vim-popped

This plugin provides four commands that use Vim's builtin popup window functionality. Vim's popup-intro provides examples of when you may want to use popup windows, so that is not explained here. The Screenshots and Mappings illustrate how to use the commands, and ways they may be useful.

Compatibility: vim-popped is built primarily for Vim 9.[1] It will not work with Neovim because Neovim has none of Vim's popup functions — refer Neovim's help (builtin.txt).

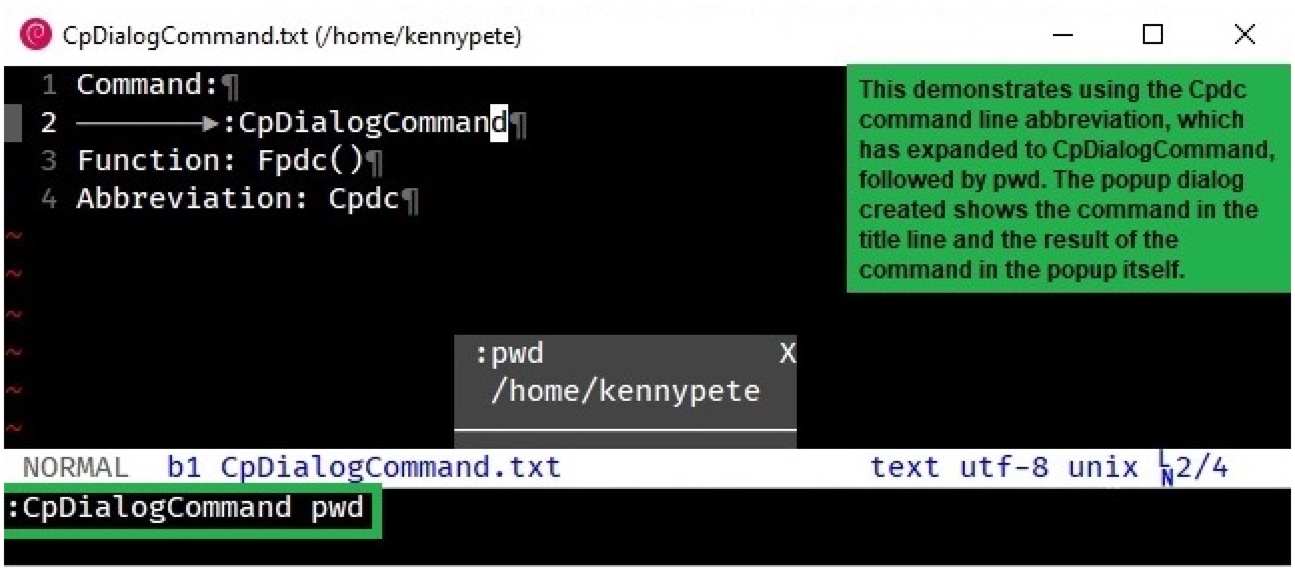
Contents

1. Screenshots

Four commands are enabled, which are associated with user-defined functions. Command line abbreviations are also provided, which, like the functions, are structured as initialisms of the command names.

Command	Function	Abbreviation
CpDialogCommand	Fpdc()	Cpdc
CpDialogCommandTimer	Fpdct()	Cpdct
CpDialogTitleCommand	Fpdtc()	Cpdtc
CpMenuBuffers	Fpmb()	Cpmb

The commands are illustrated and annotated in the following screenshots.



