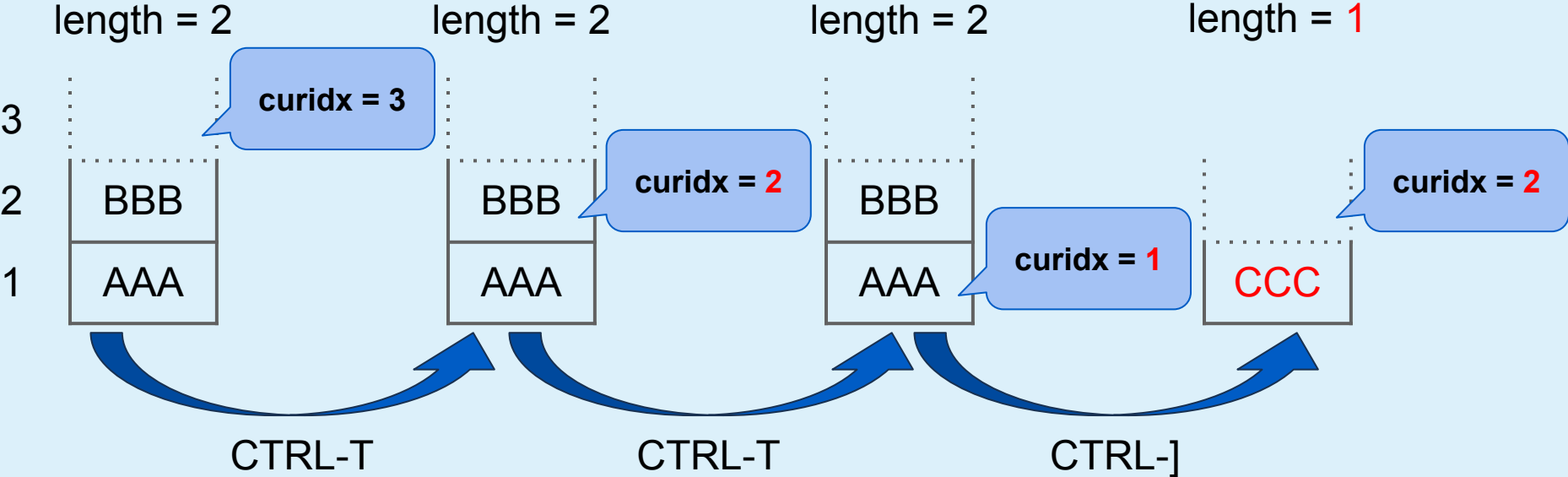
length = 1

curidx = 2		
1	tagname	'mzscheme_main'
	from	[1, 442, 12, 0]
	matchnr	1
	bufnr	1

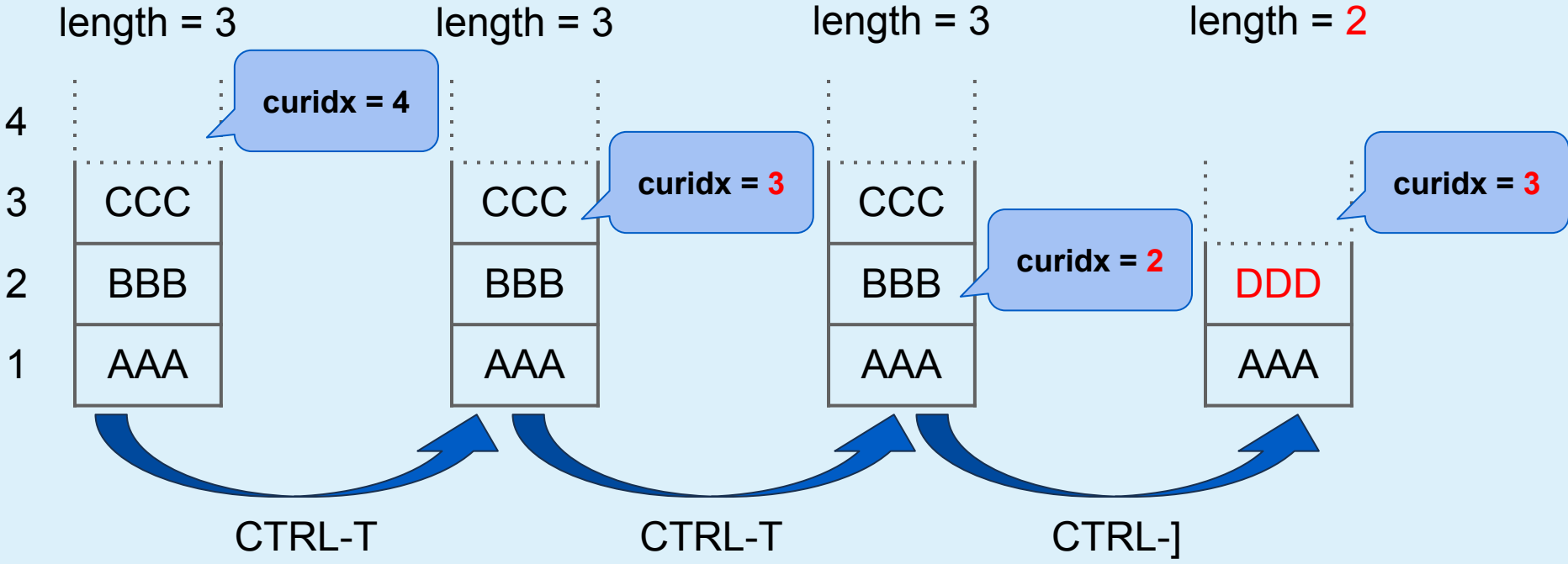
Replace top ($length == curidx$)



