Workflow Notes

Hwan Goh

17th May 2020

1 Vim Tips and Tricks

Some good references are [8, 42]. A more advanced tutorial that I haven't watched yet is [7]. A document on effective text editing written by the creator of Vim can be found at [33].

1. In Linux based systems, put 'setxkbmap -option "caps:escape' into /.bashrc to map the caps lock key to escape.
2. To set Vim as your default text editor, use 'sudo update-alternatives –config editor'.
3. Suppose the cursor is in the middle of a word. Whilst 'cw' and 'dw' will change/delete until the end of the word, 'ciw' and 'diw' to change or delete the whole word, thereby not requiring you to move the cursor to the beginning of the word first.
4. To reformat a paragraph, use 'gq'. Reformatting includes enforcing the wrap limit which can be set in your vimrc with 'set tw=<number>'. Other options can be se such as indentation. Note that this command requires a minimum of two lines, so you'll need to at least use 'gqj' and at most use 'gjG' for the rest of the document under the line currently under the cursor.
5. Use 'm<key>to set a mark to <key>then <key>to jump to it. Note that if you set it to the capitalized version of <key>then the jumping can occur between buffers. To see your list of marks, use ':mark'.
6. Use '"<key>' then an action like 'y' or 'd' to store text in the *register* <key>. Then use '"<key>p' to paste it. Note that using 'd' will automatically store it in the register at register x and 'p' will automatically paste whatever is in register 'x'. Use ':reg' to see the register list.
7. Use 'f<key>' to forward search for <key>. To backwards search, use 'F<key>'. To repeat the search, use ';'.
8. The 't' stands for 'til'. For example, 'dt=' will delete up to and not including an = sign.
9. The '*' can be used to search for the word under cursor.
10. 'J' is used to join the line below to the line currently under the cursor. This is useful for reformatting improperly wrapped lines but in general 'gq' is more useful for this purpose.
11. 'zt' or 'z¡CR¿' will put the line under the cursor to the to of the window. 'z.' will put the line to the center of the window and 'z-' will put it to the bottom of the window.
12. 'z=' will give spelling suggestions for the word under the cursor.
13. '[s' and ']s' to cycle backwards and forwards through misspelled words.
14. Use 'g$' to go to the end of an unwrapped line.
15. If you type a long line of text in insert mode, this counts as one action. Therefore, if you press undo, the whole line will be deleted. To break the undo chain, use <c-g>u while in insert mode. Alternatively, you could map space to <c-g>u so that the undo chain is broken whenever a space is added in insert mode: inoremap <Space><Space><c-g>u
16. Use the accent '<shift+6>' in insert mode to move to the first non-white space character in the line.
17. Use '%' while your cursor is over a parenthesis, square or curly bracket to move to the corresponding open/-closing parenthesis or bracket.

18. Use ':ls' to see the list of buffers then ':b' and the number to select one. A useful mapping for this process is: nnoremap <leader>b:ls<cr>:b<space>

19. Use '<number>+<c-6>' to switch to numbered buffer.

20. Use '<c-f>' in command line mode to view the command history in a buffer.

21. In general, all yank, change or delete actions such as 'y, c, d, x' etc will register the text object into the "" register. However, any yank action will also register the text object into the "0 register and any delete or change action will register the text object into the "- register.

22. Use 's' or 'S' to substitute. This is useful when you want to replace one letter with multiple letters.

23. When using search '/<word>', you can use <c-g >and <c-t>to cycle through them without confirming your search with <CR>.

24. Use '<c-r>' in command line mode to paste from the register.

25. Use '<c-r>' in insert mode to paste from the register.

26. Use <c-w>N in a Vim terminal to switch to normal mode, which allows you to navigate as if you were in Vim.

27. Use '<c-z>' to suspend Vim and return to shell. Use 'fg' in shell to resume the suspended program. You can also use

```
stty susp undef
bind '"\C-z":"fg\015"'
```

    in your bashrc to, overall, set '<c-z>' to toggle a suspended progran.