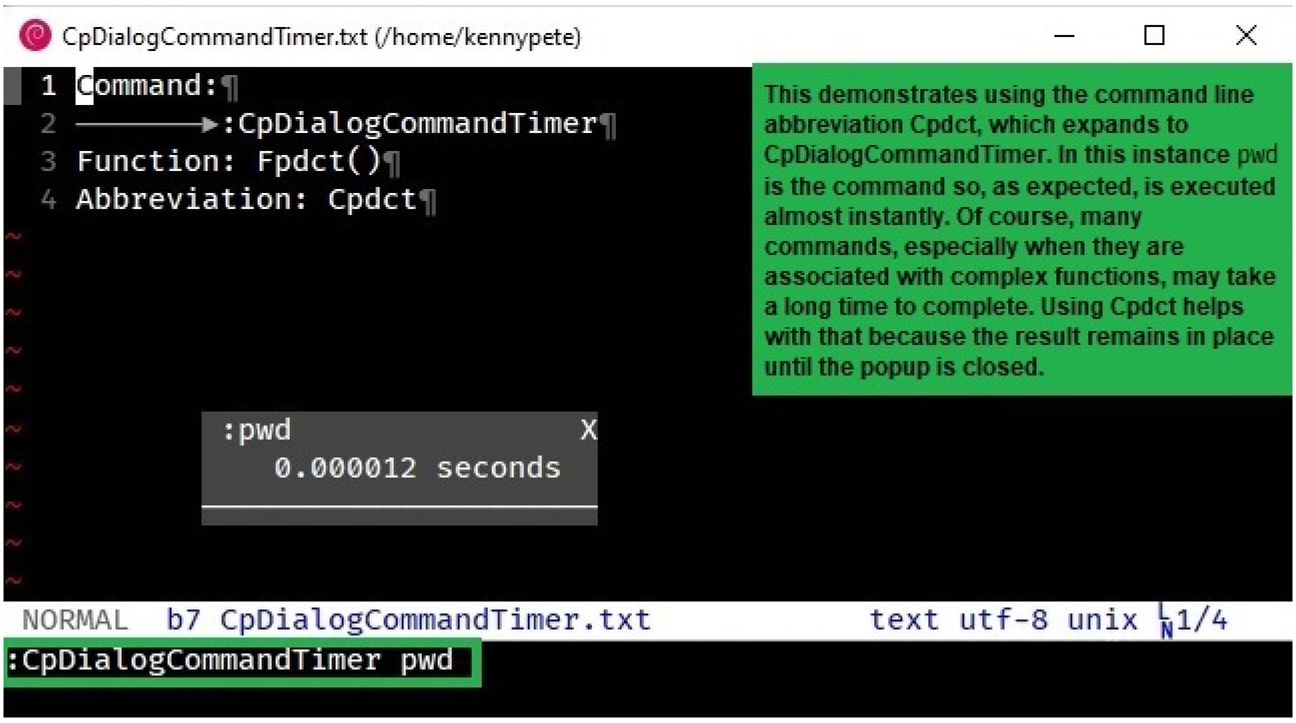
CpDialogCommand.txt (/home/kennypete)

```
Command:¶
——————►:CpDialogCommand¶
Function: Fpdc()¶
Abbreviation: Cpdc¶
```

This demonstrates using the Cpdc command line abbreviation, which has expanded to CpDialogCommand, followed by pwd. The popup dialog created shows the command in the title line and the result of the command in the popup itself.

```
:pwd                 X
 /home/kennypete
```

```
NORMAL  b1 CpDialogCommand.txt          text utf-8 unix 2/4
:CpDialogCommand pwd
```