Replace all ($length > curidx$)



Remove and replace ($length > curidx$)



How the tag stack is managed

- When jump
 - Add or replace *items*
 - *items* above *curidx* are removed
 - *length* will also be changed
 - *curidx*++
- When go back
 - *curidx*--

Manipulate the tag stack from vim script

[settagstack\(\)](#) added in [v8.1.0519](#) made it possible.

- Works without the tags file
- That means it can be used with LSP

Paired function is [gettagstack\(\)](#).

```
settagstack({nr}, {dict} [, {action}])
```

- nr ... The window number or the window-ID of the target
- dict ... A dictionary of the tag stack structure except **length**
- action
 - r ... Replace items in the current tag stack with **dict.items**(default)
 - Clear items in current tag stack before appending
 - a ... Append **dict.items** to the current tag stack

But this is a very low-level function, so **items** and **curidx** must be handled correctly.

Implementation example

```
1 let bufnr = bufnr('%')
2 let item = {
3   \ 'bufnr': bufnr,
4   \ 'from': [bufnr, line('.'), col('.'), 0],
5   \ 'tagname': expand('<word>')
6   \ }
7 let nr = win_getid()
8
9 " Update tag stack to be set.
10 +-- 16 lines: let dict = gettagstack(nr)-----
26
27 call settagstack(nr, dict, action)
```

Implementation example

```
10 let dict = gettagstack(nr)
11 if dict['length'] == dict['curidx']
12     let action = 'r'
13     let dict['items'][dict['curidx']-1] = item
14 elseif dict['length'] > dict['curidx']
15     let action = 'r'
16     if dict['curidx'] > 1
17         let dict['items'] = add(dict['items'][:dict['curidx']-2], item)
18     else
19         let dict['items'] = [item]
20     endif
21 else
22     let action = 'a'
23     let dict['items'] = [item]
24 endif
25 let dict['curidx'] += 1
```

Implementation example

```
10 let dict = gettagstack(nr)
11 if dict['length'] == dict['curidx']                                " Replace top
12     let action = 'r'
13     let dict['items'][dict['curidx']-1] = item
14 elseif dict['length'] > dict['curidx']
15     let action = 'r'
16     if dict['curidx'] > 1
17         let dict['items'] = add(dict['items'][:dict['curidx']-2], item)
18     else
19         let dict['items'] = [item]
20     endif
21 else
22     let action = 'a'
23     let dict['items'] = [item]
24 endif
25 let dict['curidx'] += 1
```

Implementation example

```
10 let dict = gettagstack(nr)
11 if dict['length'] == dict['curidx']
12     let action = 'r'
13     let dict['items'][dict['curidx']-1] = item
14 elseif dict['length'] > dict['curidx']
15     let action = 'r'
16     if dict['curidx'] > 1                " Remove and replace
17         let dict['items'] = add(dict['items'][:dict['curidx']-2], item)
18     else
19         let dict['items'] = [item] " Replace all
20     endif
21 else
22     let action = 'a'
23     let dict['items'] = [item]
24 endif
25 let dict['curidx'] += 1
```

Implementation example

```
10 let dict = gettagstack(nr)
11 if dict['length'] == dict['curidx']
12     let action = 'r'
13     let dict['items'][dict['curidx']-1] = item
14 elseif dict['length'] > dict['curidx']
15     let action = 'r'
16     if dict['curidx'] > 1
17         let dict['items'] = add(dict['items'][:dict['curidx']-2], item)
18     else
19         let dict['items'] = [item]
20     endif
21 else
22     let action = 'a'                " Append only
23     let dict['items'] = [item]
24 endif
25 let dict['curidx'] += 1
```

Summary

- Tag stack is a very useful Vim basic feature for jumping between files
 - **CTRL-]** ... Jump to tag
 - **CTRL-T** ... Go back
- Can be used with LSP
 - By plugin that using **settagstack()**
 - Let's map **CTRL-]**
- You can customize the tag stack behavior as you like, not just LSP