28. Using 'ctags -R' in terminal creates a tag file. Then, in vim, use '<c-] >' while the cursor is over a function name to jump to the code defining the function. You can also add the following shortcut to your vimrc:

```
command! MakeTags !ctags -R .
```

    so that tags can be created from the command line within Vim. Note that you need to add

```
set tags=tags;/
```

    to your vimrc so that ctags will check the current folder for tags file and keep going one directory all the way to root folder. See [2] for more details.

2 Plugin Management

The first part of this section follows [34]. First, we work towards turning /.vim into a git repository:

1. Move ∼/.vimrc into ∼/vim.
2. When vim boots, it's still going to look for .vimrc in the home directory. To ensure that it looks for vimrc in the ∼/.vim directory, we can create a symbolic link to that file using 'ln -s ∼/.vim/vimrc ∼/.vimrc'.
3. Make ∼/.vim into a git repository.

An issue now is if you install a plugin that itself is a git repository, you lose the version-control capabilities of that plugin. To circumvent this issue, we use a plugin manager; in this case *Pathogen*.

2.1 Pathogen

The pathogen plugin makes it possible to cleanly install plugins as a bundle. Rather than having to place all of your plugins side by side in the same directory, you can keep all of the files for each individual plugin together in one directory (see video from first link for example). This makes installation more straightforward, and also simplifies the tasks of upgrading and even removing a plugin if you decide you no longer need it since they are carefully segregated from each other. For a good tutorial on Pathogen, see [28].

Following the readme on the repo at [35], to install Pathogen do the following:

1. Run in terminal:
   mkdir -p ∼/.vim/autoload ∼/.vim/bundle &&
   curl -LSso ∼/.vim/autoload/pathogen.vim https://tpo.pe/pathogen.vim

2. Add the following to your vimrc: execute pathogen#infect()

Now any plugins you wish to install can be extracted to a subdirectory under /.vim/bundle, and they will be added to the 'runtimepath'. For example, to install "sensible.vim", simply run: "cd /.vim/bundle && git clone https://github.com/tpope/vim-sensible.git".

2.2 Submodules: Installing Git Repositories Within Git Repositories

This section follows [34]. Now that we can install plugins via Pathogen, let's see how we preserve the version control capabilities of our plugins. As a worked example, let us install the *Vimtex* plugin:

1. cd ∼/.vim

2. Now to clone a git repository into the bundle directory, use:
   git submodule add https://github.com/lervag/vimtex.git bundle/vimtex

Now, to upgrade this plugin, use:

1. cd ∼/.vim/bundle/vimtex

2. git pull origin master

To update ALL of your plugins, use:

1. cd ∼/.vim

2. git submodule foreach git pull origin master

2.3 Importing Your Vim Configuration and Plugins To a New Machines

One of the main benefits of version controlling your Vim configuration and plugins is the ease of which they can be imported into a new machine. To do so, use the following:

1. cd ∼

2. git clone <git repo url>∼/.vim

3. ln -s ∼/.vim/vimrc ∼/.vimrc

4. cd ∼/.vim

5. git submodule init

6. git submodule update

2.4 Vim-plug

Whilst Pathogen is the most basic plugin manager, there are limitations when porting your setup to a new machine:

1. When you run git submodule update, you'll pull the latest versions of the plugins from their respective repositories. So unless you've noted down somewhere all the commit IDs for your favourite version of each plugin, your overall plugin collection cannot be preserved when porting to a new machine.

2. Suppose one of the plugins is no longer being maintained by the owner and suppose also that you've made your own changes to the plugin. If you were to pull your configuration on a new machine, you will pull the latest version of the plugin; that is your changes will not be ported. Further, on your own repository for your configuration, the directories containing the plugins will be treated as repositories. Unless you manually install the plugins, there is no way to preserve your changes onto Github.