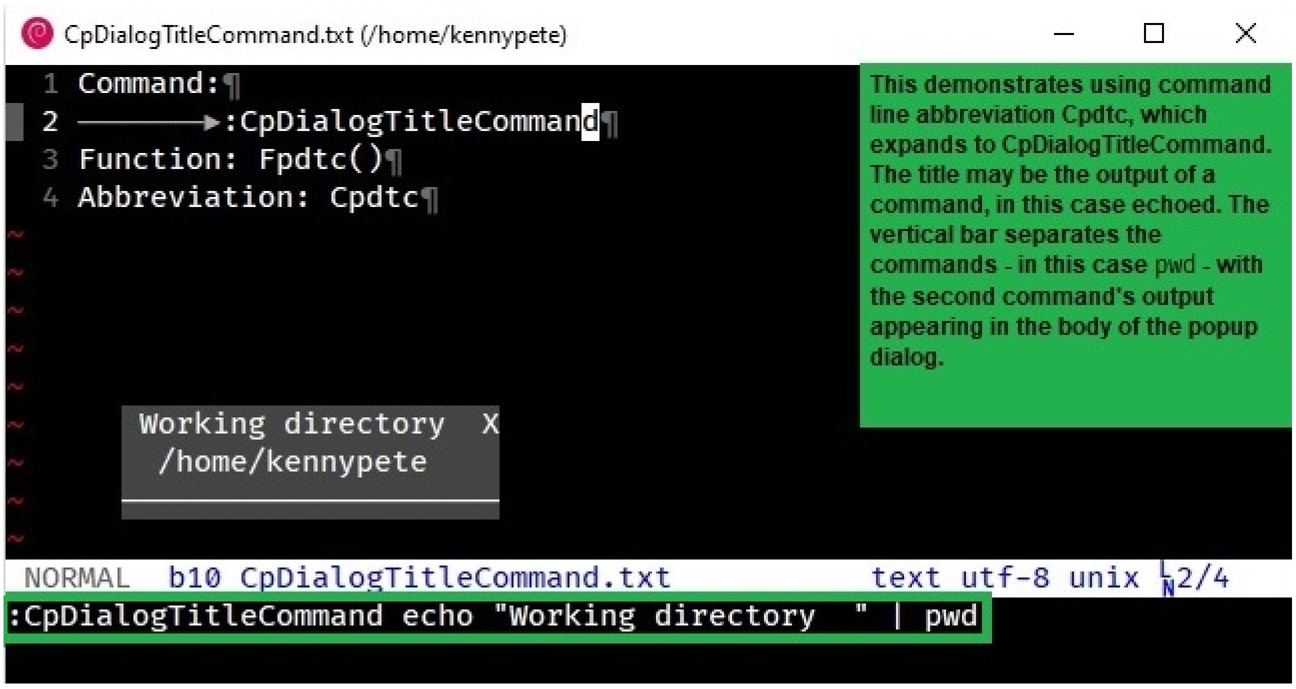
CpDialogCommandTimer.txt (/home/kennypete)

```
Command:¶
——————►:CpDialogCommandTimer¶
Function: Fpdct()¶
Abbreviation: Cpdct¶
```

This demonstrates using the command line abbreviation Cpdct, which expands to CpDialogCommandTimer. In this instance pwd is the command so, as expected, is executed almost instantly. Of course, many commands, especially when they are associated with complex functions, may take a long time to complete. Using Cpdct helps with that because the result remains in place until the popup is closed.

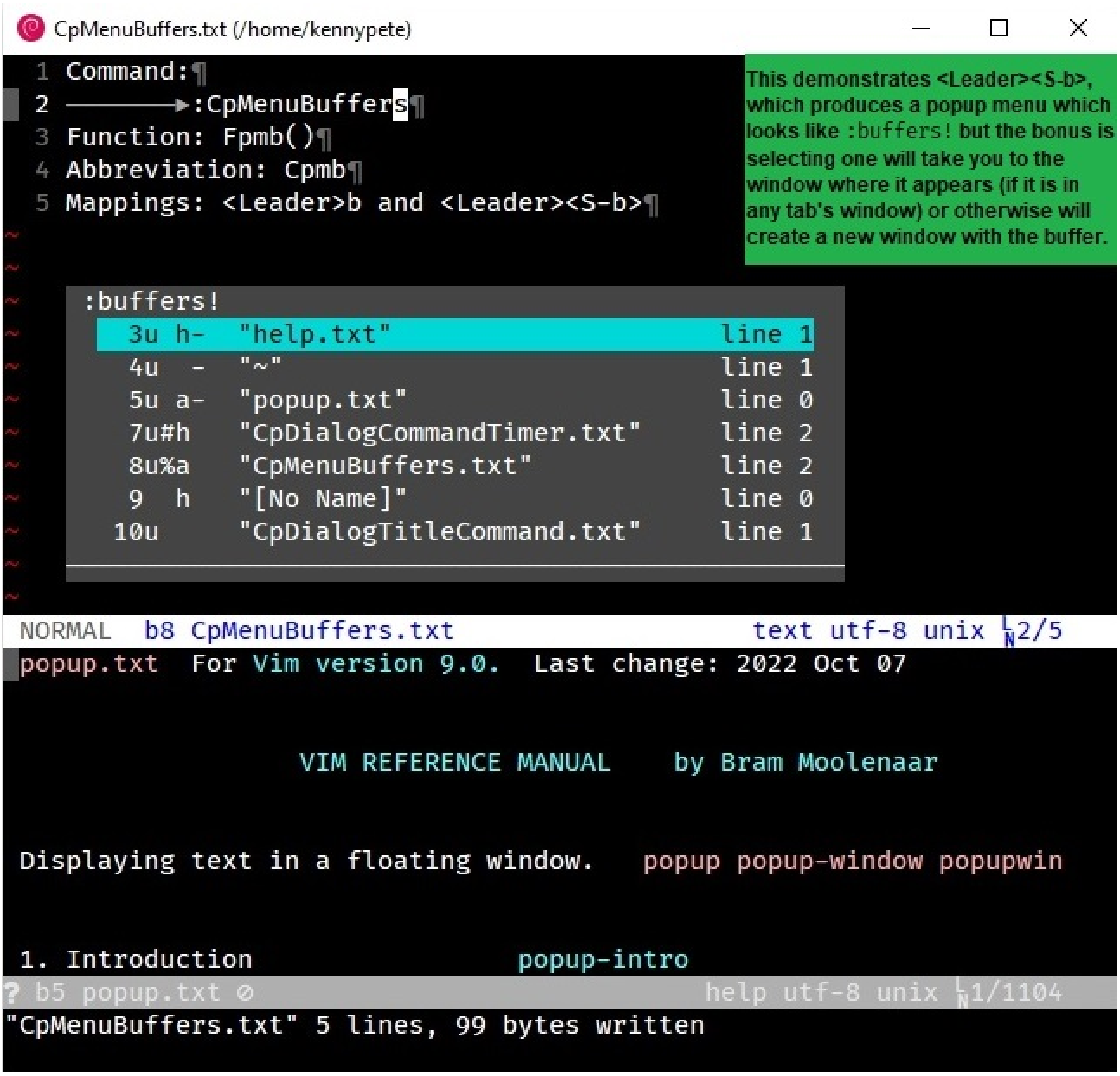
```
:pwd                 X
   0.000012 seconds
```

```
NORMAL  b7 CpDialogCommandTimer.txt     text utf-8 unix 1/4
:CpDialogCommandTimer pwd
```



```
CpDialogTitleCommand.txt (/home/kennypete)
Command:¶
        ───────►:CpDialogTitleCommand¶
Function: Fpdtc()¶
Abbreviation: Cpdtc¶
~
~
~
~
~
~    Working directory   X
~     /home/kennypete
~
~
NORMAL  b10 CpDialogTitleCommand.txt              text utf-8 unix ␤2/4
:CpDialogTitleCommand echo "Working directory  " | pwd
```

This demonstrates using command line abbreviation Cpdtc, which expands to CpDialogTitleCommand. The title may be the output of a command, in this case echoed. The vertical bar separates the commands - in this case pwd - with the second command's output appearing in the body of the popup dialog.



```
CpMenuBuffers.txt (/home/kennypete)
Command:¶
        ───────►:CpMenuBuffers¶
Function: Fpmb()¶
Abbreviation: Cpmb¶
Mappings: <Leader>b and <Leader><S-b>¶
~
~
~  :buffers!
~     3u h-   "help.txt"                      line 1
~     4u  -   "~"                             line 1
~     5u a-   "popup.txt"                     line 0
~     7u#h    "CpDialogCommandTimer.txt"      line 2
~     8u%a    "CpMenuBuffers.txt"             line 2
~     9  h    "[No Name]"                     line 0
~    10u      "CpDialogTitleCommand.txt"      line 1
~
~
NORMAL  b8 CpMenuBuffers.txt                      text utf-8 unix ␤2/5
popup.txt  For Vim version 9.0.  Last change: 2022 Oct 07


                   VIM REFERENCE MANUAL    by Bram Moolenaar


Displaying text in a floating window.    popup popup-window popupwin


1. Introduction                          popup-intro
? b5 popup.txt ⊘                             help utf-8 unix ␤1/1104
"CpMenuBuffers.txt" 5 lines, 99 bytes written
```

This demonstrates <Leader><S-b>, which produces a popup menu which looks like :buffers! but the bonus is selecting one will take you to the window where it appears (if it is in any tab's window) or otherwise will create a new window with the buffer.

2. Mappings

<Leader>b and <Leader><S-b>

Unless the user already has it mapped, `<Leader><S-b>` is mapped, in Normal mode only, to `:CpMenuBuffers buffers!`, demonstrated immediately above.