The plugin manager Vim-plug [21] has a solution to the first problem. This plugin manager is extremely simple to use:

1. Run in terminal:
   curl -fLo ∼/.vim/autoload/plug.vim –create-dirs https://raw.githubusercontent.com/junegunn/vim-plug/master/plug.vim

2. In your vimrc, add the line:
   call plug#begin('~/.vim/plugged')
3. To include a plugin you wish to install under the line above, in your vimrc add the following line:
   Plug 'https://github.com/lervag/vimtex.git'
4. Under 'Plug <url >' of your last plugin, in your vimrc add the following line:
   call plug#end()
5. Reload your vimrc and use the command ':PlugInstall' while in vim. Now all your plugins are installed.

Now to preserve the versions of your plugin:

1. In Vim, run the command ':PlugSnapshot! <filename >.vim'.
2. To restore the state of your plugins, in vim run the command:
   :source ~/.vim/<filename >.vim
   or, in terminal:
   vim -S ~/.vim/<filename >.vim

See the bradagy's answer in the reddit post [4]. To port your configuration to a new machine:

1. cd ~
2. git clone <git repo url >~/.vim
3. cd ~/.vim/vimrc
4. :PlugInstall
5. :source ~/.vim/<filename >.vim

To uninstall plugins:

1. Delete 'Plug <url >' line from your vimrc
2. :PlugClean

3 Native Plugin Management

Vim-plug provides a solution to the first of the two issues mentioned in the previous section, but not the second. To address the second issue, we can use Vim's native plugin management system to install a plugin and delete the .git repository. That way, we can track the files and any changes in our own git repository. This section follows [38]. For a more detailed explanation, see [32] or [37].

The package feature of Vim 8 follows a pathogen-like model and adds the plugins found inside a custom-path ~/.vim/pack/ to Vim's runtime path. You can check the version of Vim installed using 'vim –version'. To install using this native feature, we use the following steps:

1. mkdir -p ~/.vim/pack/plugins/start/
2. cd ~/.vim/pack/plugins/start/
3. git clone <url>or git submodule add <url >

and that's it! To remove a plugin, simply remove its directory: rm -r ~/.vim/pack/plugins/start/<plugin_directory_name >if the plugin was cloned. Or, if it was added as a submodule, use:

1. git submodule deinit vim/pack/shapeshed/start/<plugin_directory_name >
2. git rm vim/pack/shapeshed/start/<plugin_directory_name >
3. rm -Rf .git/modules/vim/pack/shapeshed/start/<plugin_directory_name >

To import your vim configuation and plugins to your new machine, simply use the same steps as required for Pathogen:

1. cd ~

2. git clone <git repo url>~/.vim
3. ln -s ~/.vim/vimrc ~/.vimrc
4. cd ~/.vim
5. git submodule init
6. git submodule update

and also, to update all your plugins, you again use

1. cd ~/.vim
2. git submodule foreach git pull origin master

If you have your own plugins that are not cloned from external repositories (i.e. you've written your own .vim files), you can simply create a directory /.vim/plugin and copy your plugins into there.

In conclusion, although Pathogen seems to be the most cumbersome, it could be the case that the servers do not have Vim 8 installed and so the native plugin manager will not work nor do they allow Vim-plug to install plugins. In such a case, Pathogen is your best bet.

4 Vimtex

This section discusses the Vimtex [30] plugin. See also the vimways article [43] for a good overview.

4.1 Installing and Running Vimtex

Assuming you are using the Pathogen plugin manager and version controlling your /.vim directory, follow the steps used in Section 2.2. Then use ':Helptags' which is Pathogen's method for generating help tags. With this, we can use ':h vimtex' to see the manual for Vimtex. To confirm that the plugin works, type ":VimtexInfo" to see a summary of the tex file.

4.2 Compiling a Tex File

The following commands are useful:

- :VimtexCompile # this is a continuous compiler meaning that everytime you save with ":w" it will automatically compile
- :VimtexStop # this stops the continuous compiler
- :VimtexCompileSS # this is a single shot compiler. Note that you have to save your file first
- :VimtexClean # Cleans auxiliary files generated in compilation process

I set the following mappings in my vimrc:

```
autocmd FileType tex nnoremap <F5> :VimtexView<CR>
autocmd FileType tex inoremap <F5> <Esc> :VimtexView<CR>
autocmd FileType tex nnoremap <F6> :w! <bar> :VimtexCompileSS<CR>
autocmd FileType tex inoremap <F6> <Esc> :w! <bar> :VimtexCompileSS<CR>
```

Here are some useful commands I use in my vimrc:

```
" Avoids opening an empty .tx file only to have vimtex recognize it as plain Tex rather than Latex
    let g:tex_flavor = 'latex'

" Use folding. Use zx to unfold and zX to fold all
    let g:vimtex_fold_enabled = 1

" Toggle Error Window On and Off
    autocmd FileType tex map <F4> \le

" Shortcut for Compiling and Viewing PDF
    autocmd FileType tex nnoremap <F5> :VimtexView<CR>
    autocmd FileType tex inoremap <F5> <Esc> :VimtexView<CR>
    autocmd FileType tex nnoremap <F6> :w! <bar> :VimtexCompileSS<CR>
    autocmd FileType tex inoremap <F6> <Esc> :w! <bar> :VimtexCompileSS<CR>
```

```
" VimtexClean on exit
  augroup vimtex_config
    au!
    au User VimtexEventQuit call vimtex#compiler#clean(0)
  augroup END
```

And here are some useful mappings for creating common math related LaTeX tex objects:

```
" Teleportation!
    autocmd FileType tex inoremap <Space><Space> <Esc>/<++><CR>"_c4l

" Inline Stuff
    autocmd FileType tex inoremap ;mm $$<++><Esc>5ha
    autocmd FileType tex inoremap ;bf \textbf{}<++><Esc>6hf}i
    autocmd FileType tex inoremap ;it \textit{}<++><Esc>6hf}i
    autocmd FileType tex inoremap ;bfs \boldsymbol{}<++><Esc>6hf}i
    autocmd FileType tex inoremap ;ct \cite{}<++><Esc>6hf}i
    autocmd FileType tex inoremap ;lb \label{}<++><Esc>6hf}i
    autocmd FileType tex inoremap ;rf \ref{}<++><Esc>6hf}i
    autocmd FileType tex inoremap ;erf (\ref{})<++><Esc>7hf}i

" Environments
    autocmd FileType tex inoremap ;itm \begin{itemize}<CR><CR>\end{itemize}<CR><++><Esc>2kA\item<Space>
    autocmd FileType tex inoremap ;enu \begin{enumerate}<CR><CR>\end{enumerate}<CR><++><Esc>2kA\item<Space>
    autocmd FileType tex inoremap ;aln \begin{align}<CR><CR>\end{align}<CR><++><Esc>2kA

" Math Environments
    autocmd Filetype tex inoremap ;def \begin{definition}<CR><CR>\end{definition}<CR><++><Esc>2kA
    autocmd Filetype tex inoremap ;thm \begin{theorem}<CR><CR>\end{theorem}<CR><++><Esc>2kA
    autocmd Filetype tex inoremap ;lem \begin{lemma}<CR><CR>\end{lemma}<CR><++><Esc>2kA
    autocmd Filetype tex inoremap ;cor \begin{corollary}<CR><CR>\end{corollary}<CR><++><Esc>2kA
    autocmd Filetype tex inoremap ;prp \begin{proposition}<CR><CR>\end{proposition}<CR><++><Esc>2kA
    autocmd Filetype tex inoremap ;prf \begin{proof}<CR><CR>\end{proof}<Esc>1kA

" Math Stuff
    autocmd Filetype tex inoremap ;T ^\mathrm{T}
    autocmd FileType tex inoremap ;sd _{}<++><Esc>6hf}i
    autocmd FileType tex inoremap ;su ^{}<++><Esc>6hf}i
    autocmd FileType tex inoremap ;frc \frac{}{<++>}<++><Esc>12hf}i
    autocmd FileType tex inoremap ;mbb \mathbb{}<++><Esc>6hf}i
    autocmd FileType tex inoremap ;lrp \left(\right)<++><Esc>12hf(a
    autocmd FileType tex inoremap ;lrs \left[\right]<++><Esc>12hf[a
    autocmd FileType tex inoremap ;lrn \left\lVert\right\rVert<++><Esc>15hi<Space>
```

4.3 Forward and Backwards Searching with Synctex

This section follows [12]. For this, you will need a Vimtex server which allows you to do forward and backward to navigate between corresponding sections of the tex file and the pdf. You will also need the pdf viewer *Zathura*. To install, simply use 'sudo apt-get install zathura'. Then, in your vimrc, add the following line 'let g:vimtex_view_method = 'zathura".