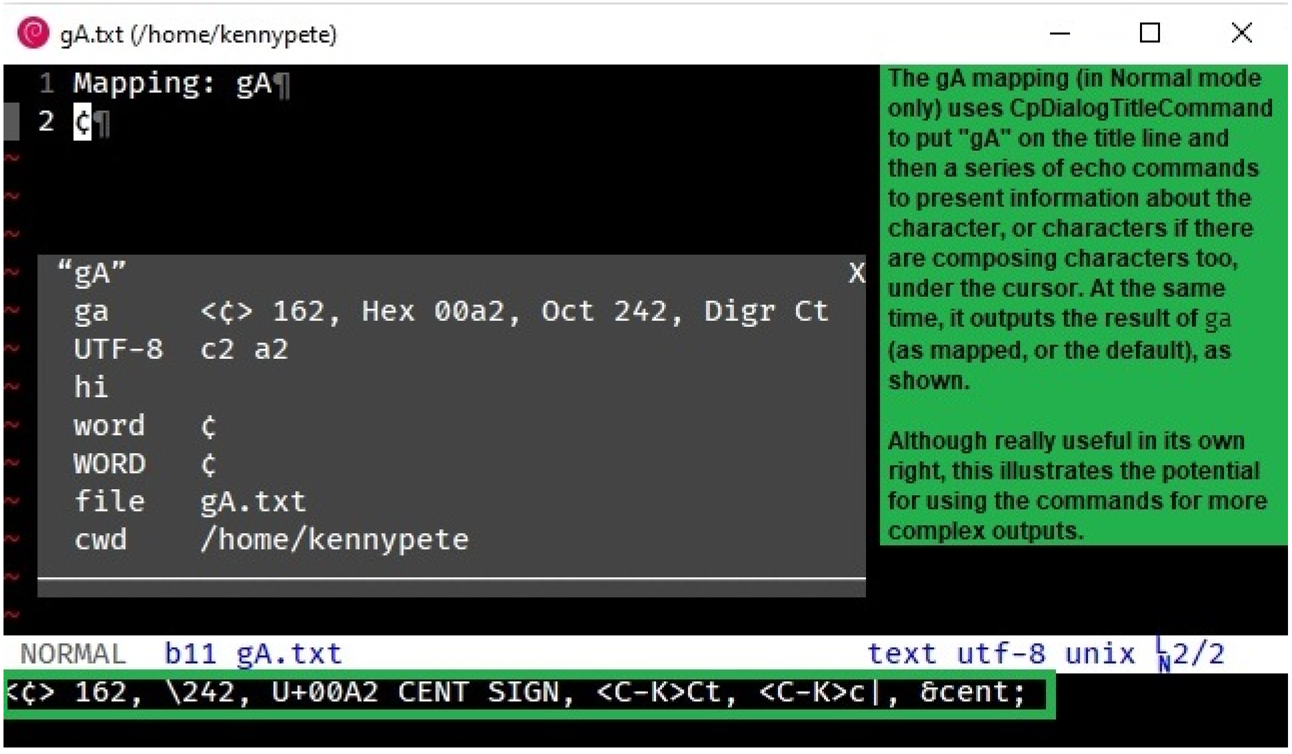
Similarly, `<Leader>b`, unless the user already has it mapped, is mapped, in Normal mode only, to `:CpMenuBuffers buffers`. (The only difference is `<Leader>b` will not show unlisted buffers, which in some instances is preferable.)

gA

The second mapping is `gA`. This provides an extension to the builtin command, `ga` (*aka* :ascii). By default, that command provides information about the character (and combining character(s), when applicable) under the cursor, i.e., the Unicode code point(s) in decimal, hexadecimal, and octal. The `gA` mapping expands on that to provide *lots* of additional information in a popup dialog window:

UTF8	The UTF8 hex values of the bytes used in the character(s) (Equivalent to the g8 command)
hi	Highlight group (only when applicable, otherwise blank)
word	The word under the cursor
WORD	The WORD under the cursor
file	The current filename. (This is blank if there's no file and is equivalent to `:echo expand('%:t')`)
cwd	The current working directory. (Equivalent to `:echo getcwd()`)

Illustrating this in action:



```
1 Mapping: gA¶
2 ¢¶

"gA"                                          X
 ga      <¢> 162, Hex 00a2, Oct 242, Digr Ct
 UTF-8   c2 a2
 hi
 word    ¢
 WORD    ¢
 file    gA.txt
 cwd     /home/kennypete

NORMAL  b11 gA.txt                    text utf-8 unix 2/2
<¢> 162, \242, U+00A2 CENT SIGN, <C-K>Ct, <C-K>c|, &cent;
```

The gA mapping (in Normal mode only) uses CpDialogTitleCommand to put "gA" on the title line and then a series of echo commands to present information about the character, or characters if there are composing characters too, under the cursor. At the same time, it outputs the result of ga (as mapped, or the default), as shown.