Following [29], to install just use 'sudo apt-get install vim-gnome'. Finally, to edit a tex file with navigation capabilities:

1. In the directory containing the tex file, use:
   vim –servername <servername><texfile>.tex
2. While in tex file, to jump to the text on the pdf corresponding to the line under your cursor, use:
   <leader>lv
3. Move your mouse cursor over some text on the pdf. Then, to jump to corresponding text on the tex file, use:
   Ctrl + left-click

Note that <leader>lv forward search works without the Vim server; it's the backwards search that requires the Vim server.

4.4 Other Useful References

- Good article overviewing Vimtex [43]
- Nice compilation of Vimtex commands [12]

- Quick guide on the basics of Vimtex [18]
- For a comparison with other Vim LaTeX plugins [30]
- Some useful mappings such as the teleportation trick [39, 40].
- Note-taking using Vim and LaTeX for math lectures [6]
- Vim and LaTeX on MacOS [10]

5 IPython

The following mapping may prove useful for starting an IPython terminal in Vim:

```
nnoremap <leader>P :botright vertical terminal ipython --no-autoindent<CR><C-w><left>
```

5.1 sendtowindow Plugin

When coding in Python with a Vim terminal on the side running IPython, I use the *sendtowindow* plugin [25] to send lines of code to the terminal. See [24] for a Reddit post from the author discussing the plugin. It's a very simple plugin that allows the use of vim motions to move lines of text to terminals left, right, above or below the Vim window. I use the following maps:

```
let g:sendtowindow_use_defaults=0
nmap ,sr <Plug> SendRight
xmap ,srv <Plug> SendRightV
nmap ,sl <Plug> SendLeft
xmap ,slv <Plug> SendLeftV
nmap ,su <Plug> SendUp
xmap ,suv <Plug> SendUpV
nmap ,sd <Plug> SendDown
xmap ,sdv <Plug> SendDownV
```

Note that these mappings must not be noremaps (mappings that are non-recursive). For an explanation on why this is the case, please see [22]. Any one of these mappings will specify to either send some text in normal mode or in visual mode. When in normal mode, to specify which text to send, use the usual Vim movements. For example, ',sr$' will send from the cursor to the end of the line to the window on the right. However, if you are in the middle of a line, it may become tedious having to first go to the beginning of the line and having to type '$'. Therefore, we can define line objects using the following maps:

```
onoremap <silent> <expr> - v:count==0 ? ":<c-u>normal! 0vg_<CR>" : ":<c-u>normal! V" . (v:count) . "jk<CR>"
onoremap <silent> <expr> i- v:count==0 ? ":<c-u>normal! ^vg_<CR>" : ":<c-u>normal! ^v" . (v:count) . "jkg_<CR>"
```

With '-' alone the indentation is left intact and 'i-' is without indentation. So, for example, ',sr-' will send the line under the cursor to the window on the right with indentation and ',sr8-' will send four lines under the cursor to the window on the right with indentation. Note that this is important for Python as if you try send a for loop with 'i-' and leave out the indentation, then it may not run.

I have extended the plugin with SendTextToTerminalRight command and SendVariableRight() and SendMarkedSectionRight() functions (I've also coded for the other three directions as well). Using these, we can send clear, %reset and run code commands as well as variable and marked sections of code to the IPython terminal.

```
" sendtowindow for IPython in Vim Terminal: clear, reset, run code, run variable, run marked section
    noremap ,sL :SendTextToTerminalRight clear<CR>
    noremap ,sD :SendTextToTerminalRight %reset -f<CR>
    noremap ,sR :w!<CR>:SendTextToTerminalRight run <c-r>%<CR>
    nmap ,sV <Plug>SendVariableRight
    nmap ,sM <Plug>SendMarkedSectionRight
```

For the map that runs the Python code, we leverage the fact that '<c-r>' in command mode will prime pasting from the register and % in the register stores the file name.

The SendMarkedSectionRight() function has been coded such that the section must be marked with 'x' on the top of the section and 'z' on the bottom. The plugin vim-signature [27] may come in handy for manipulating the marks. In particular, the plugin displays marks and the command 'dm<mark name>' deletes the mark; so 'dmq' above deletes the mark called 'q'.

5.2 Slimux plugin

There are many plugins that allow interaction between Vim and tmux. For example, there is Vim-Slime [19] and Vimux [3]. The one I use is Slimux [11]. There is a blog post about it here [41]. As discussed in the blog post, Slimux differs from Vimux in that it is more disjoint from tmux. In particular, Vimux will create a pane on which you must run your commands whereas Slimux allows you to select the pane you wish to use. Also, unlike Vim-Slime, you do not need to manually type in the pane to select it; in Slimux an interactive prompt is given. It's also worth noting that Vim-Slime supports many terminal multiplexers such as GNU Screen, kitty and Vim's native terminal.

However, with Slimux, there is an issue with sending commands to IPython where the indentation is constantly carried over [26]. There are three proposed fixes [31, 23, 44]. The first modifies python.vim, the second modifies both slimux.vim and python.vim by implementing IPython's cpaste function and the third modifies a single line of slimux.vim. Since the repo owner hasn't pulled any of these fixes, I opted for the third fix which requires changing line 328 of slimux.vim in the SlimuxSendCode() function:

```
let b:code_packet["text"] = a:text
```

to:

```
b:code_packet["text"] = "\e[200~" . a:text . "\e[201~\r\r.
```

There is also an issue with the SlimuxSendKeys() function. This function sends keys to the terminal using the 'tmux send-keys' syntax. A simple fix is to change line 228 of slimux.vim from

```
call system(g:slimux_tmux_path . ' send-keys -t " . target . " " . keys)
```

to

```
call system(g:slimux_tmux_path . " send-keys -t " . target . " " . keys)
```

It's surprising that this small typo went unnoticed. Note that this repository has not been updated in years and so all pull requests have not been fulfilled. Therefore, if you wish to track the fixes you've made, you may want to install the plugin manually so that it does not follow the original repository. This was disussed in Section 3.

I use the following mappings:

```
" Primer for Vim motions to select text to be sent
    nmap ,t <Plug>SlimuxREPLSendWithMotion

" Send text selected in visual mode
    vnoremap ,t :SlimuxREPLSendSelection<CR>

" Send terminal commands
    nnoremap ,tX :SlimuxShellPrompt<CR>
    nnoremap ,tT :SlimuxShellRun<Space>
    nnoremap ,tE :SlimuxShellRun exit<CR>

" Send keys using the 'tmux send-keys' syntax
    nnoremap ,tK :SlimuxSendKeys<Space>
    nnoremap ,tC :SlimuxSendKeys<Space>c-c<CR>

" Slimux for IPython in tmux terminal : IPython, clear, reset, run code, run variable, run marked section
    nnoremap ,tP :SlimuxShellRun ipython<CR>
    nnoremap ,tL :SlimuxShellRun clear<CR>
    nnoremap ,tD :SlimuxShellRun %reset -f<CR>
    nnoremap ,tR :w!<CR>:SlimuxShellRun run <c-r>%<CR>
    nmap ,tV <Plug>SlimuxREPLSendVariable
    nmap ,tM <Plug>SlimuxREPLSendMarkedSection
```