Although really useful in its own right, this illustrates the potential for using the commands for more complex outputs.

If you use Tim Pope's vim-characterize plugin, `ga` will display the HTML5 named character references, emoji, Unicode name, and all digraphs, when applicable. For example, `ga` on the character 😀, U+1F600, will display, in the cmdline statusmsg area:

```
<😀> 128512, U+1F600 GRINNING FACE, :grinning:
```

Another example: a̅ (an 'a' with a combining macron, U+0061,U+0305) will display:

```
<a> 97, \141, U+0061 LATIN SMALL LETTER A + < ̅> 773, U+0305
COMBINING OVERLINE
```

3. Borderchars

Popups may have several options set (refer popup_setoptions()). One of those options is `borderchars`, which is a list with characters that are used for displaying the border around a popup. The example in Vim's help is:

```
['-', '|', '-', '|', '┌', '┐', '┘', '└']
```

Those characters have the benefit of being ones that display satisfactorily with most fonts: hyphen (U+002D), vertical line (U+007C), and box drawing characters (U+250C, U+2510, U+2518, and U+2514).

A downside to using those default characters is that they do not *join* together well. They may end up looking like this, depending on factors such as your operating system, font, etc.:

```
  ┌---------------------┐
  | Default borderchars |
  └---------------------┘
```

To address this, and to provide optionality, this plugin uses the variable `g:borderchars`. It enables the user to determine, in their ~/.vimrc, their own border characters for the popups created by this plugin. If `g:borderchars` has not been set, the following default list is used (chosen because it is unobtrusive and should work with any font). It uses an em dash (i.e., U+2014) for the bottom border and bottom corners, and a space for everything else:

```
[' ', ' ', '—', ' ', ' ', ' ', '—', '—']
```

If you want no borders on the popup windows, add this to your ~/.vimrc:

```
let g:borderchars = [' ']
```

4. Installation

Vim before 8.2.3434 / Neovim: vim-popped neither works with Vim versions before 8.2 patch 3434 nor any version of Neovim. That is because:

1. Vim versions before 8.2 patch 3434 lack patches that are required to render popup windows produced by vim-popped.
2. Neovim does not have any of Vim's builtin popup window commands.

There are three installation methods outlined here. Linux is presumed, so .vimrc (not _vimrc), etc.[2]

Method 1. Using packadd! in your .vimrc

This is a contemporary way to install plugins. It uses Vim's native packadd! functionality.

Either
`git clone https://github.com/kennypete/vim-popped ~/.vim/pack/plugins/opt/vim-popped`
Or
Download the .zip from https://github.com/kennypete/vim-popped and unzip the contents within the folder vim-popped-main to `~/.vim/pack/plugins/opt/vim-popped`

In your ~/.vimrc, add the line, `packadd! vim-popped`. (If you want to turn vim-popped off, delete or comment out that line.)

Method 2. Vim's packages method, automatically

Similar to the steps above, except substitute `start` for `opt`. This is a less versatile method because to turn the plugin off you need to move it out of the `start` directory. So, it is easier in a way, though neither as transparent nor as flexible.

Method 3. Using a plugin manager

For example, vim-plug (NB: using "shorthand notation").

In the vim-plug section of your .vimrc, add `Plug 'kennypete/vim-popped'` between `call plug#begin()` and `call plug#end()`. Reload your .vimrc and then `:PlugInstall`.

5. Licence

[1] This plugin has been built with both vim9script and vimscript functions. The main script, in vim-popped/plugins/vim-popped.vim, uses vim9-mix capability, determining the script version based on v:version and has(), testing for 8.2 and patch >=4057. If 8.2 and patch <4057, but >=3434, vimscript functions/commands are used.

[2] If your operating system is Windows, instead of `~/.vim/` use `$HOME\vimfiles\` or `~/vimfiles/` (PowerShell), or `%USERPROFILE%\vimfiles\` (cmd.exe).