where SlimexShellRun awaits a command to send to shell and SLimuxSendKeys sends key inputs using the 'tmux send-keys' syntax. The SlimuxREPLSendWithMotion was not part of the original slimux plugin. My code is as follows:

```
function! s:GetTextWithMotion(type)
  let s:saved_registert = @t

  " Obtain wanted text
  if a:type ==# "char"
    keepjumps normal! `[v`]"ty
    call setpos(".", getpos("']"))
  elseif a:type ==# "line"
    keepjumps normal! `[V`]$"ty
    call setpos(".", getpos("']"))
```

```
  endif
  let text = @t

  " Restore register
  let @t = s:saved_registert

  return text
endfunction

function! s:SlimuxREPLSendWithMotion(type)
   let text = s:GetTextWithMotion(a:type)
   call SlimuxSendCode(text)
endfunction

nnoremap <silent> <Plug>SlimuxREPLSendWithMotion :<C-U> set operatorfunc=<SID>SlimuxREPLSendWithMotion<CR>g@
```

The 'g@' sets the mark '[' at the beginning of the motion and ']' at the end of the motion. This can then be leveraged in functions. The '@' in the context of functions refers to the register.

5.3 Other Useful References

- A good Reddit thread on workflow with Vim and IPython can be found at [1]. Note that the majority of the responses indicate that they use vim-slime and tmux.
- The vim-tmux-runner plugin [9] may be worth checking out.
- The vim-ipython-cell plugin [13] may be worth checking out. There is a reddit post [14] by the author on this plugin. Note that this plugin is leverages vim-slime. According to the author, the main contribution of this plugin is that it provides many ways to define and run cells, even using Vim marks.
- Another lightweight workflow can be found in [15]. For this, you will need three plugins: Vimux [3], vim-pyShell [16] and vim-cellmode [20]. However, I have trouble getting this to work. Starting and stopping a pyShell session works fine as well as sending one line of code, but I have issues sending over cells.

References

[1] abdeljalil73. Anyone uses vim for python and data science? https://www.reddit.com/r/vim/comments/gjb97y/anyone_uses_vim_for_python_and_data_science/.

[2] N. Ben. Vim: Difficulty setting up ctags. source in subdirectories don't see tags file in project root. https://stackoverflow.com/questions/5017500/vim-difficulty-setting-up-ctags-source-in-subdirectories-dont-see-tags-file-i.

[3] benmills. vimux. https://github.com/benmills/vimux.

[4] bradagy. Remember to update your plugins! https://www.reddit.com/r/vim/comments/6wy3kb/remember_to_update_your_plugins/.

[5] M. Cantor. How to do 90% of what plugins do (with just vim). https://www.youtube.com/watch?v=E-ZbrtoSuzw.

[6] G. Castel. How i'm able to take notes in mathematics lectures using latex and vim. https://castel.dev/post/lecture-notes-1/#vim-and-latex.

[7] L. Chang. Vim: Tutorial on customization and configuration (2020). https://www.youtube.com/watch?v=JFr28K65-5E&list=FLVuBBjy7NC7Q0xVUm8SEjrw&index=3&t=3083s.

[8] L. Chang. Vim: Tutorial on editing, navigation, and file management (2018). https://www.youtube.com/watch?v=E-ZbrtoSuzw.

[9] christoomey. vim-tmux-runner. https://github.com/christoomey/vim-tmux-runner.git.

[10] J. V. Dyke. Getting started with latex and vim. https://web.ma.utexas.edu/users/vandyke/notes/tex/getting_started/getting_started.pdf.

[11] esamattis. Slimux. https://github.com/esamattis/slimux/blob/master/README.md.

[12] M. Gunther. Vimtex tool for working vim, tex and zathura. https://wikimatze.de/vimtex-the-perfect-tool-for-working-with-tex-and-vim/.

[13] hanschen. vim-ipython-cell. https://github.com/hanschen/vim-ipython-cell.

[14] hanschen. vim-ipython-cell: Run python code and code cells in ipython. https://www.reddit.com/r/vim/comments/by6omc/vimipythoncell_run_python_code_and_code_cells_in/.

[15] G. Hornung. Boosing your data science workflow with vim+tmux. https://towardsdatascience.com/boosting-your-data-science-workflow-with-vim-tmux-14505c5e016e.

[16] G. Hornung. vim-pyshell. https://github.com/greghor/vim-pyShell.

[17] P. Ivanov. vim-ipython. https://github.com/ivanov/vim-ipython.

[18] jdhao. A complete guide on writing latex with vimtex in neovim. https://jdhao.github.io/2019/03/26/nvim_latex_write_preview/.

[19] jpalardy. vim-slime. https://github.com/jpalardy/vim-slime.

[20] julienr. vim-cellmode. https://github.com/julienr/vim-cellmode.

[21] junegunn. vim-plug. https://github.com/junegunn/vim-plug/wiki/tutorial.

[22] justrajdeep. Please help understand how to use ¡plug¿ mapping. https://www.reddit.com/r/vim/comments/78izt4/please_help_understand_how_to_use_plug_mapping/.

[23] karadaharu. Add an option to use cpaste for ipython #68. https://github.com/esamattis/slimux/pull/68.

[24] KKPMW. 'sendtowindow' - a small plugin with operator for sending text to repls (or any other vim window). https://www.reddit.com/r/vim/comments/au8eo5/sendtowindow_a_small_plugin_with_operator_for/.

[25] KKPMW. vim-sendtowindow. https://github.com/KKPMW/vim-sendtowindow.

[26] kmARC. Python repl indentationerror: unexpected indent # 38. https://github.com/esamattis/slimux/issues/38.

[27] kshenoy. vim-signature. https://github.com/kshenoy/vim-signature.

[28] R. Lafourcade. How to use tim pope's pathogen. https://gist.github.com/romainl/9970697.

[29] Y. Lerner. Enable servername capability in vim/xterm. https://vim.fandom.com/wiki/Enable_servername_capability_in_vim/xterm.

[30] K. Lervag. What are the differences between latex plugins? https://vi.stackexchange.com/questions/2047/what-are-the-differences-between-latex-plugins.

[31] lotabout. remove indent for selected blocks #74. https://github.com/esamattis/slimux/pull/74.

[32] manasthakur. Managing plugins in vim: The basics. https://gist.github.com/manasthakur/ab4cf8d32a28ea38271ac0d07373bb53.

[33] B. Moolenaar. Seven habits of effective text editing. https://www.moolenaar.net/habits.html.

[34] D. Neil. Synchronizing plugins with git submodules and pathogen. http://vimcasts.org/episodes/synchronizing-plugins-with-git-submodules-and-pathogen/.

[35] T. Pope. pathogen.vim. https://github.com/tpope/vim-pathogen.

[36] princker. Is it possible to skip over :terminal buffers when using :bn and :bp in vim 8? https://www.reddit.com/r/vim/comments/8njgul/is_it_possible_to_skip_over_terminal_buffers_when/.

[37] T. Ryder. Attack of the 5000-line vimrc. https://vimways.org/2018/from-vimrc-to-vim/.

[38] shapeshed. Vim: So long pathogen, hello native package loading. https://shapeshed.com/vim-packages/.

[39] L. Smith. My dank vim & latex setup (wordcucks btfo once again). https://www.youtube.com/watch?v=Mphdtdv2_xs.

[40] L. Smith. Start turning vim into a /comfy/ ide! https://www.youtube.com/watch?v=Q4I_Ft-VLAg&list=FLVuBBjy7NC7Q0xVUm8SEjrw&index=3&t=462s.

[41] E.-M. Suuronen. Slimux - tmux plugin for vim. http://esa-matti.suuronen.org/blog/2012/04/19/slimux-tmux-plugin-for-vim/.

[42] C. Toomey. Mastering the vim language. https://www.youtube.com/watch?v=wlR5gYd6um0.

[43] J. Woodruff. Latex in vim. https://vimways.org/2019/latex-in-vim/latex-in-vim/.

[44] zcesur. Fix ipython issues when sending multiple lines #80. https://github.com/esamattis/slimux/pull/80.